TURING O'REILLY

Debugging Teams
Better Productivity through Collaboration

进化

从孤胆极客到高效团队

【美】Brian Fitzpatrick Ben Collins-Sussman 著
金迎 译

程序员版《人性的弱点》

提升职业生涯软技能

探讨**领导力、合作、沟通、高效**等团队成功关键因素

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

目录

封面

内容提要

版权声明

O'Reilly Media, Inc.介绍

本书赞誉

献词

文前

第二版前言

简介

致谢

第1章 天才程序员神话

- 请帮我隐藏代码
- 天才神话
- 隐藏是有害的
- 团队为王
- 三大基石
- HRT实战
- 下一步

第2章 打造团队文化

- 什么是团队文化
- 团队文化的重要性

团队文化与人
成功团队文化的沟通模式
高层同步
日常讨论
使用问题跟踪系统
工程中的沟通
一切为了产品
第3章 群龙不可无首
自然界里无真空
经理是个贬义词
服务型领导
反模式
领导模式
人同植物
内在激励和外在激励
总结
第4章 应对有害之人
定义“有害”
巩固团队
发现威胁
去除毒素
总结
第5章 组织操控的艺术
好公司、坏公司，以及策略

理想情况
通常情况
操控组织
备用计划：撤退
希望尚存
第6章 用户也是人
管理公众认知
你的软件可用性如何
设计很重要
管理与用户的关系
记住用户
后记
推荐书目

封面

TURING O'REILLY

Debugging Teams
Better Productivity through Collaboration

进化

从孤胆极客到高效团队

【美】Brian Fitzpatrick Ben Collins-Sussman 著
金迎 译

程序员版《人性的弱点》

提升职业生涯软技能

探讨**领导力、合作、沟通、高效**等团队成功关键因素

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

内容提要

本书旨在提高人与人合作的能力。作者以其自身的经历为基础，阐明了团队合作的重要性，提出了加强合作的具体方法，并辅以实例进行了深入分析。全文主要从三个角度介绍了团队合作的方法：如何处理团队中有关人的方面；如何在良好或不佳的公司中工作；如何与用户合作以创造更优秀的产品。如果你想提高创新的效果和效率，那么本书就是答案。

本书适合所有与他人合作从事创新类工作的团队成员阅读。

本书由“ePUBw.COM”整理，ePUBw.COM 提供最新最全的优质电子书下载！！！

版权声明

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立

了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’ 回顾过去，Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也

不错。”

——Linux Journal

本书赞誉

“这本书语言风趣，引起了极客的强烈共鸣！即使你不是极客，书中的建议也值得一读。”

——Vint Cerf，谷歌首席互联网布道师

“我与工程师共事30多年了，在此期间领悟到，工程不仅仅需要科学和技术，人也同样重要，但大多数工程师很少或从未尝试了解如何与他人协作。如果你想提升创新效果，提高创新效率，那么这本书就是答案。”

——Dean Kamen，美国DEKA研发公司创始人

“公司在创业初期失败的主要原因之一是创始人之间的矛盾。这本书是保持技术人员间良好关系的自助书。书中的专业意见将帮助你把一支2人团队扩大到20 000人以上。”

——Renee DiResta，The Hardware Startup作者

“Ben和Fitz写出了我一直在实践而未能确切表述的东西。”

——Guido van Rossum，Python之父

“这是一本指导如何完成工作的手册，其应用范围远不限于工程团队。所有团队都应理解，成功的创造活动不仅需要智商，情商也同样重要。这本书思虑周全，见解深刻，而且非常具有实用价值。”

——Amanda Lannert，Jellyvision公司CEO

“如果想打造一支专注于交付优秀软件的团队，那么你就需要阅读这本书。Ben和Fitz将谦虚、尊重和信任等感性概念极好地转换为实际的建议，即便最易持怀疑态度的开发者也能领会。”

——Eric Lunt，Signal公司首席技术官

“这是一本极好的书。它讨论了计算机编程中最难的问题，即如何与其他计算机编程人员相处。我要给Samba团队的每个成员都买一本。”

——Jeremy Allison，Samba的创建者之一

“你可能听过‘10倍程序员’的说法，即顶尖程序员的效率要比普通程序员高一个数量级。做大事不仅需要经验和很强的技术能力，还要能站在同事和用户的角度思考问题。再多的聪明才智或知识也不能替代后者，但这本书将帮助你打磨自己的软技能，让你对世界产生更大的影响。”

——Bob Lee，Square公司前首席技术官

“Fitz和Ben阐述了简单的信条——谦虚、尊重和信任，并辅以大量的示例和故事。他们分享的经验和智慧将帮助团队中的软件工程师，即我们中的大多数人，变

得更加高效、多产。”

——Greg J. Badros，Prepared Mind Innovations公司CEO

“软件是由人组成的。应用这本书中的原则，一支运作良好的团队能够比任何黑客思虑得更周全，编码更优秀，交付更及时。程序员们，学习起来吧！”

——Johnathan Nightingale，Hubba公司首席产品官

“这本书是程序员版的《人性的弱点》。本书充满了清晰且可操作的建议，告诉你如何在技术团队中更快乐、更高产、更有效地工作。这是一本急需的好书。”

——Adrian Holovaty，Django创建者之一

“请寄一本给Poul-Henning Kamp，由FreeBSD核心团队转交，1994年3月前送达。”

——Poul-Henning Kamp，FreeBSD项目提交者

“Ben和Fitz不是要颂扬个人程序员神话，而是要破除它。他们用一系列文章教会右脑发达的工程师如何攻破他们遇到过的最复杂的系统：人群，并以此宣告了个人程序员神话的终结。这本书说明了最具人性的软件是由最高效的人类团队实现的，并展示了如何打造这样的团队和软件。”

——John Tolva，PositivEnergy Practice创始人兼总裁

“这是一本关于软件开发社会学的好书，内容侧重于开源软件和大型公司。书中关于向上管理和处理办公室政治的章节是新加入公司的工程师们必读的。我会向任何行业的任何一名工程师推荐此书！这是我所见过的第一本以工程师容易理解的方式

解读办公室政治的书。书中关于如何与难处之人共事的故事、轶闻以及实用技巧极具价值！这些内容是花钱也买不到的。”

——Piaw Na，An Engineer's Guide to Silicon Valley Startups及Startup Engineering Management的作者

“这是一本难得的好书。在这本书中，就程序员如何对优秀团队作出最大贡献的问题，Ben和Fitz与读者分享了他们的明智哲学。我们很幸运地看到，终于有人以如此温暖和幽默的方式对这一重要领域进行了探讨。如果我21岁时能读到这本书并将其牢记在心该有多好！”

——Bryan O'Sullivan，Facebook

“这本书是打造健康软件开发文化的蓝图。工程经理、技术主管，甚至那些需要了解团队氛围如何影响顶尖工程师去留和软件产品质量的非技术高管，都应该读读这本书。”

——Bruce Johnson，FullStory公司创始人和首席运营官

“软件开发技能能帮你找到工作，但如果再辅以与他人合作的能力，你就能改变世界。这本书讲的不是如何成为更优秀的程序员，而是如何变得卓越。”

——Clay Johnson，The Information Diet作者

“编程不再关乎代码和机器，而是将现成的部件用新的方式加以组合，而每个部件都与人相关。这本书的作者早已领悟到这一点，他们的建议丰富，观点明晰：像关注代码一样关注人，你不仅会成为更快乐的程序员，而且还能使别的程序员更快乐。这本书出版的正是时候！”

——Karl Fogel，Open Tech Strategies公司联合创始人

“多年来，我一直在博客上介绍Ben和Fitz的各种会议演讲，因为很少有人关注极客们的社交生活。我很高兴看到他们的演讲内容集结成书，以后再也不用追着他们全国各地跑了。”

——Robert Kaye，Musicbrainz公司极客

献词

“献给我的父母，他们为我带来希望和喜悦，并教会我读书和阅人。”

——Ben

“献给我的祖父，Alvin‘Nick’ Fitzpatrick，他教我如何讲故事，如何聆听。”

——Fitz

本书由“ePUBw.COM”整理，ePUBw.COM 提供最新最全的优质电子书下载！！！

文前

本书旨在提高人与人合作的能力，帮助人们将更多时间投入创新，而非浪费时间争斗。

第二版前言

本书的第一版《极客与团队》取得了巨大的成功。第一版面世后的四年间，人们逐渐认识到团队合作在工作中的重要性。在一些地方，合作甚至成为了大学课程以及领导力培训课程的一部分。（我们的书经常被用作教材！）

但我们也意识到本书第一版或许过于关注软件工程领域，因此对第二版进行了扩展，以同时面向非技术读者，并将书名改为了《进化：从孤胆极客到高效团队》。虽然仍然使用工程领域的范例，但减少了与软件开发工具相关的具体细节。基于读者的持续反馈，第二版扩展了许多讨论并使其更加一般化。我们也针对较新话题增加了一些新的章节，例如开放式办公场所、冒充者综合征，以及更新的沟通和领导技巧。

希望这些扩展内容能吸引和影响各个行业的读者。

Ben和Fitz

2015年9月

简介

“工程易做，人际关系难处。”

——Bill Coughran，前谷歌工程部高级副总裁

生活充满了难以预料的转折。我们两个人从未想过，有一天会写一本关于团队与合作的书。

和当代的许多“创客”（maker）一样，大学毕业后，我们发现自己的兴趣爱好——捣鼓计算机——是极好的谋生方式。和同时代的许多黑客一样，20世纪90年代中期，我们用散件组装个人电脑，用成堆的软盘安装预发布版本的Linux，并且学习如何管理Unix机器。在互联网泡沫初期，我们成为了程序员，泡沫破裂后则为幸存的硅谷公司（例如苹果公司）工作。后来，我们受雇于一家初创公司，全职设计和编写一个开源的版本控制系统，即Subversion。

然而在2000年到2005年间，意想不到的事情发生了。在编写Subversion期间，我们的工作职责逐渐发生了变化。我们不再整天一门心思地编写代码，而是领导一个开源项目。这意味着我们要整天呆在聊天室里与十几个编程志愿者进行沟通，了解他们的工作情况，还要基本靠一个邮件列表来协调新功能的开发。在此过程中，我

们发现，项目成功的关键不仅仅是编写优秀的代码，人们为实现最终目标而采取的合作方式也同样重要。

2005年，我们组建了谷歌的芝加哥工程办公室，继续我们的程序员事业。此时，我们已经和开源世界建立了密切的联系，不仅参与了Subversion的开发，还效力于Apache软件基金会。我们将Subversion移植到谷歌的BigTable架构上，并在Google Code旗下启动了一个与SourceForge类似的开源托管服务项目。我们参加了各种开发者大会，如OSCON、ApacheCon、PyCon以及后来的谷歌I/O，并开始在大会上发表演讲。由于同时拥有在大公司和开源项目的工作经历，我们在不经意间了解到软件工程团队的许多趣闻轶事并获得了大量宝贵经验。关于糟糕开发过程的一系列幽默故事（“Subversion Worst Practices”）最终演变为关于如何保护团队不受有害之人侵扰的演讲（“How Open Source Projects Survive Poisonous People”）。听众越聚越多，我们的演讲简直变成了软件开发者的“集体治疗”（group therapy）。

在一次又一次地发表关于创新合作的社会挑战的演讲后，O'Reilly Media的编辑建议我们把这些演讲内容集结成书。接下来的事情就不用多说了。

本书为谁而写

本书最初是写给软件开发者的，写给那些想发布优秀软件并获得职业发展的人。但在修订第二版的时候，我们发现书中的内容并不仅仅适用于软件开发者，而是可以适用更广泛的群体。只要与他人合作从事创新工作，书中的经验就对你有所帮助。你可以是社区俱乐部的一员，也可以是某个教会团体、联谊会、委员会或架构师小组的一员。我们对本书读者有两点重要假设。

□ 与他人一起从事创意工作，就职于公司或其他组织。

□ 喜欢制造新东西，认为这件事很有趣并且值得去做。如果工作只是为了挣钱，那么可能不会对自我实现或者职业抱负有什么兴趣。

我们自身的经验来自软件工程实践，因此可以想见，书中大部分的示例都是和软件

工程相关的。但几乎我们描述的所有流程和策略（或者与其类似的流程和策略）都可以直接应用于任何从事创意工作的团队。

在讨论工程师如何以最佳方式“与他人合作”的过程中，我们会涉及一些（看上去）似乎超出程序员工作范围的内容。我们会讨论如何有效地领导一个团队、处理组织关系，并与软件使用者建立良好关系。表面上，这些章节的目标读者似乎是“经理”或“产品经理”。但我们可以保证，在职业生涯的某些时候，你会发现自己也会无意中担任了这些职责。不要怀疑，接着往下读！本书的所有内容最终都会与创意工作者相关。

警告：这不是一本技术手册

在开始之前，需要设定你的期望值。上进的程序员喜欢阅读用数学表达方式精确解释相关领域问题的书，并且书中每个问题通常都有规定好的程序化解决方案。

本书并非如此。

本书研究的是创意产品开发中与人相关的方面，而人是复杂的。我们在演讲中常说：“人，基本就是一大堆间歇发作的bug。”我们讨论的问题和解决方法都很杂乱，很难进行具有完美逻辑的表述。本书读起来像一系列短文，因为它本质如此。书中每一章都会讨论许多相关问题（通常都是小故事），然后再讨论与整个话题相关的一组解决方法。要想完全理解书中内容，可能需要耐着性子看完好几页，动用右脑融会贯通，或者再多想想。

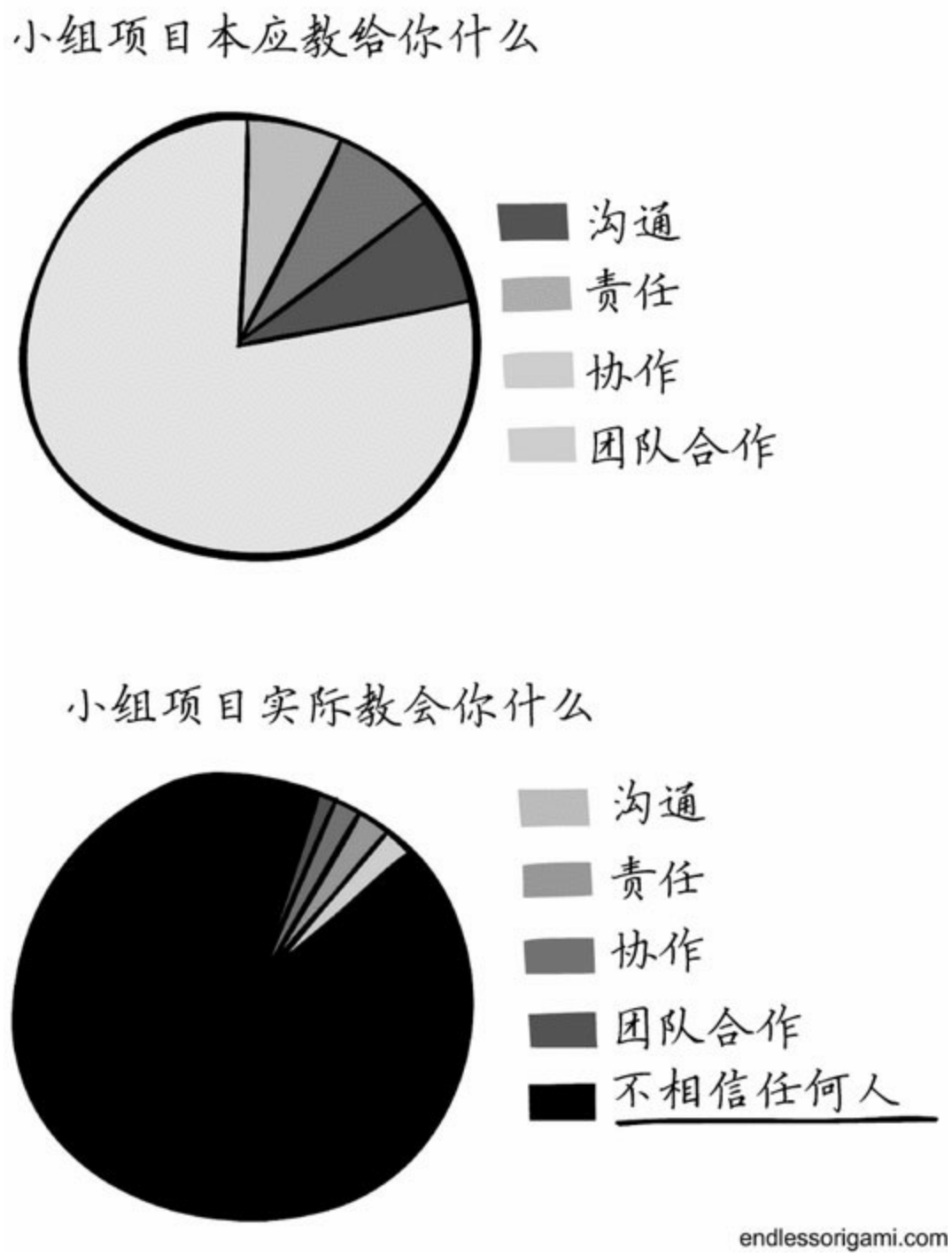
我们还应该做一些免责声明。我们在演讲时喜欢开玩笑说：“这些仅仅是基于个人经验的个人观点。如果你不同意我们的观点，欢迎你说出自己的想法。”同样，我们也欢迎大家讨论本书中的观点。如果你有任何反馈、更正、新观点或反对意见，

请访问http://www.debuggingteams.com。本书的全部内容都来自我们的亲身实践和从错误中吸取的教训。

书中示例使用的人物姓名都经过修改，以保护无辜者（或过错方）。

学校里学不到的内容

我们认识的大部分软件工程师都曾在学校花4～10年学习计算机科学和软件工程。但是，几乎没有课程实际教授如何在团队或公司中沟通与合作。当然，在学习过程中，大部分学生都要参与小组项目。但是，教授某人如何与他人成功合作与强制他参与合作，这两者之间有很大的区别。大部分学生的项目经历并不愉快。



为什么要读本书

要成为一名成功的程序员，仅靠学习最新的编程语言或写出运行速度最快的代码是不够的。职业编程者几乎都在团队中工作。虽然很多人不愿意承认，但程序员所在的团队会直接影响个人的产能和幸福感。

本书的基本理念很简单：软件编写是一项集体活动，我们认为人的因素和技术因素一样，对项目结果有很大的影响。大部分人可能花费了数十年的时间学习编程技术，却从未真正关注过人的因素。学会合作对成功同样十分重要。学习工程的“软技能”能帮助你达到事半功倍的效果。

致谢

虽然本书封面上只有两位作者署名，但本书是我们在生活和工作中与成百上千位朋友对话的结果。我们想花一点时间感谢几位对本书作出重要贡献的人（当然，书中如有错误，都是我们的责任）。

我们要感谢O'Reilly Media的工作人员。感谢Mike Loukides鼓励我们拓展目标读者，感谢无所畏惧的编辑Brian Anderson和Mary Treseler——没有Mary的鼓励、耐心和不时的鞭策，这本书便不会存在。

感谢http://sunnibrown.com的Sunni Brown和Amber Lewis用如此赏心悦目的插画使本书更加生动——和你们一起工作实属乐事。

感谢技术审阅者提出无数的建议、想法和修改方案，从而使这本书成型。他们是Dustin Boswell、Trevor Foucher、Michael Hunger、Jonathan LeBlanc、Piaw Na和Jack Welch。感谢在本书编写过程中审阅并发现一些严重错误的朋友和同事：Dave Baum、Matt Cutts、Will Robinson和Bill Duane。感谢耐心倾听

并提出建议的好朋友们：Karl Fogel、Jim Blandy、Matt Braithwaite、Danny Berlin、Chris DiBona。还要感谢Linda Stone、DeWitt Clinton、Bruce Johnson、Roland McGrath、Amit Patel的想法和建议。

感谢谷歌，特别是谷歌芝加哥工程团队，感谢他们的支持、想法和建议，和这样一群人一起工作实在是棒极了。

特别要感谢几位资深导师和教师，我们只是试图将他们集体智慧的一小部分记录在本书中。他们是Bill Coughran、Steve Vinter、Alan Eustace、Stu Feldman和Eric Schmidt。

特别感谢Brian Robinson和Yvonne Ellison-Sandler的传授、指引和教导。

感谢Apache软件基金会接纳我们，也感谢你们对社区和合作的重视。

感谢所有的好朋友，你们令我们的生活无比富足。不要这样看着我们——你知道我们说的是谁。

在美丽的芝加哥美妙、友好、舒适的Filter Cafe中，我们完成了本书大部分的构思、纲要和成稿。

Fitz

衷心感谢我的妻子Marie，感谢她的鼓励、理解和耐心——她的洞察力和同情心永远激励着我。感谢我的母亲，感谢她的支持和热情。特别感谢我的岳母Rita Gumler，感谢她“人同植物一样”的类比。

写给Ben：相识16年，同事三份工，合著三本书，我怀念与你共事的日子。感谢你与

我共度这段狂野、古怪而奇妙的旅程，你是我的良师益友。

最后，感谢我21年软件工程生涯中共事过的每一个人。这是一场不可思议的旅程，每一天我都从你们身上学到一些知识。

Ben

言语无法表达我对妻子Frances的感激之情——不仅在本书的编写过程中，而且在过去几年的十几个创新项目中，她都给予了我充分的空间。没有她默默而坚定的支持，这些工作都不可能完成。

写给Fitz：现在我们一个人说了上句另一个人就能接下句，简直就像一对老夫妻。我从来不知道和人一起演讲这么有趣，更不用说编软件和写书了。我们获得了多么令人惊叹的机遇啊！感谢你教会我这么多东西。

最后，感谢硅谷所有神奇的人和疯狂的公司：如果没有你们引领我进入这个奇异的世界，这些疯狂经历都不会发生。

关于作者

Brian Fitzpatrick是Tock的创始人兼CTO。2005年，Brian和Ben一起组建了谷歌的芝加哥工程办公室，并领导了谷歌的数项全球工程项目，包括Data Liberation Front和Transparency Engineering。Brian还领导过Google Code和Google Affiliate Network团队，也曾担任谷歌开发数据项目的内部顾问。在加入谷歌之前，Brian在苹果、CollabNet以及芝加哥本地的一家开发公司担任过工程师一职。

Brian撰写过许多文章，发表过几十场演讲，合著了《极客与团队：软件工程师的团队生存秘笈》（Team Geek: A Software Developer's Guide to Working

Well with Others，http://shop.oreilly.com/product/0636920018025.do）、Version Control with Subversion（已出版第二版，http://shop.oreilly.com/product/9780596004484.do），以及《Unix技术手册》和《Linux技术手册》中的章节。

Brian毕业于芝加哥洛约拉大学，获古典文学学士学位，专业为拉丁文，辅修希腊语，研究领域为工艺美术和陶瓷。他现居芝加哥。

Ben Collins-Sussman曾是Subversion版本控制系统的开发者之一。他参与组建了谷歌的芝加哥工程办公室，启动了Google Code项目，领导过两个展示广告团队，目前管理着运营谷歌搜索架构的团队。他现在担任谷歌芝加哥的工程运营中心主管一职，但依然爱好颇多，包括写交互式小说、演奏蓝草班卓琴和爵士钢琴、谱曲、操作业余无线电台、探索摄影技术。Ben是土生土长的芝加哥人，毕业于芝加哥大学，获理学学士学位，专业为数学，辅修语言学。目前，他依然和妻子、孩子以及猫咪一起住在芝加哥。

第1章 天才程序员神话

既然本书要讨论创新开发的社会危机，当然要关注你能控制的一个变数：你自己。

人无完人。但在了解同事们的问题之前，你要先了解自身的问题。在本书中，我们

希望你能思考自己的反应、行为和态度，并由此真正了解如何成为一个更高效、更成功的软件工程师，从而用更少的精力处理与人有关的问题，而有更多的时间编写精妙的代码。

这一章的核心内容是：软件开发是一项集体活动。要在工程团队（或任何其他创新合作）中成功，你需要根据三个核心原则重新组织自己的行为：谦虚、尊重和信任。

首先，让我们观察一下程序员的行为模式。

请帮我隐藏代码

过去十年，我们在编程大会上作过不少演讲。2006年，谷歌的开源项目托管服务启动后，许多人向我们提出了与此服务相关的问题和请求。到2008年年中，我们注意到这样几类请求。

可以让Google Code的Subversion隐藏某些分支吗？

可以让新开源项目最初处于隐藏状态，等项目准备好了再开放访问吗？

我想重写全部代码，你们能把历史记录都清掉吗？

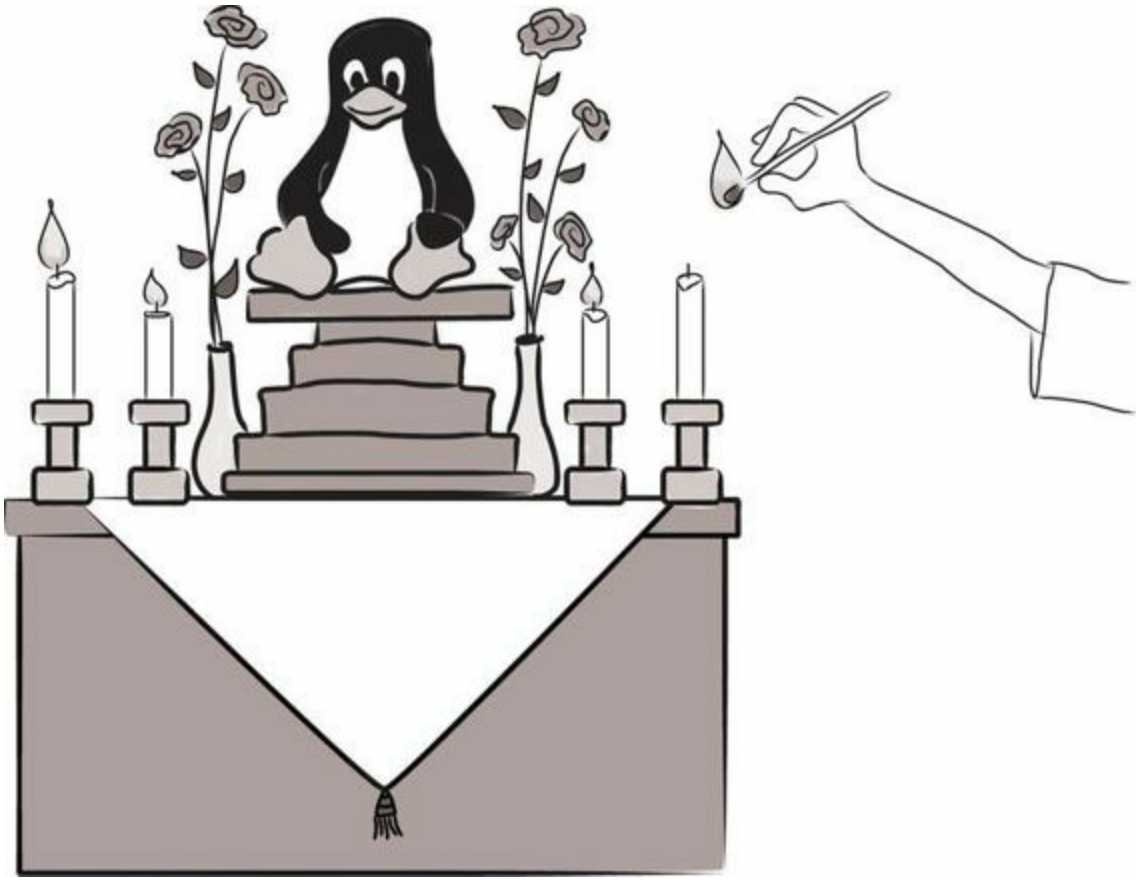
发现这些请求的共同主题了吗？

那就是缺乏安全感。人们担心别人看到并评论自己未完成的工作。在某种意义上，这只是人类本性的一部分——没有人喜欢受到批评，尤其是对自己未完成工作的批评。这种态度揭示了软件开发中的一种趋势。实际上，缺乏安全感只是另一个更大问题的表征。

天才神话

整个20世纪90年代，我们都住在芝加哥，见证了芝加哥公牛队的辉煌冠军之路。全国媒体铺天盖地地报道关于这支神奇球队的新闻，热度持续了好些年。但电视和报纸真正关注的是什么呢？不是整支球队，而是超级巨星迈克尔·乔丹。世界各地的每个篮球运动员都想成为乔丹。我们看到他在其他球员身边穿梭，看到他出现在电视广告中，看到他出演和卡通人物打篮球的烂电影。他是明星，每个在球场上练习投篮的孩子都暗自希望自己长大后能像他一样。

程序员也有同样的心理——寻找并崇拜偶像。林纳斯·托瓦兹、理查德·斯托曼、比尔·盖茨，这些人都是用伟大事迹改变世界的英雄。林纳斯自己编写了Linux系统，不是吗？



实际上，林纳斯只编写了一个类Unix的概念内核原型，并发布到了一个邮件列表。这不是一项小工程，绝对是很了不起的成就，但这些工作只是冰山一角。Linux系统比这个原型大百倍，是由几百位聪明的程序员开发的。领导这些程序员，协调他们的工作才是林纳斯的真正成就。Linux是集体工作的闪亮成果。（Unix系统本身也不是完全由肯·汤普森和丹尼斯·里奇两个人开发的，而是由贝尔实验室的一群聪明人共同编写的。）

同样，是斯托曼一个人编写了自由软件基金会的全部软件吗？斯托曼开发了Emacs的第一个版本，但其他几百位程序员开发了bash、GCC工具链，以及Linux上运行的所有其他软件；乔布斯领导一支团队设计了Macintosh；比尔·盖茨编写了早期家用电脑的BASIC解释程序，但他更大的成就是基于MS-DOS创建了一家成功的企业。他们都是集体成就的领导者和象征。

那迈克尔·乔丹呢？

迈克尔·乔丹也是这样。我们将他视为偶像，但事实上，乔丹并非仅靠一己之力赢得每场球赛。乔丹真正的天才之处在于他与球队合作的方式。公牛队的教练菲尔·杰克逊非常聪明，他的训练技术是传奇式的：杰克逊认识到仅靠单个球员永远无法赢得冠军，因此以乔丹为核心组建了一支“梦之队”。这支球队非常优秀，至少和乔丹本人一样出色。

那为什么我们要不断地将这些人偶像化呢？为什么人们要购买名人推荐的产品呢？为什么我们要买第一夫人同款裙子或乔丹鞋呢？

名人效应是主要原因之一。人有一种本能：发现领导者和楷模，将他们偶像化，然后试图模仿他们。我们都需要崇拜英雄以激励自己，编程世界也有自己的英

雄。“技术名人”（techie-celebrity）现象几乎演变成了神话。我们都希望编写出像Linux这样改变世界的软件，或者设计出下一个绝妙的编程语言。

在内心深处，我们都希望成为天才。极客的最终幻想就是获得令人称赞的新创意。在山洞里闭关几周甚至数月，努力完美实现自己的创意，然后横空出世，令所有人惊叹不已。同行叹服于你的聪明才智，大众排队使用你的软件，名利随之而来。

等等，该醒醒了。你可能并不是天才。

这话没有不敬之意。我们相信你是一个非常聪明的小伙子（或姑娘）。但是你有没有意识到真正的天才是多么稀有？的确，你会写代码，这是一门挺难的技术，掌握这门技术说明你比很多人都聪明。但即便你真是天才，也未必能够成功。天才也会犯错，拥有绝妙的想法和最好的编程技术并不能保证你的软件一定能成功。职业的成功或失败，取决于你能否与他人很好地合作。

实际上，天才神话只是我们缺乏安全感的另一个表现。大部分程序员不愿意与他人分享自己刚开始的工作，因为分享意味着别人会看到他们的错误，会发现这些代码的作者并不是天才。一位程序员在Ben的博客上写道：

如果别人看到我未完成的工作，我会非常没有安全感，觉得他们会肆意批评我，并认为我是个白痴。

这种心态在程序员中非常普遍。人们对此的本能反应是：躲在没人的地方，工作，工作，再工作。这样就没人会看见你犯错，你还有机会在完成后展示自己的杰作。躲起来，直到一切都达到完美。

另一个常见的现象是：因为害怕其他程序员偷走自己的想法，所以人们把所有的点

子都藏着。把想法隐藏起来，就不会失去控制了。

你可能在想：那又怎样？难道人们不应该按自己喜欢的方式工作吗？

实际上，这个问题的答案是否定的。在软件开发中，隐藏想法、独自工作是错误的，而且这种错误很严重。以下将列出原因。

隐藏是有害的

如果总是独自工作，失败的风险就会增加，你也会错失成长的机会。

首先，你怎么知道自己努力的方向是正确的呢？

假设你是一位自行车设计爱好者，某天有了一个好点子，想出了一种全新的变速器设计。你订购了零件，躲在车库花了好几周打造原型。当邻居（也是一位自行车爱好者）问你在干什么时，你决定不告诉他，想等到项目完成时再宣布。又过了几个月，原型设计还没完成，因为你遇到了困难，但由于你不与别人讨论这个项目，你也无法向懂机械的朋友寻求意见。

然后有一天，邻居从他的车库里推出一辆自行车，车上装有一种全新的变速器。原来，你的邻居也在研究类似的发明，但他得到了在自行车店工作的朋友的帮助。这时你终于忍不住了，向他展示了你的设计。然后邻居指出了你的设计中存在的一些简单缺陷。如果早点和他沟通，这些问题在工作开始的第一周就可以得到修正。

从这个故事中我们可以得到几个教训。如果将想法秘而不宣，坚持一切尽善尽美才公布，那么你就是在进行一场代价高昂的赌博。在项目早期很容易出现基本的设计错误，你可能会重复进行已有的设计，同时还放弃了合作的好处。请注意你的邻居是如何与他人合作而进展神速的。这就是为什么人们在跳入深水池前要先试试水：你需要确定自己努力的方向是正确的，采取了正确的方法，并且没有重复他人的工作。项目早期出现错误的可能性非常大。早期获得的反馈意见越多，出现错误的可能性就越小。请记住一句经验之谈：早失败，快失败，常失败。失败的重要性将在后面章节讨论。

尽早与他人分享想法不仅可以避免个人失误，验证想法的正确性，而且可以增加所谓的项目的巴士因子（bus factor）。

巴士因子：项目中多少人被巴士撞死会导致项目完全无法进行下去。

在你的项目中，知识和方法的分散程度如何？如果只有你一个人理解项目的原型代码是如何工作的，那你的工作可能会牢靠，但这也意味着如果你被车撞了，项目就结束了。但如果你和一位朋友一起工作，那项目的巴士因子就翻了一倍。如果一支小型团队一起设计和构造原型，情况就更理想了，项目不会因为失去某个队员而停顿。记住：项目成员可能不会真的出车祸，但仍会发生其他无法预知的事件。项目成员可能会因为结婚、搬家、辞职或需要照顾患病的亲人而离开。你需要管理巴士因子，以保证项目未来的成功。

除了巴士因子，还需要考虑项目的整体进度。人们很容易忘记，独自工作经常困难重重，进度迟缓，虽然大家都不愿意承认。独自工作时你能学到多少知识？进展速度又如何呢？网络上充满了各种观点和信息，却无法替代真实的人际体验。与他人合作可以直接提升集体智慧。当你卡在某个荒谬的错误时，要花多长时间才能迈过去？如果有几位同伴发现你的问题，并立即告诉你错误在哪里以及该如何解决，那情况该有多么不同啊！这就是为什么软件工程公司里项目团队总是坐在一起（或者进行结对编程），因为人们经常需要旁观者的意见。

