初创企业的首席技术官手册

高效工程团队的基本技能和最佳实践

Zach Goldberg 著

2023 年，扎克·戈德堡，zach@zachgoldberg.com

平装书：978-1-955811-56-9

电子书：978-1-955811-57-6

国会图书编辑号：2023918702 封面设计：Michael Rehder /www.rehderandcompanie.com/ 版面设计和排版：Paul Baillie-Lane/www.pbpublishing.co.uk 编辑：Stephen Nathans-Kelly 和 Paul Baillie-Lane

由 WorldChangers Media 出版 PO Box 83, Foster, RI 02825 <www.WorldChangers.Media>

https://ctohb.com https://startupctohandbook.com

致谢

给 Max Mintz，感谢他教会我学习和珍视生活中重要的事物。

对于我曾经直接指导过的每个下属，感谢你们的耐心和忽略我肯定犯下的许多错误。

对于我的妻子，感谢你忍受并支持我追求的众多事业，包括这本书。

赞誉

> Zach Goldberg 的《CTO 手册》为所有工程领导者提供了一个引人入胜的日常资源。无论是实际的日常框架还是有洞察力的观点，Goldberg 的书将立即帮助你解决开发高效工程团队中最复杂的问题。 麦克·洛普，randsinrepose.com

> 今天的初出茅庐的工程领导者们有了很好的建议！

> Matt Mochary，执行教练，作者《卓越 CEO 内部》 mocharymethod.com

> 扎克（Zach）在为初创组织的首席技术官们（甚至超出此范围）创建资源方面做得很出色。他为我们作为初创公司的技术领导者所面临的人员、流程、技术问题提供了可操作的建议。 托尼·卡雷，博士。 LA CTO 论坛创始人，TechEmpower 创始人兼首席执行官，Aggregage 创始人兼首席技术官

对于任何工程领导者来说，这是一本基础指南！

戈登·普雷托里乌斯（Typeform 首席技术官）

扎克简洁明了的章节中融入了几十年的工作经验，领导初创技术团队。值得一看！

> 丹尼尔·德米特里，首席执行官和 3 次初创企业的执行者

> 《首席技术官手册》是一本充满灵感的实用建议集，适用于有抱负或有经验的技术领导者。无论您是正在从零开始构建世界一流的工程团队，还是有成为首席技术官的野心，或者已经担任这个角色多年，本手册都将成为您不可或缺的指南。

> 爱瑞克·约翰森，Dama Financial 的首席技术官，《C# 8.0 简明手册》的作者

> 当我在黑暗中摸索着尝试弄清楚自己的情况，并被几本技术管理书籍压得无法承受时，这是我所需要的所有内容的简洁总结。

> 查理·冯·梅特拉特，MetricFire/Hosted Graphite 的联合创始人

目录

简介

永远保持学习态度

十四岁那年，我的父母送我去参加了几个星期的住校计算机夏令营。正如你心中所想的那样，这个夏令营非常极客：摊开的折叠桌一排排，坐着数十个（大多是）年轻男孩，他们专注于他们灰色的 CRT 显示器上的《虚幻竞技场》，而不是他们的编程课程。两年后，十六岁时，我作为一个辅导员/编程教师回到了计算机夏令营，我非常喜欢那段时光。我非常幸运，在年轻的时候，就意识到了我对计算机和软件编程的热爱，并且我的父母一直支持着我。

再往前几年，到了我进入宾夕法尼亚大学的大一之前的夏天。我确定我想要在本科阶段研究计算机科学，但我也在脑海中对商业产生了兴趣。我的父亲创办了自己的企业，我兄弟刚刚毕业于商学院，所以商业似乎是一个很好的选择。宾大以其让学生可以获得多个领域学位（如工程学和商业学）的双学位计划而闻名。

对于我这个十八岁的自己来说，这个机会似乎是完美的，所以我给我的顾问马克斯·明茨博士发了一封电子邮件，并认真地安排了一次面谈，讨论我申请双学位计划的事宜。作为一个非常慷慨和以学生为中心的教授，明茨博士热情地同意了，并邀请我去费城，在 Tuscany Cafe 喝杯咖啡谈论这个问题。 在选定的那一天，我从纽约驱车三个小时来到费城，坐在 Mintz 博士的对面，迫不及待地想听听如何玩弄这个体系。我想，关键是选择合适的课程并取得足够好的成绩来符合条件。然而，Mintz 博士却有其他想法。

充满期待并准备好接受如何完善我的简历的指示，我咕咚了一口咖啡，问他：我如何进入双学位计划？很快我就会知道的这个人，我只知道他叫 Max，拿起一张餐巾纸，在上面画了一个 X-Y 轴，然后看着我问我是否知道什么是特殊相对论。我希望我能有个视频记录那一刻，因为我想象我的脸扭曲成一个相当有趣的形状。在我还没来得及完成我的回答之前，Max 已经开始了。在接下来的两个小时里，他开始向我介绍爱因斯坦的理论。等我们讲完了，我的脑子已经支持不住了，我们从未讨论过宾夕法尼亚大学的双学位计划。

在接下来的几个月里，我们又喝了几杯咖啡，每次我问 Max 关于申请或简历的事，他总会让我重新回到真正的科学上。Max 想让我去*学习*，不只是吸收他当时讲课的主题，而是要擅长学习，而且是学习困难的东西。对于他的学生们在四年之后获得的那张纸，Max 一点也不在乎，只要每个人都准备好继续学习他们一生的知识。

到我大学毕业时，Max 已经成为我亲密的朋友和心灵的知己，他从根本上改变了我的教育道路。他不是给我鱼，而是给了我一个钓鱼竿，并教我如何装饵和投线。

没有一本书能够给你大学本科生时 Max 给我的经验。对于你现在正在阅读的这本书，我不做任何承诺。相反，我讲这个故事是为了强调专注于学习本身的基本价值和影响。 作为技术领导者，对继续学习的渴望、意愿和能力对于你的成功至关重要。在现代技术领域中，有太多的技术知识需要掌握，任何人都不可能成为所有必需技能的真正专家。我喜欢将从事科技职业的人比作 RPG 游戏中的角色，他们并非通过击败敌人来提升等级，而是需要在工作中每周花费四十小时来积累技能点。你可以选择在哪些技能树上花费所积累的技能点，但必须明智选择。技能树的种类非常广泛，无法全部解锁，因此你必须专精于某个领域。

科技领域最美妙的一点是，我们的领域不断发展演变。你所共事的人会发生变化。你使用的工具会得到更新或过时，以及新的工作技巧会不断出现和消失。当你踏上技术领导的旅程时，处理这种变化的唯一方法就是预料到它、接受它，并拥抱与你的团队以及领域本身一起学习和成长的机会。

> 我希望人们对学习是认真的。我希望他们深入钻研。我希望他们能够最重要的是增长。这不是 RSA 加密，也不是复杂的算法；那些并不重要。重要的是他们对自己有信心，能够在学术界之外发展和学习：这意味着他们不需要我。他们所需要的只是能够静下心来读书，或者现在也许是上网，自己去学习事物。
>
> 麦克斯·明茨博士，1942-2022

初创技术领导者的困境 大多数初创公司都有一位技术联合创始人。这个人撰写了大部分最初的代码库，雇佣了最开始的几名工程师，并在至少在公司完成第一轮或第二轮融资之前负责技术方面的工作。

在雇佣第三名至第十名工程师之间的某个时候，这个人将停止亲自动手编码，开始花费大部分时间来管理团队。此时，常常会出现一些问题：团队开始开发特性的速度变慢，缺陷率开始上升，系统稳定性可能受到影响，整体成本上升，其他创始人开始担心。

技术联合创始人或任何技术领导者很可能在此之前的职业生涯中一直投入时间和精力成为一名优秀的程序员，而并没有发展领导能力。因此，毫不奇怪，他们在领导技能方面处于 1 级水平时会犯错，并花费公司的时间和金钱。

无论你的职称是什么，以及你在公司的加入时间，如果你将大部分职业生涯和经验投入到技术中，并且现在承担起负责人员或部门的责任，你必须意识到你现在是一个领导者；光凭你的技术背景和才能是不足以成功的。虽然在运营软件工程团队中一些技术技能是基本的要求，但实际上，要成为一名优秀的领导者，你需要重点关注人员领导力、管理、架构和普通决策能力。

人员领导力并非适合每个人。我相信你听过一些技术创始人在公司发展壮大后退居幕后的故事。Apple 的联合创始人史蒂夫·沃兹尼亚克（Steve Wozniak）是最著名的这种情况。放下身段并退居幕后并没有什么可耻的，沃兹尼亚克认识到他喜爱的是技术工作，这也是他想要花时间的地方。你也应该考虑自己是否要做出相同的决定：确定编程是否是你的天赋领域和最能给你带来快乐的工作。如果是的话，你将会有一个很好的职业生涯，跻身最高级别的技术人员之列。 如果您或您的环境使您得出结论，管理或领导团队是您渴望的角色，那么本手册将为您提供一个很好的起点，帮助您在成为成功的技术领导者的道路上扩展您的技能。

作者

我在大学一年级暑假期间有了我第一次创业经历。我不记得为什么要找 Eduware 实习，或者为什么他们接受了我的申请。我记得的是每天早上通勤到一个位于一层办公室后面的小房间里，和其他四名年轻软件工程师一起工作。我们当中最年长的可能也只有二十五岁，而我只有十九岁。就是我们五个人围坐在一个马蹄形桌子旁边，肩并肩地在一个.NET 教育应用上工作。我可能在工程师方面毫无用处，但我很幸运，那个团队里最老也是最资深的工程师花时间教我和帮助我理解工具，并且我逐渐变得更有生产力。

那个在办公室后面一个闷热房间的经历一定给我留下了很好的印象，因为此后我选择在另外七个创业公司工作：Invite Media，WiFast（现在是 Adentro），SoChat，AutoLotto，Trellis Technologies，GrowFlow 和 Equi。在 Invite Media，一家广告展示和交易竞标公司，我与首席技术官合作，领导了一个迅速增长的阶段，最终在 2010 年以 8100 万美元的价格被谷歌收购。在谷歌，我接管了 Invite 离职的首席技术官的站点可靠性责任，并监督了该公司与谷歌的融合。

然后我继续共同创办 WiFast，一家专注于民主化和货币化 Wi-Fi 使用的科技公司，并在第一和第二轮主要融资中担任首席执行官和首席技术官。我还曾担任腾讯在中国广州的企业家驻地，并共同创办了 SoChat，一款跨平台消息应用程序。此后，我曾担任 Lottery.com、Trellis Technologies、GrowFlow（被 Dama Financial 收购）和 Equi 的首席技术官。 我以创始人的心态对待每个角色，努力建立创造性的环境，并推动工程软件应该更多地偏向科学而非艺术的理念。

在这个过程中，我也有幸从他人身上学习，包括七个出色的工程团队、无数的咨询/辅导客户以及许多聪明的联合创始人。我还积极主动通过数年与硅谷顶级教练的管理辅导以及与无数导师的交流和阅读大量相关书籍来提升自己的领导能力。

通过阅读，我明确地感觉到，尽管有数百本针对程序员和使用特定技术或工具的人的实用书籍，以及几十本针对首席执行官和首席财务官关于企业金融方面的有益书籍，但我们行业缺少一本全面而实用的初创科技领导者资源。我们需要一本涵盖核心技能之外所有主题，并解决我们工作中至关重要的领导挑战和技能范围的资源。

此外，有很多博客介绍如何编写优秀的代码、如何进行用户调查，或者关于产品市场适应性的内容。而这本书是关于技术团队建设的，它涵盖了领导者在传统技术教育或经验中没有学到的所有技能。

使用本书

作为软件工程团队的领导者，您很可能在您的角色中遇到了以下一些问题：

追踪、管理或减少技术债务。

招聘、吸引、培养和保留顶尖人才。

建立一个客观、公正和透明的绩效评估体系。

建立、管理和维持一个健康和有创造力的公司文化。 导航您与公司其他领导之间的关系。 在技术人员中忍受缓慢的决策或无休止、循环的争论，关于如何规划和构建系统。

每个技术领导人似乎都会在某个时间面临这些问题，然而如何处理这些问题的建议却几乎被遗漏在几乎所有的商业或技术课程中。

我的目标是提供对这些问题以及更多问题的看法，同时提供各种技术在现实世界中的背景。目标是让读者了解权衡取舍，有一些预见未来的能力，并提供框架来准备您做出自己的合理决策。

本书主要是为那些目前或将来可能成为软件工程团队的管理者所写的，尤其是作为风险投资支持的初创企业的推动力量。对于没有管理职位的个人贡献者和软件工程师来说，它也可能对理解管理者所面临的任务和需求有所帮助，这些任务和需求一开始可能并不明显。 我将这本书格式化为独立章节的集

合，涵盖广泛的主题。它旨在用作参考指南，读者可以选择在需要时阅读某个章节，而不一定按顺序从头到尾阅读。因此，一些材料在不同章节中有所重复，以确保每个章节都能独立存在，没有先前章节作为背景的帮助。