打个比方，请思考一下编译器的使用方式。在编写大型软件时，你会花好几天编写10 000行代码，写完改好，然后才第一次按下“编译”按钮吗？当然不会。你能想象这会带来多大的灾难吗？程序员在使用小型的反馈循环时工作效果最好。写一个新函数，编译；增加一个测试，编译；重构一些代码，编译。这样在写出代码后可以尽快修正拼写错误和缺陷。我们希望工作时编译器能随时陪伴在身边，帮助我们。有些编译环境甚至可以在程序员输入代码时即时编译。这样才能保持优秀的代码质量，确保软件一点一点正确地演化。

不仅在编写代码时，整个项目也需要这种快速反馈循环。大型项目演化速度很快，必须随时调整以适应环境变化。项目可能会遇到无法预知的设计障碍或政治灾难，或者事情未按计划进行。需求也会发生意想不到的变化。那么如何使用这种反馈循环，在计划或设计需要变更时立即了解情况呢？回答是：团队。人们经常引用埃里克·雷蒙德的一句话：“众人的审视使缺陷无所遁形。”一种更好的说法也许是：“众人的审视使项目保持其相关性，并按计划进行。”在山洞里闭关修炼的人可能某天醒来后发现，软件原型完成了，但世界变化如此之大，自己的产品已经毫无用处。

工程师和办公室

二十年前，传统观念认为工程师需要一间自己的封闭办公室才能保证工作效率。据说只有这样工程师才能拥有大段不受干扰的时间，可以集中注意力编写更多的代码。

在我们看来，对于大部分工程师而言，私人办公室并不是必需的，甚至是危险的。如今的软件由团队而非个人编写，与团队其他成员保持高带宽的即时联系甚至比保持互联网畅通更有价值。拥有不受干扰的时间并没有错，但用这些时间来做错误的

事情就是浪费时间。

不幸的是，当今的科技公司似乎走向了另一个极端。走进这些公司的办公室，你经常会看到大办公室里坐了一大群工程师（50或100人），中间连隔墙都没有。这种“开放格局”现在引起了很大的争论。在开放格局办公室里，最简短的谈话都会影响别人。人们为了避免干扰临近的十几个人，最后干脆连话也不说了。这简直跟私人办公室一样糟糕！

我们认为折中做法最好。公司可以把6～12人的团队安排在小房间（或大办公室）里，这样人们可以比较容易（而且不感到尴尬地）发起谈话。

当然，在任何情况下，单个工程师还是需要屏蔽噪声和干扰的方法。因此大部分团队都想出方法来表达“我们很忙，请勿打扰”。我们以前的一个团队有一个有声打扰协议：如果你想说话，就要说“中断Mary”，Mary就是你想打扰的人的名字。如果Mary可以停下手头的工作，她就把椅子转过来听你说话。如果Mary当时特别忙，就简单说一句“知道了”，你就先做别的事情，等她结束手头的工作。

其他团队给工程师发放降噪耳机，以消除背景噪声的干扰。实际上，在很多公司，一个人戴上耳机的意思就是“闲人勿扰”。另外一些团队把某种标记或毛绒玩具放在显示器上，以此表示“请勿打扰”。

请不要误解，我们仍然认为工程师需要不受干扰的时间来专心写代码，但他们同样需要与所在团队建立高带宽低摩擦的沟通渠道。你需要找到适当的平衡点。

总结一下：独自工作一定比多人合作更具有风险。你可能会担心别人窃取你的想法或者认为你笨，但你更应该担心的是大量时间浪费在错误的事情上。

不幸的是，这种“不愿分享想法”的问题并非软件工程领域所独有，而是普遍存在于各个领域。例如，学术界本应进行自由开放的信息交换，但由于“不发表则消亡”的严酷现实，以及对研究基金的激烈竞争，导致结果适得其反。伟大的思想者不愿分享想法，而是紧紧抓住这些想法不放，独自进行研究，沿路隐藏所犯的错误，最终发表一篇论文，把整个过程描述得轻松而明显。这种做法的结果往往是灾难性的：他们无意中重复了他人的工作，或者在研究早期犯下了错误而未察觉，又或者研究结果因过时而毫无用途。由此浪费的时间和精力是悲剧性的。

不要再犯这种错误而成为又一个统计数据。

团队为王

让我们回过头，把前面的内容总结一下。

我们一直强调的是：编程领域极少出现独行侠，即便真有，他们也不是在真空中完成超人创举的。他们的惊世之作几乎都是灵感的火花以及优秀团队协作创造的结果。

我们真正的目标是打造一支超级明星团队，而这一目标极难实现。最好的团队能让明星队员发挥极大的作用，而团队整体的能力又总是大于各个队员的能力之和。

用更简单的话说就是：软件开发是一项团队活动。

这个观点与我们内心抱有的天才程序员幻想有直接的冲突，一开始可能很难接受。你可以试着把这句话当口号念上几次。

独自在黑客巢穴里当个超级黑客是不够的，躲起来秘密发明并不能改变世界，也不能赢得百万计算机用户的青睐，你需要与他人一起工作，分享愿景，分工合作，互相学习，从而打造一支优秀的团队。

请考虑一下：你能说出多少个真正由一个人编写而又广泛使用的成功软件？（有人可能会说LaTeX，但LaTeX很难称为“广泛使用的”软件，除非你认为写科学论文的人在所有计算机用户中的统计占比是显著的！）

本书会不断重复团队活动这个概念。高效团队千金难求，是成功的真正关键。你应该尽一切努力实现这一目标，这就是本书所要讨论的内容。

三大基石

我们已经强调过了团队工作的重要性。如果团队合作是创造优秀软件的最佳途径，那么如何打造或找到一支优秀的团队呢？

答案并不简单。要达到团队合作的最高境界，首先需要学习和掌握被称为“三大基石”的社交技能。这三大原则不仅可以调和人际关系，还是所有良好交互与合作的基础。

谦虚（Humility）

你并非宇宙中心，并非无所不知，也会犯错，但愿意自我改进。

尊重（Respect）

你要真诚关心同事，以礼相待，欣赏其能力，认可其成就。

信任（Trust）

你要相信别人可以胜任工作并作出正确选择，愿意在合适的时候将权力交给他们。

我们将这些原则合称为HRT。HRT原则是为了减轻痛苦，而非伤害他人，因此HRT念作heart而非hurt。实际上，我们的主要论点直接建立在这三大基石之上：

几乎每种社交冲突的源头最终都可归结为谦虚、尊重或信任的缺失。

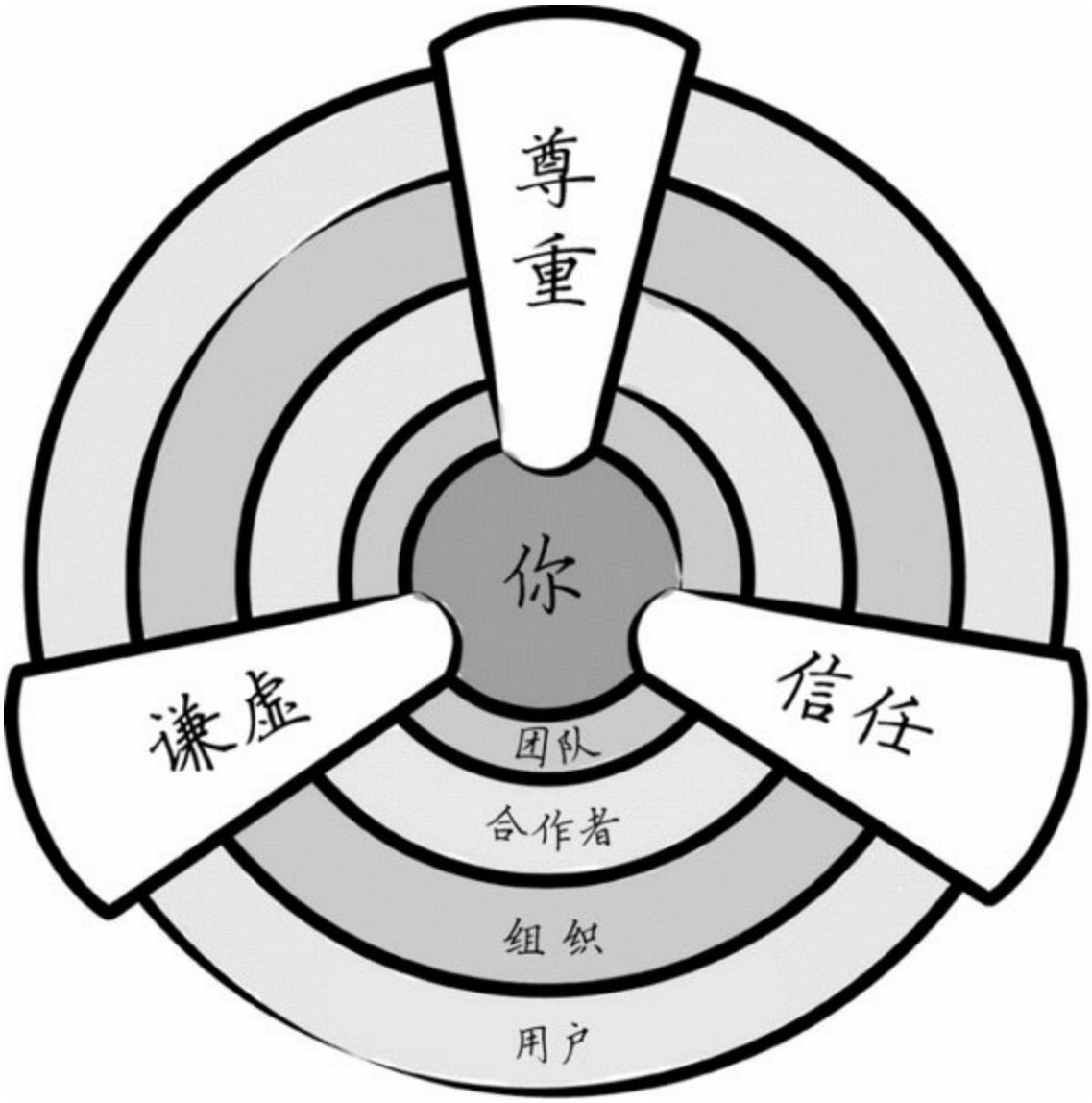
这一论点初听起来可能缺乏说服力，但你不妨一试。想想生活中正在发生的一些令人讨厌或不快的社交问题，在最基础的层面上，每个人是否足够谦虚？人们是否真的彼此尊重？他们相互信任吗？

我们认为这些原则非常重要，甚至本书都是以这些原则为核心构建的。

本书以你为起点，引导你掌握HRT，使你真切意识到如何以HRT为人际交互的核心（这是第1章的内容），随后再由内向外扩展讨论HRT的影响。

第2章讨论基于这三大基石打造团队所要面临的挑战。创造团队文化是通向成功的下一个关键，这里的成功是指之前讨论的“梦之队”。

之后将讨论与团队有日常交互、但可能不属于核心团队文化的人。这些人可能是其他团队的成员，也可能是项目中帮忙的志愿者。他们中很多人不仅不遵循HRT原则，甚至可能对团队有害！应该将保护团队不受有害之人侵扰视为头等大事。但最终目标应该是去掉他们的毒素，将他们吸纳进团队文化，这是扩展团队的极好方法。



大部分团队都位于一个更大的公司架构之中，公司环境常常会和有害之人一样阻碍项目发展。能否应对和处理这些组织机构障碍，可能会决定一个产品最终是取消还

是发布。

最后将讨论一下软件用户。有时我们会忘记用户的存在，但他们是项目的命脉。没有用户，软件就失去了目标。在团队中倡导的HRT原则，也应当用于与用户的交互，其收效是显著的。

让我们暂停一下。

当你拿起这本书时，大概没想到自己将加入某种每周互助团体吧。我们对此表示同情。社交问题很难处理，因为人是杂乱、无法预料的，常常很难打交道。与其花费精力分析社交问题，作出行为决策，选择视而不见要轻松得多。比起善变难料的人，编译器好相处多了，难道不是吗？为什么我们要纠结于这些社交问题呢？

下面这段话摘自理查德·哈明一篇著名的演讲。

通过“受累”跟秘书讲讲笑话并对他们友好些，我从秘书那里获得了极大的帮助。例如，一次因为某种莫名的原因默里山所有的复印服务都排满了队。别问我怎么回事，那天就是那样。我有一些资料需要复印，然后我的秘书给霍尔姆德尔的人打了电话后，跳上了公司的车，跑了一个钟头的路把东西复印了回来。那可真是我长期鼓励她，给她讲笑话，并对她友善的回报。这就是投之桃李，报之琼瑶。认识到使用系统的重要性并研究如何让这个系统为你所用，你就学会了如何驾驭这个系统。

中心思想是：不要低估了社交游戏的能量。社交游戏不是欺骗或操控他人，而是培养完成工作所需的人际关系。即便项目结束，人际关系依然存在。

HRT实战

以上这些关于谦虚、尊重、信任的说教听起来像布道词般高大上。让我们说点具体的，考虑一下如何将这些理念应用到实际的生活场景中吧。人们需要的是实际的建议，因此我们将给出具体的行为清单和实例，以供参考。很多建议似乎显而易见，但若仔细一想，就会发现你（和你的同伴）很多时候并未遵循这些建议。

放下自我

好吧，放下自我其实就是告诉那些不够谦虚的人别摆架子。没人想和时时表现得自己好似宇宙中心的人一起工作。就算你知道自己是讨论组中最聪明的人，也不要表现得咄咄逼人。例如，对每个话题，你是否总是第一个发表意见或最后要总结陈词？对提案或讨论中的每个细节，你是否都有评论的欲望？或者，你认识这样的人吗？

请注意，态度谦虚并不是说应该完全任人摆布。自信和谦虚并不矛盾，只是别表现得无所不知。更好的心态是追求“集体”自我。不要担心个人表现是否出色，你更应该尝试培养一种团队成就和集体荣誉感。例如，Apache软件基金有一项悠久的历史：以软件项目为核心创建社区。这些社区有异常强大的社区认同感，不接受那些意在推销自我的人加入。

自我的表现形式很多，很多时候自我会影响生产力和项目进度。理查德·哈明演讲中的另一个精彩故事完美诠释了这一点。

约翰·图基总是穿着随意。当他走进一间重要的办公室时，其他人总要花很长时间才能意识到这是一位大牛，然后才认真听他说话。有很长一段时间，约翰都不得不克服这个麻烦。这完全是浪费精力啊！我不是说你应该服从。我想说的是：“表现得服从有很多好处。”如果你选择以各种方式坚持自我，“我要按自己的方法做”，

那么你会在整个职业生涯中持续不断地付出微小代价。在你的一生中，这些微小代价会累积为大量不必要的麻烦。认识到使用系统的重要性并研究如何让这个系统为你所用，你就学会了如何驾驭这个系统。或者你可以选择不断与之斗争，用一生进行一场微不足道的无名之战。

批评与自我批评

乔刚得到一份程序员的新工作。入职一周后他就开始认真钻研代码库。乔关心业务，因此开始客气地请教其他团队成员代码问题。他通过邮件发送简单的代码审查，礼貌地质询设计假定，或者指出可以改进的逻辑。几周后，老板把乔叫到了办公室里。“出了什么问题？”乔问，“我做错事了吗？”老板面露难色：“有很多关于你的投诉。显然，你对同事太严厉了，总是批评他们。大家都很郁闷。你应该注意一下。”乔完全不明所以。在基于HRT原则的团队文化中，乔的代码审阅意见应该会受到同事的欢迎和认可。但在这个故事中，乔应该注意到这个团队中普遍存在的不安全感，采用更为委婉的方式将代码审阅引入团队文化之中。

在职业软件工程环境中，批评通常只是产品改进过程的一部分，几乎从不针对个人。困难的是如何确保你（以及你周围的人）理解对某人创意结果的建设性批评与赤裸裸的人身攻击有何区别。人身攻击毫无价值，这种行为既卑劣又没法回应。建设性批评通常是有益的，能够指导我们改进。最重要的是，建设性批评是基于尊重的：意见的提出方真切关心意见的接受方，希望他改进自身或工作。要学会尊重同伴，有礼貌地提出建设性批评。如果你真的尊重一个人，就会主动选择使用得体、有益的词句进行交流——这项技巧需要多加练习才能掌握。

作为意见的接受方时，也需要学会接受批评。要做到这一点，不仅需要对自己的技术持谦虚态度，还要相信提出批评的人是真心为你和项目好，而不是认为你笨。编

程和其他技术一样，需要练习才能提高。如果一位同伴指出你该如何提高杂耍水平，你会认为这是对你人品和个人价值的攻击吗？估计不会。同样，你的自尊不应该和你写的代码（或创建的任何创新项目）有任何联系。也就是说，你的代码并不代表你。再说一遍，你和你的产品不是一回事。你不仅要自己相信，还得让同事也相信这一点。

例如，如果你与一位缺乏安全感的同事合作，请不要对他说：“哥们，你把这个方法的控制流全搞错了，应该用标准的xyzzy代码模式，大家都这么用。” 这条反馈意见充满了反模式：你告诉某人，他做“错”了（就好像这世界非黑即白），要求他修改，还指责他与众人背道而驰（让他自感愚钝）。受批评的人将被迫采取防卫姿态，出现过度情绪化的反应。

同样一句话，这样表达可能更好：“嗨，我不太明白这部分的控制流。我想，如果用xyzzy模式，逻辑是不是更清晰，代码也更好维护呢？” 请注意此处是如何善用谦虚的，表明问题在你而不在他。他没有错，只是你不太理解这段代码。你提出的意

见只是要让自己更清晰地理解代码，同时有利于项目的长期维护。你也没有提出任何要求——对方可以心平气和地拒绝这个建议。这场讨论仅限于代码自身，不涉及任何人的价值或编程技术。

快速失败和迭代

商界有一个流传很广而且很老套的民间传说：有位经理因失误导致公司损失达千万美元之巨，第二天，他沮丧地走进办公室收拾东西准备走人。接到“CEO让你到办公室去”的电话后，他走进CEO办公室，默默地递上一张纸。

“这是什么？” CEO问。

“我的辞职信，” 经理答道，“我想你叫我来就是要辞退我。”

“辞退你？” CEO惊讶地回道，“为什么要辞退你？我刚在你身上花了1000万的学费！”

当然，这个故事有些极端，但故事中的CEO明白，辞退这位经理并不能挽回1000万的损失，反而会雪上加霜，损失一位有价值的管理人员，而这位管理人员绝对不会再犯同样的错误。

在谷歌，我们最喜欢的格言之一是：“失败是一个选项。” 我们认为，如果从来不犯错误，说明你没有足够的创新性去承担足够的风险。在我们看来，失败为下一次的努力提供了学习和改进的最佳机会。实际上，人们经常引用托马斯·爱迪生的话：“发现一万种走不通的路，并不意味着我失败了。我不会气馁，因为每次失败的尝试都是前进的一步。”

在Google X（研发极度前瞻项目的部门，如谷歌眼镜和无人驾驶汽车），失败是精

心设计在激励系统中的一部分。人们会提出疯狂的想法，鼓励同事尽快证明这些想法是错误的。部门会奖励个人（甚至提倡竞争），看大家能在固定时间内证明多少想法是错误的或不可行的。当一个概念确实无法被大家驳倒时，它才能进入下一步的初期原型设计。

从错误中学习的关键是将失败记录在案。你需要编写我们所说的“事后分析报告”。请注意，事后分析报告的目的不是罗列无用的道歉或借口。一份合格的事后分析报告应该包含学习经验的结果，解释学到了什么，计划作出什么改变。然后要确保将这份文档放在一个随手可得的地方，并按照文档提议的内容作出改变。请记住，将失败完备地记录下来，还可以让别人（在现在及将来）更容易了解当时发生了什么并避免重复历史。不要擦除你的足迹，照亮来路，以便后来者追随！

一份完备的事后分析报告应包含以下内容：

□ 概述

□ 从发现问题，调查问题，到解决问题的时间线

□ 事件发生的主要原因

□ 影响和损失估计

□ 立即解决问题的一系列措施

□ 预防事件再次发生的一系列措施

□ 吸取的经验教训

留出学习的时间

辛迪是一位超级巨星——一位在所属领域才能卓越的软件工程师。她晋升为技术组长后，承担了更多的责任，并积极应对挑战。不久之后，辛迪开始指导周围的人，在专业会议上发表演讲，很快成为多个团队的负责人。辛迪享受作为“专家”的感觉，但是，她开始感到无聊。在成长的过程中，她不再学习新事物，作为最聪明、最有经验专家的新奇感逐渐消退了。虽然辛迪在外表上依然是领域专家，享有成功，但总觉得缺少了什么。一天，她意识到自己的领域不再那么重要，人们已经转到其他方向。那么辛迪哪里出了问题呢？

我们需要面对一个事实：成为团队中最博学的人很有趣，指导别人也极为有益，问题是一旦成为团队中技术最强的人，你就不再学习了。如果停止学习，你就会觉得无聊，或者无意间变得落伍。成为最佳队员这件事很容易上瘾，但只有稍稍放下自我，才能改变方向，接触新事物。再次强调，你还是需要变得更加谦虚，既愿意传授知识，也愿意学习。请间或跨出自己的舒适区域，找一个有更多大鱼的池子，接受未知的挑战。最后，你会变得更快乐。

学会忍耐

很多年前，Fitz在编写一个将CVS库转换到Subversion（后来转到Git）的工具时，由于CVS的问题，他不断发现各种奇怪的缺陷。Fitz的老朋友兼同事Karl对CVS很熟悉，因此两人决定一起修复这些缺陷。

Fitz和Karl开始结对编程时遇到了一个问题：Fitz习惯自底向上，一头扎进代码，快速进行各种尝试，略过细节；而Karl习惯自顶向下，先摸清大局，了解调用栈中的每个方法，然后再着手解决问题。两人习惯的不同导致了一些严重的冲突、矛盾，偶尔还发生了激烈的争吵。到最后两人干脆不再结对编程了，因为这实在是太令人沮丧了。

虽说如此，但Fitz和Karl毕竟有着对彼此的长期信任和尊重，再加上耐心，他们想出了新的合作办法。Fitz和Karl会一起坐在计算机前，确认缺陷，然后分头从两个方向（自顶向下和自底向上）攻克这个问题，再回来一起研究各自所得。他们充满耐心，愿意采取新的工作方式，这不仅挽救了项目，也挽救了与彼此的友谊。

接受改变

越愿意接受改变，就越能引起改变。越是脆弱的人，越表现得坚强。这些话听来很怪，似乎自相矛盾。但是每个人都可以回想一下一起工作过的特别固执的人。别人越是试图说服他，他反而更加固执。这样的团队成员最后会怎样？按我们的经验，最后大家会把他们视为障碍物，“绕开”他们，不再听他们的观点或反对意见。你肯定不想陷入这种境地，因此必须记住：自己的观点可以受别人影响。并不是每件事情都必须坚持己见，要谨慎选择需要坚持的事情。记住，你要先聆听别人的意见，别人才会听取你的意见。在宣布自己的立场或观点前最好先听取别人的意见，如果你不断改变观点，别人会认为你优柔寡断。

关于脆弱的说法初听起来也有点奇怪。如果一个人承认自己不懂或不知道如何解决一个问题，他会给大家留下什么印象？脆弱是软弱的表现，会导致失去信任，是这样吗？

并非如此。从长远来说，承认自己的错误或能力不够可以提升地位。实际上，这也是HRT原则的表现：对外表现出谦虚的态度，不仅可以显示出可靠性和责任感，还说明你信任别人的观点，最后，人们也会尊重你的诚实和能力。有时，最好的做法就是直接承认：“我不知道。”

想想那些政治家，他们最为人诟病的就是，即便事实明显，也从不承认自己的错误

或无知。正因为如此，大多数人不相信政客说的任何一句话。政客拒绝承认自己的错误，主要是因为他们不断受到对手的攻击。但当你写代码时，就没必要时时处于防御状态了——队友是合作者而非竞争者。

下一步

读到这里，你已经掌握了“与他人合作”的艺术。你必须开始检视和反省自己的行为。一旦将这里讨论的策略应用到日常生活中，你将发现合作变得越来越自然，工作效率也明显提高。

重要的改变应当从自身开始，由内及外。下一章将讨论如何在所在团队中打造HRT氛围。

第2章 打造团队文化

团队文化种类多样，反映出不同的价值观和任务优先级。有些团队文化促进成功，有些则导致很大的失败。但是，即使在成功团队中，团队文化也不尽相同。有些团队文化极为高效，将绝大部分注意力放在当前的任务上；有些却提供很多与当前任务无关的活动。这一章将讨论团队文化，主要关注成功所需要的各种沟通技术，并阐述如何将这些技术用于团队之中，从而更高效地创造产品。

什么是团队文化

听到文化这个词时，通常会联想到歌剧之夜，或者高中生物课上培养细菌用的培养皿。其实，工程团队的文化和后者颇有相似之处。

如果你品尝过一块美味的酸面团面包并慕名找到那位烘焙师，就会发现这种面包的关键原料是一种剂子，其中包含酵母与以面粉和水为食的乳酸菌。酵母使面包蓬松，乳酸菌则给面包带来奇妙的特殊酸味。但是乳酸菌各有不同，有些使面包风味更佳。因此，如果烘焙师找到能产生完美酸面团口味的剂子（即一种乳酸菌和酵母的混合），就会精心呵护，添加更多的面粉和水，以维持和培养这种菌群。然后，烘焙师取出少量的剂子，加入所需原料，就可以得到一块美味的酸面团面包了！剂子中的菌群不仅可以产生烘焙师想要的口味，而且能够抑制面包原料或周围空气中

存在的其他酵母或细菌，这才创造出了美味的面包。

团队文化就像一块好吃的酸面团面包：剂子（创始人）向面团（新成员）注入菌群（文化），随着酵母和细菌（团队成员）的成长，就可以做出好吃的面包（团队）。如果创始文化足够强大，就可以同化新成员可能带来的不和谐因素。如果创始文化过弱，那么团队很容易受到新成员带来的未知文化的干扰。未知文化带来的影响难以预测，因此团队文化的培养还是从已知创始文化开始为宜。

团队文化不仅仅是团队成员完成工作、编写代码或彼此相处的方式，而且是成员共享的经验、价值观和目标。我们接触过的每支工程团队的文化都是独一无二的。一支团队或一个公司的创始人主导着团队文化，但团队文化会随着团队成长不断变化和发展。

组成团队文化的元素多种多样。有些元素直接与工作流程相关：编写软件时可能需要代码审阅和测试驱动开发，并在编码前完善设计文档。有些元素与人际相关，例如，每周四去某家餐馆聚餐，或者周五去酒吧喝一杯。还有些元素在外人看来完全是莫名其妙：谷歌匹兹堡工程办公室以前挨着一条货运铁路，每次火车经过（请注意，动静非常大），办公室里的所有人都会跳起来用玩具枪对射。所有这些构成了

一个团队的文化，进而影响了团队的生产力和吸引、留住优秀团队成员的能力。

看看如今任何一家极为成功的公司——谷歌、苹果、微软、甲骨文，你会发现各家有着非常不同的企业文化，这些文化的根基是由公司的创始人和初期员工奠定的。随着这些公司的成长和成熟，企业文化也在不断演化和变更，但始终保有一种特质，这种特质影响公司如何开发产品，对待员工，参与竞争等方方面面。

团队文化的重要性

简单地说，如果不花费精力打造和维护团队文化，最终会出现具有很强个性的人，在团队中培养他们的文化，接管你的团队。这样产生的团队文化可能是有效而健康的，而且能产生大量的优秀代码。但更多时候结果并不这么理想，你会突然发现，自己本应用于产品设计和生产的精力都花费在争吵和内耗上了。除此之外，团队成员应该重视并愿意维护团队文化，这一点也很重要。如果团队成员不重视团队文化，不仅难以培养很强的团队认同感和对工作的集体荣誉感，而且新成员很容易给团队文化带来不好的影响。

大部分团队成员犯的第一个错误就是认为团队领导者负责维护团队文化。这简直是大错特错。虽然创始人和领导者通常会关注团队文化的健康，但团队的每个成员都应参与其中，担负起定义、维护、保护团队文化的责任。有人加入团队时，他不仅会从团队领导者处，也会从一起工作的每个成员身上感知团队文化。例如，如果你仔细审阅新成员的工作，向他解释团队采取某种工作方式的原因，他就能很快辨别出这支团队看重产品的什么方面。通过观察其他团队成员是如何工作、交互、解决矛盾的，他也能更多地了解团队文化。

“强大的文化”能够接受提升自己的改变，而抵御带来伤害的巨变。最成功的团队

文化是那些把团队的主要精力集中在交付优秀软件上的文化。如果团队把主要精力放在别的地方（例如，玩乐、开会、勾心斗角），团队的凝聚力可能很强，但写不出什么软件。如果你在写代码和交付产品时最开心，那绝对应该加入一个看重编码和产品交付的团队，并努力维护这种良好环境。虽然没有强大有效的团队文化你也可能做出好产品，但你付出的时间和精力要多得多。好的团队文化可以帮助你集中精力，提高效率，让你充满力量，造就一个更快乐的团队。

团队文化的有趣之处在于，如果定义明确，团队文化就会自己选择成员。在开源世界，基于HRT原则，专注于编写干净、优雅、可维护代码的项目会吸引什么样的工程师呢？答案一点儿也不奇怪——那些喜欢与他们尊重、信任的人一起工作，并热爱编写干净、优雅、可维护代码的工程师。但如果团队文化充满攻击、羞辱、人身攻击，最后就会吸引更多具有这些特质的人。

我们在Apache软件基金会多次见证了自我选择的团队文化。Apache软件基金会是一组基于社区，通过成员共识运作的软件开发团队。很多次，新成员加入邮件列表后，不知是出于无知还是恶意，行为与团队文化相悖。社区成员通常会试图教导这位新成员（有时采取温柔的方式，有时就不太温柔了），如果此人对Apache软件基金会团队的工作方式不感兴趣，通常就会离开，寻找与自己更为合拍的项目。

在公司里，团队通过招聘流程进行自我选择，有的通过应聘者需掌握的技术或能力间接筛选，有的直接将团队契合度作为招聘流程的一部分进行考核。谷歌的招聘流程采取直接方式，在面试时特意评估团队契合度：如果某位面试者各方面都很优秀，但无法在团队中工作，或需要等级清晰的环境，那么面试官会在反馈意见中特别标注。

如果在招聘过程中不注意团队契合度，招进来不合适的人，最后要花大量的精力，

要么与新人磨合成功，要么让他离开团队。不管是哪种结果，都要付出高昂代价，因此在招聘时注意新人能否融入现有团队绝对是值得的。

团队契合度面试

要确保新成员能融入团队，唯一的方法是在面试时进行考察。很多公司（如谷歌）将团队契合度作为面试时的考察标准之一。有些公司为了避免招错人甚至更进一步，在技术面试之前有一轮专门的团队契合度面试，因为他们根本就不会考虑那些不能融入团队的面试者。设置这样的流程对创造和保持团队文化非常关键。这种流程不是偶然出现的，实际上，这是由公司的创始人和初期成员特意设置的。

团队文化与人

编写软件这样的创意工作与在流水线上简单拼装工具不同。有些工作只需要几天的培训和一些简单工具就能上手，如果工人辞职离开或不能胜任，安排另一个工人顶替即可。在流水线环境中，雇员完成简单任务，通常不需要什么创新思维或解决问题的能力，死记硬背就可以了。在软件行业中，制造产品的工程师需要进行大量的创新思考。要得到高质量的产品，就需要优秀的工程师。想让优秀的工程师做出杰出的工作并且留下不走，就需要为他们创造良好的团队文化，让他们放心地分享想法，并有参与决策过程的权利。

让团队拥有卓越的工程师，第一步要做的是招聘一些优秀的工程师！这话听起来有点怪，但事实是，大部分优秀的工程师希望和其他优秀的工程师在一个团队工作。我们所知的很多优秀工程师倾向于加入可以向行业巨人学习的团队。那你应该怎样吸引这样的工程师呢？

首先，优秀的工程师希望自己不仅能为产品开发作出贡献，而且也可以参与产品的决策过程，这通常要求团队的管理在一定程度上基于队员的共识。在自顶向下管理的团队中，首席工程师是团队领导者，能力弱一些的工程师是团队普通成员。雇用级别较低的团队成员花费较少，而且他们容易听从调遣。这样的团队很难吸引优秀的工程师加入，因为他在别的公司完全可以成为团队领导者，为什么要到这里受别人指挥呢？但在基于共识进行管理的团队，所有成员都参与决策制定的过程。

很多人一听到“基于共识的团队”，就会立刻联想到一群嬉皮士在营火旁唱着民谣Kumbaya，他们既不能决策也无法成事，这种印象其实更多反映的是运转不良的团队，而非基于共识的团队。我们所说的“共识”是指每个人对产品的成功都有强烈的主人翁意识和责任感，领导者能够听取团队成员的意见（重视HRT中的尊重原则）。在这样的团队中，人们有时进行很多讨论和思考，有时又需要加快进度。当需要加快进度时，可能会将大量日常琐碎的决策托付给一位或多位领导者。为此，大家需要就团队的整体任务达成共识，此举的关键是拟定一份团队的任务说明书（本章后面会详述）。

除了决策的制定过程，团队成员彼此相处的方式也很重要，这极大影响了什么人会加入团队。如果团队成员习惯互相示威，大吵大嚷，那就只能吸引并留住好胜斗勇的成员，他们在这种充满强烈自我的环境里会如鱼得水（实际上，我们认识的大部分女性觉得这种环境特别令人生厌）。如果团队文化是基于HRT原则创建的，成员彼此友善，乐于提出建设性批评，那么不仅能吸引更多的人加入该团队，而且大家能将更多的精力放在软件编写上。团队应该具有很强的集体自我，而不应该完全淹没在团队成员的个人自我中。第4章将讨论如何防止这种情况发生。

建设性批评对任何人或团队的成长和发展都很重要，但很多人尽量避免受到批评。

有时人们因为缺乏安全感而不愿接受批评，而我们看到的大多数情况是，人们觉得受到批评后，即便不同意也必须有所行动。建设性批评的优点是，你可以进行选择，自己决定接受批评中的哪些部分。举个例子，假设你穿上最喜欢的正装，准备参加一个重要的求职面试，并向一位信任的朋友寻求意见。如果朋友说“你牙齿上有片菜叶，而且这衣服好丑”，你可以快速清洁一下牙齿，但不必把衣服也换了。批评就像一件礼物，你可以接受，也可以拒绝。

如果你有意改进工作或解决私人问题，这些能提出建设性批评的朋友和同事可以帮助你找到症结所在。除非你非常了解自己或能深切自省，否则不接受批评意见会导致不断重复同样的错误，却没有人愿意告诉你。例如，在编写本书过程中，十多人审阅了本书，并提出了建设性批评，大部分意见非常详细而有价值。不管你怎么评价本书，如果我们当初不愿寻求反馈或对意见视而不见，本书肯定会比现在的版本差得多。

接受批评需要一定程度的自信，建设性批评是各种批评方式中最容易接受的。困难的是，进行建设性批评比简单的指责或嘲讽难度大多了。当然，当你请别人对你的工作进行批评时，大多数人会以为你只想得到赞扬和肯定，很难给出建设性批评。如果你有朋友或同事能够直言，千万要珍惜，这样的诤友可遇不可求。

争强好胜的人通常能在比较安静的环境中正常工作，但性格安静内向的人很难在充满争斗的环境里表现出色或安心工作。在这样的环境里，人们的意见很容易被噪声淹没，而且内向的人会不愿参与其中。要让更多的人更高效地工作，你应该打造基于谦虚、尊重、信任的团队文化。

争斗型团队文化不太容易受到性情温和的成员的影响，而基于尊重的冷静温和型团队文化则更容易受到争斗型成员的干扰。温和型团队文化需要提高警惕，不要因为

拒绝主动处理而让争斗型成员占了上风。在一些情况下，一位或多位资深团队成员需要与新成员直接交涉，以防止其作风伤害温和型团队文化。关于如何应对这种“有害之人”，我们会在第4章中进行讨论。

成功团队文化的沟通模式

在团队中，特别是工程师团队中，沟通经常是一种挑战。工程师情愿花一个下午鼓弄（可预测、富有逻辑性的）编译器，也不愿意花三分钟和（情绪化、行为难测的）人相处。很多时候，工程师将沟通视为编程路上需要克服的障碍。但是，如果团队不能达成共识或一致，就无法第一时间知道编写的代码是否正确。

如果仔细观察，你会发现，任何有效、成功的团队文化都非常重视几种沟通渠道，如邮件列表、设计文档、聊天室、任务说明书、产品手册，等等。确保每个成员都

认同团队的方向并准确理解当前任务并非易事，但这可以提高团队的生产力，并提升团队的幸福感。

关于沟通有一个通用准则：人少时使用同步沟通（如会议和电话会议），人多时使用异步沟通（如邮件、问题跟踪系统、文档注释）。同步沟通代价很高：参与者必须中断工作，按你安排的时间接收信息。而异步沟通允许参与者选择适合自己的时间和地点接受信息。每次工作中断，人们都需要花费一些时间才能回到之前的状态，当你需要打断别人工作时请记住这一点。

最重要的是，要确保所有的信息都存在项目文档中，大家都可以访问。随后我们将讨论团队在编程过程中应采用的主要沟通机制。有些沟通机制似乎很简单，但有很多细节值得注意。有一件事是肯定的：如果不花费时间建立良好的沟通机制，会有大量精力浪费在不必要或其他成员已经解决了的事情上。

高层同步

在最高层次上，团队需要保持统一的共同目标，在沟通中遵循最佳实践。

任务说明书

听到“任务说明”一词，你的第一印象可能是很多大公司使用的那种枯燥、夸张的营销式任务说明书。下面是某家大型电信公司的任务说明书。

我们致力于成为世界上最受尊敬和最有价值的公司。我们的目标是为市场带来令人兴奋的实用通信服务，从而丰富客户的个人生活，使他们的业务更加成功，并在此过程中提高股东价值。

奇怪的是，我还没见到任何人崇拜这家公司！另一家大公司的任务说明书如下。

提供实时满足客户需求的解决方案。

这句话到底是什么意思呢？可能根本就没有任何意思——如果为这家公司工作，我们不会知道究竟是洗车，还是修水管，或是送比萨更重要。正是这种似是而非的语言使任务说明书名声不佳。

在一个实干、高效的团队中，编写任务说明书是为了简明地定义工作方向，限定产品范围。编写任务说明书可能需要一些时间和精力，但任务说明书明确了团队应该做什么，不应该做什么，可以节约大量的工作。

谷歌决定将Google Web Toolkit（GWT）的开发转移到开源项目时，我们负责指导团队工作。我们对比了开源和封闭式开发的不同，特别注意到在一个任何人都能发表意见、提交补丁或对产品进行自由点评的环境中进行软件设计、讨论和编写的困难之处。考虑这些风险之后，我们让团队制定一份任务说明书，向公众描述他们的产品目标是什么（以及不是什么）。

由于前面提到的诸多原因，一些团队成员对此有些抵触，但另外一些成员表示好奇，而团队领导人似乎觉得这个主意很不错。当坐下来开始编写任务说明书时，大家就其内容、主旨和风格进行了大量的讨论。在此（以及更多的几次会议）之后，团队不仅交出了一份简明优秀的任务说明书，还完成了一份名为Making GWT Better的文档，对说明书逐字逐句进行解释，其中甚至有一节描述什么不是项目的目标。GWT任务说明书如下：

GWT的任务是，通过允许开发者使用现有Java工具为任何现代浏览器编写无风险AJAX，显著改善用户的Web体验。

这句话包含的信息非常丰富，是任务说明书的一个优秀范例。既包含了产品方向（通过允许开发者……改善Web体验），又限制了产品范围（Java工具）。几年后，我们和这位团队领导人共进晚餐，Fitz对他当时大力支持我们要求团队编写任务说明书表示感谢。他答道，其实最初他认为这件事是浪费时间，但开始和团队进行讨论后，他惊讶地发现首席工程师居然对产品的方向存在争议！

在这种情况下，编写产品说明书迫使团队面对意见分歧，就产品方向达成共识。如果这个问题没有及时解决，有可能会使以后的产品开发进度放缓（或者停滞）。GWT团队将任务说明书在网上发布，不仅明确了整个团队的产品目标，而且节省了与潜在义务编程者争论产品方向的时间——他们只需要告诉新人去哪里读Making GWT Better文档，就可以解决大部分的问题了。

随着项目推进，任务说明书可以确保事情沿既定方向发展。但是，项目说明书不应成为阻碍变化的不可逾越的障碍。如果环境或商业计划（例如在初创公司）发生变化，软件团队成员需要面对现实，重新评价当前任务是否依然有意义。人们特意将修宪过程设置得极为困难，是为了防止宪法随意变更。但在极端情况下，修改宪法至少是可能的，而且应该是可行的。如果公司或项目突然转型，项目说明也需要随

之更新。

高效会议

大部分人不喜欢开会，但又无法避免。虽然在使用得当时效率很高，但会议经常遭到滥用，通常缺乏组织，而且总是时间过长。我们希望会议就像污水处理厂一样：数量少，间隔远，且位于下风口。这一节很简短，只讨论团队会议。

先谈谈最可怕的一种：例会。这种会议通常每周召开一次，应该只包括简单的通知和介绍——参加者（无论有没有重要信息）逐个汇报进展状态纯属浪费时间，只会让大家不胜其烦，坐如针毡，巴不得会议早点结束。

任何需要深入讨论的事情都应该在会后进行，只要求相关人员出席。这样也可以避免因某人过于深入讨论某个话题而扰乱会议计划。主持会议的人应当把需要详细讨论的话题加到一个“备注”列表，在会议结束后逐个审阅。如果团队养成这种习惯，就很容易把需要另外讨论的内容放入“备注”，大家也不会觉得不妥。成功例会的关键在于，一旦主要议程结束，人们就可以离开。如果可以通过邮件发布信息或没有需要讨论的事情，例会完全可以取消。在有些文化中，出席会议是地位的象征，因此没人愿意被忽视。坦白说，这真是荒唐。

每日例会

一些开发方法（如敏捷开发方法）倡导每日例会，有些工程师对此非常支持，但要注意保持每日例会简短有物。每日例会开始时通常都很短（15分钟），每个人都会站着开会，简单汇报他们手头的工作，但如果不注意的话，这些会议很快就会变成时长30分钟而坐着开的会议，大家就像做群组治疗似的敞开心扉滔滔不绝。如果团队有每日例会，就需要有人来负责主持会议，保持会议简短。

如果会议的目标是产生新设计，请尽量保持与会者不超过五人——除非只有一个人拍板，否则房间里超过五个人时，基本不可能产生新设计或作出决定。不相信的话，你可以找五个朋友一起去城里，六个人一起决定如何走路参观几个景点。除非你指定一个人做最终决定，大家都按他说的走，否则最后结果很可能是你们站在街角争论大半天，而毫无结果。

会议经常会打断很多人所说的“创意时间”，这个词来自Paul Graham的“Maker's Schedule, Manager's Schedule”一文。如果总是需要停下工作去参加会议，会给人们（特别是工程师）造成很多困扰。对此，你可以在日历上给自己安排3～4小时的约会，标成“忙”，甚至是“创意时间”，专门用来工作。如果一定要安排会议，可以尽量把时间选择在一天中的固定休息时段附近，例如午餐，或快下班时。为了给人们留出专心工作的时间，谷歌有一个历史悠久（可惜总是遭

到忽视）的传统“无会议星期四”。这些方法可以帮助人们留出20～30小时不受干扰的较长工作时间段。

注意

主持会议的五条简单准则：

（1）只邀请必需人员参加

（2）草拟议程并在会议开始前尽早发出

（3）完成会议目标即可散会

（4）保持会议按议程进行

（5）尽量将会议安排在中断点附近（如午餐或下班时间）

如果要安排会议，请草拟一份会议议程，至少提前一天分发给所有与会者，以便他们了解会议内容。尽量少邀请与会者（请记住同步通信的高昂代价）。有些团队成员、经理，甚至总监和副总会直接忽略不提供议程的会议邀请。

只邀请那些会议必需的人员参加。有些人发现与会人员开小差看邮件，因此禁止大家带笔记本电脑来开会，但这属于治标不治本——人们开会时看邮件是因为他们可能根本就不需要参加这个会议。

主持会议的人应该切实担起责任，对于跑题或试图独占话题的人不能犹豫，要坚决（而温柔）地制止。做到这一点并非易事，但绝对值得。最重要的是，如果已经完成了议程，你完全可以提前结束会议。

分布式团队

如果所在团队分布在多个地理位置，或者你自己远程办公，那么不仅需要寻求别的沟通渠道，还需要花更多工夫进行沟通。团队有人远程工作意味着必须用书面方式记录和共享决策，常用的方式是邮件。许多讨论可能是通过在线聊天、即时消息和走廊里的谈话进行的，但必须以某种方式将重要的讨论散布给团队的每个成员，以确保他们知情并参与其中（这样做还有一个额外好处：归档邮件列表提供了文档记录）。如今，大部分笔记本电脑都有内置摄像头，人们可以通过视频方便地交谈，这也是很有用的沟通方式。

Subversion项目组有一句格言：如果讨论不在邮件列表中进行，就没有真的发生过。人们花费许多时间在聊天室里交流想法，但要使讨论的结果变得“真实”，还必须兼顾到没有参与讨论的人。将所有讨论内容在邮件列表中重新发布，整个团队都有机会看到结论是如何达成的，而且可以发表意见。如果要打造基于共识的团队文化，这一点尤为重要。

与某人远程交谈应该和走到他身边谈话一样容易。如果你远程办公，请使用各种可行的手段（如在线聊天、即时消息、邮件、视频聊天、电话，等等）与团队细致地沟通，以确保每个人不仅知道你的存在，而且知道你在做什么。最重要的是，不要低估了面对面谈话的信息量。

Fitz以前有一位工程师与科罗拉多的团队远程合作，但无法顺利推动项目。她私下向Fitz请教，Fitz建议她飞到科罗拉多，和团队一起工作一个星期，将项目启动。两周后，她从科罗拉多给Fitz发来邮件，汇报了好消息。到科罗拉多仅仅一天后，不仅项目取得了很大进展，而且她与团队共进午餐，下班后一起小酌，建立了良好的关系。

Ben曾经有一位组员Corey与异地团队合作新项目。Corey无法与新团队达成合作默契，在每周的一对一谈话中向Ben大倒苦水。Ben告诉Corey他应该飞过去和团队一起工作一周，共同启动项目。考虑到旅行和住宿费用，Corey有些犹豫，但他忽略了这趟旅行可能带来的成效。Corey花了两天时间与团队一起工作后，立刻意识到此举的价值。他不仅在面对面交谈中获得额外的丰富信息，而且还通过午餐和休闲活动增进了对彼此的了解。旅行结束后，Corey与团队虽然仍相隔千里，但交互却变得更加顺畅。

面对面沟通的重要性

有一点必须注意，虽然社交媒体和视频会议技术日益发达，但都远远比不上在现实生活中进行面对面沟通带来的亲密感和传达的丰富信息。在开始一个新项目，或与公司中某人有重要会议时，如果经费允许，那么亲身参与带来的好处总是可以抵过旅行的麻烦。面对面讨论的影响会深深留在记忆之中，其效果绝非是电话或视频聊天可比的。

人们经常因为费用过高或无法承担而反对出差。对于分布在多地的小型公司，费用可能的确是个问题，但多数大公司都能承担这种差旅费用。不与同事面对面沟通，付出的代价比想象的更高。

不管发了多少封邮件，聊了多少次天，打了多少电话，请定期飞去与团队的其他成员会面。这一建议同样适用于远程雇员、远程团队、分公司成员，请安排时间访问总部，与人们见面交流。

设计文档

如果你是一位工程师，那么一定会偶尔产生为新项目奋力编码的冲动，但这么做很

少有好结果（除非是做快速原型）。同样，很多工程师在进行软件设计之前就匆匆开始编码，结果通常很糟糕。

设计文档通常由一人负责，两三人编写，更多人审阅。设计文档不仅是未来产品的概要蓝图，而且也是与大团队沟通的低成本方式，可以传达你的设计目标及实现方法。在设计文档编写时，你还没有花费数周或数月写代码，因此更容易接受批评意见，产生更好的产品和更佳的实现。此外，设计文档一旦定稿，就可以作为项目时间的安排和工作划分的指引。但在开始编码后，你应该将设计文档视为灵活的而非一成不变的。随着项目的发展和变化，设计文档应当随之及时更新，而不是等到产品交付才一次性修改。但这也是说起来容易做起来难。大多数团队根本没有文档，其他团队开始时有完备的文档，随后就疏于更新了。

话虽这么说，但也不要把“设计文档信仰”发展到另一个极限。有的控制狂编写100行代码的程序就要写四页设计文档。如果用写设计文档的时间可以把项目重新写好几次，那么这个设计文档显然是浪费时间。在预估时间和进行取舍时，请善用自己的经验和判断。

日常讨论

如果团队已经就高层目标达成了共识，那么就需要考虑使用何种工具来协调日常工作了。这些工具都很实用，但传递的信息有限，而且通常完全没有元数据和辅助沟通渠道（如面部表情和肢体语言），因此很容易传递错误信息，影响HRT原则。但是，这些工具对大多数团队都是很有价值的，而且只需稍加努力可以大幅提高生产力。

邮件列表

如今，没有哪个团队不使用邮件列表，这里有几条建议可以使你的邮件列表用处更大。

很多成功的大型项目有多个邮件列表，用于开发讨论、代码审阅、用户讨论、公告、自动通知、各种管理杂务。有时小型项目在启动时试图采取同样的方式，创建好几个邮件列表，而实际上项目只有三个工程师和两个用户。这相当于给五个人提供六个会议室开会——结果是讨论支离破碎，回声很大，还有很多空房间。使用邮件列表最好从一个开始，当列表中的邮件多到无法管理时（通常表现是列表成员不胜其扰），才增加新列表。自动发出的邮件和通知不在此列，这些邮件应该属于单独的列表，或者至少使用容易过滤的标识。

请抽些时间约定邮件讨论的规则——保持文明，防止“聒噪的少数人”扰乱秩序。

对于共享一个办公室的团队，邮件列表并不是讨论问题的首选方式，但你可以在邮件列表上发送会议议程、会议纪要、结论、设计文档，以及其他任何相关的文字信息，从而很方便地保存记录。对于开源项目，这些邮件列表可以设置为将所有信息归档成公共可见的索引，内部项目则可以设为公司内网可见。这样就能建立起项目的历史记录，如果新成员询问一个或多个决策背后的缘由，就可以很容易地查找参考。如果不将这些讨论归档，你会发现自己要一遍一遍地解释，永无尽头。

在线聊天

使用在线聊天工具可以在不干扰同事工作（当然，假设他把聊天工具配置为非干扰模式）的情况下，向他发出一个快速请求，用于团队沟通特别方便。如果团队正在紧锣密鼓地进行一个新项目，人们在晚上或周末处理一些工作，在线聊天就特别适合作为沟通工具，这同样也适用于离开办公室一两天的人。一对一的聊天自然不

错，但我们强力推荐使用某些群聊机制。

早在即时消息流行之前，团队就经常使用Internet Relay Chat（IRC）渠道聊天，大部分的讨论是群聊。群聊消息有时多而杂，团队成员需要小范围讨论时也很容易进行私聊，但大多数讨论都是公开的。人们可以发言、潜水，也可以查看错过的前期讨论消息。使用群聊，人们很容易发起随机讨论，而且可以便捷地形成社区（即便团队在地理上是分散的）。新成员不一定要参加讨论，仅仅通过旁观或稍后阅读就可以学到很多知识。

随着即时消息工具的发展，很多之前在群组聊天室进行的讨论转为通过私人聊天进行。私人聊天是即时消息工具的默认模式。也许你觉得一对一聊天更安全，可以提出看似愚笨的问题，避免在团队其他成员面前丢脸。不幸的是，私人聊天信息无法分享，不同的团队成员可能重复提出同样的问题，增加了团队的负担。

幸好，在2014～2015年，随着Slack的兴起，群组聊天开始复兴。Slack是一款免费（但不是自由软件也不是开源软件）的群组消息客户端，用起来很像IRC。Slack与几十种其他产品集成，成为了小型公司、初创公司，甚至互联网上松散联系人群喜爱的消息工具。虽然Slack仍然可以发送私人消息，但团队所有者可以得到私有消息与群组消息占比的周报。有了这一信息，你可以适时提醒团队多进行群组讨论。

不管选用什么聊天工具，我们都强烈建议团队使用方便易用的群组聊天机制。团队可以由此获得额外的通信带宽。

群聊与一对一即时消息

今天，很多人第一次听说IRC时，对其原始的字符环境嗤之以鼻，因为即便是最新的IRC客户端，也不如iChat或Google Talk的旧版本时髦。请不要被IRC过时的外表

和用法愚弄——IRC是为多人聊天设计的，采用异步方式，这才是它最强大的功能。大多数IRC客户端能够无限回翻聊天记录，你可以阅读之前的记录，以防错过。Slack基本可算是IRC的现代版本，虽然也有图形、头像、表情等花哨功能，但内里依然是IRC式基于文本的消息系统。视频会议、共享白板等虽然很时髦有趣，但这些系统通常效率不高，而且不能像基于文本的群组聊天那样异步查看消息。如果你不用Slack或IRC，那么请选择专门为群组聊天设计的工具，而不是简单附加了群聊功能的即时消息系统。

有时人们更喜欢在线聊天。记得我们第一次参加编程马拉松时，许多开源编程者将在此会面（很多人是第一次见面）并合作项目。我们走进一间静悄悄的房间，看到十几张桌子，每张桌坐了6～8人，所有人都在狂敲笔记本电脑键盘。我们估摸着是自己到晚了，大家已经开始写代码了，于是坐下来打开笔记本，调出编辑器，登入项目的IRC频道，想看看没能来参加大会的人都上线没有。我们在频道里打了声招呼，说自己刚到会场，结果发现好几个在IRC频道里跟我们打招呼的人其实就坐在离我们不到10英尺的地方！有些人这样是习惯使然，因为大家经常在线聊天，但对有些人，这是与组里其他人沟通的最舒适的方式。由于刚在飞机上坐了四个小时，又迫切需要与他人多进行些沟通，于是我们站起来，走到各张桌前，和人们直接交谈起来。

至于何时使用聊天工具，何时使用邮件，并无一定之规。聊天要求所有参与者都在线，以便结论达成时进行快速实时讨论。如果一些参与者不在场，或者不急于做决定，邮件更为适用。请记住我们在“成功团队文化的沟通模式”中讨论的同步沟通与异步沟通的代价对比，作出自己的选择。

使用问题跟踪系统

如果使用问题/缺陷跟踪系统（应该使用），你应该建立某种处理和仲裁缺陷的流程，鼓励人们及时报告并修复缺陷。如果系统无人照料，缺陷没有优先级，人们会停止报告缺陷，发无用的牢骚。等到团队开始处理系统中的缺陷时，极有可能会修复无关紧要的缺陷，而漏掉重要的问题。

请记住，缺陷跟踪系统只是一个稍作定制的“网络论坛”或“公告牌”。因此，缺陷跟踪系统与邮件列表有许多相同之处，也适用同样的最佳实践。人们在走廊里进行的讨论应该在缺陷跟踪系统中记录为更新消息，将想法和决定“官方化”，让大家都能看到。系统中消息应当用词文明，禁止粗俗行为。

很多时候，项目经理负责检查问题跟踪系统中所有的未关闭问题。这不仅导致大量问题被更新，还使得团队成员开始在问题跟踪系统里进行回复讨论。如果消息变得很长或不再连贯，可以把讨论暂时转移到主邮件列表中——邮件客户端更适合处理复杂的讨论。

工程中的沟通

关于软件开发过程的著作成百上千。虽然我们无法都加以讨论，但要提到几点和沟通相关的重点内容。不管你采用何种开发方法，这些内容都适用。即便你不是做软件开发的，这里也有一些值得学习的经验——特别是关于不做什么的经验。

代码注释

代码注释风格的选择因人而异。详细的注释常常可以提供与原作者意图和想法相关的信息，很有用处，但需要持续维护：过时或不正确的注释会严重妨碍人们理解代码库。同样，注释过于简单或缺少注释，会导致将来的维护人员或API使用者浪费时

间进行各种猜测。人们经常在注释里指出缺失的结构和不好的命名，然后重新解释一遍代码内容。注释应当关注代码如此实现的理由，而不是代码的功能。

函数或方法级别的注释用处最大，特别是可以用作API文档。简单地说，注释可以用一句流行的希腊名言来概括：“凡事不过。”此外，你还应该为团队制定注释风格，请大家遵守——保持风格一致比选用何种风格更为重要。注释风格指南还应该说明此指南存在的原因及规定内容，例如，下面是Google C++ Style Guide的介绍。

C++是很多谷歌开源项目使用的主要开发语言。每个C++程序员都知道，C++语言功能非常强大，但也非常复杂，导致代码更易出现问题，更难阅读和维护。

这一指南旨在通过详细描述编写C++代码时应遵循的规则，降低代码复杂度。这些规则在保持代码库可管理的同时，使编程人员依然能够有效地使用C++语言的功能。

代码风格，亦为可读性，是管理C++代码的约定。风格一词其实并不准确，因为这些约定描述的远不止是源文件格式。

保持代码库可管理的方法之一是强调一致性。任何编程者都应该能够快速理解其他人的代码，这一点非常重要。保持风格一致，遵循约定，我们就可以更容易地使用“模式匹配”，推断各种符号的含义和属性。创建通用、必需的习语和模式可以提高代码的可读性。某些风格约定有时可能值得商榷，但为保持一致起见，我们依然保持现有约定不变。

这一指南解决的另一个问题是C++功能臃肿。C++是一个庞大的语言，具有很多高级功能。在某些情况下，我们会限制甚至禁止使用某些功能。此举意在保持代码简单，避免这些功能可能导致的各种常见错误和问题。这一指南列出了这些功能，并

解释了限制使用的理由。

谷歌开发的开源项目遵循这一指南中的规定。

请注意，这一指南不是C++教程，读者应已熟悉C++语言。

请注意，这份指南并未谈及使用C++的最好或最快方式，只强调了保持代码库风格一致的重要性。

署名

每个人都想因自己所做的工作而获得应有的功劳，艺术家在绘画上签名，作者把名字印在书脊上或署在博客文章上，这些行为都是这种心理的体现。希望以各种方式得到认可是人类的天性，但我们认为在源文件里到处留名会带来许多麻烦。我们都见过源文件头部版权声明中的这种署名。

```
# ---------------------------------- # Created: October 1998 by
Brian W. Fitzpatrick <fitz@redbean.com> # ------------------------
---------
```

把名字放在源代码头部是一项悠久的传统（好吧，我们俩以前都干过），在程序由个人而非团队编写的年代，这种做法可能是适宜的。但在今天，很多人可能修改同一段代码，文件的署名问题会招致很多争议，浪费时间，伤害感情。因此，我们强烈反对源代码文件中宣示所有权的署名（最多可以指定修改此文件的首选审阅人，但不要让人觉得这是在暗示代码所有者）。

举个例子，假设你在项目中创建了一个新文件，你写了几百行代码后，在文件头写上了自己的名字和版权信息，然后将文件移交审阅，后来又提交到代码库。目前为止一切顺利，没有问题，没有冲突，没有不同意见。如果你的同事Adrian之后对这个文件做了一些修改，那么他把文件修改到何种程度就可以把自己的名字放在文件头呢？修复一个缺陷？五个缺陷？写一个函数？两个函数？又需要写多少行代码呢？如果他写了一个函数，把自己的名字署在文件头上，然后又有人重写了“他的”函数呢？那这个人可以把自己的名字署在文件上吗？可以把Adrian的名字去掉吗？与其他的合作创意工作（如戏剧、小说、电影）不同，软件即便在“完成”之后也会持续修改。因此，将参与者名字列在电影末尾是静态的安全做法，而试图向一个源文件添加或删除作者名属于疯狂的徒劳之举。

当然，你可以回答前面提出的所有问题，并对每种可能出现的特殊情况都给出规定，但维护、跟踪并时刻注意违反规定的情况，是对时间的极大浪费——这些时间本可以用来编写代码。正因为如此，我们主张在项目级，而非代码级标注所有权。我们见到的大部分项目会有一个“作者”或“参与者”文件，列出对项目有贡献的

每一个人。如果需要更多细节，可以在版本控制系统中查看。当然，如果你不使用版本控制系统，所有这些瞬间都将湮没在时间的洪流中，如泪水般消失在雨中。

每次提交必有审阅

如果要制定编码标准，就需要方法对进入项目的代码进行监控。无论是在代码提交之前还是之后审阅，都应该确保进入代码库的每一行代码都经过了除修改者之外的审阅者检查，从而保证其风格、质量，当然，还要防止粗心造成的错误。请保持代码每次只进行小修改，而且修改可审阅——数千行的修改基本无法审阅，只能看看格式而已。严格的审阅不仅可以提高代码库质量，而且可以极大地提升项目成员对代码质量的集体荣誉感。关于代码审阅的更多信息，请参见“隐藏是有害的”一节中关于反馈循环的部分。

测试与发布流程

无论是用完全的测试驱动开发，还是只对产品进行简单的回归测试，自动化测试越多，在修复缺陷或增加新功能时你的自信度就会越高。一旦团队决定了测试的作用，测试就应该成为编码和审阅流程的一部分。同样重要的是，发布流程应该足够灵活，可以进行经常性的版本发布（例如，每周发布），也要足够完善，可以在产品提交给用户之前发现问题。

一切为了产品

团队文化和沟通习惯反映了我们喜欢的工作方式，可能带有一些主观喜好，但并不像你可能认为的那般主观。我们发现，如果打造一种强大、有效的团队文化，花时间注意团队的沟通，创建出的团队会将更多的时间花在编写和提交产品上，而花费

更少的时间争论提交什么样的产品。

强大的团队不会自发产生，需要团队领导和创始者理解失衡团队编写软件的高昂代价，从而精心培育和呵护团队。如果团队成立之初就多加注意，可以培育出自我选择的团队文化，使得打造出的团队有更多时间设计和创造产品，无需费力定义和维护团队文化。这种努力——沟通和过程——还可以带来一大附加好处，即显著减小新来者融入团队的阻碍。如果没有这些元素，新来者要么会浪费大量时间努力学习团队如何工作，要么放弃努力，转而使你的团队按他们之前团队的做法工作（不知结果是好是坏）。

选择合适的人加入团队，为团队注入正确的理念固然重要，但培育团队文化所需的最大努力其实就是沟通。任务说明书、会议、邮件列表、在线聊天、代码注释、文档，甚至决策过程都是团队（内部及与外界）沟通的不同方式。只是为了创造一个产品而打造一支强大的团队，竟需要如此多的沟通、情感、时间和努力。人们常常对此感到惊讶，但事实就是如此。你的产品最终与人沟通，而不仅仅与机器沟通。

不管团队文化是什么样的，团队沟通多么顺畅，每支高效的团队都要有一个领导者。下一章将探讨最高效团队的领导者具有何种特征，为何他的角色并非如你所想，以及为什么每个团队成员都需要掌握领导团队的基本知识。

第3章 群龙不可无首

就算赌咒发誓自己永远不会当“经理”，在职业生涯的某些时候你还是会不小心走上领导岗位。这一章将帮助你理解如何处理这种情况。

写给经理的管理书有很多，但这一章是写给担任非正式领导职责的基层员工的。出于各种原因，大多数人害怕当经理，但一个团队没有领导者就无法运作。我们不是要说服你成为一名经理（虽然我们俩现在都是经理！），而是要说明为什么团队需要领导者，为什么人们通常不愿当经理，以及为什么最好的领导者用HRT原则为团队服务。除此之外，我们还将详细介绍领导模式和反模式，以及激励机制。

透彻地理解领导方式是影响工作方向的关键技能。如果想主导项目而不是随大流，就需要了解如何掌握方向，不然定会搁浅。

自然界里无真空

一艘没有船长的船无异于一间漂浮的等候室——除非有人掌舵并发动引擎，否则这艘船只会随波逐流。项目和船一样：如果没有人领航，项目团队就是一群只会等待事情发生而无所事事的极客而已。

如果项目要有所进展，就需要有人坐上驾驶座，不管此人是不是官方任命的领导者。如果你有主动性又不善等待，也可能成为驾驶座上的那个人。你可能发现自己整日忙于帮助大家解决冲突，作出决策，协调工作。这种事情经常发生，而且非常偶然。你并不想成为一名“领导者”，但不知怎的就担任了领导者的角色。有些人将这种现象称为“管理炎”。

经理是个贬义词

如今的经理形象概念部分来自旧时，最初源自军队，后来又传到工业革命——那都是100年多前的事情了！工业革命时期出现了很多工厂，这些工厂需要（通常是无技术的）工人操作流水线。因此需要监工管理这些工人。有很多人愿意进工厂工作，流水线工人很容易替换，因此经理不需要善待工人或改善工人的工作条件。且不论是否人道，这种做法在员工只执行简单重复任务的时代存在了很多年。

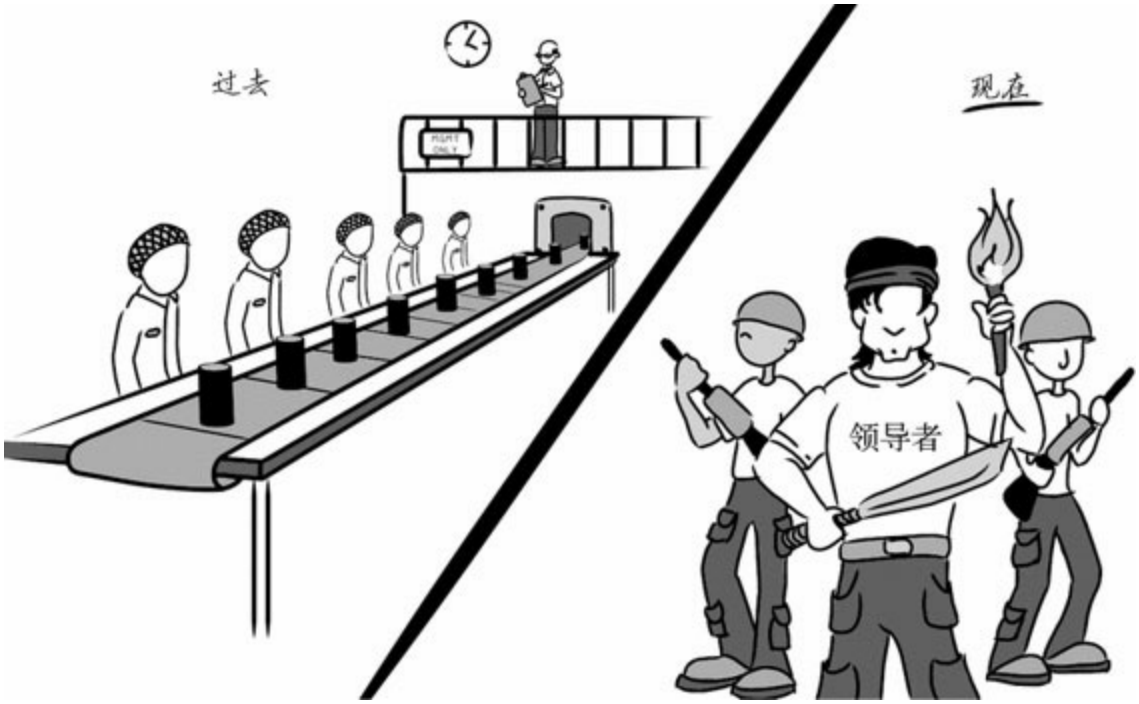
经理常常像车把式对待拉车的骡子一样对待员工：用胡萝卜激励、领导骡子前行，如果此法不好使，就改用棒子。这种“胡萝卜加大棒”的管理方法从工厂一直沿用到现代办公室。20世纪中期，办公室职员年复一年地从事同一项工作（他们通常需要依赖这份工作的养老金），像赶骡人一样强硬的经理形象大量存在。

这种管理方法在某些行业至今仍在使用，甚至包括一些需要创意思考和问题解决能力的行业，但大量研究表明，这种胡萝卜加大棒的过时方式用于创新人群效率并不高，且对生产力有损。以前的流水线工人也许可以快速上手，并随意替换，但知识工人需要数月才能跟上新团队的工作节奏。与可替换的流水线工人不同，知识工人需要培养，以及足够的时间和空间才能进行思考和创新。

新时代的“经理”是“领导者”

虽然“经理”这个头衔已然过时，但大多数人仍然在使用它。我们认为，经理这个词应该替换为领导者。虽然我们无意追求政治正确性，但经理已经成了一个贬义词——经理这个角色的存在就会促使新的经理去管理自己的下属。经理最后就像家长一样，而员工也会表现得像一群孩子。用HRT原则来解释就是：如果经理很明显地表现出他信任员工，那么员工感受到的压力是正面的，希望自己能不辜负经理的信任。就这么简单。领导者为团队创造条件，照看成员的安全和健康，同时确保大家的需求得到满足。如果说这一章什么内容最重要，那就是：

传统的经理关心如何做事，而领导者关心做什么事（并相信团队知道如何将其实现）……



几年前，一位工程师Jerry新加入Fitz的团队。Jerry的上一位经理（在另一家公司）要求Jerry每天必须在办公室从早9点呆到下午5点，而且认为，如果缺勤就说明工作未完成（当然，这种想法很可笑）。上班第一天的下午4：40，Jerry来到Fitz办公室，结结巴巴地说自己有一个重要约会，必须提前15分钟离开办公室，并为此表示道歉。Fitz看了看他，笑了，直接告诉他：“只要完成了工作，我不在乎你几点下班。” Jerry傻傻地看着Fitz几秒钟，然后出去了。Fitz以成年人的方式对待Jerry，只要Jerry按时完成工作，Fitz并不关心Jerry是不是在办公室，因为Jerry不是孩子，不需要保姆。

作为一名“领导者”，你并不一定需要为所有事情负全部责任。领导方式有多种，有些走技术路线，有些走人事管理路线。在软件开发领域，领导团队的人有两种不同的角色（和头衔）：技术主管（tech lead，TL）和技术主管经理（tech lead manager，TLM）。TL通常负责产品的整体（或部分）技术方向，而TLM除了负责产品的整体（或部分）技术方向外，还要兼顾团队成员的职业发展及心理健康。如果人们愿意，可以选择担任负责项目技术的主管，避开人事管理。