每个章节的目标并不是对该主题进行详尽的讨论或评论。相反，目标是介绍主题，提供概述或思考该主题的结构，提供一些最佳实践，并建议参考资料以深入研究该主题。将这本书视为与技术领导力相关的广度优先主题集合。读者可以确定哪些主题对他们最有趣，并在了解一些背景和观点的基础上，深入研究与实践相关的最重要的知识。

归根结底，这本书是我个人经验和我发现有用的资源的综合，同时融入了同行、导师和顾问的建议和意见。如果您对本书中的某些事物持不同意见或认为有错误，并希望告诉我，或者如果您发现本书有帮助并希望直接与我沟通，请随时与我联系，邮箱为zach@ctohb.com。我也很愿意在同一地址讨论顾问、教练和指导的机会。

商业流程

在本书中，您将找到许多关于商业流程的描述。我在概述这些流程时的目标是为您提供一个开始解决面临问题的方案的起点。 根据您团队和公司的规模而定，这里所描述的可能会显得繁重且繁琐，或者可能显得过于简单和不成熟，无法满足您的需求。实际情况是，随着您的公司和团队的发展壮大，您需要重新发明经营方式。当您的公司从 5 人发展到 20 人、50 人、100 人或 1000 人时，它将会运营得截然不同。我已经强调了核心原则，现在将它们留给您，以便您根据现有团队的情况进行调整，并在未来根据您的业务需求和限制来扩展您的方法。

人员和文化

管理基本原则

推荐阅读：《管理人类》（Michael Lopp 著） 管理的黄金法则：尽一切可能激发团队的最佳表现。在技术领导和其他领导角色中一样，作为经理，衡量你绩效最好的标准就是团队的表现。这意味着你应该思考并花时间做一切必要的事情，帮助团队成员在个人和集体层面上做出最好的工作。

帮助团队取得成功需要谦卑，因为这意味着你要始终把直接下属的需求置于自己之上。你需要调整你的风格、行为、思维和行动，以适应工程团队成员的需求。这将包括愿意犯错、开放心态，并从直接下属身上学习。

如果你加入这个旅程，要知道你会犯错误。向团队坦诚这些错误，他们将更加信任你。同时也要明白，成为完美的经理是不可能实现的目标；你最好的希望是通过不断的小改进来提升自己。经过执掌人员管理的职业生涯，你将学习到关于技术和人类的无数经验，使你成为一个更有能力的经理。

迈克尔·洛普在《管理人类：一个软件工程经理的幽默故事》一书中写道：

与你一起工作的每个人都有各种不同的需求。满足这些需求是让他们感到满足和有生产力的一种方式。你的全职工作就是倾听这些人，并在心里记录下他们的特点。这是你最重要的工作。我知道高级副总裁对你说项目的截止日期是第一要务，但你不会编写代码、测试产品或记录功能。这些事情将由团队来完成，而你的工作是管理这个团队。 在这个简洁的段落中，洛普涵盖了管理的所有关键要点。首先，你是一个倾听者，一个个人和职业发展教练，一个能够防止外界干扰、压力或其他情况阻碍你的团队发挥最佳水平的屏障。

专业技能树

许多视频游戏都涉及到技能树的概念。对于那些不熟悉的人来说，技能树是一个随着玩家在游戏中进展而解锁的技能或能力的序列。每个技能都需要花费技能点解锁。问题是：在任何给定的时间，有更多的技能需要解锁，而你只拥有有限的技能点来投入。技能树迫使你在其他技能之前选择某些技能。技能树同样适用于你的职业发展。在任何一份工作中，你可能会积累技能点用于某些技能而不是其他技能。

在你成为技术领导者的道路上，你已经在技术/工程分支上投入了许多技能点。我对你的关键见解是，管理分支同样广阔，如果你到现在还没有在这个领域投入技能点，即使你是一名 100 级的工程师，你在担任新的领导职位时也将以一个 1 级经理的身份面对一棵巨大的尚未解锁的关键技能的橡树。一旦你的公司拥有超过一小撮的工程师，这些技能将决定你是否能够与团队一起实现规模化增长。

改进：持续不断的进步

*改善*是日语中的一个词，意思是改进。该短语在丰田生产系统中得到了广泛推广。在丰田，所有员工都被赋予了一个（字面或比喻性的）红手柄，用于停止整个生产线。如果一名工人发现生产中的问题，他们的任务是拉动红手柄，召集同事和资源来诊断问题，然后在工作继续之前解决它。通过赋权团队中的每个人来改善流程并使其投入有效性，丰田可以以经济高效的方式打造更高质量的汽车。

我并不是第一个认为软件工程与传统制造业有很多共同之处的人（参见 Gene Kim 的《凤凰计划》）。在这种情况下，将隐喻变为现实：为团队提供一个数字化的红手柄，并鼓励他们专注于不断改进所有工作。优秀团队的成员明白，随着时间的推移，团队将变化，客户需求将变化，工具将变化，团队将需要重新审视过去的决策并进行改进。

*改善*不仅适用于团队的流程，也适用于个人。你最优秀的团队成员将接受不断学习和不断改进的理念，并将错误视为改进的机会，而不是失败。

辅导

作为经理，你的主要角色是发挥团队成员的最大潜力，因此在许多情况下，将你的角色描述为教练而不是经理更为恰当。教练是支持你的人，是团队中每个人的智慧和指导来源。教练迅速提供关键反馈，但也是第一个庆祝和赞美成功的人。 你与你直接汇报对象的互动目标，无论是个体贡献工程师还是经理本身，就是成为他们曾经拥有的最好的教练。

寻找管理导师

在你作为领导者进行过渡、指导和管理他人时，一种快速启动的方法，就是找一个管理导师，而不是通过试错来学习。市面上有很多不同方法的管理教练，挑战在于找到一个与你共鸣的教练。

在我第一次担任 WiFast（当时叫 Zenreach，现在叫 Adentro）的业务领导者角色时，我们很快招聘了一个由十名全职员工组成的团队，其中大多数是工程师。作为一名首次担任经理的人，我知道我有很多东西要学习，我渴望利用一切资源成为一个更好的经理。唯一的问题是，我讨厌大多数关于管理的建议。我发现它们要么过于教条（先做 X，然后做 Y，最后做 Z），没有上下文或洞见，要么完全没有实质内容，有点像浮华的废话。直到我遇到了我的第一个管理教练乔纳森之前。

故事是这样的，我们的一个投资人 First Round Capital 在旧金山举办了一次管理高峰论坛，离我们办公室只有约 30 分钟车程。直到那时，我一直认为 First Round 的人员素质很高，所以当我看到邀请时，他们的支持暂时缓解了我对废话的过敏反应，于是我报名参加了。 当我驱车到达山顶时，我对观众的人数感到鼓舞，只有大约 30 个人，足够容纳一个感觉像高中教室的人数。我坐在高中用折叠式桌面的桌子前，打开笔记本，拿出钢笔，在页面顶部写下日期和 First Round Capital Management Summit。遗憾的是，那将是接下来四个小时里我唯一写的东西。

上半天有三到四位演讲者谈论各种话题，所有这些话题都缺乏可行的建议或深入的见解。午餐休息的时候，我在考虑是否早点返回办公室，工作半天。我查看了议程并注意到下午的演讲者完全不同，所以我决定至少听完第一个人的发言。

午餐后的第一个发言人是乔纳森。与之前的演讲者不同，他没有幻灯片，走上教室前似乎有点匆忙，或许稍微准备不足，或者只是紧张。然而，他嘴里说出的第一句话却讲了一个不同的故事：

让我透明地操纵你们。

我永远不会忘记那一刻。这样说真是有趣；这是一个看似自相矛盾的说法，就像说“这句话是假的”。（如果你感兴趣，这被称为说谎者的悖论）。他接着解释说这正是他想要达到的效果：他想说些能够吸引我们作为观众注意力的话，这一点他完美地做到了。接下来的三十分钟里，我做了大量笔记，不是关于操纵人，而是关于了解人的一般性质。 我对乔纳森说的每一个字都非常关注。在半小时的会议结束时，乔纳森说他要赶飞机，匆忙地走出房间。我低头看着笔记本，意识到我在过去的三十分钟里记了三页笔记，然后站起身跟着他跑出去。

我设法赶上他，恰好看到他上了一辆黄色的出租车。有些恼怒，乔纳森问我想要什么。我问他是否提供私人辅导。他回答说，让峰会组织者联系我们。聪明的他没有当场答应或拒绝辅导，还给自己留了个机会，通过 First Round 来对我进行尽职调查，再决定我是否值得他花时间。幸运的是，当我请求 First Round 让我们联系时，联系人对我说了一些好话，乔纳森同意进行一次介绍性的辅导会议。

一对一会议

1:1 会议是你与直接下属之间的私人会议。人们常常把 1:1 会议视为状态检查会议，并且议程完全集中在手头的业务或技术主题上。如果议程包括这些主题也是可以的，但这是你与直接下属建立辅导关系的机会。你应该利用这个时间真正了解和理解你的下属的思维方式，发现并确定他们的优点，并认识到可以解决的弱点，帮助他们发挥最佳水平。

跳级会议

定期（每月或每季度）与直接报告给你的经理们的下属举行会议是一个很好的实践。这些会议被称为跳级会议，因为你直接与他们见面，跳过了组织图上的一个级别。实际上，你并不是试图通过跳级会议削弱你的经理们，相反你是想通过收集更多数据和听取不同的观点，更好地与他们合作，以改善业务。

一些跳级会议议程的快速思路：

- 确保员工知道会议的目的，让他们感到放心，你不是为了解决问题或做出更好由他们真实经理处理的决策而在那里。
- 告诉他们你想建立关系，并听取他们对领导力、文化、战略和公司发展方向的见解。
- 与员工建立联系；提问并怀有好奇心。 以下是一些从 managementcenter.org 推荐的互联网上有许多好的现实模板/议程可供跳级使用： ctohb.com/skip。

辅导经理

随着你的组织的发展，你很可能会达到这样一个点：你不再拥有任何直接报告的个人贡献者。每个真正编写代码的直接贡献者都由中层管理人员管理。因此，很明显，高效的中层管理人员对于你的组织的绩效非常重要。你的工作就是确保你的经理们得到支持、资源、培训和指导，使他们能够最好地辅导他们团队中的工程师。

当然，培养高质量的中层管理人员的最大贡献因素是招聘正确的人才，但其次是持续的培训和支持。如果你可以监督一个管理团队，我鼓励你在你的组织中建立以下内容：

- 建立持续学习的文化。 例如，鼓励您的经理们建立一个内部以管理为重点的读书俱乐部。 定期与管理团队共享您自己正在学习的见解，并要求他们与他们的团队做同样的事情。如果您正在使用一个公司聊天工具，则可以在一个专门的频道（如 #管理见解）中进行这种对话非常好。

确立一个较高的辅导和管理标准。 明确告诉您的经理对管理的期望，包括对辅导、1:1 会议、绩效管理等的期望。 将您的管理期望明确地编纂在内部文档中，并将其作为管理聘用和入职的一部分。

提供详细且易于获得的管理培训材料。

- 为您的经理提供资源，以进行持续学习和职业发展。这可能包括购买公司订阅学习计划、赞助员工参加会议、聘请管理教练或制定内部或外部的导师计划。
- 在您每个团队成员的常规预算过程中考虑这些培训材料的费用。
- 构建一个面向外部的思想领袖文化。
- 鼓励您的经理在您的行业成为思想领袖。这可以通过公司的形式，参与技术或管理播客节目，或在会议上发表演讲来实现。

与工程师的 1:1 会议

您的工程师应该经常向您倾诉，所以如果他们这样做了，请不要惊慌，这是完全正常的，实际上非常可取的。您应该每两周至少与您团队的每个成员举行一次 1:1 会议，如果可能的话，每周一次。您在这些会议中的目标是为您的工程师创造一个安全的空间，让他们告诉您他们心里想的事情，而您要积极倾听并参与这些话题。

对于有实力的工程师来说，这意味着他们意识到周围世界的不完美之处，并希望向您提出意见。您的任务不是解决他们提出的每个问题；您的任务是倾听，用问题澄清自己的理解，并说服他们您确实理解，然后引导他们寻找解决方案。偶尔会有直接的要求或者您可以直接帮助的事情，但这不是常态。

您在这里提供的价值是让您的直接下属感到被倾听，并教导他们有效地处理问题。

1:1 会议内容和议程

在 1:1 会议中，您的最终目标是与他人建立关系，并进行脆弱和关键的对话，从而帮助他们做出最好的工作。如果您的直接下属有一个广泛的议程，那很好，就从那里开始。然而，如果他们的议程一直局限于战术性的进行中的工作事项，而您没有进行更高层次的关于我们如何工作的对话，那么我鼓励您补充他们的议程，以便他们更好地理解您会议的目的，并在未来的会议中提出更多实质性的问题。 桥接你的议程和他们的最简单的方法是有一个共享文档，可能带有一些结构/模板，以引出你认为重要的讨论主题。在会议之前提供这个文档，也可以使你和员工在会议之间有一个共享的地方来记录想法，在会议之前组织思路，这些都有助于使会议时间更加高效和有效。

还有一些 SaaS 工具可以帮助促进一对一的对话。值得注意的例子包括 Culture Amp 和 15Five。但你并不需要一个工具；一个简单的文档同样有效。我使用的模板可以在 ctohb.com/templates 上找到；它包括讨论个人、部门和公司层面的喜欢/期望的项目的提示，以及经理和员工之间的双向反馈。

1:1 游戏规则

为这些工程师 1:1 会议建立游戏规则是确保这些会议涵盖一致的议题，不偏离主题的另一种有用方法。你的游戏规则应该确保你的 1:1 会议涉及以下内容：

冲突：你的团队内部，工程团队之间，跨职能部门之间的冲突。

绩效与发展:通常是你的工程师寻求建议，关于如何改善某些方面。

清晰度:工程师可能对某些事情有一些一般想法，正在寻求你的观点，或者想知道你是否对某些事情有不同的信息。

背景:公司整体的情况，以及贡献者的工作与公司目标/目标的关系。

激进坦诚

"激进坦诚"一词由金·斯科特在她的书《激进坦诚》中定义。该书将激进坦诚定义为同时包含赞美和批评的沟通方式，并确保在传递时既关心个人又直接挑战。我认为这一观点最好通过与斯科特书中概述的其他三种沟通方式进行对比来阐述。

令人讨厌的攻击性：有时被称为残酷的诚实或正面刺伤，以直接挑战但缺乏个人关怀为特征，可能表现为虚伪的赞美或不友善的批评。

毁灭性的同情心：以个人关怀为出发点但缺乏直接挑战的沟通方式。

操纵性的虚伪：也被称为背后捅刀子或被动攻击行为，既不在乎个人情感也不直接挑战。

我鼓励你阅读斯科特的书，但如果你不愿意，至少要了解这些术语，并将它们作为一种辅导工具，帮助你的团队定期实践激进坦诚。

过度沟通的好处

没有什么比雇员感觉经理和他们沟通不足更糟糕的了。缺乏信息时，人们往往会本能地假设最糟糕的情况；缺乏信息也是焦虑和困惑的主要来源。

相比之下，过度沟通几乎没有什么后果。对于过度沟通，最糟糕的评价可能是会干扰或变得多余，但这些问题可以通过考虑过度沟通的形式来很容易地解决。因此，毫不奇怪，大多数初创公司都大力投资于在他们的文化中建立过度沟通的环境，通常将其作为公司的核心价值观之一。

电子邮件

现在几乎与任何人的交往都离不开电子邮件，不管是使用了二十五年还是从小学开始使用，所以他们肯定知道如何有效地使用电子邮件，对吧？然而，有效地在工作中使用电子邮件并非是常识。因此，你需要帮助推动最佳实践。以下是一些关于有效使用电子邮件的一般建议：

- 不要让电子邮件成为你的工作。
- 不要整天开着邮箱或持续监控它，每天固定时间查看邮箱。
- 关闭手机上的邮件通知。尽管这个做法可能听起来令人难以接受，但我鼓励你尝试一下。这不仅会显著减少你收到的通知数量，而且你会发现自己正在养成一个主动检查邮箱的新习惯。这样，邮箱就成为一个有意识的活动，而不是持续存在的干扰。
- 每天将收件箱清零。
- 投入时间来学习你的邮箱工具，或者使用可选的邮箱助手插件，帮助分类和筛选邮件，这样，每天结束时，你的未读邮件数量将为零。
- 清空收件箱并不意味着要对每封邮件做出反应或回复。如果你将邮件用作待办事项清单，那也可以（尽管这并不理想，请参考《会议和时间管理》第 28 页，了解更好的待办事项替代方法）；只需确保将待办事项清单从核心收件箱中筛选出去，这样就不会将其与未筛选的邮件混淆。
- 不要在邮件中解决问题。 电子邮件是一个不理想的媒介，特别是在涉及超过两个人的深入讨论时。群发邮件最适合用于协调和过度沟通，而不是解决问题。 理解电子邮件往往缺乏细微差别和语气，这使得意图容易被误解。 若想参与或撰写一份细致入微的群发邮件讨论，这表明以同步对话的形式来解决问题更合适。一个 15 分钟的讨论往往可以解决一个包含 20 封邮件的讨论只触及皮毛的问题。 写下自己的思考过程通常是一个非常有效的练习，但是电子邮件并不是促进和捕捉这种书面头脑风暴的好方法。鼓励团队使用维基百科来撰写备忘录，以促进深度思考。

不要过分依赖电子邮件进行长时间或深入的沟通。 总的来说，电子邮件不适合长篇内容的传递。长篇备忘录最好放在内部维基或文档中，以便进行评论、更新，并且可以在将来轻松引用。

尽量保持电子邮件简短，最好使用项目符号列出关键观点。 请不要犹豫使用基本格式，如加粗或高亮显示请求/行动项。

请考虑你的受众。

- 一般来说，工程师更喜欢编写代码而不是阅读/回复电子邮件。问问自己，电子邮件是否真的是与你的受众沟通的正确方式。总的来说，与某人的沟通最好是使用*他们*偏爱的方式，而不是你自己的方式。
- 很容易在电子邮件中忽略同事，无论是故意为了不淹没收件箱，还是无心之失。如果你正在考虑哪些人应该被添加到/移出电子邮件线程，那可能是电子邮件不适合的一个征兆。

同步聊天

很有可能你的公司已经采用了某种形式的同步聊天平台；在 2000 年代初，通常是谷歌聊天或者 MSN 通讯工具，而在 2020 年代，更常见的是 Slack、微软 Teams 或者 Meta 的 Workspace。如果你现在还没有使用这些平台之一，考虑采用它们是值得的。

从初创公司到拥有十万人以上的巨头公司，绝大多数公司都采用了这些平台，并取得了巨大的成功。

要实现这种成功，就要在注意力上下功夫，并针对它们固有的缺点进行规划：同步聊天程序要求双方停下手中的工作并参与，对话内容组织混乱，并不能产生长期可供团队参考的文档。你可以并且应该意识到这些缺点，并通过为团队设置基本礼仪和使用这些工具的预期来弥补它们。

Slack 官方博客包含一篇介绍常见最佳实践的优秀文章，网址是 ctohb.com/slack。

以下是在使用同步聊天工具时的几个建议： 请尽量在消息中包含所有必要的信息以继续对话。如果您向同事提问，请在问题中提供足够的背景和信息，以使他们有最好的机会全面回答。这样做可以减少发送的通知数量，减少来回沟通的次数，缩短解决问题的时间。像 loom.com 这样的工具非常有帮助。

使用消息格式化功能，如项目符号和标题，使较长的消息更容易浏览，相关信息更易于找到。

将对话集中在特定的频道或线程中。尝试同时跟踪多个人在多个话题上的对话是低效和令人沮丧的。

充分利用通知时间表和免打扰功能。您还应该鼓励团队成员在任何同步聊天程序中设置免打扰时间表，以减少对集中/流动时间的打扰。

本着过度沟通的精神，将默认通信设置为公共频道。更好的做法是，建立一种文化和标准操作程序，将对话和相关决策转化为长期存在且有组织的公司维基或其他适当的文档/信息存储中。

谨慎使用向多个人发送通知的消息，例如在 Slack 中使用@here 或@channel。特别是当您的公司发展壮大时，发送此类消息可能会打扰并打断数十名员工。

异步通信

异步通信是指不立即获取回应的任何通信方式。为了有效，接收方应该有足够的时间来处理信息，然后进行深思熟虑地回复。异步通信的一个关键要素是初始消息是一个完整的思想，并包含足够的上下文，让对方能够做出回应。

一个陈词滥调的例子是可怕的功能损坏的错误报告。几乎所有情况下，错误报告应该发送到一个票务系统而不是直接发送消息。工程师在消息中收到错误报告时没有上下文，以知道哪个功能已损坏或以何种方式未能满足期望。因此，工程师的回复很可能包含一些问题，需要与报告人进行更多的往返交流，耗费时间并引发沮丧。

相比之下，一个错误报告包含完整的书面复制步骤以及一个用户尝试使用该功能并演示失败的视频。很可能，这种方法将使工程师能够在不需要进行进一步跟进的情况下进行修复。 这是底线：无论何时您以异步格式向某人发送消息，请提供给该人所有必要的信息，以便他们能够理解、处理和回复，以推进对话。

异步文化

您会惊讶于经过深思熟虑的异步沟通有多经常地取代了同步的聊天或会议。良好的异步沟通不仅意味着更少的会议和干扰，还可以为他人留下全面的书面文档以供将来处理。一些初创公司，例如 Levels Health，实际上已将默认异步作为其核心文化的一部分，并取得了巨大的效果（ctohb.com/async）。

文档

文档是扩大组织规模的关键要素。写下事情的好处有很多：书面文档可以帮助入职、培训、过度沟通、深思熟虑、彻底性、建立文化、避免不必要的错误等。您的角色不仅仅是相信文档的价值和投资回报，还要建立文档文化和重视文档的团队。 建立良好文档文化的一些建议：

- 自己践行价值观，并为团队树立榜样。我曾经带领一个团队，在大约八周的时间里将每周零篇公司内部维基文章的数量增加到了每天数篇。促进这种文化变革的唯一举措，就是开始自己撰写文章。我将所有有意义与团队分享的事情都写成文章，并在适当的时候分享文章的链接。很快，其他经理也开始效仿，两个月后，团队中的每个人都开始每周做出贡献。

- 在流程中加入编写和阅读文档。不论是用于新员工培训、技术规范、代码审查、内部评论请求(RFCs)还是备忘录等，标准操作都应该是将其写下来并保存在整理有序的档案库中，方便随时查阅。

- 在适当场合制定维护文档的流程。文档很容易过时，而在许多情况下，这是可以接受的。但在其他情况下，文档保持更新至关重要，唯一的方法就是建立一个包括更新文档在内的流程或检查清单。在每个文档上附上最后更新日期是向读者表明其是否最新、已废弃、过时还是失效的好方法。

- 鼓励团队遵守童子军规则（随手保持营地/代码的整洁优于你找到的状态）。如果他们发现文档有错误，应该自己进行更新，或明确将文档标记为已废弃。

(Note: The translation provided is a simplified Chinese translation of the given content.) 文档的一个关键领域是你应特别注意的，即开发人员如何在特定项目或存储库中开始编写代码。我建议每个存储库都有一个 README.md 文件，其中解释了至少以下四个方面：

安装： 如何将应用程序安装并在本地运行 **目录结构：** 如何在该代码库中找到所需的位置 **开发：** 在该代码库上的开发/运行/测试循环是什么样的 **部署：** 如何将更改应用到较高的环境中

关于工作中的首字母缩略词

每个组织都有自己独特的文化，并且有着内部和外部交流的风格。作为一个领导者的关键责任之一就是确保文化始终支持组织的目标，而不是阻碍它们。

一个技术组织内部文化中容易失控的一个因素就是虚构的缩略词的产生，随着时间推移它们会不断增多、使沟通变得模糊和过于复杂化。这可能看起来只是一个小烦恼，但它是一个不良沟通策略的症状，可能会失控，尤其当它会在那些熟悉情况的人和不知道这些缩略词意味着什么的团队成员之间造成障碍时。作为一个组织的技术领导者，你的工作是设定基调和定义文化，虽然虚构缩略词的蔓延可能不是从你这里开始的，但你的工作是在事情变得失控之前认识到它，并制止它。

在 2018 年 1 月给 SpaceX 员工的备忘录中，埃隆·马斯克呼吁禁止使用缩略词。我从那时起就开始实行同样的政策，并十分赞同。以下内容来自一封名为《缩略词真的很糟糕》（ctohb.com/acronyms）的电子邮件：

> SpaceX 越来越倾向于使用虚构缩略词。过度使用虚构缩略词严重阻碍了沟通，而良好的沟通对我们不断发展至关重要。单个的缩略词可能看起来并不那么糟糕，但如果有一千个人在瞎编缩略词，经过一段时间，我们将不得不向新员工发放一个庞大的词汇表。[...] 这对于新员工来说尤其困难。[...] 判断一个缩略词是否有助于沟通的关键测试是询问它是有益还是有害。像 GUI 这样大部分太空探索技术以外的工程师已经知道的缩略词是可以使用的。实际上，大部分缩略词作为一个障碍而不是增进清晰沟通的好处。它让新员工更难理解正在讨论的内容。团队需要付出努力来维护一个缩略词定义的列表，总体而言，写和说起来都不像乍看起来那么节省时间。

这可能听起来很亵渎，或者是一项过分和愚蠢的规定试图在一种文化中加以实施。我不是提议你惩罚那些使用缩略词的人，或者在办公室的走廊墙壁上写下这个规定。恰恰相反，在一个规模较小的组织中，只需轻轻一举就可以将不使用新缩略词作为文化的一部分。获得高层团队的支持，不要创造缩略词，并鼓励他们向自己的经理发出温和的提醒，要求他们做同样的事情，你会惊讶地发现每个人很快就能改掉这个习惯。在你的入职文档中加上一两句话通常就足够为新员工提供一个提示，由于入职中的温和提示以及周围缺乏缩略词的见证，他们将不太可能自己创造缩略词。 会议和时间管理