唯一需要担心的是，所有事

除了大多数人听到经理这个词会感到不舒服外，人们不希望当经理还有另外一些原因。在软件开发行业，最主要的原因是当经理会减少很多写代码的时间，不管当技术主管还是技术主管经理，都确实存在这个问题。我们稍后会对此进行更多讨论，现在先介绍一下大部分人避而不当经理的其他原因吧。

如果你的大部分工作是写代码，那么一天结束时你可以指着某样东西——代码、设计文档或刚刚关闭的一堆bug——欣慰地说：“这是我今天的工作成果。”按照这种产出指标，一天忙碌的“管理”之后，你通常会想：“我今天啥也没干。”这就好像你多年来一直按每天摘的苹果计算工作量，工作变成摘香蕉之后，你收工时对自己说“我今天一个苹果也没摘”，却对身边的一大堆香蕉视而不见。对管理工作进行量化比计算组装好的工具要困难得多。你不能把团队的工作成果算成自己的功劳，但保持团队成员心情愉快、工作高效是衡量你的工作的重要标准。请不要犯摘香蕉时数苹果的错误。

还有一个很少明说的重要原因，源自著名的“彼得原理”。彼得原理称：“任何层级组织里，每个人都将晋升到他不能胜任的阶层。”大多数人都遇到过不称职或管理水平很差的经理，还有些人就没遇到过好经理。如果你工作中见过的经理都很差劲，你又怎么会想当一名经理呢？为什么要晋升到一个你无法胜任的角色呢？

当经理有很多好理由：首先，当经理可以扩展自己。就算你很擅长写代码，能写出的代码总量也有个上限。想想领导一支优秀工程师组成的团队能写出多少代码！其次，说不定你很擅长当经理——很多人由于缺人不得以当了经理，结果发现自己特别善于为团队提供所需要的指导、帮助和氛围。

服务型领导

有时候新晋经理会忘记自己的经理曾做过的不得人心的事，突然开始用同样的手段“管理”向他们汇报的人，就像犯了病一样。这种病的症状包括（但不限于）微管理、忽视效率低下者、雇用容易控制的人。若不尽快治疗，这种病会扼杀整个团队。当我们自己初次在谷歌担任经理职位时，得到的最好建议来自一位工程总监Steve Vinter。他说：“首先，不要急于管理。”刚当上经理的人最大的冲动就是对下属进行积极的“管理”，毕竟这就是经理的工作嘛，对吧？但这样做的后果通常是灾难性的。

治疗“管理”病，需要大量使用我们所称的“服务型领导方式”，也就是说，领导者能做的最重要的事是为团队服务，就像管家照料家人的健康和福利一样。作为一位服务型领导者，你应当努力营造一种HRT氛围。为此你可能需要消除团队成员无法自己移除的管理障碍，帮助团队成员达成共识，甚至在团队成员加班时为大家订餐。服务型领导者为团队搭桥铺路，在需要时给出建议，而且依旧愿意从事具体工

作。服务型领导者唯一需要管理的是团队的技术和人际关系的健康。虽然你可能只想关注团队的技术健康，但团队的人际关系健康也同样重要（但管理难度常常大得多！）。

反模式

在列举成功领导者的各种“设计模式”之前，先回顾一下若要成为成功领导者不应陷入的一些模式。我们从一些不称职的领导者身上观察到这些毁灭性的模式，并且我们自己也曾犯过不少错误。

反模式：雇用软弱者

如果一位管理者（出于某种原因）对自己担任的角色缺乏安全感，若要确保无人质疑其权威或威胁其地位，方法之一是雇用那些容易控制的人。雇用不如自己聪明或有野心，又或者比自己更缺乏安全的人，即可达到目的。这样做的确可以确保你作为团队领导者和决策者的地位，但会增加大量工作。没有你的严密监管和领导，团队就寸步难行。如果团队里都是些只会听从命令的成员，你可能无法休假，因为一旦离开，所有工作就会即刻停顿。但这只是获得工作安全感要付出的微小代价，对吗？

其实，你应该努力雇用比自己聪明，可以替代自己的人。做到这一点很难，因为这种人会不断提出挑战（还会毫不客气地指出你犯的错误）。同时，这种人也会不断展现能力，令你刮目相看。他们能拓展自己的能力，有些也有领导团队的意愿。你不应将此视为权力旁落，而应将其视为一个机会，使你可以领导更多团队，寻求新机会，或者休个假，不用每天检查团队是否完成了工作。

反模式：忽视表现不佳者

Fitz刚开始在谷歌带领团队时，有一次负责给团队发奖金，他咧着嘴高兴地告诉自己的经理：“我真喜欢当经理！” Fitz的经理，一位业界老将，当时就回答说：“有时你扮发钱的牙仙，有时你还得扮拔牙的牙医呢。”

拔牙是件苦事。有些团队领导者尽心尽力打造团队，却因为一两个表现差的成员拖了团队后腿（甚至导致队伍分崩离析）。人的因素是软件编写中最难的部分，而管理人最难的部分则是管理那些无法达到预期的成员。有些人是因为工作时间太少或不够努力未完成任务。有些人则是能力不足，无论怎么加班或努力工作也于事无补，这种情况最难处理。

谷歌中负责所有服务运维的团队有一句格言：愿望不是策略。在处理表现不佳者时，人们常常将愿望当作策略。大部分的团队领导者会咬紧牙关，睁一只眼闭一只眼，暗自希望表现不佳的成员会突然有所长进，或者自行离开。而这两种情况都极少发生。

当领导者在希望中等待时，表现不佳者没有任何改进（或离开团队），团队中表现优异者则浪费宝贵的时间弥补他人的工作，团队士气逐渐消散无形。虽然你尽力忽视表现不佳者，但团队肯定知道他们的存在——大家对谁是表现不佳者心知肚明，因为他们拖了大家的后腿。

如果忽视表现不佳者，新的表现优异者不会加入团队，已有的得力成员也会离开。最终剩下的整支团队都是表现不佳者，因为只有他们无法主动离开。最后，把表现不佳者留在团队中其实对他们没有任何好处，有些人在这支团队表现不佳，到了别的地方却能大施拳脚，这种事情经常发生。

尽快处理表现不佳的团队成员，就有机会帮助他改进工作或离开团队。如果及时处理，常常只需要一些鼓励或指导就可以提高他的生产力。如果拖得太久，他与团队的关系可能会恶化到无法挽救的地步，最后只能让他离开。

如何有效指导表现不佳者呢？通过痛苦尝试和失败，我们俩都（不幸）获得了不少这方面的经验。最贴切的比喻是，帮助一个瘸腿的人重新学习走路，然后慢跑，最后与团队一起并肩驰骋。这一过程总是需要一些微管理——但也需要大量的HRT，尤其是尊重。你可以设定一个具体的时间计划（如两三个月），列出希望他在此期间实现的具体目标。目标设定应该小而渐进，以得到许多小的成功。每周与这位成员会面以检查进度，每个里程碑都设定非常明确的期望值，使失败或成功容易衡量。如果这位表现不佳的成员无法跟上进度，那你们俩在此过程初期就能发现。此时，这位成员通常会认清现实，决定离开，也可能痛下决心，奋发努力。不管具体结果如何，直接面对和处理问题，可以帮助表现不佳者作出重大而必需的改变。

反模式：忽视人际问题

前面谈到，团队领导者对团队的管理有两大方面：人际交往和技术。团队领导者通常在技术方面较强，而且大部分领导者是从技术岗位（工作的主要目标是解决技术问题）提拔的，因此容易忽视人的问题。当基层员工时，你把主要时间都花在了解决技术问题上，成为团队领导者后也很容易把全部精力集中在技术方面。上学时，课堂讲授的都是工作中所需技术的方方面面。但成为领导者后忽视团队中人的管理是不明智的。

先讲一个领导者忽视人际问题的例子。多年前，Fitz的一位好友Jake有了第一个孩子。Jake和Fitz共事多年，既远程合作过也共享过办公室，因此孩子出生后的几周Jake选择在家办公。这极大地帮助了Jake和他的妻子，Fitz已经很习惯和Jake远程

合作，因此也无异议。Fitz和Jake工作依然高效，直到他们的经理Pablo（在另外一间办公室办公）发现Jake大部分时间在家办公。虽然Jake和以前一样完成工作而且Fitz也没意见，但Pablo对Jake不来办公室很不高兴。Jake试图向Pablo解释他来不来办公室都能完成工作，而且在家办公一段时间的话，他和妻子可以更好地照料孩子。Pablo对此的回答是：“哥们，谁还没生过孩子啊。你必须来办公室上班。”可想而知，（平时性情温和的）Jake非常愤怒，不再那么尊重Pablo了。

Pablo本来可以用很多方式更好地处理这个问题：他可以对Jake希望在家照料妻子的想法表示理解，如果Jake和团队的工作不受影响，让Jake继续在家办公一段时间。他也可以和Jake商量，让Jake一周来办公室一两天，直到忙过这阵子。不管最终结果如何，如果Pablo能显示一点同情心，Jake可能会容易接受得多。

反模式：与所有人为友

大多数人第一次担任管理职位是成为所在团队的领导者。很多人不想失去此前与团队培养的友谊，因此在成为团队领导者后会对团队成员刻意维护。这样可能带来不良后果，导致友谊破裂。请不要将友谊与温和的领导方式混为一谈：当你手握影响某人职业发展的权力时，他可能会对你曲意逢迎。

请记住，不成为团队成员的伙伴（或铁腕强权）也能领导一支团队。同样，不抛弃旧日友情也能成为强硬的领导者。要与团队成员保持人际交往而又避免使大家不自在，同进午餐是一种有效的方式——大家有机会在通常的工作环境之外随意地交谈。

有时，成为好友或伙伴的经理很难办。如果经理的朋友不善自我管理或工作不勤奋，大家都会感到压力。我们的建议是，要尽可能避免陷入这种局面。

反模式：放宽招人标准

史蒂夫·乔布斯曾说：“一等人招一等人，二等人招三等人。”这句话特别准，尤其是在急着招人的时候。一种常见的情况是：团队需要招五名工程师，因此筛选收到的简历，面试40或50人，选其中最好的5人，不管他们是否达到了招聘标准。这是组建一支平庸团队的最快方法。

找到合适的人选代价固然不菲——不管是找猎头、发广告，或是找人推荐——但其成本远远低于处理一名本不该招进来的员工。后者的“代价”体现在团队生产力降低、团队压力、培训这名员工所花的时间，以及解雇员工要走的流程和所受的压力上。当然，这是假设你要避免将其留在团队中而导致的巨大开销。如果你无权决定所管团队招聘什么人，又对招来的人不满意，必须寸土不让，据理力争，才能得到更优秀的工程师。如果总是塞给你不够格的工程师，也许你应该换个工作了。没有构建优秀团队的原材料，只能注定失败。

反模式：把团队当孩子管

要让团队知道你不信任他们，最好的办法就是像对待孩子一样对待他们。你怎么对待人们，人们就会怎么做。因此，如果你把他们当孩子或囚徒对待，他们也会表现得像孩子或囚徒一样。你可以试试对团队事无巨细地管理，或者干脆不尊重团队成员的能力，不给他们任何承担工作责任的机会，这样就可以使他们表现得像孩子一样。如果总是因为无法信任团队成员而需要微管理，那你就招错了人。当然，如果你的目标就是组建一个需要耗尽自己后半生照看的团队，那就不算招错人。如果招聘值得信任的人并委以重任，他们通常都能很好地把握机会（基本前提是你招到了优秀的团队成员）。

Fitz在芝加哥主持的一个会议曾经租用了一家当地机构的场地。Fitz第一次去的时候，物业经理带着Fitz简单地转了一圈，介绍各种设施都在什么地方，然后把大楼的钥匙交给了Fitz，告诉他下周交回钥匙。这位经理没有列出条条框框，也没有全程跟进监督。回报这份信任，Fitz和他的团队自觉有义务像对待自己的地方一样照料这里，尽心尽力地保持场地整洁有序。

这种程度的信任不仅适用于大楼钥匙，也适用于办公室和计算机耗材的管理。例如，谷歌提供装满各式各样办公材料（如笔、记事本，以及其他传统的办公工具）的柜子，员工可以随意取用。IT部门设置了很多“技术点”，以提供类似微型电子商店的自助区域，其中有很多方便拿取的计算机零配件（如电源、数据线、鼠标、USB盘等）。但因为公司信任大家自行取用这些设备，员工觉得自己有责任自律。很多传统公司的人听到谷歌的这种做法极为震惊，认为谷歌肯定会因为员工“偷窃”这些办公设备而暗中蒙受损失。这种可能性当然存在，但如果员工表现得像孩子一样，那会带来多少损失呢？代价肯定高于几支笔和USB线的价值。

领导模式

通过观察其他成功领导者，尤其是自己的导师，我们从经验中学到了一些成功领导方式的行为模式。这些模式不仅最具实效，而且是我们在追随的领导者身上最为看重的。

放下自尊

第1章介绍HRT时讲过“放下自尊”，这一点对服务型领导者尤其重要。人们经常将这一模式误解为鼓励领导者毫无原则，任由团队指挥，但其实并非如此。我们承认，谦虚和让人利用之间的界线很模糊，但谦虚绝不同于缺乏自信。不必成为自大

狂也一样能拥有自信和主张。任何团队中过强自尊心的人都很难相处，更别说是团队领导了。实际上，应该努力培养强大的团队集体自尊和认同感。

要做到“放下自尊”，一部分是指要做到前面介绍过的：信任团队。这要求尊重团队成员的能力和成就，对新成员也是如此。

如果你没有对团队进行细致的管理，那么负责具体工作的成员必然比你更了解任务细节。也就是说，虽然你负责促使团队达成共识，帮助设定工作方向，但具体执行的人更明白实现目标的具体步骤。这样一来，他们不仅能增强主人翁意识，还能对项目的成功（或失败）产生更强的责任感。如果团队能力很强，并可以自己设定工作的质量标准和进度，那么他们取得的成果将比“胡萝卜加大棒”管理方式下取得的成果要多得多。

大多数人在刚担任领导角色时都觉得自己必须做对所有事情，了解所有情况，并解决所有问题。相信我，你不可能把所有事情都做对，也不能解决所有问题，而且如果你总是抱着这种想法，很快就会失去大家对你的尊重。克服这种想法，需要对自己的角色有基本的安全感。回想一下当普通员工时，你会发现自己的不自信有多么明显。请试着接受别人的质询：当有人对你的决定或声明提出疑问时，请记住，这个人通常只是试图更好地理解你。如果鼓励别人提出质询，就会有更多机会得到建设性批评，从而成为一个更好的领导者或能够自我改进。能提出建设性批评的人本来就很少，要从“为你工作”的人那里得到建设性批评就更难了。请想想为团队设立的愿景，敞开心胸接受意见和批评，不可固步自封。

“放下自尊”的最后一点很简单，但很多工程师宁愿下油锅都不愿做：犯错时道歉。道歉不是将“对不起”挂在嘴边，而是要真心实意。犯错是必然的，不管你承认与否大家都会知道。无论大家是否和你谈论此事（有一点是肯定的：他们会私下

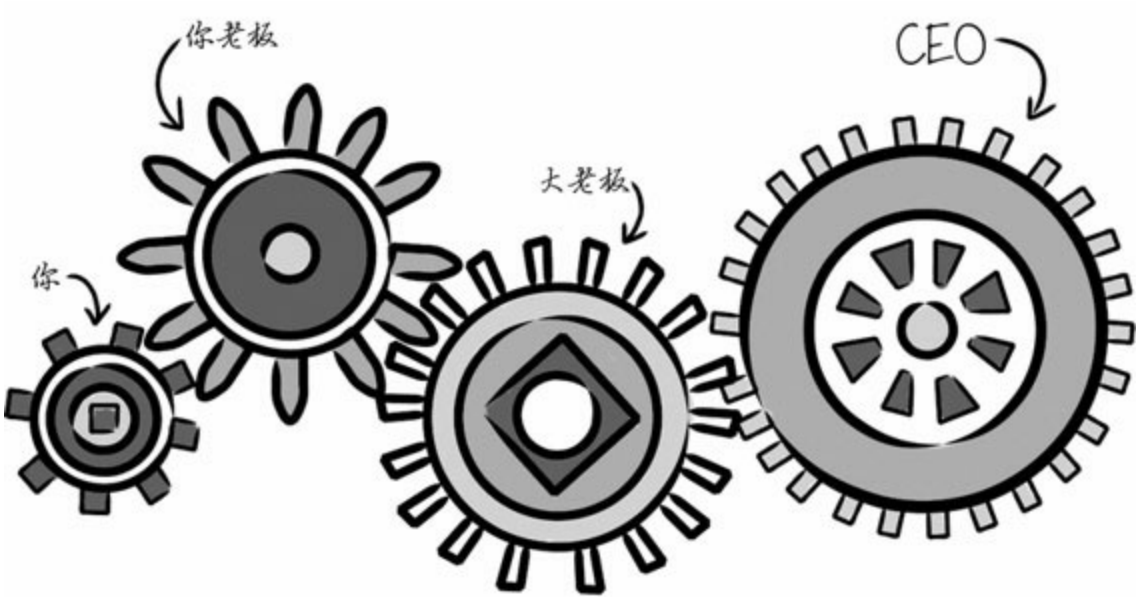
谈论此事），他们确实知道你犯了错。道歉没有成本。人们对犯错时敢于道歉的领导者怀有莫大的敬意。与大家想的相反，道歉并不会使你变得脆弱。实际上，道歉通常会赢得人们的尊重，因为道歉表明你头脑冷静，能够审时度势，而且——回到HRT原则——态度谦虚。

成为禅意大师

作为一名工程师，你可能惯于持怀疑态度，还有些愤世嫉俗，但这种心态不适合领导团队。不是说要时时保持盲目乐观，而是要少发表一些怀疑言论，同时要让团队知道你了解工作中的复杂情况和困难障碍。当领导较多人时，整个团队都会（有意无意地）把目光投向你，以寻求如何对周遭采取行动或作出反应，因此仔细斟酌如何应对和保持冷静更为重要。

你可以把公司架构看作一组齿轮，一边的齿轮齿较少，代表基层员工，其上的每个

经理是与之咬合的另一个齿轮，而CEO是有几百个齿的最大齿轮。每次某人的“经理齿轮”（可能有几十个齿）转动一圈，这个“个人齿轮”要转两三圈。CEO的小小举动，就会导致六七层齿轮下的倒霉员工疯狂转动！你在齿轮上所处的层次越高，下面的齿轮转动得就越快，无论你愿不愿意都是这样。



另一种说法是领导者总是身在舞台之上。这种说法表明，如果处于一个明显的领导岗位，人们总是在观察你：不仅仅在开会或讲话时，甚至是当你坐在桌前回邮件时。同事会观察你的身体语言，闲聊时的反应，以及吃午饭时的表现，以寻找蛛丝马迹。他们看到了自信还是恐慌？作为一名领导者，你的工作是激励大家，而激励是一项需要时刻进行的工作。你对于任何事（无论多小的事）的态度都会在无意间受到关注，从而影响整个团队。

Fitz曾经的经理Bill就完全掌握了在任何时候保持镇定的能力。不管出了什么事，事态有多疯狂，波及有多广，Bill都不会乱了阵脚。大多数时候他会一手抱胸，一手支颐，向完全慌了神的员工提出相关问题。这种做法能让这位员工镇定下来，帮他集中精神解决问题，而不是像无头苍蝇一样乱转。Fitz开玩笑说，如果有人告诉Bill，公司有19处办公室被外星人袭击了，Bill的反应将是：“你觉得他们为什么

不袭击20处呢？”

这是禅意管理的另一个难点：提出问题。当团队成员向你寻求建议时，你通常会很兴奋，觉得终于有机会解决点问题了！这正是你在走向领导岗位之前从事多年的工作，因此你会直接进入解决问题模式，这恰恰是不应该做的。寻求建议的人通常不是希望你替他解决问题，而是帮助他解决问题。最简单的做法就是问他问题。如果问的问题不在点子上，只会让人抓狂，毫无用处。你可以使用HRT原则，尝试细化和探究问题，帮助他自己解决问题。最终，这些问题会引领这位员工找到答案，而且答案是他自己找到的。如前面讨论的，这种方式可以增强员工的主人翁意识和责任感。无论你是否知道答案，使用这种技巧总是能让员工觉得你知道答案。有趣吧？苏格拉底都会以你为傲。

成为催化剂

在化学中，催化剂是一种可以加速化学反应但并不损耗自身的物质。催化剂（如酶）的一种工作方式是将反应物拉近：如果有催化剂将反应物拉近，反应物将不需要在溶剂中随机碰撞，从而使交互更容易发生。作为领导者，你需要经常扮演催化剂的角色，具体方式有很多种。

团队领导者最常做的一件事是促成共识。你可能需要从始至终驱动整个过程，或往正确的方向略做推动以加速过程。帮助团队达成共识是非正式领导者经常使用的一种领导技能，因为只有通过这种方式才能在不具有任何实际权力的情况下领导团队。如果有权力，你可以指挥或发出指令，但整体效果不如促成共识有效。如果团队需要加快进度，成员有时会自愿服从权威，听从一个或多个团队领导的指挥。这可能看起来有点像独裁或寡头政治，但如果大家自愿如此，就是一种共识。

知道在哪里做记号

有一个故事讲的是一位退休已久的机械大师。他的前雇主遇到了一个无人能解决的问题，因此请这位大师帮忙解决。大师检查了出问题的机器，倾听良久，最后摸出一个粉笔头，在机器的一边画了个小小的×。他告诉技师，在记号的位置有一根松了的电线需要修理。技师打开机器，拧紧了那根松了的电线，问题就解决了。当大师开出的10 000美元账单送到时，CEO大怒，要求列出做这个简单记号的天价账单的明细。于是大师又送来一份账单，列出做记号的粉笔价值1美元，知道在哪里做这个记号价值9999美元。

对我们而言，这个故事讲的是智慧的价值：一个简单、慎重思考后的调整可能产生巨大的影响。Ben尝试在管理员工时使用这一技巧。他想象团队乘坐着一艘飞艇，正朝某个方向缓慢而坚定地移动。Ben没有用微管理持续调整航向，而是用一周的大部分时间仔细观察、聆听。在这一周结束时，他在飞艇上一个具体位置做了一个小小的粉笔记号，在此处作出微小而关键的一“触”，调整前进的方向。

有时，团队已经就需要做什么达成了共识，但可能因碰到一个障碍而无法前进。这个障碍可能是技术问题或规章限制，全力帮助团队克服障碍继续前进是常见的领导技巧。有些障碍对团队成员而言基本无法跨越，对你却可能是易如反掌，要让团队知道你很乐于而且有能力帮助他们克服这些障碍。

有一次，Fitz的团队花了几周试图绕过公司法律部门的一个障碍。最后他们无计可施，去找Fitz，Fitz花了不到两个小时就解决了这个问题，因为他知道应该找谁。另一次，Ben的团队需要一些服务器却无法得到。幸好，Ben认识公司里的另一支团队，当天下午就帮团队拿到了所需的服务器。还有一次，Fitz团队的一位工程师搞不定一段难懂的Java代码，虽然Fitz自己不是Java专家，但他帮这位工程师介绍了

另一个正好懂这个问题的工程师。要帮助团队克服障碍，不必知道所有的答案，但认识知道答案的人通常很有用。很多时候，认识正确的人比知道正确的答案更有用。

失败是一个选项

催化团队的另一个方法是让他们觉得安全无虞，能够尝试更大的风险。风险是件有意思的事情——大多数人极不擅长评估风险，大多数公司竭尽所能规避风险。由此导致的结果是，人们惯于保守行事，关注小的成功，即便较大的风险能带来指数级的更大成功也不愿尝试。谷歌有个常见的说法是：如果尝试完成不可能的任务，那么很可能会失败，但如果因此失败，获得的成果仍然可能比只尝试自己能力范围内的任务要多得多。要打造敢于冒险的团队文化，一个好办法是让团队知道失败也没关系。

我们不妨说清楚：失败也没关系。实际上，我们喜欢将失败看作快速学到大量知识的方法（前提是不要重复同样的失败）。另外，要将失败视为学习的机会，而不要趁机指责或怪罪别人。快速失败是件好事，因为不会有太多风险。缓慢的失败也很有价值，但由于风险过大损失过多（通常损失的是项目时间），过程比较痛苦。如果影响到客户，这种失败可能是我们最不愿意遇到的，也是最需要从中吸取教训的。前面讲到，谷歌每次出现产品失败后，都会进行事后分析报告。这一过程记录了导致失败发生的事件，并总结出防止此类事件再次发生的一系列步骤。其目的不是追究责任，也不是建立不必要的官样检查，而是关注问题的核心，一劳永逸地解决这个问题。这很难做到，但非常有效（而且令人浑身舒畅）。

个人的成功和失败稍有不同。称赞个人成功没什么关系，但在失败时指责个人会分化团队，挫败大家承担风险的精神。失败并不可怕，但要让团队整体承担失败，并

从失败中学习经验。如果某个人成功，请在整个团队面前表扬他；如果某个人失败，请私下给他建设性批评。无论如何，请借此机会应用HRT原则，帮助团队从失败中总结教训。

成为老师和导师

作为团队领导者，最痛苦的事情之一就是看着经验较少的团队成员花三个小时研究你自己20分钟就能解决的问题。指导他人，给他们学习的机会，一开始会异常困难，但这是有效领导能力的一个关键组成部分。对于刚进入团队，正在学习团队的技术和代码库，同时也在学习团队文化，了解自己应承担职责的成员，这一点尤为重要。

就像很多人因为偶然的机会当上了经理，大部分人也是无意间担当了导师的角色——在团队主管找人带新人时他们成了导师。当导师并不需要很多正式的培训或准备。实际上，当导师主要有三个条件：熟悉团队的流程和系统；能够向别人清楚地解释事情；能够判断新人需要多少帮助。最后一条特别重要——你应该提供给新人足够的信息，但如果解释过多或太啰嗦，对方不会客气地告诉你他已经明白了，反而很可能会听而不闻。

设定清晰的目标

这个模式看似明显，却被众多领导者忽视。要想团队向一个方向快速前进，需要确保队员都明白并认可前进的方向。你可以把产品想象成一个大卡车（而不是一列管子）。每个团队成员手持一根系在卡车前部的绳子，当工作时，就将卡车朝自己的方向拉。如果目标是将卡车（产品）尽快向北移动，就不能让团队成员向各个方向使劲——他们应该都把车向北拉。

设定清晰的目标，使团队将产品朝同一个方向拉，最容易的方法是为团队创建一份简明的任务说明书（详情请参见第2章中“任务说明书”一节）。帮助团队定义了方向和目标之后，你就可以靠边站，给团队更多自主权，只需要不时了解一下情况，确保大家仍在正确的轨道上就可以了。这样不仅为你节省时间来处理其他的管理工作，还可以极大提高团队的效率。如果没有清晰的目标，团队（确实）也可以成功，但是各个成员把产品拉向略微不同的方向，通常会浪费大量的精力。这会令人沮丧，拖延进度，使你不得不把越来越多的精力放在修正工作方向上。

以诚相待

强调以诚相待不是说你对团队不诚实。只是终有一天你会发现有些事情不能告诉团队，或者更糟，不得不告诉团队一些大家不想听的话。Fitz以前的一位经理会对团队的新成员说：“我不会对你说假话，但如果有事情不能透露，或者我也不知情，我会如实地告诉你。”

如果一位团队成员向你询问某事而你不能直言相告，你可以直接告诉他自己知道答案但不能说。更常见的是，有些成员问的事情你也不知道，那你也可以直接告诉他自己不知道。这个建议看起来再明显不过，但是很多人担任领导角色后，觉得如果不能回答某个问题，就说明自己能力不够或者没跟上形势。实际上，做不到无所不知只证明了领导也是人而已。

给出负面的反馈是件很难做的事情。当第一次告诉手下他犯了错误或者工作没有达

到预期时，你会觉得压力异常大。大部分管理教材建议，在给出负面反馈时可以使用“三明治评价法”缓和气氛。“三明治评价法”示例如下。

“你是团队的中坚力量，是最聪明的工程师之一。虽说如此，但你的代码太晦涩了，组里几乎没人能看懂。但你很有潜力，而且这把大胡子太帅了。”

的确，这么说显得不那么严厉，但这么旁敲侧击之后，大多数人散会时只会想：“耶，我的胡子好帅！”我们强烈反对使用三明治评价法，这不是认为你应该毫无必要地显得粗鲁或严厉，而是因为大多数人会错过其中的关键信息，这就是需要改进的地方。我们可以应用HRT原则：在提出建设性批评时，不使用三明治评价法也表现出善意和同情心。实际上，如果你希望对方能听取批评，不产生反感，善意和同情是关键。

很多年前，Fitz的团队转来一位新成员Tim。Tim原来的经理认为无法与之合作，他告诉Fitz，Tim从来不接受意见或批评，告诉他不该做某事他还一直坚持不改。Fitz旁听了这位经理与Tim的几次会议，观察两者间的互动，发现这位经理为了避免伤害Tim的感情，大量使用了三明治评价法。Tim转到Fitz的团队后，Fitz非常清

楚地向Tim说明，他需要作出一些改进才能和团队更有效地合作。Fitz没有表扬Tim或美化问题，但同样重要的是，Fitz也没有表现得刻薄——他只是列出了基于Tim在之前团队的表现所观察到的事实。你猜怎么着？短短数周（和另外几个“提醒”会议）之后，Tim的表现有了大幅改进。Tim只是需要非常清晰的反馈和明确的方向而已。

在提供直接反馈或批评意见时，要想确保消息传达到位，表达方式是关键。如果让对方产生反感，他就不会思考该如何改进，而是想方设法与你辩解，证明你说得不对。Ben曾经管理过一位叫Dean的工程师。Dean特别坚持己见，总是和团队其他成员争辩。辩论的焦点大到团队任务，小到Web页面上一个微件的位置。无论事情大小，Dean总是抱着同样的热情和信念进行辩论，寸步不让。如此数月之后，Ben找来Dean，告诉Dean他太好斗了。如果Ben说“Dean，不要这么混”，Dean肯定会置之不理。Ben仔细思考了该如何让Dean明白他的行为对团队产生了不好的影响，最后采用了下面的比喻。

每次作出一个决定，就像一列火车穿过城镇——如果你跳出来拦火车，火车会减速，可能还会惹开火车的工程师生气。每15分钟就有一辆新火车经过，如果每辆火车你都要拦，不仅会花费很多时间，而且最终会有开火车的工程师忍不住朝你碾过去。所以，虽然可以拦火车，但你应该有所选择，确保自己只拦下那些真正关键的火车。

这个故事不仅为谈话带来了一丝幽默，而且帮助了Ben和Dean讨论Dean的“拦火车”行为所耗费的精力，以及对团队产生的影响。

关注幸福度

作为一名领导者，要提高团队的长期效率（并降低人员流失率），评估成员的幸福度是个不错的方法。我们接触过的最优秀的领导者都是业余心理学家，他们不时关注团队成员的心理状态，确保其工作得到认可，尽量保证大家工作得开心。一位主管做了一张表，列出了所有需要完成的吃力不讨好的任务，确保这些任务平均分配给团队的每个成员。另一位主管关注团队的工作时间，安排调休，组织有趣的团队活动，避免大家过于辛苦和劳累。还有一位主管安排与团队成员一对一的谈话，先处理技术问题打破僵局，然后了解每位工程师是否得到了完成工作所需的各项条件。在聊开之后，再接着了解工程师是否喜欢现在的工作，以及下一阶段有何计划。

评估团队幸福度最有用的一个工具是在每次一对一谈话结束时问：“你需要什么？”这个简单的问题是结束谈话的极好方式，可以确保每个团队成员都能保持高效、快乐，但可能还需要仔细询问以获得具体细节。如果每次一对一谈话你都提出这个问题，最后团队成员会将其记在心中，有时甚至会带着改进工作所需的事项清单来找你。

意料之外的问题

Fitz在谷歌开始工作不久后，第一次与时任CEO的Eric Schmidt开会，会议结束时Eric问Fitz：“你有什么需要？”Fitz本来为难题或挑战准备了上百万个回答，却完全没有料到这个问题。因此他呆坐在那里，瞠目结舌。当然，下一次再问这个问题时Fitz就有准备了！

关注团队在办公室以外的幸福度也是有用的。请不要认为人们没有工作之外的生活——对人们花在工作上的时间抱有不切实际的期望会导致人们失去对你的敬意，或者更糟，导致大家过于劳累。我们不是建议你打探团队成员的个人生活，但是留意

团队成员的个人生活情况可以帮助你理解他们为什么在某段时间工作效率提高或降低了。对家中有事的团队成员适当宽容，他会更愿意在以后工期紧迫时加班。

关注团队成员的幸福度，很大程度上是关注他们的职业发展。如果问一位成员五年之后的职业规划，大部分时候得到的是不置可否和茫然。当具体问到这个问题的时候，大多数人都说不出什么，但有几件事是大部分人都希望在今后五年实现的：升职、学习新知识、开始重要工作、与聪明人共事。不管是否明确地说出来，大部分人都会考虑这些事。如果想成为一位有成效的领导者，你应当思考如何帮助团队成员实现这些想法，并且让大家知道你为他们所想。最重要的是将这些模糊的目标变得明确，在为团队成员提出职业发展建议时，给出一组切实的指标，对环境和机会进行评估。

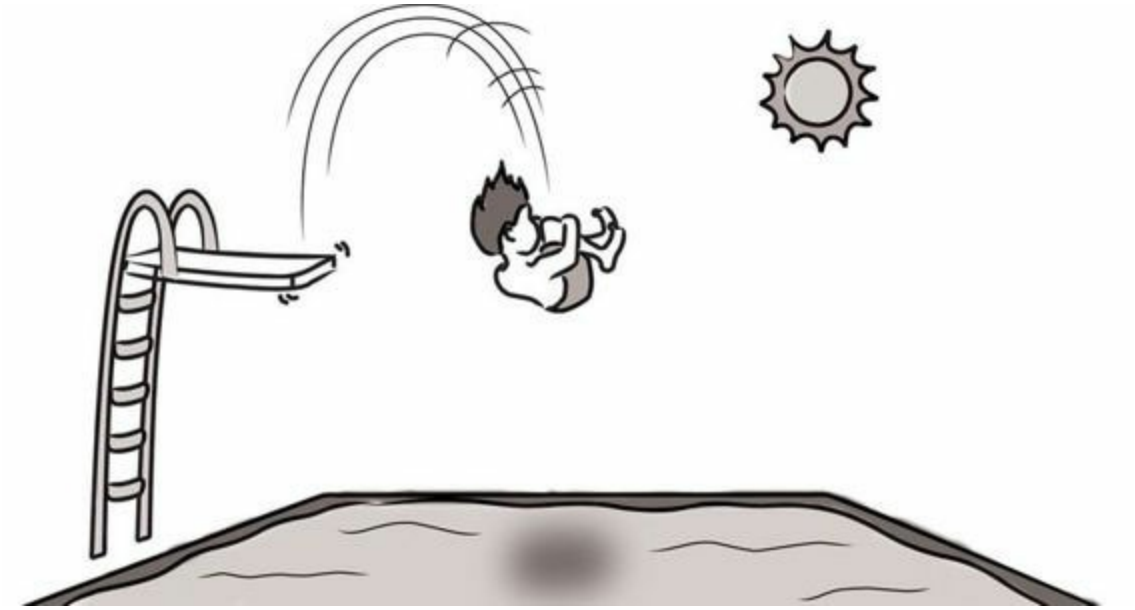
关注团队成员的幸福度，不仅要关注他们的职业发展，还要提供机会，使他们的能力得到提高，工作得到认可，而且在工作中获得乐趣。