广义上说，有三种类型的会议：定期安排的信息会议、冲突解决会议和临时/即时会议。作为经理，您的工作是根据会议类型设定与参与人员的期望。对于信息会议，要问自己是否开会是传达信息的最佳方式；有时是，但并非总是如此。如果是的话，确保以多种方式传达信息，例如提前在公司的维基中提供书面资料。如果是冲突解决会议，请确保提前确定讨论要点，以便与会者能够准备充分讨论和解决问题。临时会议通常在开始之前已经有明确定义的目的，无需进一步介绍。

无论会议类型如何，您设置的任何会议都应该有一个明确的目标，参会人员事先知道这个目标。理想情况下，每个人在会议之前都应该有足够的信息，以判断对他们来说是否有价值参加。同样重要的是，您的企业文化应该赋予员工权力，使他们能够决定不参加那些他们认为不是对时间的有效利用的会议。

时间管理 身为初创公司的领导者，你会很快发现自己的时间被分配给许多不同的工作，并且每周可能有数十个小时的会议时间。如果你还没有找到适合自己的工作系统，现在是投资一些良好习惯和组织的时候了。我推荐斯蒂芬·柯维的《高效能人士的七个习惯》和大卫·艾伦的《完成事项的艺术》作为你开始这个旅程的起点。

会议时间安排

你可以通过创造或允许大块的自由时间给工程师们来提高团队的生产效率。上下文切换（我们从一个不相关的任务转到另一个任务的倾向）是有代价的（详见 ctohb.com/myth 中的《多任务神话》），因此你为工程师们创造的不需要切换到其他任务（邮件、电话、会议）的工程工作时间越多，你所付出的上下文切换代价就越少。

我喜欢为团队宣布一个非正式的会议时间窗口。鼓励工程团队和跨职能团队在每天的两三个小时内安排会议，并尽量不安排工程师在这个时间窗口之外的会议。这样每天都会有足够的时间供你的工程团队专注于核心工作，并为必要的信息交流和冲突解决会议腾出空间。如果你的团队每天的会议时间超过三小时（每周十五小时，几乎占据了他们一半的时间！），你应该仔细审查这些会议，并问问自己是否可以合并和减少它们。

其他团队通过设置无会议日来取得成功，即每周的某一天或多天不安排任何重复性会议。但请记住，在一个每周工作四十小时的时间里，你的目标是尽可能保留更多的连续集中时间给你的工程团队。一个无会议的工作日意味着有八个小时的集中工作时间，但还有另外三十二个小时需要考虑，所以它并不能解决所有的问题。

工程师的时间建议

考虑一个软件工程师的假设性一周：

- 一小时：与经理进行 一对一会议
- 两个半小时：每日三十分钟的站立会议
- 两小时：平均花费在其他敏捷仪式（迭代计划、回顾等）的时间。
- 四到八个小时：审查他人的代码 四个小时：电子邮件/聊天沟通

总之，每周将花费十三到十七个小时进行会议和沟通。如果再加上几个小时用于切换任务和意外打断，你会发现在一周工作时间（四十小时）中，实际能够专注的时间最多只有一半。如果不仔细安排会议时间，工程师们不仅会只剩下二十个小时进行核心任务，而且这些时间也不是连续的，进一步降低了工作效率。

我提出这个虚构的例子是为了强调，为工程师提供较长时间的专注工作并不是偶然发生的。作为领导者，你决定他们如何利用时间，需要发展一种文化和流程，以整合和最小化这些干扰，并最大程度地提供给个人贡献者进行实际的工程工作的时间。

HIPPO

HIPPO 是一个非正式的行业术语，缩写为最高薪酬人士的意见。无论你是否为最高薪酬人士，你的职位会暗示你是，大多数员工不愿挑战 HIPPO。我强烈建议你在讨论中尽量减少这种影响，经常敞开大

门接受挑战，明确表示可能错误，并采纳和支持不同于自己的观点和想法。当你觉得你经常这样做时，就意味着你已经到达了大多数员工实际上需要的最低要求。 尽管你尽力显得平易近人并对其他方法持开放态度，但你在会议中的出现往往仍会对其他出席者产生潜意识的影响，特别是如果他们在组织架构中比你低一级或更低。要注意这种影响，并尽力只在团队确实需要你参加并能为其增加价值的会议上出席。对于其他事项，你可以在会议结束后查看会议记录/录音。

任务清单

总体而言，任务清单是一种非常简单的任务管理形式，缺乏结构、优先级或时间因素。我建议使用基于日历的任务清单。不要将工作事项放在一个通用列表中，而是将它们放在你实际使用的日历中。

这有几个优点。它可以为你实际完成任务的时间段设置专属时间，确保你不会过度承诺时间。它可以通过移动事项来进行优先级排序，也更容易预测何时完成任务。大多数日历系统还具有内置的提醒机制，将在你计划执行特定任务时通知你。

日历回顾和时间平衡

不时地说，每个月，我鼓励你对你的日历进行历史回顾，并衡量你如何花费时间。例如，Google 日历内置了分析功能，只需要非常少的调整你的日程习惯，就可以提供关于时间花费的准确摘要。在回顾这些数据时，问问自己在不同类型活动上花费时间的比例是否符合你想要实现的目标。同时，确保你花时间的方式符合你的优势，并带给你个人满足感也是很重要的。通常，仅仅是简单呈现这种数据就能够有效地激发你的组织和积极变化的动力。

迷你管理框架

作为经理，你会遇到很多相同的、重复的问题。拥有一个思维框架来解决问题可以加快决策过程，提高决策质量，并为解释决策提供上下文和观点。以下是我在管理道路上发现有用的一些框架。

管理问题解决的三个阶段

当面临一个庞大而模糊的挑战时，比如接管一个新团队的管理，或者诊断和改进个人的表现不佳，我喜欢使用一个三步骤的流程。这些步骤应该按顺序完成，以制定解决特定问题的计划。

1. 吸收

- 尽可能多地获取信息。阅读可用的文件，无论是维基文章、绩效评估、代码，或者与问题相关的任何内容。安排一对一会议，运用积极倾听的技巧，并对所发现的情况做详细记录。
- 当你开始多次看到相同的事情，并且不再看到新的模式时，你就知道自己已经吸收了足够的数据。
- 例子：几个人都对你团队的某个成员的表现发表评论，经过几次积极倾听和好奇心的激发，你已经不再获得更多关于绩效问题的新信息。

2. 综合

- 一旦你收集了与你的问题相关的足够多的数据，暂停收集信息，并给你的大脑一些时间来处理。我建议在这个阶段至少花几天时间。试着有意识地停止获取新信息，而是花时间从不同角度看问

题。做笔记，画图，打高尔夫球，洗个淋浴，或者任何能够帮助你思考问题并得出与数据相符且可行的分析的方法。

- 继续以上的例子，在这一点上，你可以尝试提出不同的假设，解释为什么那个人表现不佳：他们是如何花时间的？是技能不匹配，期望不匹配，还是他们的个人生活出了问题等等？

3. 行动

- 一旦你有了一个论点，就是时候真正实施计划了。在采取行动时，验证你的计划是否达到了预期的结果非常重要。尽可能地进行测试、验证，如果需要的话，重新开始循环。

基于团队的决策模型

有三种模型用于与团队一起制定决策的材料。作为经理，你可以完全自己制作材料和决策，并将结果作为*既成事实*呈现给团队。也可以采取相反的方法：从零开始，与团队中的一些或全部人员完全共同开发材料。第三种方法是前两种方法的折中：自己起草并将其作为一个草稿呈现给团队，作为开始收集反馈并迭代以达到最终版本的起点。这些技术之间的主要区别在于它们所需的时间以及团队的购买情况。我鼓励你优先考虑购买：确保团队中的每个人都理解决策并能够成为决策的支持者，这是确保大家朝着同一个方向前进的唯一方法。正如硅谷产品集团的马蒂·卡根所说，你希望你的团队表现得像传教士而不是雇佣兵。

独立模型： 作为经理独立开发需要的总时间最少，但也产生的购买程度最低。

草稿模型： 从草稿开始共同制定决策需要中等时间，并且根据执行方式可以产生健康的购买程度。

共同开发模式：从零开始共同开发可能需要很长时间，但可以获得最多参与开发的支持。

任何决策所使用的模式取决于您，我鼓励您在做出选择时要思考和仔细考虑。如果您觉得选择了错误的模式，不要害怕重新考虑。

两种决策类型

在 1997 年给股东的年度信中，杰夫·贝佐斯提出了一种将决策分类为类型 1 和类型 2 的框架。贝佐斯写道，类型 1 的决策是不可逆转的，应该经过系统、谨慎、缓慢的思考，并经过深入的协商。例如，您选择使用哪种编程语言是一种类型 1 的决策。

类型 2 的决策则相反。它们可以并且应该由具有高判断能力的个人或小团队迅速作出。例如，按钮使用的确切灰色是一种类型 2 的决策，因为它可以在后续被轻松更改。 贝索斯的建议是，我在这里重申一下，即使用第一类决策思维处理第二类决策会导致缓慢和失败，无法进行实验和创新。

大部分日常技术决策都属于第二类，最好快速做出决策，并在收集更多数据（通过原型或 MVP 实施）后重新考虑或确认。这是因为大部分初创公司技术决策中最昂贵的部分是工程团队在解决方案上投入的时间。如果你有意地限制验证可逆决策所需的时间（和成本），则只会损失一小部分。大部分第二类技术决策只有在你在其上投入了大量时间和新工程之后才会变得不可逆，所以要在早期严格评估进展。如果有疑问，尽早做出改变的决策。

解决冲突

作为团队的领导者，最终你要对团队的目标负责。如果整个团队无法达到里程碑，那就是你的责任。因此，当团队内部发生冲突或意见不合时，你需要审慎参与，并准备好根据以下三个故事之一做出决策：

1. 我们将按照你的方式进行，因为你提出了明确且令人信服的论点证明它更优秀。
2. 我们会按照我的方式进行，因为它更优越，并且我将利用作为一个具有更广泛责任范围的经理所拥有的附加背景来解释为什么。
3. 我们会按照我的方式进行，因为我们无法找到任何客观原因来判断一种方式比另一种更好。换句话说，结果是平局，由于最终我是在这里负责成功的一方，我们将采用我的方法。我将对这个决策的成功或失败负责。

任务分类 紧急/重要矩阵

在《高效能人士的七个习惯》一书中，史蒂芬·柯维博士引用了紧急/重要矩阵的概念，该概念改编自 1955 年德怀特·艾森豪威尔总统在一次演讲中提出的观点。在紧急/重要的二分法中，工作根据其紧迫性（即时间敏感性）和重要性（即影响力）进行分类。结果是一个四象限图表： 我在这里提供这个框架，以提醒大家要考虑到各种任务的价值。技术领导经常被功能请求、优先级债务、缺陷等问题所困扰，能够有这种观点来问任何一个项目是否重要和/或紧急，是一种非常有用且快速的诊断机制。

人员拼图解决者

作为一名管理者，你在工作中将主要扮演人员拼图解决者的角色。你需要想办法让两个人能够有效地合作，指导某人改进技能，或者设计一个能够促进协作的团队结构。人们的行为很难预测，有时候团队成员可能会说一件事情，但思考或感受又完全不同。

鉴于这种不断变化的复杂性，我鼓励你像侦探或科学家一样思考：时刻收集数据。形成一个假设，如果你采取某种行动，可能会发生什么，然后采取行动，并收集结果的数据。有意为之，这将鼓励你倾听，观察人们在各种情况下如何回应。这将为你下次与同样的人在类似情境中相处提供有用的数据。

加入一个团队

成为技术领导者有两种方式：一种是从非领导地位开始加入团队，并逐步成长或晋升为领导角色；另一种是被聘用为团队的领导者。如果你是通过晋升而成为领导者，可以推测你已经具备良好的业务背景和技术能力，但在管理和领导方面尚未证明自己。相反，如果你是被雇用为领导者，你可能在管理方面有一定的履历，但缺乏对组织的业务、技术和人员的上下文、历史和背景了解。因此，对应不同的弱点，你在开始角色时的方法和策略应有所不同。

晋升为管理/领导职位

如果你被晋升为管理职位并且已经花时间培养了管理技能，或者你有管理经验和良好的履历，那么你可能不会感到这个转变非常可怕，你的目标将是继续成长为一名更高级的管理者。然而，如果这是你的第一个管理职位，你将面临更大的挑战。

人员管理是与获得晋升的技术技能完全不同的技能集合。当然，技术技能是成为一名出色的技术经理的必备条件，但远远不够。理解这一点，并发展你新角色所需的额外技能集，将是作为一名管理者成功的关键。

如果你是从编写 C 语言代码的后端工程师晋升为编写 TypeScript 代码的前端工程师，你会做哪些事情？你可能会阅读一本关于 TypeScript 的书籍，做一些 TypeScript 编程练习，加入 TypeScript 用户组，阅读一些 TypeScript 博客等等。 从编写 C 语言到管理人员实际上比从 C 语言到 TypeScript 更大的变化。成功地进行这种转变需要同等或更多层次的积极学习。对于大多数人来说，成为一个优秀的人员经理是一次终生的旅程，而不是一个周末就能掌握的技能。接受这个挑战，把这份工作当作一个终身学习的任务，从现在开始。

实用的管理技巧

牢记德国总理俾斯麦的格言“愚人通过经验学习；我更喜欢通过他人的经验学习”，这里有一些新管理者的可行技巧：

- 寻找一个管理导师或教练。通过与他人讨论问题并获得更多的观点是无价的。（见边栏“寻找一个管理导师”，第 13 页。）
- 学会委派任务。罗伯特·凯根的《免疫力》详细讨论了委派任务的障碍以及学会委派任务对经理成功的重要性。关键在于，你对组织的价值不再是个人的产出，而是团队的产出。对于许多技术领导者来说，这是转变中最困难的一部分，但也是最关键的一部分。
- 照顾好自己。
  - 管理他人可能会消耗大量情感。为了能够始终保持最佳状态，保持冷静、积极和高效，务必要照顾好自己。
  - 找出适合自己的方法；也许是冥想、打高尔夫、玩电子游戏或与家人共度时光。做任何让你感觉良好并能让你恢复并准备好面对下一个挑战的事情。
  - 注意疲劳的迹象，并在它出现之前休息一下。管理他人并不意味着每周工作八十个小时。

请查看本书末尾的推荐阅读列表，以获取更多发展管理技能的资源。

被雇用来领导

如果你被雇佣来领导一个团队（而不是晋升为领导职位或成为初创公司合伙人才担任领导角色），想必你既有技术经验又有管理经验。作为一个被雇佣来领导现有团队的领导者，你面临的挑战是尽可能顺利地融入新团队，并与新同事建立信任。毫不奇怪，我的建议是在管理新团队时更多地关注人员而不是技术。

以下是我为外部雇佣的技术领导者制定的一些短期目标，以及您应尽早回答的一些问题。

目标：

- 与技术团队建立信任。倾听，并且在增加价值或改变事物的时候要考虑周到。
- 与其他团队/领导建立信任，做出合理的承诺并兑现。
- 了解与你一起工作的人以及他们在公司/产品/技术方面的历史。

- 诊断团队内对人员产生最大影响的挑战。是否有员工的职位评级不合适，表现不佳或超出了他们的角色范围？是否有任何文化方面的问题需要进行修正？及早采取果断行动来纠正文化问题是与团队建立信任的一种很好的方式，因为团队成员很可能有意识或无意识地受到文化问题的困扰。
- 诊断团队整体上技术问题的最大影响区域，并为团队制定一组短期、中期和长期目标。

问题：

- 以前是谁负责技术工作？这个人还在团队中吗？
  - 技术领导者经常发现他们想要成长为人员管理者。在这种情况下，你将填补人员管理空缺。另一种常见情况是，要么之前没有技术领导者，要么他们因表现不佳而离职，而你将继承大量技术债务。
- CEO 说你被聘用来解决什么问题？你认为你被聘用来解决什么问题？
- 当前技术团队与公司其他部门之间存在哪些痛点？
- 技术团队中哪些痛点优先级最高？

对朋友/陌生人提供技术建议

在简历上有一段时间担任技术领导角色之后，你很可能会开始接到朋友或朋友的朋友寻求建议的请求。在大多数情况下，我建议接听这些电话，不仅因为人际网络很有价值，还因为你被问到的问题可能会迫使你思考并用言语表达你下意识正在努力思考的观点。人们说教授他人是真正学好某个知识的最佳方式。 请注意给非技术创始人的建议：你可能会不时地被某人以自称价值数十亿美元的点子找到。接听这些电话所需的代价很小，而且可以是建立社交和人际资本的好方式。然而，请注意，优秀的点子并不能自己成为成功的业务。在每个伟大的点子和成功之间，都有一个巨大的执行难题，而大多数人并没有准备好征服珠穆朗玛峰。所以在对一个有好点子但没有攀登装备的人做出承诺时要非常谨慎。

流量：漏斗和雨伞

你可以成为一个糟糕的漏斗，也可以成为一个糟糕的雨伞。 - Todd Jackson，Gmail 产品经理（参见 ctohb.com/umbrella 和 ctohb.com/keytogmail）

如果没有严格的流程将问题、疑虑和想法引导到其他地方，它们会找到管理层。这不仅包括你，还包括组织中的所有管理人员。管理者是默认的收件箱，Jackson 的观点是你的团队是默认的发件箱。你听到有人说：“嘿，X 有个错误”，你会想：“好的，工程师 Y 写了那个功能，去把错误报告发给他们。”这就是直接将问题引导到你的团队的漏斗化的一个例子。

一个更好的策略是充当团队的雨伞。与其实时将问题直接引导到团队，一个好的经理会组织、优先排序，并为团队提供一个有结构的队列来处理问题。你的目标是帮助团队集中精力，限制分心，并提供一个问题应该去的地方，以便高效处理问题。 只有在罕见的情况下，作为经理，你才应该向某个工程师强调一个 bug。如果你有一个 bug 队列和一个处理该队列的流程，你可以基本上消除常规的单独提升。

管理应该监控 bug 队列的处理流程，确保队列保持在一个可管理的长度，并根据产品质量是否达到目标来调整人员配置或流程。

你应该根据重要性和紧急性对队列进行优先排序。如果出现了具有极高时间压力的至关重要的事情，应将其放入队列并提升至首位。然后根据常识来处理它。如果你需要打电话给某人确保他们知道有 bug，那就这么做吧，只要这只是个例而不是规则。

你的第一个团队

作为技术领导者，你的工作不仅仅是管理技术团队；它还包括充当公司高层管理层中的技术代表。你的角色是在公司最高层级的战略讨论中代表工程和技术，并确保工程和技术朝着正确的方向发展。有时，这意味着与其他领导进行艰难的对话，这些对话需要你展示脆弱和谦卑的一面，以及通过相互信任解决冲突的对话。这些对话对于工作来说至关重要，为了能够进行这些对话，你必须与领导团队紧密合作，将他们视为你的第一团队。

你可能是一个技术水平很高的人。你可能最喜欢参加技术会议，或者至少觉得与团队一起解决技术问题是熟悉而且非常令人满意的。因此，很容易陷入一种模式，花费大部分时间与技术团队在一起，并采取“我的团队就是技术团队”的思维方式。这是在技术团队中建立融洽关系的好方法，在某些情况下，这是最关键的关系需要投入。

我的建议很简单：不要让技术上的诱人事物分散你的注意力，或者限制你与非技术领导建立关系的投入。你与其他领导的关系以及与非技术同行的信任将为你赋予信誉，并使你能够作为整个业务的技术决策指导人。与技术团队以外的人建立信任关系的方法与建立任何其他信任关系的方法相同：良好的沟通、定期设定期望并履行承诺，以及在发生错误/失败时承担责任。

那些将其时间都花在与工程深入编码，几乎不参与领导团队的技术领导者或首席技术官将很难在试图说服其他首席层级继续投资工程方面获得信誉。甚至更糟糕的是，当需要做出艰难决策时，他们甚至不会被征求意见。其他领导者将无法理解技术团队所要求的价值或工程在当前情况下的运作方式，并且他们将缺乏对未来可能发展的共同愿景。只有通过定期与领导团队保持互动，共享相关背景信息，并参与讨论过程中，作为技术领导者，你才能确保领导团队对工程如何帮助组织以及其如何随着时间推移发展有共同的理解。 每个 CTO 和 CEO 的关系都是不同的，尽管有一些共同的元素，以及一些健康关系的关键先决条件。

对齐特定目标

CEO 必须完全信任你领导技术团队以实现业务目标的能力。建立这种信任意味着你需要能够与 CEO 进行良好的沟通，既通过主动沟通，也通过确保 CEO 始终具有足够的背景知识来提出好问题。

良好的沟通意味着学会用商业语言进行沟通，在领导层交流中避免使用技术术语。你希望能够让 CEO 与你进行沟通，如果 CEO 不是技术人员，你越能说他们的语言，你们的互动中获得的信息就越多，你们也会相处得更好。

对齐业务方向

你和首席执行官需要对企业的发展方向有共同的理解，并能够进行建设性、甚至具有争议性的对话，以确保对该理解的深入。信任不仅适用于整体业务方向，也适用于具体目标。

建立信任的方式有很多，但无论你采取哪种方法，你都需要建立信任。参与非工作的共同活动，找到你们对个人价值观有共同点的领域，并使用特定的工具和练习来建立信任（参考 Brene Brown 的

BRAVING 清单，网址为 ctohb.com/braving）。

对齐企业文化和价值观

与所有高级管理人员一样，首席技术官（CTO）和首席执行官（CEO）应该在公司文化和价值观上保持强烈的一致性。特别重要的是，在工程领域内以及工程团队与公司其他部门之间建立积极的文化。在初创公司的预算中，技术团队往往是一个非常大，甚至是唯一的支出项。技术人员也往往是最具竞争力的职位，这使得在工程领域的招聘和不可避免的员工流动比其他部门更加昂贵。

CTO 与其他高管在文化和价值观上的高度一致是确保技术团队感到受到尊重和融入公司的关键因素，这反过来应有助于提高员工留住率。 传递坏消息

在与其他领导或高管合作时，一个通用的建议是不要回避向你的同行领导，尤其是 CEO，传递坏消息。由于你对团队的表现负责，你可能会想美化现实，宣传一切都很好。这种方法的问题很多：

- 如果事情不顺利，意味着常常错过截止日期或质量低于预期，你的同行将知道，并会想知道为什么你不承担这些失败，并解释你会如何改进。现实与你所代表的方式之间的差异削弱了对你的领导能力的信任。
- 时不时地，你需要为软件工程团队创造时间进行非面向用户的工程投资，无论是在技术债务还是未来架构方面。你需要拥有其他领导的信任和可信度，以便他人相信你，并理解这段时间上的投资回报率。

有关拥有失败的重要性，请参阅 Jocko Willink 和 Leif Babin 的《极致领导：美国海豹突击队如何领导并取胜》第一章的原则部分。 讲述受众的语言

技术主题往往非常微妙，细节至关重要。技术术语很好地帮助传达这种微妙之处，所以当工程师解释技术问题时，他们经常使用其他部门成员难以理解的语言，这并不意外。我相信你一定见过那些充满热情和激情的工程师，试图向非工程师解释他们的项目，结果只得到茫然的眼神，没有共同理解的增加。作为技术领导者，你必须做得更好。

例如，如果你面临着重大的技术债务，并希望在高层团队中倡导花一个月的时间重新设计该代码区域，你需要以一种可理解的方式传达你的理由。如果你在这次对话中讨论延迟、RPC 调用、依赖注入以及云服务提供商的首字母缩写，那么你的 CEO 和 CFO 几乎会立即对你置之不理。

然而，如果你将这次对话的框架建立在开发者生产力和团队士气的基础上，并将债务偿还的内容放在接下来六个月团队速度的背景下解释，那么你的观点将会更具说服力。

技术沟通最佳实践 确保与非工程师进行技术讨论和演示时更成功的几个一般性建议：

事先建立共同的语言/词汇。如果你需要使用一些普通高中生不会理解的词汇，确保你的听众已经知道它们，或在解释之前明确定义它们。

使用可理解的概念。技术挑战通常会与其他技术挑战进行比较，但在与非技术人员交流时这并不奏效。与其描述你的数据传输速度，以每秒字节数为单位，不如将其比作公路上的交通。

确认理解过程中的情况。在解释过程中向听众提问。如果他们能在你之前说出关键内容，那么你就知道你走在正确的道路上了。

不要因为你掌握了技术语言而自以为是，更不要让你的听众因为缺乏知识而感到自卑。"我不想给你讲得太专业"这样的说法会让听众失去兴趣。 通常，尽量将您的解释保持简单和简洁。避免纠缠于可能会分散注意力或让观众失去兴趣的细枝末节。

招聘和面试

作为技术领导者，有效招聘是您最具影响力的活动之一，也是最具挑战性的。您经常会发现自己在一个供给不足的市场中招聘人才，并与可能比您更有深度的其他公司竞争。您的优秀候选人很可能会收到其他竞争的工作机会，这意味着您不仅需要评估候选人的资质，还需要让他们相信您的机会是适合他们的。

在这种情况下，招聘既是一个销售活动（候选人评估您/您的公司），也是一个筛选过程（您/您的公司评估候选人）。在制定团队的招聘流程时，牢记这一点非常重要。 本书的这一部分按照顺序介绍了招聘和面试过程中的各个环节，从头部规划到入职。

像创业公司一样招聘

作为一家创业公司，您在招聘方面享有几个关键优势，并且在整个过程中必须充分利用这些优势，以确保能够吸引顶级人才。以下是几个使您具有竞争优势的特点：

- 您的规模较小，这意味着您在招聘过程中应该更灵活、更关注细节，并且比大公司更快速。
- 您可以推销一家非常吸引人的公司和个人成长轨迹。您可以推销一种富有创造力和激励人心的工作环境文化。 以下是利用这些优势的一些建议：
- 为了快速推进，发布职位描述之前，培训和征召同事进行面试。确保每个人都了解日程安排、面试脚本和评分标准，以及如何在开始之前使用应聘者跟踪软件（ATS）阅读并留下反馈（详见《求职者招聘》，第 59 页）。
- 预先安排好与每个应聘者的所有面试时间。如果您的流程包括四轮面试，请在开始时将所有面试都安排在日历上，最好在五个工作日内完成。如果您的团队能快速留下面试反馈（而且您应该坚持要求他们这么做），那么您可以提前通知未通过的候选人，并取消待定的日历邀请。如果将安排下一个面试推迟到候选人通过每一轮之后，每个步骤之间会增加多个天，很容易将原本能在一周内完成的流程延长至三周或更长时间。 一个好的经验原则，尤其在创业公司：没有人会太忙或太重要而无法抽出时间与候选人会面，尤其是对于更高级别的聘用。如果一个有实力的候选人要求与你的首席执行官和首席运营官交谈，那么你应该安排他们的会面。
- 确保针对每个面试者都有独特的脚本或指南，涵盖不同材料或从不同角度进行面试。（关于这个更详细的内容，请参考《面试最佳实践》第 63 页的“只提问新问题”部分。）