其他提示和技巧

放权，但也要做具体工作。在从基层员工转换为领导角色时，最难的是找到平衡点：开始时你想自己完成所有工作，而在长时间担任领导工作后，又容易养成什么具体工作也不做的习惯。如果你刚成为领导不久，可能需要特别注意将工作分配给团队里的其他工程师，即使他们比你花的时间长得多。这不仅是你不至于累崩溃的唯一方法，也是团队其他人学习的机会。如果你已经领导团队一段时间或接手了一个新团队，要获得团队的尊重，尽快进入状态，最简单的方法就是参与实际工作——通常是接受一个没人愿意做的苦差事。即便你有一份好看的简历，上面列出的成就有一英里长，要让团队知道你有多能干多投入（多谦虚），最好的办法还是身体力行，实际完成一些困难的任务。

寻找接替自己的人。除非你想一辈子做同样的工作，不然就得找人接替自己。前面提过，这一过程要从招聘开始：如果你希望团队中的一员可以接替自己，就需要招聘具备这种能力的人，也就是说要“招聘比自己聪明的人”。招到了有能力完成你的工作的团队成员后，还要给他们机会承担更多责任，或偶尔让他们领导团队。这样你可以很快看出谁最有能力而谁的领导意愿最强。请记住，有些人宁愿作一名高绩效的普通员工，这也没关系。有些公司选拔旗下最好的员工——不顾他们的意愿——将他们推上领导岗位，这真令人吃惊。这种做法通常导致团队少了一位伟大的工程师，而公司多了一位不称职的经理。

知道何时出手。你（不可避免而且经常）会遇到一些困难情况，你希望自己可以置之不理。这可能是团队里的工程师技术能力不够，可能是有人凡事都要辩一辩，也可能是有个惫懒工程师一周只工作30个小时。“再等一等，情况会好转的。”你安慰自己，“事情会自己解决的。”请不要掉进这个陷阱——有些事情你必须有所动作，而且必须立即采取行动。这些问题极少会自己消失，事情拖的时间越长，对团队其他人的不利影响越大，你就越担心，甚至夜不能眠。等待只是让必然发生的事情延后发生而已，而且还会导致更多的损害。因此该出手时就要出手，而且行动要快。



给团队一方净土。走上领导岗位后，你发现的一件事通常是，团队之外是一个你之前从未见过的混乱动荡（甚至疯狂）的世界。20世纪90年代，Fitz第一次成为经理时（后来他又重新当了基层员工），公司的不确定因素和组织混乱的程度令他大吃一惊。他问另一位经理是什么导致以往平静的公司陡生波澜，那位经理对Fitz的幼稚感到非常好笑。其实混乱一直存在，只是Fitz以前的经理将混乱屏蔽在团队之外罢了。

保护团队。让团队知道公司在他们“之上”的动向非常重要，但也需要保护他们不受过多不确定因素和可能强加于他们的无谓要求干扰。你可以尽量与团队分享信息，但不要让那些不可能真的影响到团队的事情干扰他们。

肯定团队的成就。很多新主管过于关注团队成员的缺点，而忽视了要经常提供正面反馈。犯了错要指出，做得好也要表扬，团队成员表现优异时要让他（和团队其他人）知道。

最后，对于具有冒险精神、总想尝试新事物的成员，最优秀的团队领导者了解并经常使用的方法是：同意尝试容易恢复的事情。如果有人想花一两天时间，试用一个

可能提高产品性能的新工具或代码库（而且项目工期不紧），你可以很简单地回答：“行，试试吧。”但如果他想做的是一个你今后十年都必须维护的新产品，那么你可能需要多考虑一下。真正的好主管能够准确估计什么事情能够恢复。

冒充者现象

所谓的“冒充者综合征”或“冒充者现象”，据维基百科称是人们无法将成就内在化的一种心理现象，相关资料很多。罹患这种综合征的人，虽然有外部证据证明其能力，但患者内心深信自己并非实至名归。

我们更愿意称之为“现象”，虽然这种心理状态让你觉得自己是冒牌货，随时有被揭发的危险，但冒充者现象常常使你工作更加努力，取得通常情况下难以达到的成就。

在新晋领导岗位，特别是因形势所需被推上（正式或非正式）领导岗位的人中，这一问题极为常见。冒充者现象如此之多，人们总会在我们的演讲之后提出相关问题。“我不知道自己在做什么，”人们说，“该怎么做才能不觉得自己是冒牌货呢？”我们的回答是，每个人在职业生涯的某些时候都会觉得自己不称职，稍许的不自信也许可以使我们工作更加努力，帮助我们获得更大的成功。

Ben很喜欢给我们讲他父母的婚姻故事。在结婚前夜，Ben的父母都打了退堂鼓，向对方承认自己犯了天大的错误——但要取消婚礼为时已晚。于是他们约定“假装”结婚，扮演快乐的新婚夫妇，然后过几天再分手。几周之后，他们决定再试一个月，这样过了一个月，又过了一个月。最后这个约定成了他们婚姻中不时调侃的笑话。每年的结婚纪念日他们都会说：“让我们再试一年吧，如何？”

不管担任何种领导角色，这种“装着装着就成真了”的技巧都能发挥很好的作用。

Ben第一次受命管理一个大型团队时，脑海中也有过类似的想法：“让我管理这个项目？这也太扯了。不过，好吧，我想我可以假装当一阵子领导。”之后每年绩效评估的时候，他都会回头看看自己的成果，然后说：“好吧，我想我还可以再多装一会儿——好像干得还不错呢！”

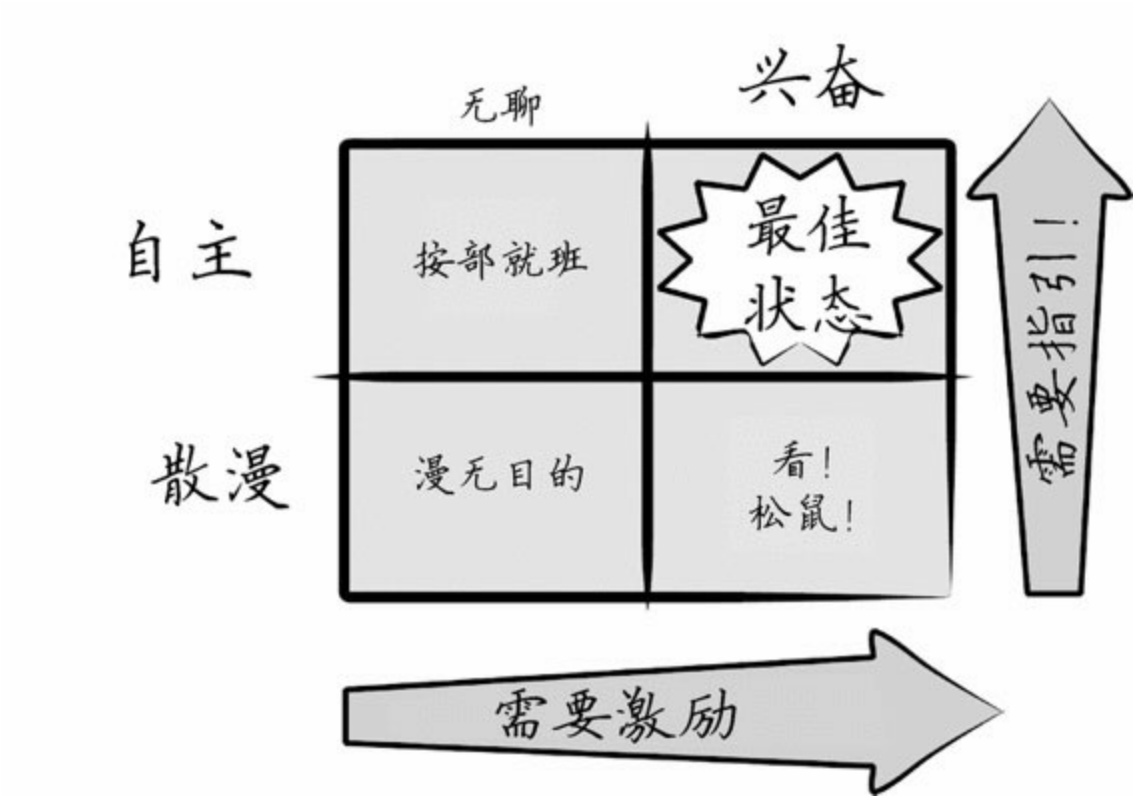
人同植物

Fitz的妻子是家里六个孩子中最小的一个，她的母亲担负着养育六个孩子的艰巨任务，且这六个孩子个性迥异，各有不同的需求。Fitz问岳母是如何管理（请注意此处的用词）孩子的成长的，她回答说，孩子就像植物，有些像仙人掌耐旱喜阳，有些像非洲紫罗兰喜湿忌暴晒，还有一些像西红柿宜施薄肥。如果你有六个孩子，给每个孩子同样多的水、阳光和肥料，虽然他们得到了同样的待遇，但很可能一个也没得到真正需要的。

你的团队成员也和植物一样：有些需要更多阳光，有些更需要灌溉（有些需要更多的牛粪，呃，肥料）。作为团队领导者，你的责任就是发现谁需要什么，然后满足

他们的需求。

请看下面的矩阵：



要让所有团队成员进入最佳状态，就需要激励那些落在矩阵中“按部就班”一格的成员，为“看！松鼠！”那部分成员提供更强的指引。当然，那些“漫无目的”的成员既需要激励也需要指引。因此，你需要为团队成员提供的不是水和阳光，而是激励与指引，让他们保持快乐、高效。也请注意不要矫枉过正——如果他们不需要激励或指引，而你还要施与，那只会惹人厌烦。

提供指引非常简单——需要对待办工作的基本理解，一些简单的组织技巧，以及足够的协调能力将工作细化为可管理的任务。有了这些工具，你就可以为需要方向性帮助的工程师提供足够的指导（好吧，其实并非这么简单，但本章前面已经涵盖了部分相关内容）。而激励则略为复杂，需要多加解释。

内在激励和外在激励

激励有两种：源自外部因素（如金钱）的外在激励和源自内心的内在激励。在Drive一书中，Dan Pink阐述了使人们最幸福最高效的方法不是外部激励（例如，给他们很多钱），而是增强他们的内在激励。Dan宣称有三样东西可以提升人们的内在激励：自主性、卓越性和目标。

如果一个人有能力自己采取行动，无需他人微管理，那他就拥有自主性。对于有自主性的员工，可以给出构建产品所需要的大致方向，让他们自己决定如何具体实现。采用这种方式，员工与产品的关系更加紧密（可能比你更了解如何构建产品），而且对产品具有更强的主人翁意识，从而起到更好的激励作用。产品成功与否对员工的影响越大，他们希望产品成功的意愿就越强。

卓越性最基本的形式是指，你需要给人们提供学习新技能、提高已有技能的机会。给人们足够的机会精进技能，不仅可以激励人们，而且可以使他们日渐优异，团队也随之强大。员工的技能就像刀刃：你可以花数万美元为团队寻找技能最强的人，但如果常年“用”刀而不磨刀，最后利刃会变成低效甚至无用的钝刀。为团队成员提供足够的机会学习新事物并提高技能，可以使他们保持技术先进，工作高效，成果显著。

当然，如果缺乏工作目标，则自主性再强，卓越性再高，也无法激励一个人，因此你需要为其工作提供目标。很多人所做的产品意义深远，但他们却无法体会该产品对公司、客户甚至全世界可能产生的积极影响。即使产品产生的影响可能不那么大，你也可以探索该工作的意义并传达给团队成员，以激励大家。如果你能帮助团队成员看到自己所从事工作的意义，就能极大地提高大家的主动性和效率。我们认

识的一位经理时刻留意公司得到的产品（“影响较小”的一个产品）反馈邮件，如果看到客户谈及该产品对其个人或业务有何帮助，她就立刻将这封信转发给工程团队。这一举动不仅能激励团队，而且常常能启发他们思考如何改进产品。

总结

无论你是否有领导团队的意愿，我们都希望本章可以帮助你理解如何成为一位优秀的团队领导者，并驱除一些关于如何领导团队的误解。即使你下定决心不当领导者，熟悉本章列出的一些概念也不无裨益，不管你自己的团队主管是否称职，这些内容可以帮助你理解他的所作所为。花点时间审视一下所在的团队，看看团队主管用了哪些模式和反模式导致团队的成功（或失败），你会对团队运作的关键获得更实际的理解。

理解每日所处的团队及其领导者只是与他人合作的一个方面——与团队以外的人打交道会更困难，特别是那些专门来搞破坏的人。我们称这种人为“有害之人”，下一章将对此进行详细讨论。

第4章 应对有害之人

正如本书开头引用的那句话，创新工作中最困难的部分是人。

本书从开篇到现在采取的都是自省方式，先审视了自己的个人行为，讨论如何将HRT

原则应用其中，然后探索了如何围绕这些原则打造沟通良好的团队文化，第3章解释了如何在需要时将自己塑造为这种团队的高效领导者。

本书的第二部分会将视线投向外部。如何与核心团队之外的人打交道？几乎总会有人想加入你的团队或进行合作，你需要处理组织结构中的冲突，而且，当然还需要与最重要的外部人员——软件的用户——打交道！

本章将讨论防止外来者破坏（精心打造的）团队合作文化的重要性。也许更为重要的是，如何应对已经存在团队之中的有害之人。

定义“有害”

我们已经回顾过构建稳固且沟通良好的团队文化的重要性。我们用了大量篇幅介绍良好的团队文化应该包括什么：基于共识的开发、高质量代码、代码审阅，以及鼓励大家进行新尝试并从快速失败中学习的环境等。

讨论团队文化不应包括什么同样重要。如果试图打造一支高效、敏捷的团队，关注避免什么也很重要。优秀的工程师能使项目进展更快，团队工作更加高效，但某些不好的行为会使项目进度变慢，效率降低，而且使公司工作环境变差——最终破坏团队的凝聚力。

最初就软件开发的社交挑战发表演讲时，我们做了一个名为“如何与坏人打交道”的演示。会议的一位主席建议我们将演示更名为“如何从有害之人手中拯救项目”，希望这个夸张的标题能吸引更多眼球。他是对的：我们在不同的会议上一次又一次地进行这个演示，每次会场都挤得水泄不通。不仅仅是因为“有害”这样的贬义词吸引听众，而且是因为每个人似乎都有过和讨厌的人打交道的个人经历。这

些演讲最终都变成了一种集体治疗，听众互相交流经验，寻求建议。

但这么做也存在着某种危险。通常，将时间花费在负面情绪中是不健康的——这会耗尽你的精力，长此以往会产生更多的冲突。“有害之人”这个词是个不好的标签，自动在“我们”（好人）和“他们”（那些坏人）之间划了一条分界线。我们可以用更好的方法来思考这个问题。不要将团队当作一个力图“驱除恶人”的精英兄弟会，而是要创造一种拒绝容忍特定负面行为的团队文化，这样比较有助于大家的心理健康。需要去除的是行为，而非特定的人。将人视为纯恶或纯善是幼稚的行为，发现和谴责不可容忍的行为则更为有效和实际。

为简单起见，我们将暂时继续用“有害之人”这个词指代行为不佳的人。但在实际工作中，最好避免在日常谈话中用这个词！

巩固团队

回想一下那个酵母比喻：团队文化是如何从一个重要的初始文化逐渐成长的。对团队长期文化影响最大的是最初的那支团队，如果初始团队没有建立足够强大的文化，就会受到其他文化的压制。如果初始团队建立了可接受和不可接受行为的严格标准，那么这些行为预期就会一直沿用下去。

我们俩参与过很多开源项目，我们的个人经历与上述想法非常契合。

我们参与最多的项目——Subversion——由几个人发起。这些人非常谦虚，很自然地彼此信赖且互相尊重。过了15年，这个项目的参与者至少换了三四拨（最初的发起人大部分都离开了），但团队文化依然未变——每个人都友善、礼貌，并对他人的行为也抱有同样的期望。这一文化得到保存，不仅因为其标准高，而且因为它

具有自我选择性。友善的人会受到友善社区的吸引。

自我选择也很容易向相反的方向起作用。如果一群暴躁的混蛋组建了一支团队，结果会吸引越来越多同样的人。Linux内核社区就是个很好的例子，那里面各种斗争无休无止，我都懒得再提。是的，这样的团队可能完成了很多工作，但其运作的整体效率令人生疑。如果如此之多的能量没有浪费在个人攻击上，可以多完成多少工作啊！因为有礼貌的人对这样的团队敬而远之，所以项目又失去了多少潜在的参与者！

我们老调重弹，是为了强调其中的利害：有害之人对高性能团队具有直接的威胁。如果允许不良行为存在，不仅工作效率会降低，而且团队文化也会逐渐恶化。最好的防御手段是制定一组有效的最佳实践和流程，巩固团队文化。第2章已经讨论过这部分内容，在此做一个快速回顾。

□ 发布任务说明书，明确目标和非目标。

□ 为邮件讨论建立适当的礼节规范。保存归档，请新成员阅读归档邮件，防止少数人干扰讨论。

□ 记录所有历史：不仅是代码历史，还包括设计决策、重要缺陷修复，以及以往的错误历史。

□ 有效合作。使用版本控制软件，保持代码少改动，可审阅，增大“巴士因子”，防止知识集中在少数人手中。

□ 明确缺陷修复、测试、软件发布的政策和流程。

□ 降低新成员的磨合障碍。

□ 依赖基于共识的决策方法，但也要制定备用流程，以解决无法达成共识的冲突。

团队越深入执行这些最佳实践，社区对有害行为的容忍度越小。当捣乱份子出现时，就可以从容应对。

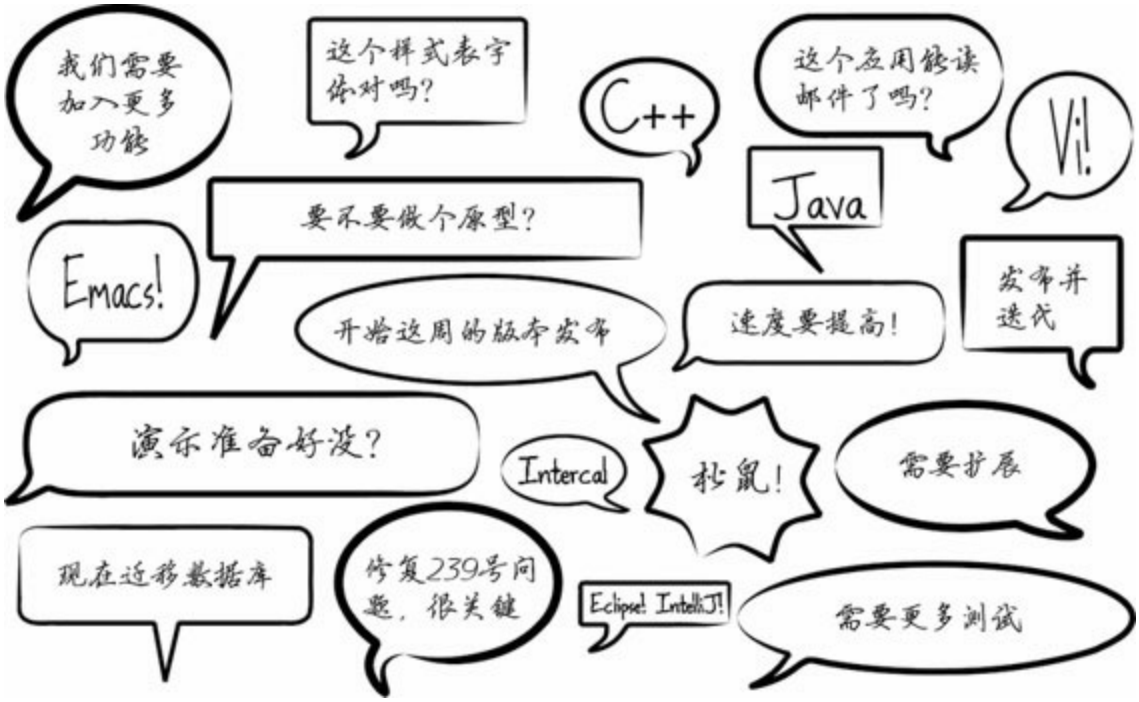
发现威胁

保护团队不受有害之人的侵扰，要做的第一件事是了解构成威胁的因素是什么，什么时候应提高警惕。

最可能受到影响的是团队的注意力和专注力。

注意力和专注力是最为稀缺的资源。团队规模越大，团队拥有的可集中于构建产品和解决有趣问题的能力就越强——但这个能力总是有限的。如果不主动保护这些资源，有害之人可以很容易地干扰团队的运作。团队成员最后可能会争吵，无法集中

精力，而且觉得筋疲力尽。每个人都将所有的注意力和专注力浪费在了创造优秀产品以外的无谓之事上。



同时，人们可能会想：有害之人长什么样呢？为了采取防卫措施，你需要知道注意什么问题。

按照我们的经验，很少有人是有意搞破坏（即故意挑事）。我们管这种蓄意捣乱的人叫“troll”，通常不予理会。但大多数行为不良的人要么是自己没意识到，要么就是根本不在意。很多时候，问题的根源是无知或冷漠，而非恶意。大部分的不良行为都可以简单地归结为HRT的缺乏。

对于有害之人，有一些典型的迹象和模式可循。一旦发现这些模式，我们就会对此人“做个标记”——也就是说，暗中记下此人总是有不良行为，与他打交道时应格外小心。

不尊重他人的时间

有些人总是搞不清楚项目进行的状况。他们造成的危害通常是浪费别人的时间。他们不肯花几分钟读一下基础的项目文档、任务说明书、FAQ，或最新的邮件讨论内容，而是不断地提出自己本可以轻松找到答案的问题，打扰整个团队。

在Subversion项目组中，我们遇到过一位成员将开发人员的主讨论论坛当成自己每日感悟的留言板。Charlie没有实际提交过任何代码，而是每隔两三个小时，就发表自己最新的想法和点子。不可避免地，会有几个人回复他，解释为什么这个想法不正确，不可能，已经在实现中，以前讨论过，而且/或者已经有文档记载了。更糟的是，Charlie甚至开始回复来访社区用户提出的问题，而且答案还是错误的。项目核心成员不得不一次又一次更正他的答案。相当长一段时间之后我们才意识到这位可亲、热情的参与者实际上对项目有害，耗费了项目组的精力。这一章后面将谈到我们是如何处理这个问题的。

自大

自大这个词用在这里可能并不妥当，我们想描述的是那些无法接受集体决定，不能听取、尊重相左意见，或不愿作出让步的人。这种人常常会因为自己当时没有参与决策，就重新开启早已得出结论（而且在邮件归档中有记录）的讨论。这种人根本不阅读或思考历史成因，却要求团队为他重演整个辩论。他们经常会对项目的成功作出断言，声称如果不按他说的做项目就会立刻失败。

Subversion项目发生过一件很有名的事。有一天项目邮件列表里出现了一位睿智的程序员，声称整个产品的设计有问题。他已经考虑清楚，知道应该怎么改，要求整个项目推翻重新开始。他甚至主动请缨领导这一过程。他说，如果没有他的领导，整个项目很快会遭遇彻底的失败。

项目创始人们浪费了整整一周的时间，与此人无休止地争论，为他们最初的设计决策进行辩护。大量的注意力和专注力就此流失。最后大家明白了，这个人不愿意让步，也不愿意把他的任何想法集成进当时的产品中，这个产品（已经是beta版本，进入了广泛试用阶段）也不可能重新开始。到了某个阶段，我们只能退出辩论，回去继续工作。具有讽刺意味的是，数年之后，事实证明这个人的预测在很多层面上都是正确的，但这也没妨碍Subversion取得极大的成功——至少在企业软件开发领域如此。事情的关键并不是谁对谁错，而是不同意见是否必须得出结论，是否值得花时间继续讨论。请时刻提醒自己这类问题。在某些时候，你需要作出决定，及时止损，继续前行。

颐指气使

如果碰到一位来访者要求你完成某事，那么就应该提高警惕了。如果一个人整天抱怨一个软件的不足之处，又不愿意给出任何直接帮助，这个人就不太对劲。

这种理所当然的感觉有时会演变成无理行为。运营谷歌的Project Hosting服务时，我们遇到过一位项目所有人请求封杀一位用户。这个开源项目是一个视频游戏模拟器，但该用户喜爱的视频游戏在此模拟器上运行效果不佳。这个用户在问题跟踪系统里报告了谷歌的一个缺陷，言词相当粗鲁。项目开发人员有礼貌地解释了为什么这款游戏尚不受支持，以及短期内无法修复该缺陷的原因。这个用户不愿接受这个回答，开始每天骚扰开发人员。他就同一问题又报告了一次缺陷，还在别的缺陷报告中添加评语，说开发人员不修复他的问题，是一群白痴。尽管开发人员和谷歌的管理员不断提出警告，这位用户的用词仍旧变得越来越粗俗。虽然我们尽力消除他的破坏行为，他还是拒绝悔改，我们最后不得不——实在是没有别的办法——彻底封杀了这位用户。

沟通幼稚或混乱

这种人发言不用真名，而是使用幼稚的昵称，如SuperCamel、jubjub89或SirHacksalot。更糟的是，这些人在不同的地方经常用不同的昵称——邮件用一个名字，即时消息用另一个名字，代码提交可能再用一个名字。有的时候，这些人还会在发言中使用网络缩写词，用数字替代字母，全篇大写，或者使用过量的标点符号。

疑神疑鬼

前面所举的例子中提到，有时候某种不合时宜的颐指气使会直接导致对项目的公开恶意。很多次，我们看到这种情况升级为完全的偏执。如果现有团队与来访者意见不合，这位有害之人有时就会开始抛出阴谋论。项目团队会认为一位来访者如此重要，以至于要费尽心思地对付此人，想想也是挺有趣的。如果团队已经具有公开、透明的沟通文化（正如第2章倡导的），这种指控就显得更可笑了，因为所有的对话都有公开记录。推荐的做法是根本无需对这类指控作出回应。这位有害之人已经偏执到这个地步，你说的任何话都只会令他更加疑神疑鬼，又何必再多费唇舌呢？还是回到重要的创新工作中去吧。

完美主义

从表面上看，完美主义似乎全然无害。确实，完美主义的信奉者偶尔会有些古怪的强迫性举动，但通常情况下都表现得谦虚、有礼、恭谨，善于聆听。他似乎完全遵循了HRT原则，充满了善意。那问题出在哪儿呢？问题出在完美主义者有可能使团队丧失行动能力。

让我们来看一个前同事的例子。Patrick是一位卓越的工程师，他设计能力很强，

写出的代码和测试质量很高，且为人非常随和。不幸的是，设计新软件时，他可能会花费下半辈子的时间来调整和改进设计。他无法对计划感到满意，似乎永远不能开始编码。虽然他确实很有道理，对大家试图解决的问题也有优异的见解，但最后团队其他人总是会变得无比沮丧。这样下去，我们永远也不能真正开始写代码。项目组的几个人琢磨了好一阵子该怎么办。一方面，Patrick对团队贡献很大；另一方面，他妨碍了团体的进度。每次我们开始编码时，他都会很有礼貌地表示反对，指出可能在遥远的将来造成影响的潜在理论性问题。他使我们丧失了行动能力而不自知。我们将在下一节介绍这个问题的解决办法。

去除毒素

我们不建议仅仅因为某些人不擅社交或行为粗鲁就将其踢出社区。前面提到，创建一种“我们”（友善的人）对抗“他们”（刻薄的人）的派系之争是不利于社区健康的。在前面的例子中，我们没有把注意力放在如何驱逐人，而是关注如何消除行为。你需要清楚地说明，恶劣行为将不会得到容忍。如果受到多次警告后行为依旧没有改善，此时才应考虑正式的制裁。

很多时候，集中精力杜绝有害行为，可将有才华（虽然可能不善社交）的人变成团队中的高效成员。几年前，我们团队中一位优秀的工程师有个坏毛病——有时会冒犯其他成员。我们没有将他逐出社区，而是请人单独问他是否知道自己的言辞不妥。他似乎对此略感惊讶，并不完全明白为何自己的行为会产生这样的影响。但他同意尽量注意自己的行为，以成为更好的团队成员。事情得到了完美的解决，他改变了自己的行为，问题迎刃而解。并不是每个问题的解决方法都是赶人走！

现在你应该学会了如何识别有害之人。也许当前正有人在扰乱团队的注意力，消耗

团队的精力，应该如何有效处理这种情况呢？下面是一些行之有效的策略。

引导完美主义者的能量

原有的问题一旦得到可行的解决方案，就应当将完美主义者的注意力引导到另一个需要解决的问题。

这一策略对Subversion项目中的完美主义者非常有效。到某个时候，我们会将Patrick拉到一边说：“好了，我们现在要按现有的设计开始工作了，看看效果如何。但愿你能帮我们解决随后出现的任何问题。”没想到，Patrick对此并无异议，转而投入到另一个问题的研究中。大家的情绪都得到了照顾，Patrick也不断对整个项目作出贡献。

有句老话说不要让“完美成为良好的敌人”，在努力打造高效团队时，既要警惕较为明显的干扰行为，也要注意避免完美主义。

这种转移注意力的方法也可用于那些抱怨批评多于实干的人。对于这种人，你可能很想回复一句标准的“欢迎提交补丁”——开源社区里让某人要么干活要么闭嘴的委婉说法。但最好还是引导他正式测试软件找到问题，并对此产生兴趣，这样他既可以接着抱怨，又可以对项目提供帮助。

不要喂食能量生物

不要喂食能量生物，这句格言源自Usenet。这一策略对故意捣乱的人（即那些有意激怒你或团队的人）特别有效。你回应得越多，对方消耗你的能量就越多，你浪费的时间也就越多。最好的简单方法通常是保持沉默。不管你有多想用一句一针见血的回应置对方于死地，请尽量克制。当对方意识到没人注意他时，通常就会失去兴

趣，一走了之。请注意，对挑衅不予理会通常需要极大的意志力，坚持住！

不要过于情绪化

即使人们不是故意捣乱，也很容易让人生气。如果有人指责你的设计决定不好或指责你搞阴谋，又或是问很多答案显而易见的问题浪费你的时间，你很容易觉得不高兴。请记住，你的工作是创造伟大的事物，而不是安抚每位来访者或不断证明自己的存在合理。你的情绪越强烈，就越可能浪费几个小时或几天时间书写充满感情的回复，而该人其实并不值得你付出如此多的关注。请仔细选择哪些挑战值得回应，并时刻保持冷静；谨慎判定何人值得回复，何人应当无视。

在愤怒中寻找事实

继续前面不要过于情绪化的讨论，延伸出的必然结论就是积极寻找事实。如果有人抱怨，那就仔细聆听。不管人们使用的言辞是否愤怒或粗鲁，先假设他有其道理。此人是否言之有理？他的经验是否有可借鉴之处，或有值得回应的想法？很多时候答案是肯定的——虽然此人言辞尖刻，但其中确有智慧闪光。请将争论带回到技术

讨论中。

关于这一策略，我们最喜欢举的例子是，某天，我们收到来自开源社区某位知名领导者的邮件，这封邮件充满敌意。虽然这封信算是某种缺陷报告，但看起来更像是对团队集体智慧的指责。信中充满了不实指控和夸大其词，似乎其目的是为了激怒团队，而非解决问题。但是，我们的一位团队成员只用了几个关注该缺陷的具体问题回复了这个报告。这位缺陷报告者在回复中进行了更多说明，但仍是满篇恶语。该队员仍然完全忽略这些辱骂，对问题进行了调查，并简单回复道：“感谢你的缺陷报告，我知道该如何修复这一问题了——很快就会发布一个补丁。”

我们对该队员处理这一问题的方式感到无比自豪。只要保持完全冷静，着眼事实，随着对话的发展，最初那位发信人只会显得更像一个疯子。到对话最后，这位缺陷报告者在听众面前完全失去了可信度，就不会再有兴趣继续呆下去了。

以德报怨

将前面的方法（保持头脑冷静并关注事实）更进一步，有时表现得格外友善就可以把人吓走！下面是来自Subversion IRC频道的一段真实聊天记录。

Harry：Subversion糟透了，一点儿都不好用。

Sussman：如果你需要帮助，愿闻其详。

Harry：我想cvs某人的文件。不，我就是想发发牢骚。这个家伙非要用什么Subversion，只有svn，没有cvs。

Sussman：那就下载一个svn客户端，签出他的源代码。

Harry：我去下了这个什么Subversion……你能配置make，让make像安装cvs一样安装Subersion吗？当然不行。我不怪Subversion，都怪他。

Sussman：你不能运行./configure; make; make install，并不意味着Subversion有普遍存在的严重bug。人们每天都这么安装svn tarball。

Harry：我没说这是bug。

Sussman：如果有这么基本的问题，你觉得我们会发布tarball吗？

Harry：我只是抱怨一下这个混蛋。我还得装expat或者libxml。唉。

Sussman：大多数系统通常都预装了这些软件。

Sussman：他用的是Apache服务器吗？也许你应该抓个二进制文件。

Harry：我不知道，他只说用了svn……

Sussman：你用的什么版本？

Harry：FreeBSD。

Sussman：那直接cd到ports树，make这个port。

Harry：你们还让不让我发牢骚了啊……我是来找茬吵架的……你这也太友善、太乐于助人了吧。

Sussman：:-)

Harry：IRC频道里的人啥时候都成了雷锋了？郁闷。

Harry退出了。

适时放手

有时候，不管你如何努力，最后还是需要选择放下，继续前行。即使已经花费了大量的注意力和精力试图纠正不良行为，你仍需要知道如何识别败局。

让我们回到Charlie的例子，这位友善的哲学家在Subversion邮件列表中过于频繁地发信。当我们最后对邮件讨论进行分析时，发现这位参与者在两个月内的发帖数排名在第三位。排名第一和第二的发帖人是项目的核心成员，而他们70%的帖子都是回复Charlie的！很明显，虽然Charlie本身并无恶意，但他浪费了我们的精力和注意力。我们的最终解决办法是给Charlie发了一封私信，（很客气地）请他不要这么频繁地发帖。这场对话进行得很艰难，主要原因是他无法看到自己导致的干扰有多大。数周后，没有看到明显的行为变化，我们中的一员和他通了电话，进行了一场漫长（甚至更为困难）的讨论，请求他停止发帖。他最终尊重了团队的意愿，略带伤感和困惑，按我们的要求停止了发帖。每个人对此都有些负罪感，因为Charlie不曾确切理解自己造成的伤害，但每个人也都觉得这是正确的决定。这一情况解决起来颇为微妙，但我们尽力遵循HRT原则，保持礼貌、客气。

放眼未来

通向项目成功的路上有千百个障碍。如果对有害之人干扰的处理存在一个共同主题，那就是很容易陷入当前局势的冲突。如果见到你觉得可能有害的行为，你需要向自己提出两个关键问题。

□ 抛开团队注意力和精力的短期损失，就长远来说，你确实相信项目将受益吗？

□ 你相信这个冲突最终会以一种有益的方式解决吗？

如果以上两个问题中的任何一个答案为否，那么你需要出手干预，尽快停止这种行为。我们很容易说服自己，认为容忍有害行为的短期收益是值得的，但经常并非如此。例如，某人可能是极好的技术人员，但也有不良的行为。你可能很想对这种行为睁一只眼闭一只眼，以便充分利用他的良好技术能力。但要当心！基于HRT的强大团队文化是不可替代的，但技术能力绝对是可以替代的。引用我们前队友的一段话。