同时，切记没有免费的午餐。创业公司招聘的好处也伴随着风险。候选人几乎肯定会问你关于公司产品市场适应性、现金流或资金保证期、公司文化以及工作和生活平衡的问题。我鼓励你对候选人坦诚地谈论这些因素，与你的执行团队一起处理实际情况，并对这些问题有好的回答。

速度是你的朋友

记住，当招聘顶级人才时，速度是你的朋友。如果你能在一个星期内将某人完整地进行招聘流程，那么你的创业公司就会在决策方面拥有比那些流程常常需要数月才能做出决定的大公司更大的优势。

何时招聘：队伍规划

现金是一家年轻且尚未盈利的初创企业的命脉，因此决定承诺年薪超过 10 万美元的新工程师薪水这样一个重复性开销的决策不可轻率对待。几个关键因素共同影响着人员配置的决策，其中包括需求、优先级、时机和预算。

角色需求/团队缺口

决定招聘的第一步是识别团队中的缺口。缺口有几种形式。通常，在公司发展的早期阶段，这仅仅是一个技能上的缺口。例如，您的企业决定移动应用将成为市场推广策略的关键要素，而创始团队之前从未从事过移动开发工作。当然，他们可以学习并逐渐提高效率，但在短期和长期来看，雇佣一位有经验并有意愿从事移动开发的高级工程师来构建和维护该项目会更加高效。

其他种类的缺口包括资历缺口（缺乏资深经验做出正确决策，或没有足够的初级人才处理较为简单的任务），管理缺口（一个经理负责过多的员工），或专业知识缺口（团队中没有人足够了解某个领域的行业以指导决策）。 雇佣中的另一个主要正当理由是为了增加团队的总带宽。这种雇佣应该与某种业务目标或产品路线图保持一致，这样才能证明在特定时间引入新的团队成员是有正当理由的。

角色优先级和时机

一旦确定了一个空缺，下一个问题就是需要什么时候填补这个空缺。考虑到需要一段时间来找到一个优秀的人才，何时开始招聘过程才是合理的呢?

通常情况下，不是总是，回答是现在！每增加一个新员工，都会给团队增加复杂性和开销。假设忍受空缺的痛苦不是很严重，如果你能再过六个月继续运用一个较小的团队，并延迟雇佣，那可能是一个好主意，因为这样既能降低成本，又给你更多时间为雇佣提出理由。

你的团队的编制或招聘请求通常需要与其他团队的请求竞争，所以在公司内发展一种共同的语言来讨论一个雇佣的紧急程度或重要程度是很有用的。这并不需要非常复杂；它可以是一个 0 到 5 的评级系统，其中 0 代表紧急需求，而 5 表示一个对方便但可以在几个月或几个季度后再考虑紧急的雇佣。

新员工预算编制

在一家不盈利的初创公司中，您应该拥有一个合理化支出与收入的财务模型，并预测当前手头现金可以持续多久，直到您需要进行另一轮筹资。大多数首席执行官和首席财务官对这个模型有着深入的了解和亲身经历；作为技术领导者，您不需要花费太多时间来研究它。

但是，维持对部门对该模型的贡献的清晰认识是至关重要的，这主要表现在员工编制支出（现在和未来）方面。该模型应提供某种程度的约束，以年度预算或费用运行速度的形式来指导您的招聘时机。

招聘目标与目的

就像设计软件系统一样，在设计面试流程之前，您应该首先考虑您的要求和目标。虽然每个公司都应该有自己的要求和观点，但以下是我在设计面试流程时考虑的一些因素：

效率：招聘一个候选人需要多少时间和成本？

成功率：被聘用的候选人在工作中的成功率如何，他们在公司工作多长时间？

候选人体验：无论是否被录用，候选人在经历完整个过程后是否对贵公司有很高的评价？

公平机会：您是否确保每个人都有公平机会被聘用，并尽可能避免无意识偏见？

可扩展性：是否可以让其他人来负责运营这个过程，并有与您类似的效果/成功率？

效率

对于您的公司来说，招聘是一项昂贵的活动。这涉及到实际的成本，包括招聘人员、职位发布和招聘广告的费用。同时它还包括时间成本，主要体现在雇员花费在面试上的时间。在设计面试流程时，考虑每个步骤的意图，您要筛选什么以及如何以最有效的方式进行筛选。

降低或者分散时间投入的一种方法是让其他团队成员参与招聘过程。根据面试的主题，您并不总是需要最资深的工程师在场。一个经过适当培训的招聘协调员可以像一名资深工程师或高管一样有效地进行电话筛选、文化面试或者背景核查。

成功率

并不是每个招聘都能为您的公司带来成功。有些职位安排错误，有些人与公司文化不匹配，有些人在第一年被解雇或者辞职。特别是在扩大面试流程时，您需要衡量有多少次招聘是成功的。作为一名技术领导者，这是少有的能够计算明确、一致且具有指示性的指标之一，所以请充分利用，并确保您的流程处于最佳状态。考虑追踪招聘时间（从发布职位描述到新员工开始工作的时间）、全员留任情况、新员工离职或岗位下调的情况，以及在前两年内有多少新员工获得晋升。

正如安迪·格鲁夫在 *《高产出管理》* 中讨论的那样，即使是世界一流的面试流程，成功率也只有约 70%。基本上，招聘中存在许多风险：你试图根据仅有的几次谈话和面试过程中收集到的一些数据点来预测一个人在每周工作四十个小时、一周又一周的表现。

最好的领导者会跟踪他们的成功率，不害怕承认雇佣错误，并会慢慢地招聘，迅速地解雇。

无论如何，解雇一个刚刚雇佣但无法胜任工作的新员工对于每个人来说都是社交上尴尬和不舒服的。然而，对于你的团队来说，这是负责任的做法。一些有助于为新员工提供透明度并帮助经理做出明智决策的做法包括实施正式的 90 天试用期或引进期，并要求每隔 15 或 30 天进行新员工/经理的检查，或使用合同到期雇佣结构。

候选人体验

候选人体验指的是候选人在经历和完成你的招聘过程后对你的公司的感受。许多候选人在申请或面试之前会进行尽职调查。他们很可能会查看在线论坛和社交媒体，看看其他经历过你的招聘过程的候选人或员工对你的评价如何。 你无法总是控制别人对你的评价，但你仍然希望提供一种候选人体验，让

他们更有可能在网上看到好的评价，亲自获得良好的体验，从而更愿意继续参加你的面试并接受你的聘用邀请。

公平机会

有句谚语说人们往往偏向雇佣与自己相似的人。这往往是由于不成熟的面试评分方法，仅仅依赖面试官的直觉，而直觉往往会受到无意识偏见的影响。这种偏见可能会对其他种族、性别、族裔等候选人造成不利的影响。

在设计面试流程时，你应该侧重于基于与职位描述要求相一致的评估标准，而不仅仅是面试官的直觉。了解更多关于避免偏见的信息，请参考《面试最佳实践》中的避免偏见部分，第 63 页。

可伸缩性

如果你个人有能力有效地招聘，那就一切都很好。但随着时间的推移，可能会出现比你个人能够承担的更多职位空缺，这时你就需要扩充招聘流程并引入其他人。为了有效地这么做，你必须构建一个可重复利用的系统，让其他人能够利用该系统以与你相同的效果和成功率来识别顶尖人才并进行招聘。

这意味着其他人需要能够进行与你相似的面试，并在整个过程结束后得出与你类似的结论，就好像你亲自进行了面试一样。

本节的目标是帮助你创建一个可扩展的面试和招聘系统，以适应你组织的目标，并在你不直接参与的情况下（一旦你花时间进行过校准）无缝运作。通过定义和部署这种结构，创建深思熟虑的文档和模板，你可以使其他人能够进行面试并生成与你相似的候选人排名评分。

职位描述

很多公司低估了出色的职位描述的价值。一份真正优秀的职位描述对你有两个主要作用：它帮助你在公司内部明确和对齐对该职位的任务和价值的认识，同时它也会为你的公司进行宣传，并吸引符合条件的申请者。

调查市场

在开始编写工作描述之前，我鼓励你对市场进行调查。看看竞争或类似公司以及他们在类似职位上的工作描述。

通常，这些职位可以提供良好的灵感和校准，尤其是当你在雇佣不常见的角色时，可能无法被你建立的可扩展系统完全满足的时候。

明确角色

传统的工作描述中包含了角色的简短描述，然后是对候选人的要求的条目列表。我鼓励你写得更多。与其专注于一个成功候选人在特定职位上做什么，不如思考该角色的目的。这个角色会产生什么样的结果？你对这个角色在三、六或十二个月内带来什么样的影响有什么期望？你可以选择是否将这些问题的答案作为工作描述的一部分发布，但是对期望进行详细描述的练习无论如何都会有价值。 请与贵组织的其他领导共同交流这些问题的答案，并确保他们对这些答案表示赞同。如果在职责和岗位成果

的首个版本上得到了重要的反馈意见，也不要感到意外。在大多数初创公司，在正式开放编制职位之前，通常会就某个特定职位进行一次高层次、非结构化的讨论。哦，我们需要聘请一名高级 JavaScript 后端工程师。草拟和社会化职位描述的过程可以使您的团队准确地了解公司真正的需求，因此需要进行一些修订。

职位描述作为广告

贵公司公开发布的所有内容都会反映其文化和品牌形象，职位描述也不例外。职位描述的目标是公司文化和品牌中最有价值的客户：现有和未来的员工。理想情况下，合适的候选人不仅满足您的工作要求，而且与公司的文化非常匹配，他们阅读了职位描述后对该角色和公司本身感到兴奋。

以下是几种帮助职位描述反映公司文化的方法：

- 包括公司的核心价值观、使命或愿景——无论您拥有哪种核心能力。
- 包括该职位对您的团队、公司和客户的影响。
- 宣传您的团队结构、工作环境和规模。
- 强调您的薪酬和福利（在某些市场上，公布薪资范围是法律要求的）。

除了旨在引起候选人兴趣的要素外，还可以包括一些常被忽视的细节，以帮助候选人自行分级：

- 包括职级和薪酬范围。
- 包括地点、现场需求，以及是否允许远程工作，并到何种程度。
- 包括时区/工作小时要求。

招聘候选人

在寻找组织中的职位时，您将以三种方式寻找合格的候选人：内部招聘、外部招聘和推荐。一个有效、可扩展的招聘流程应该设计来利用这三种方法。

内部招聘

入职招聘是指推销您的职位空缺并收集自愿求职申请。与任何其他营销活动一样，单一渠道的方法可能不足以取得结果。

因此，将职位描述发布在招聘网站上只是最基本的要求。根据市场状况、招聘职位数量以及职位描述的质量和清晰度，仅仅发布职位描述可能就足够了。通常，您需要采取更多措施吸引顶级人才，包括在专业技术社群中积极推广您的职位，或通过参加会议/赞助、公司博客、社交媒体等方式来营销您的品牌。

并没有一个通用的最佳分类广告平台，吸引优秀员工的方式因地区而异。请密切关注最常见的平台/招聘网站，并相应地发布职位描述。一个良好的应聘者跟踪系统（ATS）将帮助您追踪每个候选人的推荐来源，并提供有关哪个招聘网站带来更好/更多深入流程的候选人的指标。特别是在招聘设计师时，与一些工作中的设计师交谈了解最受欢迎的作品集托管网站，并在这些招聘网站上保持存在，以找到最佳候选人。

您还需要监控每个职位收到的申请数量。至少，您的招聘经理应该每周查看他们职位的招聘漏斗状况，并相应调整招聘策略。如果某个职位没有收到足够的申请（或者吸引到了错误的申请者），那就要做一些改变！尝试调整职位标题或将职位描述发布到新的渠道。强大的招聘流程的一个关键要素就是与您团队构建的其他流程一样：不断检讨过去的决策，并不断改进。

外向招聘是主动联系目标候选人并鼓励他们申请你的职位。这可以由你、你的团队、内部招聘人员和/或外部招聘人员完成。我鼓励团队首先从内部招聘和直接招聘开始招聘流程。通过实际招聘、与候选人交谈并倾听他们对你的推销的反应，你将了解市场的情况以及顶级候选人对你的销售物品的看法。你还将了解你的职位的竞争力如何，以及找到符合你的工作描述的候选人有多容易或困难，这甚至可能促使你对其进行调整。一旦你调整完善了职位，并清楚知道你要寻找的人和条件，你就可以为外部招聘人员提供优化指导，帮助他们更有效地代表你找到候选人。

并非所有外部招聘人员都一样。你希望找到符合以下所有标准的人：

- 组织能力强
- 能够有效地推销你的职位（你的工作是培训他们并追究他们能否胜任这项工作）
- 有动力在不推销或打扰的情况下坚持跟进
- 倾向于重视与您和候选人的关系，而不仅仅是单个聘用所带来的佣金。

内推

雇佣中最高的回报来自于内部推荐，即来自您现有员工的推荐。人们更愿意与一位现有团队成员赞美的公司做生意，而且当候选人已经通过熟悉您文化的人的审核时，更容易找到符合文化的人才。您可以通过提供现金激励（参见侧边栏）或与推荐人之间良好的沟通，告知他们推荐的状态，来鼓励内部推荐。

鉴于推荐具有如此高的成功概率，您希望提供最佳候选人体验。您还可以考虑简化（但公平的）招聘流程。跳过或压缩任何顶层筛选工具，如电话筛选或资格确认表，可能是合适的。您还可以鼓励推荐人写一两段文字，用于证明他们的推荐。

关于激励推荐的数学说明

根据您可能已经拥有的数据，很容易就可以近似计算出雇佣一名新工程师的成本（包括时间和实际花费的美元）。如果考虑到推荐往往有更高的成功转化率，就可以清楚地看到，推荐可以节省数千到数万美元的费用，这可以帮助您为任何成功推荐并最终被雇佣并在岗位上工作数月以上的员工辩解为数千美元的奖金。

面试最佳实践

面试流程是您确定候选人与您招聘岗位的匹配程度的能力的重要环节。请记住，没有完美的面试。在短短几个小时内，面试官收集到的数据当然无法完全预测一个人在未来几个月甚至几年的全职工作表现。

在本节中，我介绍一些高级面试最佳实践，并为面试的各个步骤提供背景和上下文，包括候选人/简历接收表、电话筛选、文化面试、技术面试、编码作业或带回家完成的作业、高管面试，最后是背景核

查。

被拒绝的候选人的意见很重要

在设计面试流程时，候选人的体验应该是首要考虑和优先事项。即使你选择不雇佣某个候选人，他们都会对你和你的公司留下好坏印象。这种印象可能会导致他们向他们的专业网络中的人力资源推荐你，这些人员可能在将来会申请你的职位。或者这种印象可能会导致他们在任何机会都对你进行负面评价。

职位招聘网站和谷歌评论中充斥着许多面试失控的证据，一旦声誉受到损害，很难挽回。虽然对于某些候选人，无论你如何尊重和考虑，他们都无法避免对你的拒绝感到痛苦和愤怒，但这些人是少数。对大多数进入面试阶段的候选人来说，一个尊重和周到的面试过程会让他们对你的公司产生中立或积极的感觉，并帮助你避免在网上受到负面评论。

及时安排并简化日程安排

理想情况下，你/你的团队应该事先向候选人传达招聘流程的步骤和范围，并利用简单可靠的解决方案实时安排这些步骤。例如，你可以选择(A)指派一位招聘经理在工作小时内处理所有日程安排，(B)提前安排所有面试，或者(C)提供一个在线工具，候选人可以据自己的方便时间异步安排面试。

实际上，任何方法都比要求每位面试官在每次面试之前向每位候选人发送电子邮件以便顺序安排日程要好，这种方式可能会拖延整个面试过程数周或数月。 只提出新问题

每次面试接触点都应该给候选人一种对话的延续感，而不是重复之前讨论过的细节。避免重复需要在面试之前进行深思熟虑的结构设计和仔细的计划。

理想情况下，后续的面试应该用来深入挖掘和探索与候选人或角色相关的领域，双方都希望全面了解关键优势和劣势。通过应聘者跟踪系统（ATS）与后续的面试官分享建议关注的领域或新的问题，可以确保连贯性、效率和良好的候选人体验，从而显示您的潜在雇佣者是否真正适合您的团队。

避免偏见

如果您对“无意识偏见”一词不熟悉，我鼓励您阅读丹尼尔·康纳曼的《快思慢想》。这是我了解我们大脑产生的许多类型系统错误的首选书籍。 在没有充分理由的情况下，无意中对候选人造成优势或劣势是非常容易的。必然会导致更差的招聘结果，甚至可能引发昂贵的法律纠纷。

偏见有很多形式。大多数偏见是无意识的，可能涉及性别、种族、校友身份或社会经济背景。但偏见也可以意味着，面试官在面试前根据之前面试官的评分结果对候选人做出的结论。没有任何系统可以确保消除所有有害的偏见，但你可以采取一些步骤来最小化无意识偏见，例如在简历筛选过程中遮盖候选人的姓名或照片（这通常暗示性别和种族）。

为了避免影响后续面试官的主观评价或引发偏见，我鼓励面试官在候选人交谈后留下两种不同的反馈信息：

1.详细的笔记和评分 2.给后续对话提供建议性问题。

大部分面试反馈应该包含针对具体职位的详细笔记和评分，而这些评分是根据事先准备好的面试问题进行的（更多信息请参阅《技术面试》，第 76 页）。为了避免偏见，这些反馈信息理想情况下不应在后续团队成员面试之前阅读。例如，如果你知道之前的面试官对候选人给出较低评分，你可能会出现确认偏见，并夸大候选人在你面试中表现不佳的方面。 候选人追踪系统（ATS）的使用

当同时面试两个或三个以上的候选人时，需要投入大量的工作来管理候选人在流程中的位置，协调面试官的笔记，并与候选人持续、迅速地沟通。如果没有一个精密调校的系统来管理所有这些物流，候选人的体验很容易受到影响，招聘成本也会上升。这是一个普遍存在的问题，有几种高质量、现成的候选人追踪系统（ATS）解决方案已经开发出来，价格和复杂程度各有不同，以解决这个问题。

这里的建议很简单：尽早选择和启用一个 ATS。不要等到你的流程已经陷入困境才采取行动。培训你的团队，要求系统被广泛采用，并与人力资源、招聘经理和面试官共同设定使用期望。

推销候选人 如前所述，我强烈鼓励招聘经理将面试过程视为销售过程。这自然会导致一些良好的习惯，无缝地从销售过程转化为面试中：

- 在与客户的销售过程中，无论您正在对客户进行资格评估，您始终专注于向潜在客户推销产品。良好的销售过程将资格评估视为一个漏斗，在漏斗顶端进行轻度的资格筛选，并逐步进行更加细致和时间密集的资格筛选，同时提供更加定制和个性化的销售推荐。

- 您应该始终向候选人销售加入贵公司以及您提供的职位/机会的优势和积极影响。通过他们通过所有面试环节后，他们应该渴望在贵公司工作，并且对于接受您的工作机会胜过其他已收到或可能收到的工作机会感到兴奋。

- 确保您在面试早期至少提出一些关于候选人对他们下一个角色的期望的开放式问题。这将帮助您的面试官综合评估候选人的期望与您招聘的角色的匹配程度。这些信息应在候选人的个人资料中记录，并在途中用于定制和个性化的推荐。

- 例如，一个初级候选人来自一个主要由初级和中级团队组成的团队，可能正在寻求与更资深的 JavaScript 工程师合作的机会，并处于一个能促进他们成长的环境中。如果您的团队提供资深支持，并且您的文化崇尚指导，确保您特别强调这一优势，尤其是在提供工作机会阶段。 请帮我翻译：“* 在面试结束时，务必留给候选人一些时间（五到十分钟）提问。大多数合格的候选人都会准备好问题参加面试，通过他们的提问，您可以了解对方关心的内容。这是一个让面试官在回答问题时向候选人推销公司福利的好机会。

- 在整个过程中，确保候选人感到受到尊重，并逐步接触到公司的更多信息。最优秀的候选人需要感觉到他们经受过智能审核，并且从公司了解到足够的信息以引起他们的兴趣。理想情况下，您希望被拒绝的候选人也能在 Google 和 Glassdoor 上留下积极的评价。要实现这一点，您需要在整个面试过程中推销公司的福利，尊重他人的时间，像对待自己的时间一样对待他人，保持一致和及时的沟通，并确保每个人都感觉到整个流程尽可能公正和透明。

招聘表格

面试漏斗的开始是一个表格，它实现了两个目标：向候选人提供有关公司及其招聘流程的一些信息，从而展示公司的文化；并从候选人那里收集一些信息作为廉价的粗略过滤器。

招聘表格导言部分” 在您的申请表格顶部，您应该概述几个关键的信息给候选人：

- 重申他们所申请的职位及其关键要求和影响。
- 重申贵公司的核心价值观并提供文化样本。
- 对招聘过程设定期望，包括需要多长时间，有多少步骤，以及大致的过程。

问卷调查申请表格

问卷应包括请求候选人的简历（或领英个人资料链接），以及与就业资格有关的法律和人力资源要求的一些问题，然后最好再问一些对候选人进行资格审核的问题。这些资格审核问题应该是轻松、自由形式的，甚至可以是技术性问题，以确保候选人的能力与职位要求大致相符。例如，对于需要 JavaScript 经验的职位，询问候选人在一个从不舒服到非常舒服的范围内，你在使用 JavaScript 方面的熟悉程度如何，不失为一个合理的确认经验的问题。

这些问题似乎与职位描述中列出的要求重复，的确如此，但令人惊讶的是，有多少简历缺乏基本资格。这些问题对于候选人来说很快/简单地回答，对于招聘经理来说也能很快用来筛选应聘者。

如果你的候选人数量太多，想在这个阶段进行更多的初步筛选，调查问卷还可以包括一两个更有趣或更具挑战性的问题。如果包括这些问题，请确保仍然让它们简洁；你不希望因为这些问题太过费力而失去候选人。如果你被申请者们压倒了，那么在这里更加偏向于提供更多的数据来进行筛选，否则也许最好把更细致入微的资格要求留到更进一步的阶段。

以下是一个涵盖广泛匹配度和自我识别的技术熟悉度的信息收集表的一些示例问题（我在 ctohb.com/templates 中提供了一个样本）。

- 作为一个优秀的候选人，您将收到很多的聘用邀约。在薪酬和福利相等的情况下，是什么因素会让您选择一家公司而不是其他公司？ 以下是你下一步行动中的交易推手和交易破坏者是什么？

在你的工作中，什么给你提供了能量？什么让你感到精力不济？

你对于地理位置的期望是什么（地点，远程，现场）？

你对基本技术资格有多熟悉：将[相关编程语言或工具]的熟悉程度从 1 到 10 进行排名。

电话面试

初次电话面试，和面试过程中的任何事物一样，具有双重功能：一方面是了解候选人的机会，另一方面是候选人与贵公司的首次互动（以及对人员的评估）。

考虑到这是候选人与贵公司的第一个互动对象，值得慎重考虑由谁来进行面试。这个阶段的问题不需要非常技术性，因此并不一定需要由技术团队的成员来进行面试。通常由人力资源部门或专门招聘团队负责。

无论谁来进行电话面试，都要确保这个人是您团队/公司良好文化代表，并且了解到技术候选人可能在此阶段提出的问题，包括：

软件平台的架构，包括关键语言、工具和目标客户（例如移动端、桌面端等）。面试官应该具备对所使用的术语有初步的了解，而不仅仅是读出一个清单。

技术团队的规模，包括公司整体以及在技术方面。

请提供以下内容:（Please provide the following content:）

技术团队规模，包括公司整体规模和技术团队规模。 候选人将与谁一起工作。这也应该包括对招聘预测的总体描述以及随时间新增人员的大致数量。

候选人将向谁报告工作。提供一些关于该经理的基本背景，包括他们在公司的任期，以及可能在公司工作之前从事的工作。

对于公司的核心价值观/文化以及工作方式有一种很好的感知。

面试官的目标应是向候选人介绍公司、公司的文化、职位以及招聘流程。他们还将向候选人提出一些高层次的问题以确认其在该职位上的结构适应性。您希望候选人在面试的其余过程中都能充满动力，并对在贵公司工作的想法充满激情。

因此，电话面试中问的确切问题并不是非常重要。以下是一些您可能希望涵盖的领域的大纲： 他们的招聘时间是否有限制（例如，其他工作机会）？ 候选人的所在地是哪里，他们是否愿意搬迁？ 大概何时可以开始工作，或者他们正在寻找开始工作的时间？ 确认薪酬预期是否一致，并解释福利和优惠。

除了对这些问题的回答，面试官还应该评估候选人在这个职位中的整体适应性。候选人能否清晰地沟通，他们是否看起来适合公司文化，他们声称的经验是否与简历上的相符，以及他们是否对公司和机会感兴趣?