我有几位对他有所了解的朋友。其中一位说：“他常常游走在天才和疯子的边界上。”问题是，如今天才是件商品，举止怪异已经行不通了。

——Greg Hudson

当然，Greg这里所说的不是字面意义上的“天才”。他指出，这个世界充满了能力极强的程序员。如果你找到的天才行为不当，或者长此以往会影响团队文化，那么最好还是等待下一个天才出现。

我们曾在Subversion项目中遇到过这种情况。团队有一条严格的规定，不可将名字写在源代码文件中（就是第2章中讨论过的！）：我们觉得这种行为会产生无法管理的领域性。如果一段代码带有其他人的署名，大家会不愿意进行修改，而且还会降低巴士因子。我们的做法是，用版本管理的历史适当记录人们的工作，并用一个顶层文件记录所有编程人员的名字。

有一天，项目来了一位聪明的程序员，主动承担了一个迫切需要而且规模颇大的新功能的开发。他将代码提交审阅后，我们的主要反馈意见只是要求他把自己的名字从文件头部去掉——我们会和对待其他人一样记录他的贡献。但他拒绝了这一要求，讨论陷入了僵局。最后，我们决定拒不接受他的代码，他就此离开，把自己的代码也一起带走了。当然，大家都很失望，但我们不想仅仅为了尽快得到这个新功能就违背自己的策略（并淡化团队的文化）。几个月后，其他人重新实现了这个功能。

这里要明确说明：为了短期效益而损害团队文化是不值得的——特别是涉及无视HRT重要性的优异贡献者时。

总结

这一章讨论了不少场景，在了解了这些内容后，人们很容易变得非常多疑。请记住，世界上大多数地方并没有那么多混蛋。我们的一位朋友曾评价道：“是的，其实外面只有少数几个疯子，但网络使你觉得他们好像都住在你隔壁。”

或者，如Robert J. Hanlon所说：

不要将愚蠢归结为恶意。

我们更愿意使用无知而非愚蠢一词，但意思其实是一样的。这一章开头提到，将人们分为好或坏是幼稚的。只有极少数人会故意破坏团队文化——大多数人只是产生了误会或受到了误导。又或者他们只是希望得到认可，却不善社交，因而无法融入。不管是哪种情况，你的任务不是培养优越感，将较为朦昧的“农民”锁在项目之外，而是做到对破坏性行为零容忍，明确阐明HRT预期。要理解其中的差别需要智慧，将其实施则需要真正的技巧。

第5章 组织操控的艺术

读到这里，你已经了解了如何管理你和团队中关于人的方面。我们回顾了领导团队所需的基本人际能力和处理有害之人侵扰的危险。除此之外，你还需要理解如何在良好或不佳的公司中工作。大多数人所在的公司都存在各种问题，需要应用某些操控技能才能有效地完成工作。有些人将此称为政治，也有人称之为社会工程。

我们将其称为组织操控。

好公司、坏公司，以及策略

大型公司有如迷宫般复杂，至少需要一个GPS，一只手电筒，以及一卡车的面包屑才能从公司的一端走到另一端。

首先，我们将介绍在理想的公司中团队通常是如何运作的，然后讨论有问题的公司

是如何阻碍团队成功的。我们还将介绍在这两种环境中完成工作的策略，最后谈到在其他各种方法都失败时的备用计划。

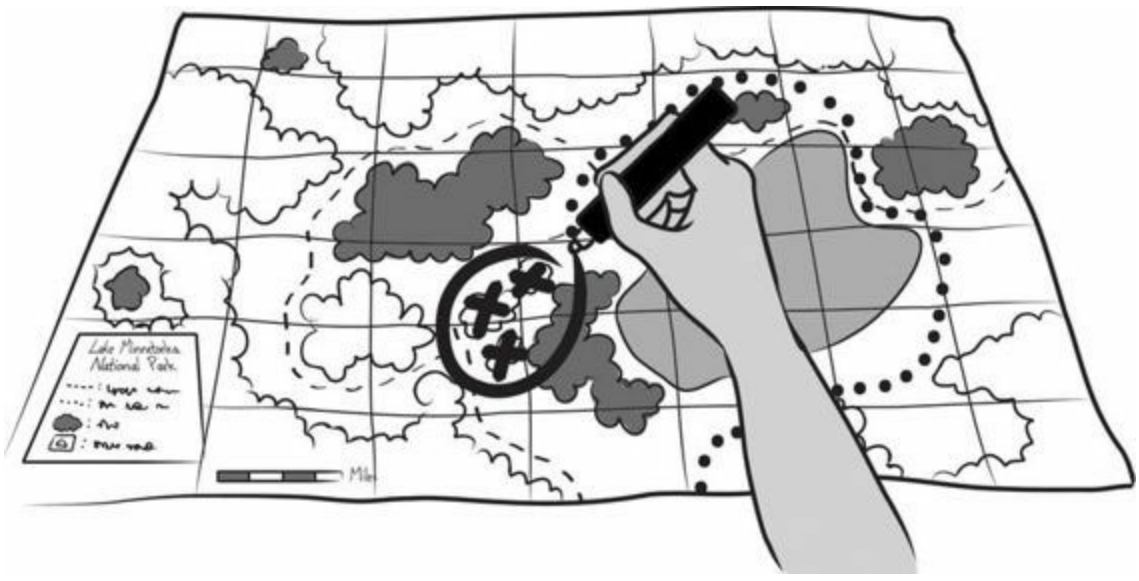
理想情况

正常运作的公司可以分为两层：经理（你大多数时间与之打交道），以及经理之外的部分，包括知识型员工、中层经理、高管、销售人员、律师，等等。

理想的工作体验

如果你有一位服务型的经理，遵循HRT原则，真心实意想帮助你成功（参见第3章），那么你可以做到几件简单的事情，使他的工作更容易，从而也使自己工作更高效，在团队中更有价值。也许更重要的是，做到这些可以使你的工作成绩更优异，职业更成功。

完成本职工作后寻求更多职责。打个比方，你是一位初级护林员，护林员让你去森林中砍掉一棵生病或受损的树。如果只关注手头的工作，你的做法是进入森林，砍掉那棵病树，然后收工。但如果拓展一下思路，你可以进入森林，砍掉病树，然后带回一张标柱了途中所见其他病树的地图，以及一份高效清理这些病树的计划，供资深护林员参考。如果你这样做，下一次有需要这种职责的任务时，护林员很可能会优先考虑你。这不仅仅是因为他知道你有这个能力，而且因为这是阻力最小的路径——可以减少他的工作。



这种主动寻求更多职责的做法可以使你的经理少关注一件事情，减少他的工作量，还可以展示出你有能力完成当前职责之外的工作。但这也意味着你可能不得不离开自己的舒适区，尝试新事物。如果你所在的团队鼓励大家承担风险，快速失败，这么做也没问题。

承担风险而且不惧失败。我们在第3章和第4章详细讨论了承担风险和快速失败的重要性。如果你有一位开明的经理，那么失败是快速学习的一种渠道，你可以通过失败发现自己的能力和不足，并逐渐拓展自己的极限。我们的朋友Steve Hayman常常出差，他经常说：“如果一年一次航班都没误过，就说明你到机场太早了。”这一言论也可用于产品创造：如果一年一次失败都没有发生过，就说明你没有承担足够的风险。和寻求更多职责一样，承担风险也是展示你有更强能力的一种方法。

如果在工作中不承担风险，你会少经受一些失败，但也会少获得一些大的成功。一位优秀的经理需要团队愿意挑战极限，发现他们的能力和局限（并在此过程中得到学习），他也会在你失败时提供缓冲。如果失败了，要勇于承担，不要试图推卸责任，请记录下发生了什么，以及该采取何种措施来避免同一失败再次发生。不断重复这一过程。

表现得像成年人一样。还有一个看似非常明显的建议：别人采取什么行动，如何对待你，其实取决于你自己。不称职的经理经常会压制创新，推卸职责，或禁止任何违规行为，将团队培训得如孩子一般。如果你在这样的经理手下呆过，常常会认为所有的经理都是这样的。

对不确定的东西提出疑问。如果你不同意经理的决定，不要害怕与他争论或对他的决策依据提出疑问。如果你掌握着经理不具备的信息或经验，却只是唯唯诺诺，曲意逢迎，虽然不一定会造成障碍，但也不会对他有任何帮助。

经理不是透视眼：公司中极少有人沟通过多，请及时向团队领导者汇报工作进度，不要等着他来问你。当你遇到问题、取得成功、有所需要，或有所预期时，都可以向经理汇报。这可以确保经理了解你的工作动态，而且有些聪明的工程师甚至将此技巧发挥到极致，用于对付微管理：如果经理总是来检查工作，你可以按他检查工作的频率主动给他发邮件，这样他肯定就不再来了。

在理想的公司环境里这些技巧行之有效，但如果你所在的公司环境并不理想呢？

通常情况

幸福的家庭总是相似的，不幸的家庭却各有各的不幸。

——列夫·托尔斯泰《安娜·卡列尼娜》

你所在的公司环境可能有数不清的因素会阻碍你的成功，但通常可分为两大类：不好的人和不好的组织。

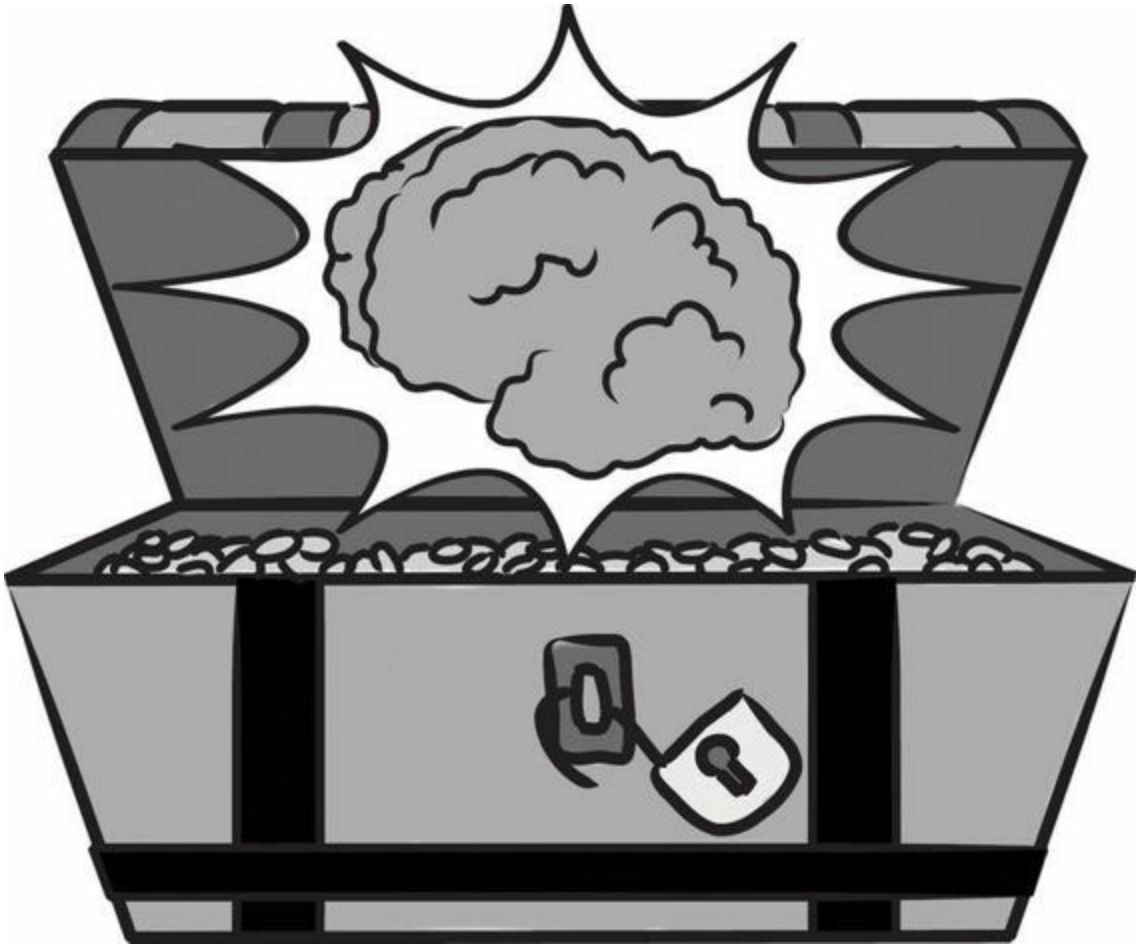
不称职的经理

不称职经理的恶行真是罄竹难书——有专门的电影和电视剧是以讽刺世上不称职经理为主题的。我们中的大多数人在工作中都至少遇到过一位不称职的经理。即便对最优秀的团队成员而言，一位不称职的经理也会使人“生不如死”，因此我们只介绍影响工程师的不称职经理的几个典型特征。

害怕失败是不称职的经理最常见的一个特征。不安全感使得他们极度保守，而这正好与典型工程师的工作风格相反。如果经理不希望你冒险，那你就很难有机会将自己的想法付诸产品，最后工作通常会变成机械地实现其他人设计的产品。

很多时候，缺乏安全感的经理会坚持插手你与团队以外人员的交互，从而阻止你在没有“经过指挥链”的情况下与其他团队直接交谈。这种经理会将你与团队外人员——特别是其他经理——的任何直接联系都视为类似策反或不服从管理的行为。如果你需要团队或组织之外的任何东西，这位经理都希望能经他之手，这样他可以提高自己的重要性并使你服从管理，从而获得更多权力。

大多数人在学校都经常听到这样一句话——知识就是力量。不称职的经理对此深有体会，但理解角度比较独特：不管这种力量对你的工作会产生多大的帮助，他只希望将这种力量掌握在自己手中，而不是与你分享。这种经理藏匿信息，不让你接触信息，以此确保自己可以参与和这些信息相关的所有活动，这样不仅妨碍你完成工作，还可以帮助他维持自己的重要性和权力，至于这样做会怎样延迟开发进度，他并不关心。



藏匿信息使经理自己成为信息传达和沟通的渠道，不称职的经理还会在你成功时夺取功劳，在你失败时撇清干系（有时还将他们的失败归罪在你头上）。很多时候，这种不称职的经理只是单纯将你的存在视为自己向上爬的手段，他并不关心你的职业发展，更不在乎团队的幸福度。

我们的朋友Susan在一位很糟糕的经理手下工作过几年，她的经理经常把新项目交给Susan，但不交待项目背景，不提供如何完成项目的信息，也没有可以答疑的联系人。即使Susan对新任务的背景和技术一无所知，他也还是这种做法，因为这样Susan就不得不依靠他获得信息，与其他团队进行交流。Susan的经理并不见得希望Susan失败：实际上，如果他将Susan所需的与项目相关的所有信息都告诉她，Susan的生活会变得更加轻松，工作效率也会更高。另一方面，他极有可能担心的是，这样做后Susan能在工作中轻易绕过他！如果Susan能够与相关团队直接联系，他们就会知道做这个项目的人是Susan而不是她的经理。Susan一次又一次费力地熟悉新项目，将其完成，累得筋疲力尽，结果通过小道消息发现她的经理领取了本属于她的功劳。

与第3章中讨论的服务型领导截然相反，不称职的经理想知道的是你最近为他做了什么。那团队中的那些效率低下者呢？只要他们不使团队工作停摆就可以安稳地呆着——不称职的经理才不会费力气对付他们。问题归结为：是经理为你服务，还是你为经理服务？答案应该是前者。

办公室政治家

虽然我们支持大家信任他人，或者至少假定他人值得信任，但信任办公室政治家会严重限制你的职业发展。

你可能很难在第一次见面就识别出办公室政治家，因为他们通常非常善于与人相处，经营关系——初识可能显得相当友好。他通常特别注意维系与领导的良好关系，更善于利用同事和下属作为自己升迁的手段。他会毫不犹豫地指责别人，若有机会则会更为麻利地窃取功劳。他通常不会直接与人发生冲突，而是说些你爱听的话以博取你的好感。如果你对他没有用处或无法操控，他要么忽视你，要么将你视为威胁而试图削弱你。一起工作一段时间后，你就可以很容易地识别出办公室政治家：他整天作出有影响的样子，而不是实际产生影响。

我们建议你避开办公室政治家：如有可能，尽量绕行，但不要对他的上级说他的坏话，因为你很难知道谁已经受到了他的蒙蔽而谁是明眼人。如果你是那种喜欢埋头工作、专注于创建有趣技术的人，当周围有办公室政治家存在时，你也许想重新思考这一策略。如果你不愿意“同流合污”，不追求升迁，可能会发现办公室政治家比你先升职，然后你就同时有了一个不称职的经理和一个办公室政治家。更多相关讨论请参见“操控组织”一节。

管理不当的组织

随着发展，公司会设置机构和流程，以管理利润、降低风险、提高可预测性，并适应组织自身的庞大规模。久而久之，这些机构会发展到阻碍公司成功的地步。和不称职的经理一样，人们已经对管理不当的组织做了大量描述和批评，因此这里只回顾几个最常影响基层员工的组织问题案例。

大部分公司不是以工程师为主导的，这是一个简单事实。也就是说，工程师是完成商业目标的手段，但这些目标通常是非技术的。这意味着管理公司的人很可能不理解系统底层的技术细节，只知道业务需求，因此会对工程师提出不切实际的要求。这种公司即使来了一位有技术能力的高管努力维护组织，经常也会被替换为愿意牺

牲员工健康和理智以满足商业需求的人。这一问题的直接表现通常是不切实际的工期，缺少称职的技术人员按时完成项目。你可能难以获得有效运行项目所需要的足够硬件，或者发现团队花费数周重写软件，解决一个购买几百美元硬件就能解决的问题。很不幸，在不重视工程师价值，将他们视为“工作单元”或“资源”，不让他们对公司运营发声的公司里，这样的问题非常典型。

在特别糟糕的公司里，指挥和控制结构僵化有如割据。多年前，我们的朋友Terrence工作的公司对团队之间bug的传递有严格的规定，最终另一支团队产生的一个bug导致Terrence的产品在几个小时内就耗尽了内存。他没有发邮件给负责此事的团队成员，也没有查看源代码或提交日志，而是花了整个晚上重现这一bug，收集数据，说明情况。Terrence将邮件发给他的经理，经理把邮件发给总监，总监再把邮件发给产生这个bug的团队的总监。这位总监发邮件给那支团队的经理，该经理找出团队中谁应该负责这部分有问题的软件。10多天后，Terrence和两位经理、两位总监、另外三位工程师开会讨论这个bug，以及是否能在下一次发布前及时将其修复。听起来挺荒谬吧？可悲的是，这种事情时刻都在发生。与此相对，Fitz在谷歌工作的第一天发现了Gmail中的一个拼写错误，他打开源代码，修正了这个错误，然后给Gmail团队寄去一个补丁，并得到了衷心的感谢。

很多公司充满了极力维护组织层级的人。这导致了无休止的权力争斗，经理经常阻止工程师调去别的团队，以免自己的团队失去有一位价值的成员——即便对于公司和这位工程师来说这一调动都是正确的决定。

公司是否像对待调皮孩子一样对待过你？你是否因为公司防火墙设置过严而无法访问无害的外部网站？你是不是需要填写详细的时间卡汇报每个时间段的工作？有些公司甚至用一些无意义的（而且通常是极为不准确的）方法来计算工作量，例如，

每周编写的代码行数。

还有一些公司培养出来的员工用受邀参加的会议数量，而不是发布产品的数量和质量，来衡量自己有多么成功。

最后，你的公司可能缺少一些重要的东西，如焦点、愿景或方向。这常常是大师过多或“委员会设计”的结果，导致传达了冲突的命令。因此，大家最后可能是原地打转，而不是向一个一致的方向前进。

很多管理不当的公司都有以上提到的情况，而且问题远不止这些。这些公司仍然是由人组成的，你可以尝试一些技巧和方法，让人们帮助你完成工作。

操控组织

这是一个陪练程序，类似母体的程控现实。它具有同样的基本规则，例如，重力。你必须了解的是，这些规则与计算机系统的规则没有什么不同。有些规则可以改变，其他规则可以打破。明白了吗？那就来战吧。

——Morpheus

与陪练程序很像，公司是由规则组成的：有些规则可以改变，其他规则可以打破。如果你只注意公司中应该如何做事，通常只会得到挫败和失望。相反，你应当承认事情的现状，专注于探索公司结构，寻找可以用来完成工作的机制，在公司中为自己打造出一个安逸的位置。不管你处在一个好公司还是管理不当的公司，总有一些策略可以用于有效完成工作。

“取得原谅比获取许可更容易”

首先也是最重要的是，要了解公司允许什么行为——虽然请求确实让你有机会将责任推给别人，但也给了别人对你说“不”的机会。了解在没有明确获得上级批准的情况下，什么行为在公司内是可以容忍的，是很重要的。但只要有可能，我们还是建议你做对公司有益的事。

即使你准备好事后寻求原谅，也还是要谨慎行事——每次你为自己辩护或与公司内其他人作对，都会消耗自己的政治资本。如果耗尽资本只为赢得一些无关紧要的斗争，你会发现当重要事情发生时已无资本可用。请有策略地选择那些重要的或有些胜算的事情进行斗争。把所有的政治资本都花在一场自知无法获胜的斗争上毫无意义，会让人充满压力，会限制职业发展，而且没什么道理。更多细节请参见“政治银行账户”一节。

如果你决定要走“请求原谅”这条路，可以在公司里寻找一些同事和朋友，听听他们对你的想法的意见——特别是比较冒险的那些想法。

这些人应当很了解公司能够容忍或不能容忍什么行为，而且知道哪些想法行不通。

当市场部有人建议Fitz引起谷歌高管对他的Data Liberation团队的关注时，Fitz和他的智囊同事探讨了自己的想法：给高管一个Data Liberation牌的螺栓剪钳和锁着的宝箱（钥匙当然锁在盒子里）。他决定执行这个想法并获得了很大的成功。几年之后，当Fitz考虑打印一些仿制的钱币时，该智囊团认为这个计划风险过大，于是Fitz取消了计划。如果你打算未经允许擅自行动，相信自己的直觉固然不错，但从你信任的人那里获得一些意见也是非常重要的。

另辟蹊径

在公司内作出改变的另一个策略是想办法使你的想法被底层接受。如果你能使足够多的人支持你的想法或使用某个产品，通常管理机构就来不及“镇压”你，管理层不得不引起注意，要么接受要么反对。（对，你猜对了，这会消耗他们的政治资本！）多年来，很多工程师应用了这一策略，例如，将开源工具引入他们的日常工作流，以改善工作条件。

说客

如果你想说服某人，提高成功率的一个好方法是找几个支持你的人，让他们在和此人交谈时说起你的想法（提议或请求）。即使你的目标对你的举动心知肚明，但因为这个想法由多方提及而不仅由你提出，按常理他还是会对此更加重视。

想法是一个特别有趣的东西：如果你不在意功劳归谁，想法可以流传很广！有时人们会因为自己可以把一个想法归为己有才将其散布，因此你需要决定什么更为重要：功劳归你，还是想法得到传播。虽然听到自己的话从另一个（也许是你鄙视的）人嘴里说出来很令人心痛，但这往往是想法得到传播的最容易的方法。这种情况在大公司和小公司中一次又一次出现：从高管口中说出的高深概念和想法其实源自于其组织内的某个人。想象一下，在这种情况下，你的（本来会无人了解的）想法能接触到多广大的听众群啊！

和个人积习难改一样，公司也很难杜绝坏习惯。Ben早年的一位老师有个说法：“停止一个坏习惯是不可能的，你需要将其替换为一个好习惯。”试过戒烟的人都非常熟悉这一现象。公司也一样——如果要成功消除一个坏习惯，你需要找到一个更好的习惯进行替换。不喜欢每周例会？那就换成另一种会议或其他更有效的形式。不喜欢无用的报表过程？不要抱怨，你可以写一个好用到令人无法忽视的有用过程。一旦找到好的替代习惯，你需要克服大家不愿作出改变的惰性，因此我们推荐你将新形式给大家“试用”几周。这种做法使新事物显得不那么固定，不那么吓人，如果大家都喜欢这个新形式，那么等到“试用”期结束时，他们会忘记这其实是试用。

学会向上管理

无论你是管理人员还是基层员工，都需要花些时间向上管理。所谓向上管理，是说你需要确保自己的经理和团队外的人不仅知道你在做什么，而且知道你做得很好。

有些人觉得这种“推销自我”的模式很可鄙，这一点也许无法改变，但是这么做确实有很大的益处。

第6章中将提到，你需要尽可能少承诺，多提交。我们不是鼓励你多估工作量，拖延工期，而是说尽量避免承诺无法完成的事情，即便这意味着你不得不多说几次“不”。如果你总是耽误工期或无法提交功能，公司里的其他人会降低对你的信任，而且当他们寻找能完成工作的人选时，很可能不会考虑你。

我们建议你将推出产品放在首位。在公司里，提交产品比任何其他事情都更能带来可信度、声誉，以及政治资本。推出自己的产品是可见度极高的事件，能显示出你的成就。虽然你可能很想花许多时间清理代码库，进行重构，但根据我们的经验，如果你将大多数时间花在这种防御性工作上，很难得到团队外的人（包括你的上级）的欣赏。然后你会发现自己处在一个很尴尬的境地，几乎没有任何（政治上）重要的成果可以展示。这样做不仅无法得到认可，还可能导致你的产品被取消。

“进攻性”工作和“防御性”工作

Ben刚当上经理时，团队的工作效率似乎被以往累积的大量技术问题拖累了。他认为团队的首要任务是清理旧债。他的上级对此略表同意，然后大家就开始工作了。但事情进展并不顺利。虽然之前表示了同意，但Ben的经理在几个月后开始变得不满，失去耐心——为什么团队“什么都没完成”？Ben的团队其实非常高效，他试图向大家展示团队解决的大量历史问题是有成效的。但是这种工作根本无法让人产生深刻印象，在感情层面上也基本属于枯燥无味的类型。

在这次失败经验之后，Ben开始将工作分为“进攻性”和“防御性”。进攻性工作通常是新用户可见的功能——容易展示给外人并使大家兴奋的漂亮东西，或者是可以

提高产品吸引力的可见改进（例如，改进的用户界面，更短的响应时间）。防御性工作针对的是产品的长期健康（例如，代码重构、功能重写、格式修改、数据迁移，或者改进的紧急情况监控）。防御性活动能提高产品的可维护性、稳定性和可靠性。然而，虽然这些绝对是至关重要的，但完成防御性工作无法得到任何政治功劳。如果把时间都花在防御性工作上，人们会觉得你的产品没有任何变化。篡改一句老话：“法律的十分之九是认知。”

我们现在遵循一个简单准则：不管欠了多少技术债，团队花在防御性工作上的时间不应超过全部工作时间的三分之一到二分之一。超出这个范围都是自寻死路。

运气和人情经济

不管信与不信，无论你处在哪种公司，为自己创造某种运气并不是一件难事。Richard Wiseman在The Luck Factor一书中记录了他做的一项测试，该测试对人们发现偶然机会的能力进行了实验。

我给运气好和不好的人都发了一份报纸，让他们浏览报纸，并告诉我里面有多少张照片。运气不好的人平均花费两分钟数照片，而运气好的人平均只花费几秒钟。为什么呢？因为报纸的第二页上有一条信息：“不要数了，这份报纸里共有43张照片。”这条信息占据了半个页面，字体足有2英寸高。这信息太显而易见了，而运气不好的人总是对它视而不见，运气好的人却能看见。

他随后指出，运气好的人“善于创造和捕捉偶然机会”。我们认为同一原则也适用于在公司中创造机会：如果你循规蹈矩，只知道完成工作，完全不关注其他，那么就很少能获得偶然出现的机会。如果有机会，即便不是自己份内的工作，你依然可以帮助其他人完成工作，他们不一定会给你回报（你也不应有此期望），但在将来

有机会时会有很多人乐意回报你的好意。

政治银行账户

每个公司都有存在于组织架构之外的灰色市场人情经济，这些人情是用来充实政治银行账户的主要货币之一。公司里经常有一些位于别人工作范围内，但你可以快速轻易完成的工作，如果你注意寻找机会完成这样的工作（很多时候，有人会直接跑来请你帮忙），就可以在这个人情经济的银行账户中挣得一点积分。你可以把这些积分看成一些小赌注：有些打了水漂，有些能回本，还有一些能大赚。很难知道哪些赌注能获利，但有一件事情是肯定的，人们会记住你是帮他们解困的人。将来，在你受困而向他们求助时，他们很可能会愿意——甚至迫切希望——帮助你。如果你在他们寻求帮助时直接回绝“不是我份内的事”，结果就会大不同了。即使你没有得到“回报”，常常也会在帮助别人的过程中学到新知识，而且助人为快乐之本，除了一点时间和精力，你又会有什么损失呢？

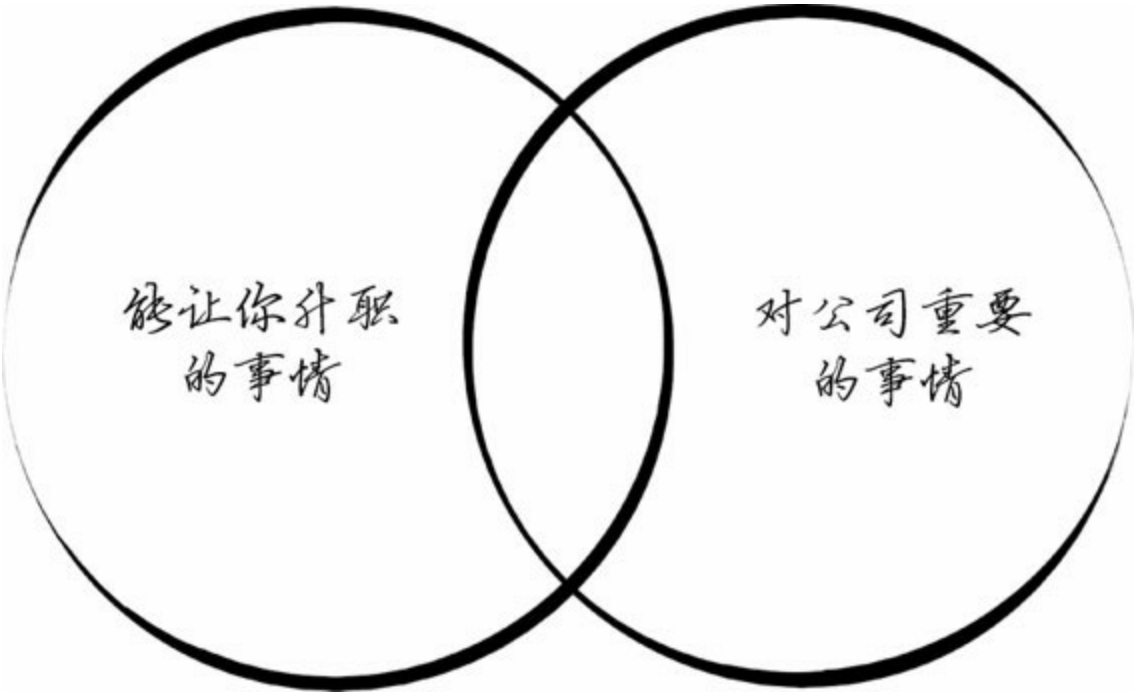
在你需要请公司的其他人帮忙时，这个政治银行账户的金额就会减少。可能是你需要别人帮你做某事，或者你做的某事冒犯了别人，甚至可能是你与公司内某人意见不合而已。如果你能意识到自己什么时候获得了政治资本，什么时候花费了资本，将会大有用处。如果缺乏这种意识，那么你的账户可能在你未察觉时就已透支，使你在公司内以及职业发展中举步维艰。

人情经济最有趣的一点是，银行账户并不会在你结束一份工作或离开一家公司时清空——在离开后你还能经常找公司的人帮个忙。正因为如此，你在离开一家公司时，不管当时有多气恼，都绝不应该断了自己的后路。

晋升到一个安全职位

如果你和大多数工程师一样，那么你会期望一个有逻辑的晋升流程——只要工作表现优异就能升职。不幸的是，这样的理想世界只存在于最开明的公司里。在大部分公司里，你需要作出一定的努力，“参与升职游戏”（通常还要工作表现优异）才能得到提拔。

如果你对自己的工作、薪水以及所在团队很满意，可能会选择不参与这种升职游戏，安于现有职位和级别，专心工作。但这种做法会使你在很多情况下轻易受到伤害——例如，公司重组，你被发配到新团队，你可能会在不称职的经理手下工作，或者受制于办公室政治家。



在组织中的层级越高（无论是作为独立员工或管理人员），就越能掌控自己在公司中的命运。当你在当前职位上游刃有余时，适当努力寻求晋升，是一种很好的投资，可以确保自己在公司和团队发生震荡时的安全和幸福。你可以留意自己的工作成就，在自我评估时有所展现。更新自己的简历，并分享给经理或选拔委员会。研究选拔流程，找经理谈谈你还需要满足哪些条件才能升职，并有系统地实现这些条件。虽然提拔过程很主观，具有非确定性，你还是可以作出很多努力，增加自己的机会。

寻找有影响力的朋友

每个公司都有一个“影子”组织架构，虽然没有记载，却具有很大的能量和影响。这种架构中只有很少的几类人。

联系人熟知公司上上下下的所有人物，如果他们不认识团队中的某个人，也能为你找到合适的人。有时候，完成某事的关键就在于找到正确的人，联系人就能帮你找到这个人。

老员工不一定级别很高或头衔很大，但他们通常知道很多过去的信息，因为在公司的时间很长，所以就拥有很大的影响力。如果你想知道公司以某种方式运作的原因，或者需要德高望重者的支持，就应该找这些人。

人们大多数时候说起来都像在开玩笑，但行政助理实际上是他们为之工作的高管的代理，因此在公司中拥有极大的权力和影响力。更重要的是，他们通常需要完成极多的工作以保持事情运转顺利，如果惹恼了他们，你和你的职业前途可能会不太妙。记住，永远不要放过向行政助理示好的机会——他们是人情经济的基石。

如何写信找忙碌的高管帮忙

在任何大公司工作足够长的时间后，你会发现自己有时需要给高管或者任何你不认识的忙人写信，请他帮忙。其目的可能是为自己的产品或团队求得某事，或者纠正错误。不管具体情况如何，这可能是你第一次和此人通信。此时，几乎每个人都会犯同一个新手错误：啰哩啰嗦、唠里唠叨、喋喋不休。

14多年前，Fitz（在苹果公司工作时）给他母亲买的一台iMac电脑有质量问题，在一位同事的建议下，他给史蒂夫·乔布斯写了一封“短”信。这封电子邮件可以作为一个粗略的原型，展示如何有效地向高管请求帮助。

日期：2001年2月1日，星期四

收件人：sjobs@apple.com

主题：公司硬件的糟糕客户体验——我该怎么办？

如果您能就如何解决这个问题提供一些建议，我将不胜感激。这个问题非常令人尴尬，对苹果公司和我本人都是如此。

去年母亲节时我给母亲购买了一台iMac电脑。她是新奥尔良一家蒙氏学校的副校长，在学校有一台旧麦金塔电脑。得到这台iMac电脑后她非常兴奋，甚至为所在的学校申请了基金用于为实验室购置iMac电脑。

但是，我给她买的这台草莓红的iMac电脑却出现了质量问题。

七月，它进入睡眠状态后就无法唤醒。母亲将电脑送到一家苹果授权的经销商处，诊断结果是一个逻辑板有问题并进行了更换。

她把电脑带回家，插上电，电脑开始启动，然后就出现了一个哭脸图标，发出了死机的声音。她把机器送回经销商处。诊断结果是一个模拟板有问题并进行了更换。

九月，我终于说服她再次使用睡眠功能（而不是关机或重启）。但这台iMac电脑无法唤醒。如果将电脑电源拔掉然后重新插上，机器就能够重启。于是我们完全禁用了睡眠功能。

十二月，显示器开始闪烁，从黄色变成绿色又变成蓝色。她昨天将电脑送回了经销商处，到现在还没修好。

目前情况就是这样。我母亲认为是我在故意整蛊她，她告诉每个认识的人她的iMac电脑是垃圾，而我在苹果认识的同事没有一个人知道该怎么处理这种情况。

（除了再买一台）有什么办法可以让她得到一台好用的iMac电脑吗？

致敬！

Fitz

不到20小时后，Fitz就收到了史蒂夫手下某人打来的电话，两周后他母亲拥有了一

台（没有质量问题的）新iMac电脑。

告诉你一个大秘密：当有机会纠正错误时，身居高位的人通常会很乐意去做——工作繁忙的高管（他们很多人喜欢纠正错误，人人都明白获取一点额外政治资本的价值）也是如此。不幸的是，这些人的电子邮件收件箱永远都像是在遭受分布式拒绝服务的攻击，如果他们看到从未谋面的人发来的长达3000字却一张图都没有的邮件，很可能会在读上15个字后就按下删除键，然后接着处理下一封邮件。

但是，如果他们能用10秒读完一封邮件然后挥动魔法棒（即给手下发封邮件处理此事）就修正问题，那么他们很可能会付诸行动。他们只要花几秒把任务分派下去，就能从你那里得到一大堆的政治资本。

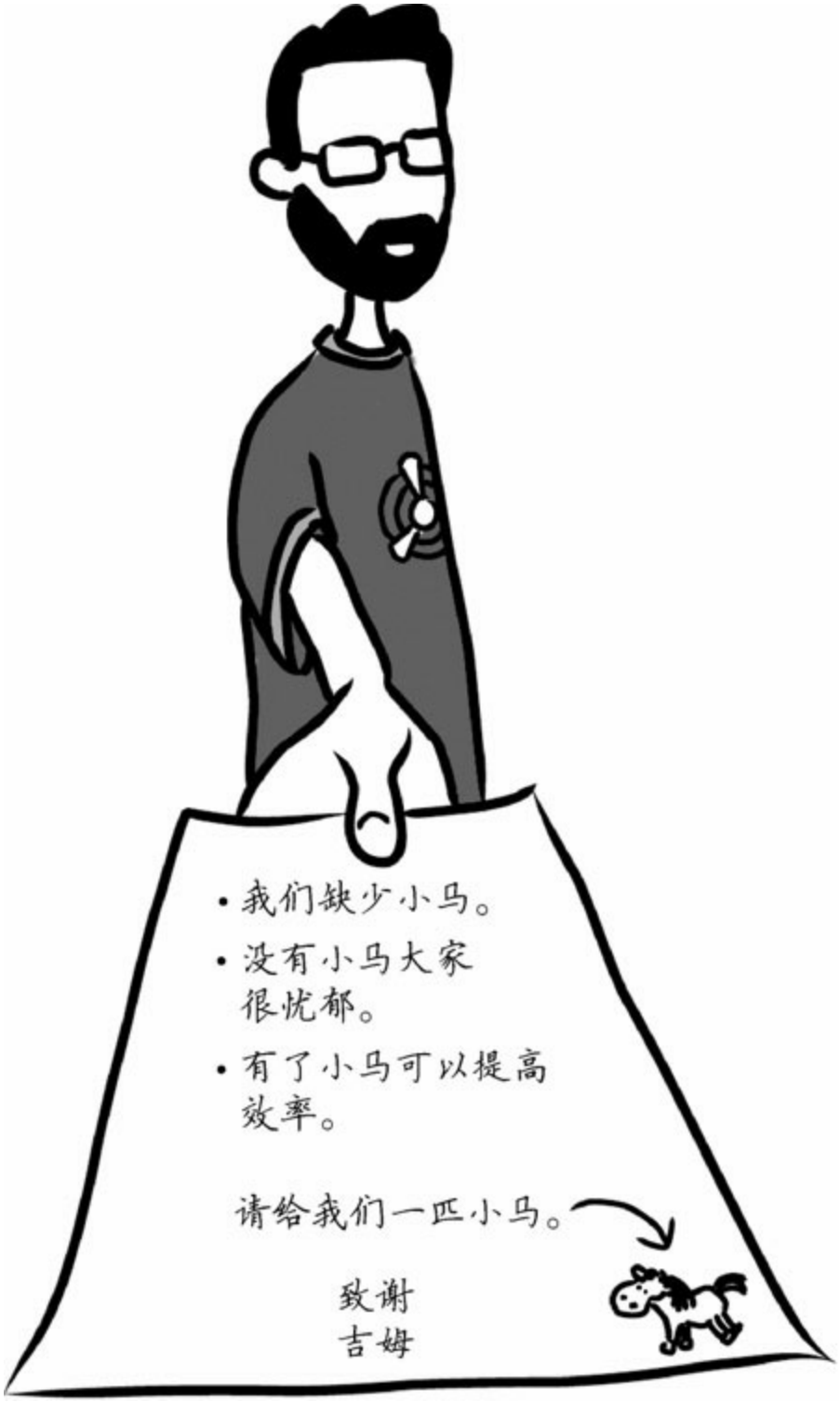
通过多年的尝试摸索，我们发现邮件越短越容易得到回复。

我们将此技巧称为“三条要点和一个行为召唤”，这一技巧可以极大地提高收件人采取行动的可能性——或者至少给出回复，不管你冒昧地发邮件给谁（不光是高管）以寻求帮助。

一封使用“三条要点和一个行为召唤”技巧的优秀邮件应最多包含三条要点说明当前的问题，以及一个——只有一个——行为召唤。这些就是全部，不能有更多内容——你需要写出一封很容易转发的邮件。如果你太啰嗦或者在邮件里提到了四件完全不同的事情，那他们肯定只会回复其中一件事，而且那件事正好是你最不关心的。或者更糟，由于读起来太费脑，你的邮件直接被删除了。

要点应当是短句子（每个句子应该能用一行显示，无需折行），行为召唤也应当尽量简洁。如果你想得到任何答复，就应当让对方容易在行内进行回复，最好答案只需一个或两个字。不要在一段里问好几个问题——一段限制在一个问题，或者理想

情况下，一封邮件只包含一个问题。最后，你的邮件应当体现HRT原则：礼貌、谦恭，没有语法和拼写错误。如果你实在忍不住，非得加上更多的背景和信息，那就放在邮件最后（甚至在签名之后），并清楚标明是“更多细节”或“背景信息”。



事后再看，我们觉得Fitz的原型邮件写得有点长。如果今天将其重写，可能会是下

面的版本。

日期：2001年2月1日，星期四

收件人：sjobs@apple.com

主题：糟糕的客户体验——寻求您的帮助

我为任职学校管理员的母亲购买了一台iMac电脑。她对此很激动，甚至为所在学校申请了基金用于为实验室购买更多iMac电脑。

七月，苹果公司为这台电脑更换了有问题的逻辑板，一个月后又更换了模拟板。

九月，这台电脑无法正常休眠。十二月，显示器出现了问题。电脑目前仍在经销商处。

我母亲告诉所有她认识的人，说自己的iMac电脑是垃圾，而我认识的苹果同事没有一个人知道该怎么办。

我该如何做才能让她得到一台好用的iMac电脑呢？

致敬！

Fitz

重写的版本行文不那么生动，但一位忙碌的高管可以在10秒钟内将其读完。

在职业生涯中，我们一次又一次地使用了上述各种技巧来完成工作。但有的时候所有的诀窍和方法都于事无补。

备用计划：撤退

在我们和大家讨论如何在管理不当的公司里完成工作，以及如何与不那么好的人共事的数年间，每次演讲后都有人找到我们，一副筋疲力尽的样子，告诉我们他已经尝试了各种方法，但情况就是得不到任何改善，什么事情也无法完成，到底该怎么办呢？答案很简单：如果你的确无能为力，那就不要再耗着了，走为上策。

如果你无法改变系统，那么再多努力也是无用。换个思路，你可以努力离开这个系统：更新简历，向关系好的朋友打听其他公司有没有适合你的职位。学习一些新知识。作为知识型人才最好的一点是，今时今日对我辈佼佼者的需求很大，由此给了我们掌握自己未来的能力。

一旦你意识到未来是由自己掌握的，就会觉得海阔天空。如果你四处打探，发现自己还有其他的工作机会，可能就会发现自己突然（在很少压力下）完成了很多工作，因为如果丢了现在的工作也不是世界末日！在谷歌长期担任“Jolly Good Fellow”的Chade-Meng Tan有一篇博文非常振奋人心，极大影响了我们的工作方式。

做正确的事，等着被炒

谷歌的新员工（我们称他们为“Nooglers “）经常问我，是什么使我能有效地工作。我半开玩笑半认真地告诉他们，答案很简单：我为谷歌和世界做正确的事，然后休息一下，等着被炒。如果没有被炒，说明我做的事对大家都好。如果被炒了，说明我本来就不该为这家公司工作。所以，不管结果如何，都是我赢。这就是我的职场策略。

如果你有备而来，知道自己的选择，那么你就是世界上最自由的人。不要害怕离开。五年来，我们总是在演讲后给大家提出这样的建议，可以很高兴地向大家汇报：好几个人写信告诉我们，他们考虑了我们的建议，现在正从事着他们热爱的工作。这些信当属我们收到过的最棒的信——下面是我们最喜欢的一封。

日期：2011年12月1日，星期四

主题：感谢信

发信人：Alex Mrvaljevich

收信人：Brian Fitzpatrick

Brian，你好！

你可能不记得我了，但是你给了我两条最好的职业建议，改变了我的人生。

我听了你在谷歌IO 2010上的技术演讲，结束后找到你，就当时的工作状况寻求意见。你简单地告诉我应该“走人”，所以我给你发了自己的项目经理求职简历。此后你告诉我谷歌极少雇用没有技术背景的项目经理。

因此我重新对自己的职业前途进行了长时间的严肃思考：不管谷歌的策略是什么，大多数公司肯定也差不多。

变更职业方向是我唯一的选择，最靠谱的方向是产品管理。我拼命学习，不仅从当时的公司“走人”，而且离开了所在的国家（委内瑞拉的产品管理职位不多），结果这成了我人生中最明智的举动……

此举使我获得了一份极好的工作，现在我在日本担任产品主管，二月份就要搬去神

户了。

我曾暗自许诺，如果你的建议真起了作用，就要给你写一封感谢信，因此现在我要说一声：

谢谢你。

Alex

希望尚存

说了半天辞职、等待被炒的问题，不是说如果现在的工作干得不开心就应该更新简历立马走人。相反，你的首要目标应当是作出所需的改变，使自己开心，完成工作目标，本章为你准备了很多能派得上用场的工具。如果不去努力了解如何操控所在的组织结构，就无法掌握自己的职场命运。

第6章 用户也是人

前面探讨了很多对产品开发成功至关重要的因素。

从一小群聪明、富有创造力的人开始；注入谦虚、信任和尊重的强大团队文化；以服务的方式领导他们，帮助他们达成合作，作出明智的决策；给他们提供水、阳光、方向，以及所需的内在驱动力；保护他们免受负面影响——威胁团队文化、影

响团队前进的破坏性行为或环境——的干扰；保持72°F烘烤六个月，你就可以得到一些优秀的软件。大功告成了，是这样吗？

很多程序员的努力止步于此。他们为自己编写软件，对最终结果非常满意，然后就认为自己成功了。

不幸的是，真实世界并不是这样的。“良好的软件”是过于狭隘的成功定义。如果要养家糊口或者只是充实简历，你还需要很多其他人来使用你的软件并对其感到满意。软件开发过程并不结束于产品交付，实际上，这一过程永远不会结束。人们使用你的软件，你就需要对用户作出回应，并逐渐改进产品。如果不知道如何掌控这个反馈循环，你的工作就白费了。

这一章将讨论用户参与的三个阶段。首先需要唤起用户对产品的注意——他们知道这个产品的存在吗？在使用前他们对产品有何看法？然后需要考虑用户在开始使用后的体验。产品是否达到预期？产品可用度如何？是否帮助了用户？最后，我们要讨论如何与忠实用户进行高效互动。所有这些交互都是产品开发循环中的一部分。

要记住一点：合作不限于团队，要创造优秀的产品，你还需要积极地与用户合作。

如果疏于此道，你所得到的不过是一个没人使用的漂亮软件。若果真如此，也许你该质疑一下自己的职业选择了！

管理公众认知

听到市场营销这个词，你想到的第一件事是什么？

如果你和大部分人一样，这个词可能会使你想起满嘴跑火车的推销员的形象，一脸

假笑，大背头梳得铮亮：这种人的工作就是为客户或产品打造形象。如果你的产品是待售的生“肉”，那么市场营销人员的工作就是给牛排加上诱人的嘶嘶声，从而使更多人争相购买。

为什么我们这么讨厌这种形象？为什么我们一想到市场营销人员就会不寒而栗？

原因在于，在程序员看来，营销人员代表着工程文化的反面。我们专注于事实。代码要么编译成功、要么编译失败；软件要么具备某个功能、要么不具备；问题要么得到解决、要么没解决。我们不会“虚构”对世界的描述；我们陈述事实，然后着手进行改变工作。我们在市场营销人员身上看到的都是谎言，而我们不喜欢听到谎言。当需要作出决策时，我们希望事情井然有序，结果可预见，并且描述准确。

我们将市场营销视为扭曲事实的行为，因此其与创新者对精英文化的直觉需求相悖。我们总是相信最好的产品应该胜出。所谓“最好”，即按客观标准质量最好、效率最高，而不是广告做得最漂亮的产品。但我们一次又一次失望地看到较先进的技术失去市场：很多人相信Betamax比VHS好，Laserdisc比DVD好，或者Lisp仍然是最好的编程语言。（我们只是需要多做宣传！）即便在版本控制工具世界中也是如此，虽然较新的系统（如Git）技术更为先进，但Subversion还是占据着企业市场。

更严重的是，我们认为营销人员总是向客户作出过多承诺，导致工程师交付的产品好像总是不够好。想想就让人火冒三丈。

我们要告诉你一些坏消息，还有一些好消息。

坏消息是：你不能忽视市场营销。市场营销确实很重要，你必须做好这方面的工作。好消息是：积极与市场营销合作是可能的。如果采取正确的方式，市场营销行为不一定要那么低俗。实际上，市场营销可以成为通向成功的强大工具！

程序员总是逻辑感过强，但大多数人的行为决策受到逻辑和情感的双重影响。市场营销人员是情感操控的大师，这也是他们成功的秘诀：他们将感觉与事实调和在一起，获取人们的注意。如果想让新用户使用你的软件，就必须考虑用户对软件的感性认知。人们做决定的方式是无法改变的。

在获得非专业用户对技术的感性认同方面，苹果公司是公认的大师。身处2015年，我们要问：客观地说，iPhone手机真的比安卓手机更好吗？它们的功能几乎是一样的。但是如果一位普通用户相信iPhone是神机，那它就真的是神机，至少对这位用户是这样的。认知即现实。或者如前文所言，“法律的十分之九是认知”。

你可能会认为获胜的唯一途径是退出，但这是一场不容忽视的游戏。你至少需要一个最简单的营销策略，才能让自己的软件进入市场。如果你精于此道，那么会发现营销可以大大增强杰出工程技术的效果。下面将介绍一些基于HRT原则的基本技巧。

注意第一印象

如果你肚子饿，正在找地方吃饭，那么一个餐馆从大街上看是什么样就相当重要了。如果一个餐馆显得令人厌恶或者不讨喜，你根本就不会进去。如果一个餐馆显得温暖又友善，服务生热情好客，你很可能会愿意进去尝尝。请不要低估精心设计的产品体验的感性影响——如果你有过iPad或Nest温控器的开箱体验，就会明白我们在说什么了。

对于一个新手而言，你的产品是什么形象？是否显得亲切并欢迎探索？相反，如果一位专家坐下来初次试用，你的软件是否显得熟悉而合理？你的应用程序是第一眼看上去就能立刻上手，还是需要大量的学习和无数的尝试？更具体地说，用户在打开你的软件后最初30秒内的体验如何？不要光给出客观的回答（ “他看到一个选择

菜单，还有一个登录框”），还要给出感性的回答。新用户在一分钟后感觉如何？自信满满还是满怀挫败？对此你该如何作出改进？退后一步，看看你的产品网站。网站是否像漂亮的临街店铺一样既专业又吸引人？你需要认真对待这些问题，别人才会认真对待你的软件。

少许诺多提交

不要让市场人员先发制人。如果用户问到将实现的功能或产品发布时间，你可以借此机会给出非常保守的估计。如果任由市场人员胡说八道，最后会出现与《永远的毁灭公爵》类似的局面——这个软件发布时间拖了15年。但是如果你抢先发布自己较为准确的消息，用户就会很惊喜。谷歌在这一点上做得就很好，特意制定了一个政策，任何产品都不许预先公布功能。当新功能上线时，总会给用户带来意外之喜。这种做法也可防止团队内部拼命加班以赶在对外宣扬的不切实际的发布日期交付产品。软件应该在真正完成并可用时发布。

与工业分析师谦恭合作

很多程序员厌恶媒体，他们认为媒体只是变着法的市场营销而已。当行业杂志或研究机构上门拜访时，很多公司会放下全部工作去满足他们的所有要求。他们认识到一篇好的（或不好的）访谈可以造就（或毁掉）一个产品的公众认知。但工程师总是对媒体的这种权力和公司对他们的恭顺感到不满。

例如，Apache软件基金会的成员有一段时间与分析师打交道出现了问题。如果分析师想找Apache软件基金会索要一份描述Apache HTTPD服务器的行业标准白皮书，成员通常会没好气地回答：“和别人一样，到网站上看文档去。”这种做法虽然满足了开源开发者对精英文化的深刻认同心理，但整体效果对公众认知不利，特别是

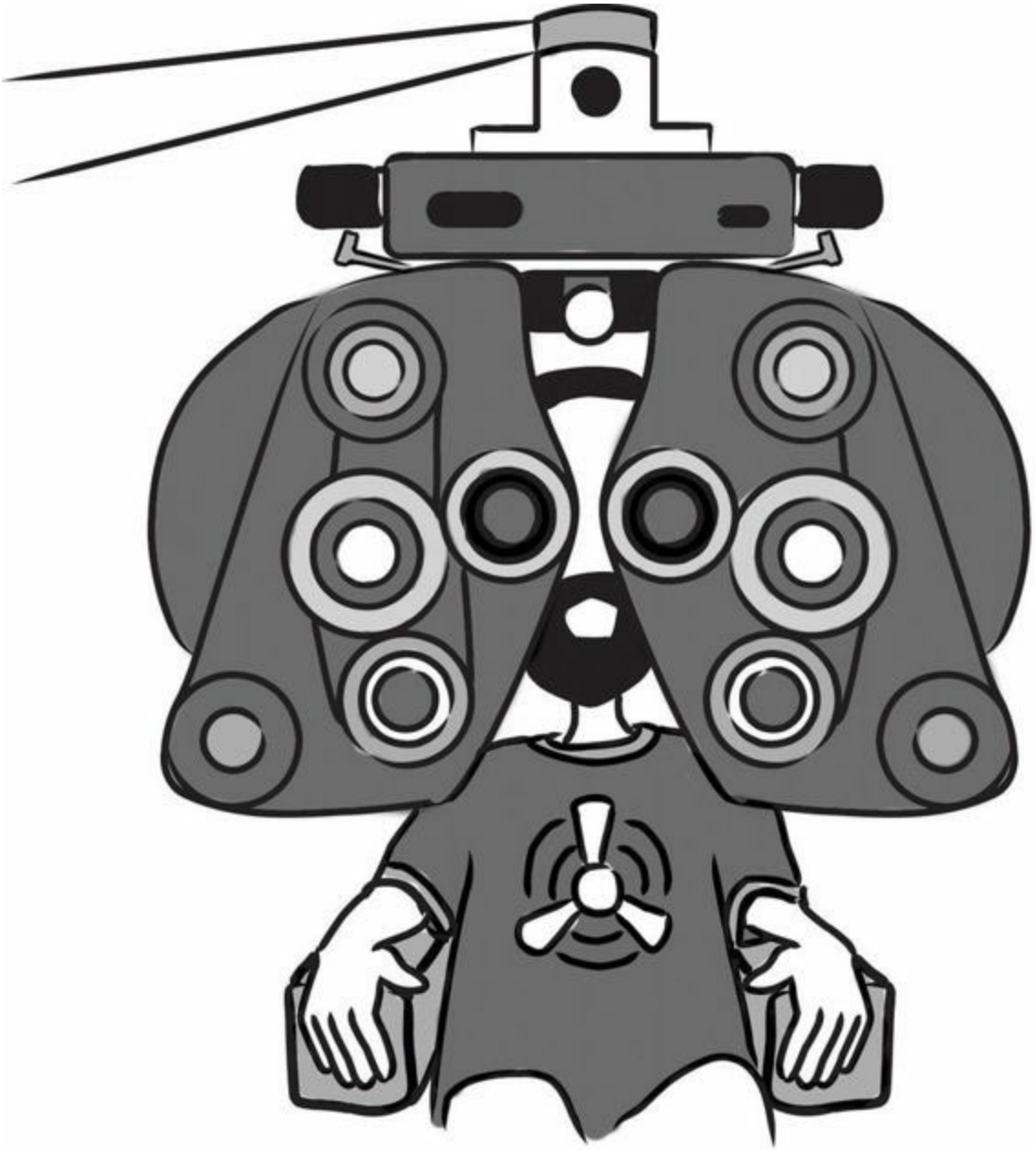
对企业用户。最后Apache软件基金会的“公共关系人员”对很多社区成员重新进行了教育，以帮助他们摈弃这种态度，更高效地与分析师合作。用非暴力、不合作的态度对抗一个系统——不管这个系统有多烦人——没有任何意义。这种做法和让餐馆评价师排队等位是一样的效果。餐馆评价师应该得到特别优待吗？也许不应该。但是有必要将这作为原则问题吗？绝对没必要。这么做吃亏的是你自己。请谨慎选择哪些抗争是值得的。

你的软件可用性如何

告诉你一个残酷的真相：除非你开发的是软件工具，否则工程师不会是软件的用户。因此，作为一名工程师，你对软件可用性的评价基本不靠谱。一个对你而言完全合理的界面很可能会让你那技术盲邻居抓狂。

假设“成功软件”的条件是“很多人使用并喜爱你的软件”，那你就需要对用户格外留意。谷歌有一句名言：

关注用户，其他便都将水到渠成。



听起来有点故作姿态，但我们在工作中一次又一次在多个项目中看到这条准则生效，见证了因这条准则成功和失败的项目。

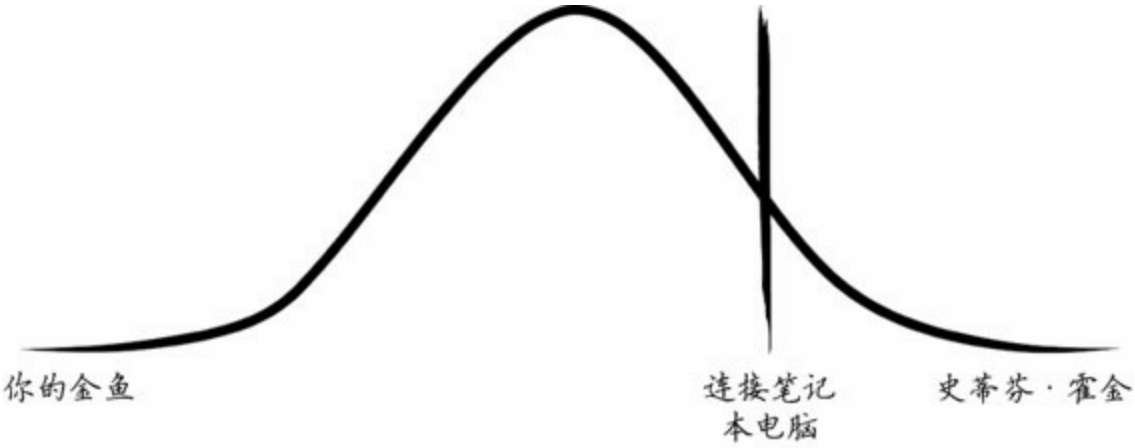
谷歌的一大突破是开始量化搜索广告的有效度。如果用户点击一个特定的广告，说明该广告对他们有用；如果一条广告无人点击，说明它令人讨厌或是无用。谷歌从系统中移除不好的广告，并向广告投放者提供反馈以促进广告改进。最初这种做法在短期内起到的是反作用：谷歌自己拒绝了收入来源。但通过使搜索者（而非广告投放者）成为关注焦点，从长期来看，这种做法极大提高了谷歌搜索广告系统的可

用度（和实际使用度）。

下面将讨论直接关注用户的一些重要方法。

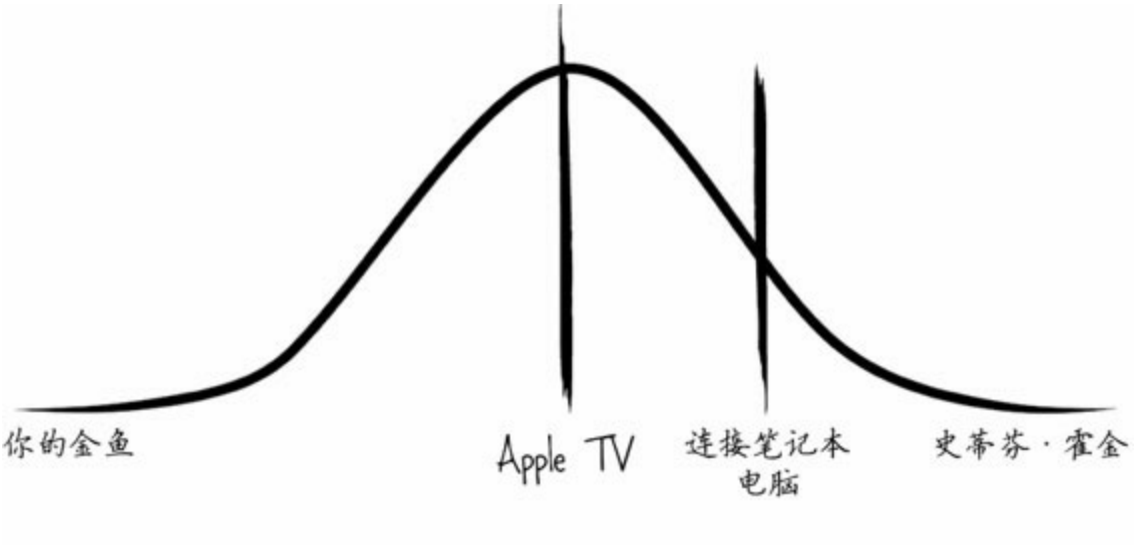
选择听众

先说最重要的：想象你的用户落在不同的技术能力范围内。



如果在图中画一条竖线标识你的产品最适合哪部分用户，这条线会在哪里？画在这条钟形曲线中心的线表明大约有一半的用户（即：线右边的用户）能愉快地使用你的产品。

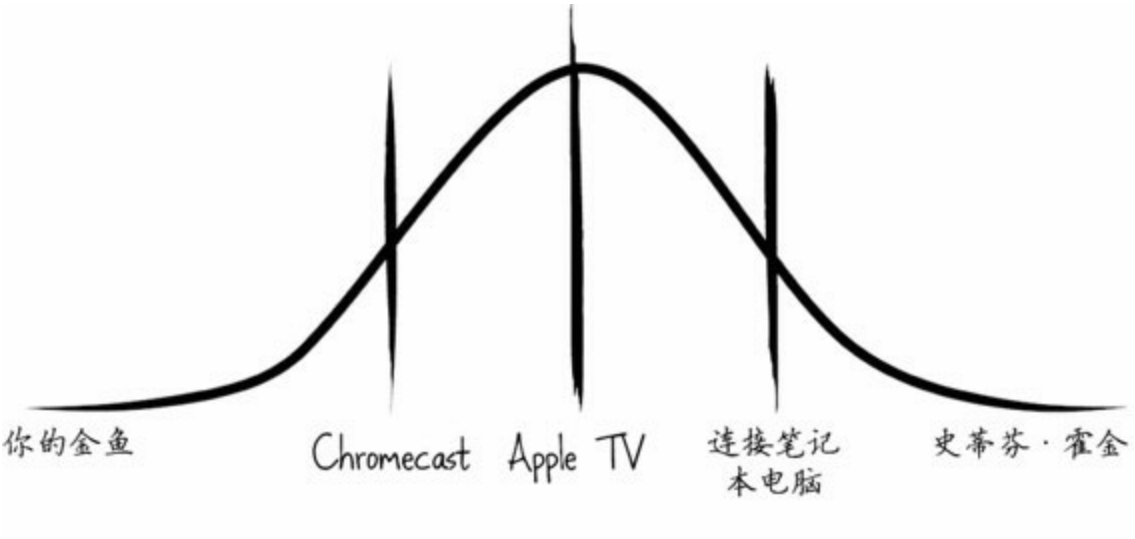
举个例子，考虑一下把网络内容显示在大屏幕电视上的功能。各种竞争解决方案的“可用性”是如何拓宽潜在用户群的？最初人们需要把笔记本电脑直接连接到电视。这需要用户理解模拟和数字信号输入，并且要准备合适的音频和视频线。



于是苹果推出了Apple TV——一个像计算机一样的小巧设备，可以一直连着电视。你可以从计算机或智能手机控制它，可以播放私人的多媒体文件或实时的网络内容。这个设备解决了更大范围（更不懂技术的）用户的问题：它自带接线，连接以后就可以不用再管了。

谷歌随后更进一步，推出了Chromecast，这是一个可以直接插在电视HDMI接口的小电视棒。Chromecast更容易安装，并且可以把更多苹果和非苹果设备的屏幕“播”出去。在我们编写此书时，市场上已经有新型号的电视机自带内建WiFi和网络流媒体。Ben的孩子可能永远不知道曾经有过不带Netflix的电视机！

好产品开发的目标是将这条竖线尽量向左侧移。一般来说，你拥有的用户越多，就越成功（公司挣的钱越多！）。当考虑用户时，要认真考虑产品针对的目标客户是谁，是不是有尽可能大的用户群？这就是为什么简单体贴的用户界面如此重要——还有好的文档和容易获取的教程等。



考虑进入的壁垒

现在想想初次使用软件的用户。第一次运行有多难？如果用户无法很容易地进行尝试，你就不会有任何用户。初次尝试的用户通常不会去想你的软件比同类竞争者强还是弱，他只是想很快完成某项任务。

为了说明这个问题，让我们看看流行的脚本语言。大多数编程者认为Perl或Python比PHP更“好”。他们宣称Perl/Python/Ruby程序的可读性和长期维护性更好，拥有更成熟的代码库，处于开放网络中时固有安全性更好，且拥有更强的抗攻击能力。但PHP要流行得多，至少在Web开发中如此。为什么？因为任何一位高中学生都能通过复制朋友的网站自己琢磨学会PHP。不需要读书，不需要培训教程，也不需要认真学习编程模型。PHP很方便修修补补：你只需要改改自己的网站，从同伴那里学会各种PHP用法。

另一个例子是文本编辑器。编程者应该使用Emacs还是vi？用哪种编辑器重要吗？其实不重要，但一个人为什么会选择某个编辑器而不是另外一个呢？有一个真实的故事：Ben刚开始学习Unix（在1990年当实习生）时，需要找一个文本编辑器用。他第一次启动vi打开了一个已有的文件，20秒后就觉得无比挫败——他可以浏览文

件内容，却无法输入任何东西！当然，vi用户都知道你得先进入“编辑”模式才能修改文件，但对一个新手来说这仍然是一场糟糕的初体验。当Ben启动Emacs时，他马上就能开始编辑文件，和使用熟悉的家用文本处理器一样。因为Emacs的初始用法和他以前的经验一致，Ben立刻就成了Emacs的用户。这个选择理由很傻，但这种事每时每刻都在发生！使用产品的最初一分钟的体验至关重要！

当然，还有其他因素会毁掉第一印象。软件第一次运行时，不要让用户填一张巨大无比的表格，也不要让他们设置一大堆必填选项。强迫用户创建某种新账号也会让人反感，这意味着用户什么都没做就得承担长期责任。另一个让人讨厌的是，打开网站两秒就会弹出“立即订阅！”的模式对话框。所有这些都会让用户落荒而逃。

TripIt网络服务设计用于管理行程，是近乎零进入壁垒的一个极好例子。你只需要将已有的行程确认邮件（航班、旅馆、租车，等等）转发到plans@tripit.com，嗖地一下，你就可以开始使用TripIt了。这项服务会为你创建一个临时账号，解析邮件，创建一个完美的行程页面，然后发邮件告诉你一切都准备好了。这就像是突然有了一位私人助理，而你做的不过是转发了几封邮件而已！你不费吹灰之力，就已经深度介入，以一名参与用户的身份浏览了该网站。到了这个时候，你就会心甘情愿地创建一个真正的服务账号了。

如果你对自己产品的进入壁垒持怀疑态度，可以尝试一些简单的测试。让普通人——包括技术人员和普通用户——试用你的软件，观察他们使用时的第一两分钟。你也许会对自己的发现大吃一惊。

衡量使用量，而非用户数

在考虑用户群大小和软件是否易上手时，你还应当考虑如何衡量使用量。请注意，

我们说的是“使用量”，而不是“安装数”——你希望很多用户使用你的产品，而不是很高的下载量。你可能经常听到有人说，“嘿，我的产品有3百万下载量——那就是3百万的用户啊！”等等，先等一下。这3百万用户中有多少人在实际使用你的软件？这才是我们所说的“使用量”。

举个极端的例子，有多少台机器安装了Unix归档工具“ar”？答案是：大概每台使用基于Unix的操作系统（包括Linux的所有版本、Mac OS X、BSD，等等）的机器都装了。那么有多少人使用这个程序呢？又有多少人知道这个程序是什么吗？这就是一个安装量达百万而使用量却接近零的软件。

很多公司（包括谷歌）投入大量时间衡量使用量。常用的度量指标包括“7日活动”和“30日活动”——即有多少用户在过去一周或一个月内使用了这个软件。这些数字很重要，它们实际说明了软件的使用情况。你可以忽略下载数，转而寻找衡量持续活动的方法。例如，如果你的产品是一个网站或Web应用，那你可以试试像Google Analytics这样的产品。Google Analytics不仅能提供这些指标，还可以帮你了解用户来自何处，停留多长时间，等等。这些都是反映产品使用情况的极为有用的指标。

设计很重要

在互联网兴起之前，将产品投入市场的最大挑战之一是发行。很少有公司拥有足够的财力编写一个产品并将它投放到世界各地成千上万的店铺中，因此，当公司推出一个产品时，会做大规模的市场营销。这通常会导致每类软件有一两个“胜出者”（例如，Microsoft Word与WordPerfect，Excel与Loutus 1-2-3，等等）。这种情况下，你选择产品的主要标准是功能和价格，不会考虑软件是否丑陋

或不直观。

然而，这一切，已经改变了。

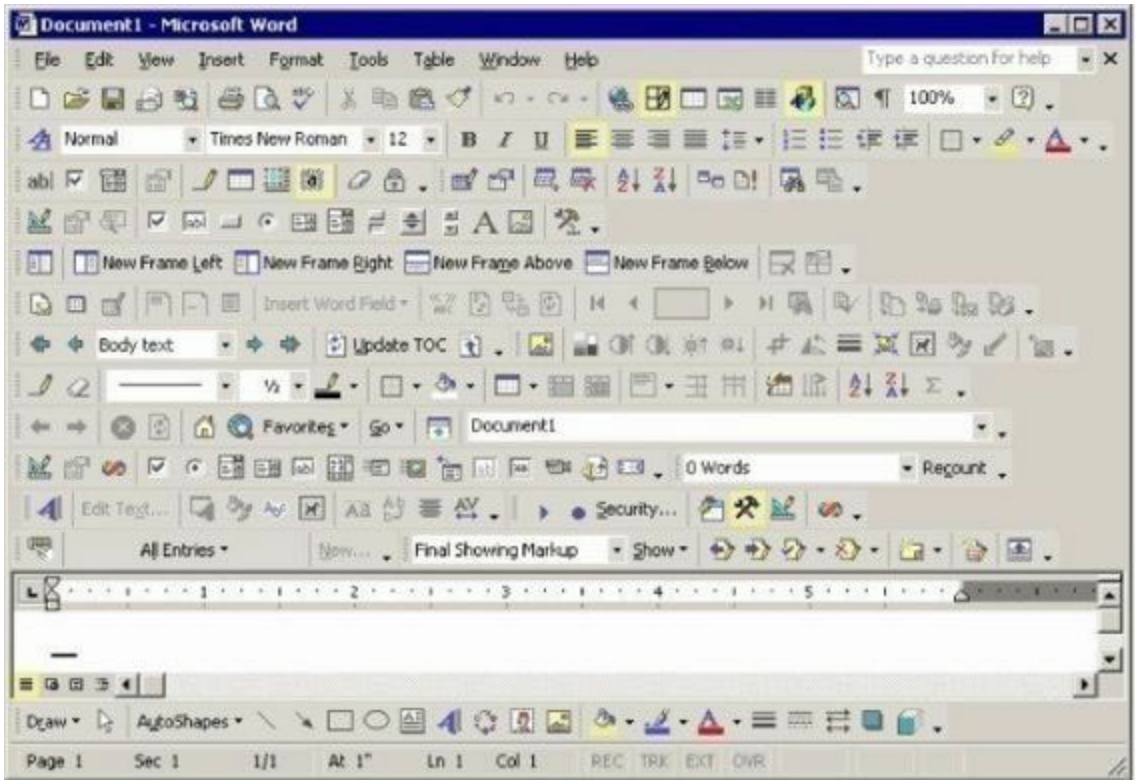
互联网是一个全球发行网络，在这里，寻找和下载软件的成本几乎为零。社交网络使人们很容易将自己对于各种产品的感受瞬间分享到全球范围内。这两大变化（以及许多其他较小的因素）的结果意味着今天的消费者能够选择使用什么产品。在这个竞争激烈的环境中，仅仅推出具有必备功能的产品已经不够了——你的产品需要漂亮的外观而且使用要简单方便。如今，再多的市场营销也无法拯救一个蹩脚的产品，而令用户愉悦、设计精良的产品则会将这些用户变成为你推销产品的传道者。

因此，良好的设计是关键，但良好设计的很大一部分是以用户为先，隐藏复杂性，提高产品速度，最重要的是，不要一味求全。

以用户为先

“以用户为先”的意思是，你和团队应当在产品开发中付出一切努力，令用户使用产品更容易。这些努力可能是某种困难的工程工作，但更多时候是作出困难的设计决策，以免让用户在每次使用产品时进行选择。我们将这称为产品懒惰。有些人认为懒惰是工程师的美德，因为它导致了工作的高效自动化。另一方面，它也很容易创造出给用户带来很多痛苦的产品。产品易用性是产品开发的最大难题之一。

这种懒惰的典型例子是给用户提供过多选项。人们最爱嘲笑20世纪90年代后期的Microsoft Office产品的版本：按钮工具条！你可以立刻使用每个可能的菜单项……方便极了！用户界面设计师很喜欢取笑这个设计，特别是将这个功能用到极致时。



选项过多会令人无所适从，心生畏惧，裹足不前。甚至有一些关于选择过多如何导致焦虑和痛苦的书。连软件的偏好对话框你都需要小心设计。（你知道流行的电子邮件客户端Eudora有30个不同的偏好值面板吗？）如果让某人填写表格，请对可接受值尽量宽松：处理多余的空格、标点或破折号。不要让用户做解析工作！这是对用户时间的尊重。如果程序员本可以做出对最终用户友好而易用的产品，却没有费神去做，其实用户很容易发现这一点（而且对此很恼火）。

速度不容忽视

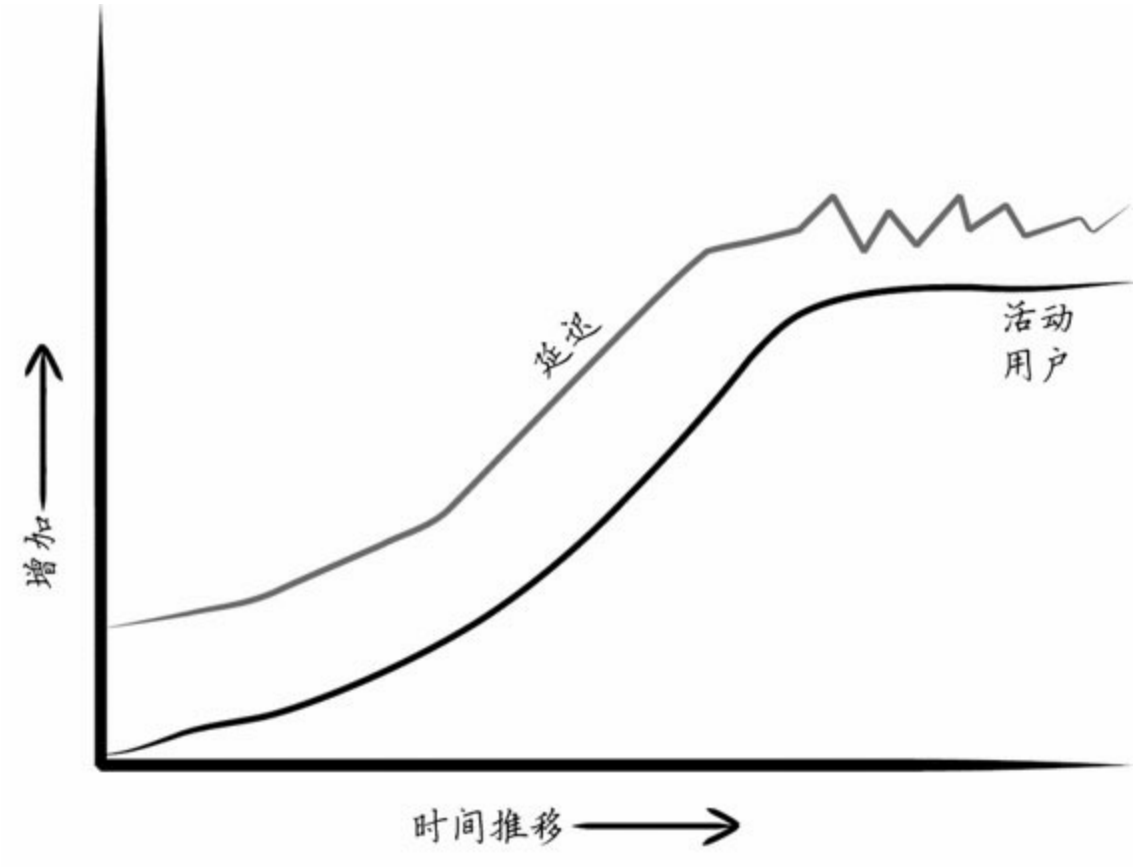
大部分程序员都极大地低估了应用程序速度（或延迟，这个词听起来更科学一些）的重要性。延迟的影响既基础又意义深远。

首先，延迟是另一类“进入壁垒”。我们已经习惯对Web页面速度要求极高。当初次访问一个新网站时，如果三四秒还没完成加载，人们经常会停止加载，并对此网站失去兴趣。这种行为都不需要借口。Web浏览器使得人们很容易离开，将注意力转到

另外的12个地方。我们有更好的事情要做，没空等一个页面加载。

其次，如果一个程序响应快速，会对用户产生很深的潜意识影响。他们开始更经常地使用这个程序，因为它感觉特别平顺。该程序会成为他们能力的一种无意识扩展。另一方面，随着时间的推移，运行缓慢的程序会变得越来越令人沮丧。用户会越来越少地使用这个软件，经常连用户自己也没有意识到这个变化。

一个产品发布后，目睹使用量逐渐增加会令人非常兴奋。但一段时间之后产品使用量常常会达到一个极限——开始原地踏步了。这时候，市场营销人员经常会跑来嚷着要增加功能，换个配色，用个新字体，或者做更多“弹出”动画。然而，有时候这种使用量停滞的真正原因是延迟。程序变得缓慢，不如人意。下面的图形显示了用户使用量随着延迟增加而降低。



有一个来自谷歌的真实故事：某天，一个工程团队发布了一些能极大降低Google

Maps延迟的改进。没有正式宣布，没有博客发帖，这些改进完全是默默地偷偷发布的。但之后几天内活动图表就显示出使用量极大（而且持久）的增加了。大家的心理一定发生了极大的改变！

如果你提供的是基于Web的应用程序，即使微小的性能提升也是很重要的。假设你的应用主界面加载需要750毫秒，这似乎挺快的，对吧？对任何单个用户都不算太长。但如果你能把加载时间降低到250毫秒，这多出来的半秒聚集起来就能产生很大的不同。如果你有100万用户，每人每天发出20个请求，那就一共为用户节省了116年的时间——不要浪费用户的时间！降低延迟是提高使用量，保持用户满意度的最佳方法之一。正如谷歌创始人喜欢说的一样：“速度是一个功能。”

切勿求全

你的软件是否功能太多？这个问题初听起来有点傻，但是市面上有些最糟糕的软件确实是因为野心过大而失败。这些软件想为所有人完成所有功能。有些最好的软件因为精确定位自己并完美解决问题而获得成功。这些软件没有试图解决每一个问题，而是为大多数用户解决十分常见的问题，而且效果明显。

我们经常拿杂志广告上的一些工具开玩笑：嘿，瞧，营地灯，还自带天气收音机！……啊，还自带电视、秒表和闹钟，还有……呃？真是一团糟。相反，你可以把软件看作一个简单的烤面包炉。它能烤任何东西吗？肯定不行。但是它能烤很多十分常见的食物，几乎对每个人都有用，又不会令人困惑。把软件做成烤面包炉吧！少即是多。

隐藏复杂性

“但是我的软件很复杂，”你可能会想，“能解决很复杂的问题，为什么要隐藏这一点呢？”这个问题很合理，但也是优秀产品设计的核心难题之一。优雅的设计能轻易地完成简单工作，又提供完成困难工作的可能。即使在完成复杂工作时，你的软件也应当显得无缝而容易使用。（再次强调，我们关注的是用户的感受。）

这就是我们所说的“隐藏复杂性”。对于一个复杂问题，将其分解，隐藏起来，或者用其他方法使软件看起来很简单。

再看看苹果公司。苹果公司的产品设计具有传奇性，它最聪明的设计之一是创新性地解决了MP3音乐管理的问题。在iPod出现之前，市场上有过一些试图在移动设备上直接管理音乐的不成功工具。苹果的天才之处在于，它意识到MP3管理问题过于复

杂，无法在小屏幕上解决，因此将这一问题转移到计算机上。iTunes就是解决方案。使用（带有大显示器、键盘和鼠标）计算机管理音乐，然后用iPod单纯进行回放。于是iPod可以设计得简单、优雅，管理音乐的工作也不再那么费劲。

Goolge Search是隐藏复杂性的另一个知名案例。谷歌的接口（以及进入壁垒）几乎不存在：只有一个可供输入的神奇文本框。但在这个文本框背后有分布在全球成千上万台机器的并发响应，在你每次敲击按键后进行搜索。当你敲击回车键时，搜索结果已经出现在了屏幕上。这个文本框背后的科技含量非常惊人，但这个问题的复杂性对用户而言是隐藏的。在用户看来，这个功能如魔法般神奇。这是值得创新性团队追求的一个伟大目标，因为它基本上就是产品可用性的代表。

最后，我们应当提出关于复杂性的一个警告。虽然隐藏复杂性是值得称道的，但封闭产品，束缚用户并不可取。隐藏复杂性几乎总会需要创建聪明的抽象，作为程序员，你需要假设这个抽象最终总是会“泄漏”。Web浏览器返回404错误就是一个泄漏的抽象，假象破灭了。但是不要慌——你可以坦然接受抽象最终会泄漏的事实，提供精心设计的方法绕过这层抽象。其中一个很好的方法是给别的程序员提供API。或者，对于高级用户，创建一个“专家模式”，为那些想要绕过表层抽象的用户提供更多的可选项和决定权。

我们不仅要保持接口灵活可选，而且也要使用户数据可访问。Fitz投入了很大热情确保谷歌的产品提供“数据自由”——即用户可以从一个应用程序很方便地导出数据然后离开。无论接口如何优雅，软件都不应当将用户封闭其中。允许用户打开封口，自由处理自己的数据，可以迫使你诚实地竞争：人们使用你的软件是因为他们愿意这么做，而不是因为他们没法离开。这种做法可以使用户对你产生信任，而用户的信任是你最宝贵的资源（稍后将详述）。

管理与用户的关系

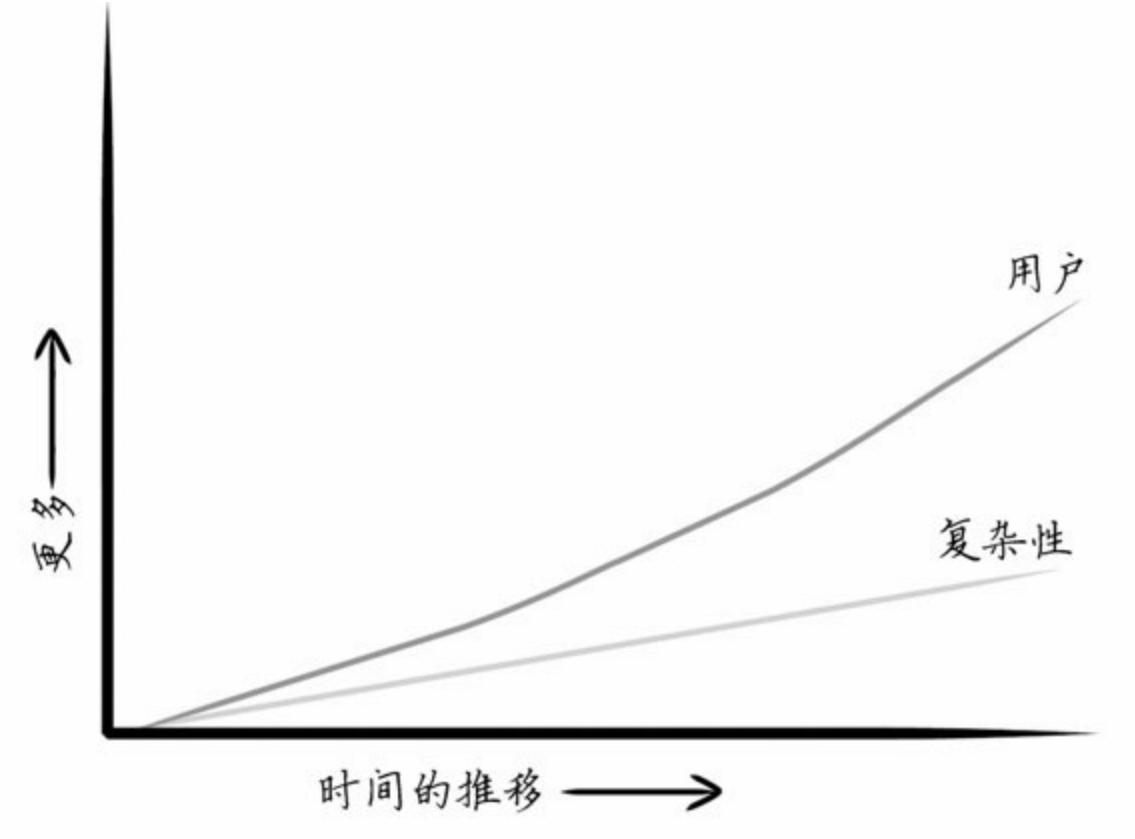
好了，现在你的产品给人的第一感觉非常好，而且容易上手。一旦人们开始使用，体验非常愉快。那么几个月之后会发生什么呢？应该如何与数年来每天使用产品的用户进行交互呢？

不管你是否相信，很多用户希望与产品的公司或团队建立关系。满意的用户希望知道软件的演化情况，不满意的用户希望有个投诉的地方。程序员常犯的最大错误之一是交付软件之后就对用户反馈置之不理了。

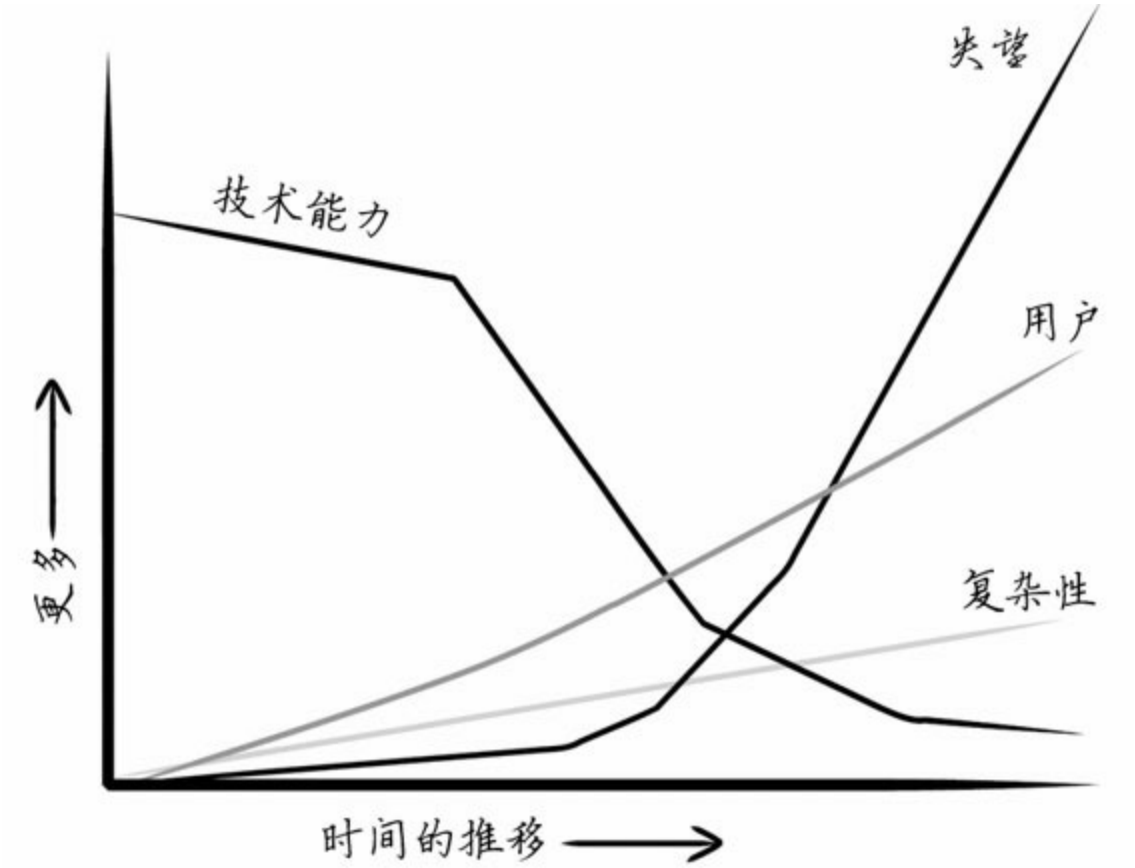
和市场营销一样，客户服务这个词通常也带有负面含义。人们想象的“客户服务”人员的形象经常是咖啡店的服务生或者满屋接听支持电话的机械人员。而在实际生活中，客户服务并不是一项肮脏、令人不快的任务，也不是其他人（职位描述较差的）从事的工作。客户服务是一种人生哲学——一种看待自己与用户持续联系的方式。作为一支创新性团队，你需要主动提供客户服务，而不是将客户服务简单视作对外部投诉的回应。

工程师经常害怕与用户直接打交道。“用户啥也不懂，”他们认为，“他们很烦人，无法进行沟通。”虽然没人要求你对每个用户都充满爱意，但一个简单事实是，用户希望有人倾听。即使用户提出疯狂的建议或让人摸不着头脑的投诉，你能做的最重要的事也许是关注他们。用户参与讨论和开发过程越多，他们的忠诚度和满意度就会越高。你不一定要同意他们的意见，但仍然需要倾听。这就是HRT原则中的R——尊重！在这个社交媒体年代，各家公司很快认识到了这一点——要作为一个普通人而不是形象高大、面目模糊的企业与用户进行交流。人们喜欢企业公开显示出HRT原则。

我们喜欢用另一个简单（略微非科学一些）的图形来说明长期管理用户的重要性。随着时间的推移，软件会获得越来越的用户。当然，随着不断“改进”，产品的复杂性也会不断增加。



问题是，随着用户数的增加，用户的平均技术能力水准会下降，因为产品的目标用户覆盖了越来越多的普通人群。将这一因素与不断增加的复杂性加在一起，就得到了严重的用户失望问题。



更多的失望意味着更多的投诉、更不满意的用户，以及日益增长的与软件开发者进行公开沟通的需求！

如何避免这一趋势呢？

首先，不要拒绝承认这个问题的存在。很多公司不自觉地尽一切可能在程序员和用户之间建立一道机构障碍。这些公司创建了语音留言树让用户迷路，或者为投诉建立“支持请求”，由一层层并未实际编写这个软件的人来跟踪。消息只能通过这些层次间接传递，就好像直接接触这些危险暴民会对开发者产生危险（或者不必要地干扰他们）一样！就这样，用户最后感到受忽视、受打击，而开发者完全与用户脱节。

更好的交互方式是直接关注用户。提供一个公共的缺陷跟踪系统，供用户进行投诉，并在其中直接作出回应。创建一个邮件列表，让用户互相帮助。在社交媒体上

与用户直接互动。如果你的产品可以开源，这也会有极大的帮助。你对用户表现得越“人性”，用户就会越信任产品，不满情绪也会降低。时刻注意并发现以预料之外（并极佳）方式使用产品的人。只有通过真正的对话，才能发现用户是如何实际使用软件的，可能会有很聪明或令人欣喜的用法。

尊重用户的智商

请在默认情况下尊重用户。导致我们害怕与用户直接交互的一个常见误解是，用户都很笨。毕竟不是他们写的软件嘛，所以他们只是“无知的用户”，这样理解对吗？如果最终有机会与用户交互，要记住，最重要的事情是不要表现得傲慢。作为计算机用户的能力如何并不能代表一个人的整体智慧程度。有很多聪明人只是将计算机作为工具使用而已。他们对调试程序或者依循科学方法确定问题并无兴趣。请记住，大多数人都不知道如何拆解和修理自己的汽车。认为用户愚笨是很荒谬的，就和汽车技师因你既不会重建传动装置又不关心如何诊断传动问题而认为你很笨一样。汽车是一个黑盒子——你只想驾驶汽车。对大多数人而言，计算机（以及软件）也是一个黑盒子。用户并不想参与诊断过程，他们只想完成自己的工作。这和智商一点关系也没有！

耐心

那么结论就是，要学会保持耐心。大部分用户只是不知道用什么词汇简明地表达遇到的问题。你需要学习数年的时间以理解他们说的是什么：问问那些想通过电话给父母提供计算机技术支持的人（本书的大部分读者大概都有过这种经历！）就知道了。一半的谈论时间都花在了让双方用词达成一致上。很多人不知道什么是Web浏览器，认为那是计算机的一部分。他们将应用程序描述为动作，或者认为屏幕图标是神秘的工作流名称。问题是，即便最聪明的人也会创造自己的逻辑空间（以及词汇

表）来解释计算机的行为。他们会用只存在于自己脑海中想象的分类和规则来诊断问题。

父母：“我觉得我的计算机变慢是因为磁盘满了。”

你：“你怎么知道磁盘满了？你检查了吗？”

父母：“嗯，满屏幕都是图标，大概没有地方下载我的邮件了。也许我可以删掉一些cookie，空出点地方？上次好像就是这样解决问题的。”

你：[掩面无语]

最关键的倾听技巧在于学会理解人们想说的是什么，而不一定要试图解释他们实际说的是什么。这不仅需要一些语言翻译，而且还需要情商，以及读心术。

Fitz讲过一个他祖母的故事。祖母（在电话里）问他：“Brian，你祖父的旧电脑还有什么用吗？” Fitz说没用了，那就是一台很旧的没联网的Mac Classic——也许最好送去回收了。祖母的回答是：“OK，那我只在削铅笔的时候打开它好了。”

在莫名其妙了很长时间后，Fitz决定问问她，搞清楚她到底是什么意思。

原来这台Mac电脑和祖母的电动削铅笔器插在同一个插线板上。祖母每周都会带着铅笔到这个房间来打开这个插线板。这台Mac电脑就会滴地一声开始启动。祖母削完铅笔后就会关掉插线板，离开房间，在这台Mac电脑完成启动之前突然断电。这是非技术人员试图用有限的词汇以及她与计算机间的特定关系模型解释一个情况的极佳范例。

很多人对谷歌的搜索服务也有着神奇的看法。许多人认为这只是他们计算机的一部分。2005年，当我们告诉人们我们是谷歌的工程师时，他们总是面露不解之色：“哦！我不知道还有人在那儿工作？！”另一方面，有一次Fitz祖母的一位朋友听说我们整个公司要去外地滑雪而很不高兴。（当时公司还很小）“太糟了！他们怎么能都跑去滑雪呢？”她问道，“那谁来帮我做搜索呢？”显然，谷歌这是玩忽职守，没有留下足够的接线员保持线路畅通。

创造信任和愉快

还有两个关键字应当成为你与用户交互的基石：信任和愉快。

信任是个微妙的词。我们在介绍HRT原则时已经讨论过信任——是否需要以及如何向同事展示信任。这里我们讨论的是从用户处得到信任。用户信任你的团队（或公司）主要是一种感情状态：很少人会说，“我信任产品X，因为这一长串事实证明了我与它的关系没有风险。”他们信任你是因为他们与你之间交互的累积达到一种整体感情上的正面状态。

想想你的朋友和家庭。他们中有多少人有真正信任的汽车技师？如今这个答案几乎为零。几乎没有人信任汽车技师，这是因为我们多年来受所谓的“mailboxing”所害：当你去店里预约某项服务时（例如，更换机油），却被强加了许多其他的维修服务，就像邮箱里塞进垃圾邮件一样。没人再相信汽车技师，因为他们总是利用一切机会使利润最大化。请记住，不存在所谓的暂时失去诚信。

这个例子说明了长期关系会如何被短期获利轻易破坏。时不时地对顾客略做手脚，最终他们会透过一层累积的鄙视之意来看待你们之间的关系。另一方面，每次团队完成对客户有帮助或有用的工作，或回应了客户的意见，他们心中的假想银行账户就会增加一点对团队的信任。当买了12个甜甜圈而烘焙师又给你添上第13个时（新奥尔良人称这个为lagniappe，意为赠品），你的脸上会露出微笑。多年相处下来，这个信任账户会逐渐增长，最后一提到你的产品，用户心中就会产生温暖舒适的感觉。

然而，信任也可能很脆弱，因为它可以毁于一旦——就像一次愚蠢的冲动购物可以透支一个银行账户一样。如果公司的行为显示出对用户全然缺乏尊重（即便是无意之举），信任银行也会一夜清空。

一个很好的例子是，Netflix在2011年末如何暂时毁掉了其客户关系。Netflix既提供互联网流媒体电影服务，又提供通过邮政信件租借DVD的业务。它在十年间变得日益流行：简单、方便而且新颖。价格也很便宜。到2011年初Netflix已经拥有超过2300万的用户。

后来业务人员意识到他们的DVD租借和流媒体服务其实是不同的业务，具有不同的盈利模式、管理需求等。因此他们决定开始对这些业务分开收费，将某些用户的月费提高了60%。顾客为此大为恼火。然后Netflix宣布要拆分成两个单独的公司，以获

得更好的清晰度和便利。在用户看来这就是说，“现在你得付两份而不是一份账单了。”意识到公司的公共关系危机后，Netflix随后又取消了公司拆分，但为时已晚。伤害已经造成了。虽然此前用户数一直持续增长，但三个月内Netflix就失去了800 000位用户。这几个小举措看似为简单而必要的商业决定，但没有考虑已有的客户关系，使他们失去了累计十年的大部分信任。（幸好，他们在之后几年格外注重服务和内容，完全重建了信任银行，而且势头更猛了！）

信任是最宝贵的资源。要小心看护它；要度量信任银行账户的大小；在每个举动之前都要考虑到此举将如何影响信任银行账户；关注你的长期形象，而非短期便利。

和信任一样，愉悦是另一种可以大幅提高你与用户关系的感觉。它可以增加温暖舒适感，并使你的团队显得更加有人情味儿。

你必须从别对自己太严肃开始。谷歌有个传统，每年四月愚人节会发布一些古怪的产品。例如，有一年，YouTube首页上的每个视频都会播放“rickroll”。或者去看看http://www.woot.com，这是一个每日促销网站，但其广告充满了自我调侃和冷幽默。

尝试给用户带来令人惊讶的美妙惊喜。（这就是愉悦的定义，不是吗？）虽然谷歌是硬派计算机科学的顶尖公司，但偶尔出现的为节日或周年庆祝所做的“涂鸦”总是令用户无比兴奋。这只是投入人们生活中的一丁点艺术创作，却带来了无尽的反馈信件和办公室讨论。

当然，只要表现手法幽默，少量恐怖也能激发用户。一家试图建立社交网络的公司一度鼓励新用户上传自己的照片，最后该公司决定给每个没上传照片的用户显示一张作咆哮状的迪克·切尼——然后大家就突然都开始上传照片了！

（委婉地）加入点滴的愉悦和幽默能显示出你确实关注用户，关心你与他们之间的关系。

记住用户

本章讨论了很多想法，但最后可归结为三个简单概念。

市场营销

了解人们对软件的看法；这决定了他们会不会愿意尝试。

产品设计

如果软件做不到容易尝试、速度快、友好，而且用户面广，用户就会流失。

客户服务

主动与用户建立长期的良好关系能影响软件的演化和用户保持率。

程序员的工作充满了各种杂务——审阅代码、讨论设计、调试工具、解决产品相关问题和处理bug——我们很容易忘记自己编写软件的初衷。编写软件不是为了你，也不是为了团队或公司，而是令用户的生活更轻松。关注用户对产品的想法和观点，以及他们的长期使用体验，是至关重要的。用户是软件成功的命脉。一分耕耘，一分收获。

后记

读到这里，你应该很清楚，本书中的大部分建议并不只适用于产品开发。

我们主要讨论的是，维护健康、功能正常的社区——任何社区——的艺术。你可以任取书中的故事，去掉与产品开发相关的部分，替换为任何其他活动。我们讨论的可以是社区俱乐部、教堂团体、兄弟会或建筑团队；它们存在同样的社会问题，也适用同样的解决方法。无论在什么环境中，人们总是行为难测，不易相处。产品开发具有任何其他团体活动存在的社区健康问题。

因此，当你忙着在日常工作中实践HRT原则时，请记住，这些原则也可用于生活的其他方面。

谁知道呢？也许我们真正的天分是写布道词。但目前我们将继续编写软件，获得最佳合作成果。现在你也拥有了同样的能力。

最后一点想法

本书介绍了很多内容。读完之后，你也许很难决定应该将哪部分用于日常生活。毕竟，如果读完之后工作方式毫无变化，本书还有什么意义呢？那么现在应该怎么做呢？

简单地说，如果你读完本书只记得一点内容，那么请记住HRT原则：谦虚、尊重和信任。

如第1章所述，这三个核心特点应该成为你所有社交行为和全部人际关系的基础。如果仔细观察，你会发现，几乎每种社交冲突的源头最终都可归结为谦虚、尊重或信任的缺失。

请记住，HRT原则适用于各种不同的影响“范围”。HRT是最基本的原则：这些特点影响你的每次沟通。HRT原则适用于团队：建立在谦虚、尊重和信任之上的团队文化能令团队专心编码，少些争斗。HRT原则适用于领导团队：有能力的领导者为团队服务，而不是团队为领导者服务。HRT原则也适用于与团队之外的人员的短期合作，不管这些是好人、混蛋，还是运转不良的机构。最后，这些原则可直接指导你与头号重要团体——产品用户——的交互。

如果在工作中时刻遵循HRT原则，可做到事半功倍。将大部分时间投入热爱的工作（交付产品），以最少的时间处理社交冲突、机构琐事和其他人际纷争，HRT原则就是助你达成此目标的最好方法。

推荐书目

我们根据自己与大量团队及他人合作的经验完成了本书，但读过的很多书籍和文章也辅助了书中想法的形成。以下是影响过我们的一些书籍和文章。

□ Critical Chain，作者Eliyahu M. Goldratt（North River Press出版）

□ Drive: The Surprising Truth About What Motivates Us，作者Daniel H. Pink（Riverhead出版）

□ Mastery: The Keys to Success and Long-Term Fulfillment，作者George Leonard（Plume出版）

□ Project Retrospectives: A Handbook for Team Reviews，作者Norman L. Kerth（Dorset House出版）

□ Startup Engineering Management，作者Piaw Na（自出版）

□ The Art & Adventure of Beekeeping，作者Ormond Aebi（Rodale Press出版）

□ The Luck Factor，作者Richard Wiseman（Miramax出版）

□ The Significance of Task Significance: Job Performance Effects,

Relational Mechanisms, and Boundary Conditions（2008），作者Adam M. Grant（Journal of Applied Psychology 93:1, pp. 108–124）

□ “Maker's Schedule, Manager's Schedule”（http://www.paulgraham.com/makersschedule.html），作者Paul Graham

□ “You and Your Research”（http://www.cs.virginia.edu/~robins/YouAndYourResearch.pdf），作者Richard Hamming

□ 《安静：内向性格的竞争力》，作者Susan Cain

□ 《编写可读代码的艺术》，作者Dustin Boswell、Trevor Foucher

□ 《重来》，作者Jason Fried、David Heinemeier Hansson

□ 《怪诞行为学》，作者Dan Ariely

□ 《硅谷最受欢迎的情商课》，作者Chade-Meng Tan

□ 《你就是极客！软件开发人员生存指南》，作者Michael Lopp

□ 《人件（第2版）》，作者Tom DeMarco

□ 《人月神话（第2版）》，作者Frederick P. Brooks

□ 《软件开发者路线图》，作者Dave Hoover、Adewale Oshineye

□ 《三双鞋：美捷步总裁谢家华自述》，作者Tony Hsieh

- □ 《无从选择》，作者Barry Schwartz
- □ 《拥抱变革》，作者Mary Lynn Manns