文化面试

在面试过程中，你寻找的主要标准之一是文化适应性。文化适应性是候选人除了经验和技能之外，使他们能够成功的个性特质。 在设计面试流程时，应将候选人体验放在首要位置并作为一项重要任务。即使选择不录用某位候选人，他们也会对你和你的公司留下积极或消极的印象。这种印象可能会导致他们向其职业网络中的人赞美你并在将来申请你公司的岗位。或者这种印象可能会导致他们随时都在负面地抱怨你。

就业网站和谷歌评论上满是面试出现问题的证据，一旦声誉受损，要恢复就非常困难。尽管对于某些候选人而言，你的尊重与考虑无法阻止他们因被拒绝而对你产生不满，但这些人只占少数。对于大多数进入面试阶段的候选人，一次尊重和周到的面试流程将使他们对你的公司产生中立至积极的观感，并帮助你避免在线负面宣传。

准时并简化日程安排

理想情况下，你/你的团队应事先向候选人明确说明招聘流程的步骤和范围，并利用一种简单可靠的解决方案来实时安排这些步骤。例如，你可以选择（A）指定一位招聘经理在工作时间内负责全部安排，（B）提前安排所有面试，或者（C）提供一个在线工具，候选人可以根据自己的时间异步安排面试。

实际上，任何一种方式都比要求每位面试官在每次面试之前给每位候选人发送电子邮件以依次安排日程要好，这可能导致面试过程拖延数周或数月。 只问新问题

每次面试接触点都应该让候选人感觉像是对话的延续，而不是重复之前讨论过的细节。避免重复需要事前经过深思熟虑的结构和仔细计划。

理想情况下，后续的面试应该用来更深入地探索候选人或职位相关的领域，双方都希望全面了解主要优势和劣势。通过求职追踪系统（ATS）与后续面试官分享建议的关注领域或新的问题，可以确保连续性、高效性和良好的候选人体验，从而揭示你的潜在雇佣是否真正适合你的团队。

避免偏见

如果你对无意识偏见这个词不熟悉，我鼓励你阅读《快思慢想》（Daniel Kahneman）这本书。这是我了解我们的大脑犯下的许多系统性错误的最佳读物。 很容易在不合理的情况下无意中给候选人带来优势或劣势。不可避免地，这将导致雇用结果变差或潜在的昂贵的法律纠纷。

偏见有很多形式。大部分都是无意识的，可以涉及性别、种族、校友身份或社会经济背景。但是偏见也可以意味着面试官在面试之前仅仅基于之前一个面试官的评分来得出对候选人的结论。没有一个系统能确保排除所有有害的偏见，但是有一些步骤你可以采取来减少无意识的偏见，比如在简历筛选过程中隐藏候选人的姓名或照片（这些常常暗示性别和种族）。

为了避免给后续的面试官带来先入为主或有偏见，我鼓励面试官在候选人交流中留下两种不同类型的反馈：

1. 详细的笔记和评分
2. 推荐用于后续面试的问题

大部分面试反馈应该是针对特定岗位的评分指南做出详细笔记和评分（关于这点的更多内容，请参见《技术面试》，第 76 页）。这些反馈最好不要被后续团队成员在他们的面试之前阅读，以避免偏见。例如，如果你知道之前的面试官对候选人的评分很低，你可能会产生确认偏见，并过分看重候选人在你的面试中表现不好的方面。 候选人销售 正如前面提到的，我强烈鼓励招聘经理将面试过程视为销售过程。这自然而然地导致了几种良好的习惯，这些习惯可以从销售中无缝转化为面试中：

- 在与客户的销售过程中，您始终专注于向潜在客户销售产品，即使在对客户进行资格评定时也是如此。一个良好的销售过程将资格评定视为一个漏斗，顶部进行轻度资格评定，底部进行逐渐细致和耗时的资格评定，同时提供越来越个性化和定制化的销售陈述。

- 您应该始终向候选人推销加入您的公司和所提供的角色/机会的优势和积极效益。当他们通过所有的面试环节后，他们应该渴望在您的公司工作，并且对您的职位提议比其他他们收到（或可能收到）的提议更感兴奋。

- 确保您至少在面试初期提出几个开放性问题，询问候选人对他们下一个角色的期望是什么。这将帮助您的面试官综合考虑候选人的期望与您正在招聘的角色的匹配程度。这些信息应记录在候选人的档案中，并用于在途中量身定制和定制销售陈述。

    - 例如，一个来自主要由初级和中级团队组成的初级候选人可能正在寻找与更高级的 JavaScript 工程师一起工作的机会，并且在这种环境中促进他们的成长。如果您的团队提供高级支持，并且您的文化倾向于导师制度，请确保突出这一优势，尤其是在提供阶段。 请务必在面试结束时留给候选人一些时间（五到十分钟）来提问。大多数合格的候选人都会在面试时准备好问题，通过他们选择提问的内容，你可以了解到他们关心的事情。这也是一个非常好的机会，让面试官在回答问题时推销贵公司的优势。

请在招聘过程中，逐步让候选人感到受到尊重，并逐渐让他们了解更多关于贵公司的信息。最好的候选人应该觉得他们经过了明智的筛选，并且已经了解到足够让他们激动的公司信息。理想情况下，即使是未被录用的候选人也应该能够在 Google 和 Glassdoor 上留下积极的评价。你可以通过在整个面试过程中推销公司的优势，尊重他人的时间，保持沟通一致和及时，以及确保每个人都觉得整个过程公平透明来实现这一目标。

受理表

面试漏斗的开始是一张表格，它可以实现两个目标：向候选人提供关于贵公司及其招聘流程的一些信息，从而展示公司的文化；并且从候选人那里收集大量信息，作为一种廉价的、粗粒度的筛选工具。

受理表前言 在您的申请表的顶部，您应该概述候选人的几个关键信息：

- 重申他们所申请的角色及其主要要求和影响。
- 重申贵公司的核心价值观，并提供文化样本。
- 对招聘流程的期望进行设定，包括需要多长时间，有多少步骤，以及整个过程的大致情况。

申请表问卷调查

问卷应包括要求候选人提交简历（或领英个人资料链接），并向法务和人力资源部门询问一些与就业资格有关的问题，然后理想情况下也要向候选人提几个资格问题。资格问题应简要轻松，基本上是自由形式的，甚至可能是技术问题，以确保候选人具备所需的能力。例如，对于需要 JavaScript 经验的岗位，通过问一个问题来确认候选人的经验，比如：“在工作中使用 JavaScript 时，您的舒适程度如何（从不舒适到非常舒适）？”这种问题可能与职位描述中列出的要求重复，但你会惊讶地发现有多少简历没有基本资格。这些问题对候选人来说非常简单/琐碎，对招聘经理来说也能快速用来筛选申请人。

如果你面临大量候选人并希望在这个阶段进行更多筛选，问卷中也可以包括一两个更有趣或更具挑战性的问题。如果包括这些问题，请确保保持简短；你不想因为问题太过艰难而失去候选人。如果申请人过多，则偏向于增加更多数据以供筛选，否则或许最好将更细致的资质要求留到后面。

以下是一些涵盖广泛兼容性和自我技术熟悉度的入职表格示例问题（我在 ctohb.com/templates 中提供了一个示例）：

- 作为一位优秀的候选人，您将会接到很多职位邀请。在薪酬和福利相等的情况下，您会选择哪家公司？

在你下一步的行动中，什么是促成合作和阻碍合作的因素？

工作中什么会给你带来活力？什么会消耗你的能量？

你对地理位置有什么期望（是否需要在现场工作，或可远程工作）？

你对基本技术要求有多熟悉：请将你熟悉的[相关编程语言或工具]的程度从 1 到 10 进行排名。

电话筛选

电话初筛，就像面试过程中的一切一样，具有双重功能：它是了解候选人更多信息的机会，也是候选人与贵公司的第一次互动（和评估）的机会。

考虑到这是候选人在贵公司第一次与人互动，值得仔细考虑由谁进行面试。在这个阶段，问题并不需要非常技术性，因此不必由技术团队的成员来进行面试。通常由人力资源部门或专门招聘团队来进行。

无论是谁进行电话初筛，确保此人是团队/公司的良好文化代表，并具备技术候选人在此阶段可能会问的信息，包括：

软件堆栈的样子，包括关键语言、工具和目标客户（例如移动设备、桌面等）。面试官应该对他们在这里使用的术语有一定的了解，而不仅仅是背诵一份清单。

技术团队的规模，无论是在整个公司中还是在特定部门中。 候选人将会与谁合作。这应该包括一般的雇佣预测，以及随时间增加的人数大致是多少。

候选人将向谁汇报。对该经理的基本背景提供一些介绍，包括在公司的任职时间，也许是在进入公司之前做过什么。

对公司的核心价值观/文化以及工作方式有一个较好的了解。

面试官的目标应该是向候选人介绍公司、公司的文化、角色和招聘流程。他们还会向候选人提问一些高层次的问题，以确认候选人在该角色中的结构适应性。您希望候选人在这次面试后能够对接下来的面试流程充满动力，并对在贵公司工作的想法感到兴奋。

电话面试中所问的具体问题并不是非常重要。以下是你可能想涵盖的一些领域的纲要： 请帮我翻译：

- 他们是否有任何限制他们的招聘时间表（例如，其他职位提议）？
- 求职者目前所在的地点在哪里，如果需要的话，他们是否愿意重新安置？
- 大致何时可以开始，或者他们想什么时候开始？
- 确认薪酬期望是否一致，并解释福利和额外津贴。

除了对这些问题的好回答外，面试官还应该评估他们在岗位上的整体匹配度。求职者是否能清楚地沟通，是否看起来适应公司文化，他们声称的经验是否与他们的简历相符，以及他们是否对公司和机会感兴趣？

文化面试

您在面试过程中寻找的主要标准之一是文化适应度。文化适应度指的是除了求职者的经验和技能之外的所有个性要素，这些要素将使他们能够取得成功。 在您的组织中，为了有效地筛选适合公司文化的候选人，您的公司应该对自己的文化有一个相当清晰的了解。这可以是许多形式，例如核心价值观列表、使命宣言、愿景陈述、指导原则等。无论是什么形式，它们应该是真实和符合公司实际情况的。如果您对此感到困惑，

我建议您阅读 Stanley McChrystal 的《团队的团队》、Laszlo Bock 的《工作法则！》和 Jonathan Raymond 的《好的权威》。

目前，有几种正式结构化的面试程序被广泛使用。其中一个比较常见的是称为"topgrading"的方法，它涉及至少两个不同的方面：topgrading 方法和 topgrading 面试。topgrading 方法是一种完整的招聘方法论，据说是由通用电气在 20 世纪 80 年代/90 年代开发的，并在 Verne Harnish 的《扩大规模》一书中有所介绍。topgrading 面试（ctohb.com/interview），也被称为文化面试，是一种特定的面试议程、方式和结构，旨在了解候选人的背景和文化适应性。

按照正式设计，topgrading 面试会逐步询问候选人的就业历史，并针对候选人最近几个角色提出同样的一套问题。根据候选人的经历和他们在过去几个角色中所花的时间长短，您应该涵盖从两个到五个之间的前几个职位。您希望涵盖一个足够长的时间段，以便试图识别趋势并观察成长，但也不要让候选人在面试中花费三个小时讨论二十年前他们在大学实习的经历。

对于每个角色，topgrading 要求面试官询问以下问题：

- 这个职位有哪些显著的成功或成就?
- 在这个职位上有过哪些错误或失败？你的主管的名字和职称是什么?
- 你认为主管对你的优点和缺点会做出怎样的诚实评估?
- 你觉得你的主管的优点和缺点是什么?

除了这个面试模式，Topgrading 建议采用两个面试官的方法：主导面试的人会积极与候选人互动并表现出好奇心，另外还有一个专门的记录员。 无论您使用一位还是两位面试官，都非常重要的是记笔记。为了公正地评估候选人，您希望在面试第一位候选人之前制定一个评分卡，评估候选人的答案，寻找与公司文化的一致性。例如，如果尊重挑战是公司核心价值观，问候选人是否能够找到任何有挑战性但尊重的例子。或者他们是否曾言辱骂任何过去的同事？在面试后使用笔记完成和证明评分卡上的分数是必要的。

编码挑战

要求完成回家作业，也称为编码挑战或面试作业，是一个有争议的话题。回家作业通常需要候选人投入大量时间，因此是招聘过程中候选人流失的重要原因。很容易想象，受欢迎的候选人会被要求完成多个回家作业，每个作业都需要耗费很多小时甚至几天的工作，最终可能持续数周的时间。面临这些要求时，候选人优先选择那些他们最感兴趣和/或任务相对简单的公司的作业是可以理解的。

尽管存在这些结构性挑战，但从招聘经理的角度来看，拥有这种要求至关重要。您如何雇佣一名软件工程师而没有看到他们为您编写的代码呢?

总结起来，存在三个竞争因素： 建立预测能力：雇主希望在软件工程面试过程中让候选人真正编写代码，以预测他们在工作中的表现。

最小化流失：雇主希望候选人能够真正完成编码任务，而不是在招聘过程中退出。

改善候选人体验：候选人希望感到他们的时间被尊重，分配的任务也是合理的。理想情况下，应通过这个任务让候选人更多地了解贵公司，并对机会更加兴奋。

预测能力

编码面试或任务有几种不同的形式。任务范围从家庭作业项目到使用在线平台进行编程练习（有时也称为代码卡塔），再到实时配对编程。在没有关于这些形式的预测能力的任何经验数据的情况下，我鼓励您设计一个与贵公司日常工作尽可能相似的练习。如果贵公司没有进行任何实时配对编程，那么直觉上来说，在面试中与候选人共同编程的表现如何，并不高度相关/可预测。最起码它收集了相关的信号。 作为一名经理，你的目标是充分发挥与你一起工作的人的最佳水平。在此背景下，试着回想一下你上次面试时的情况，并在设计编码任务时发挥你的同理心。对于大多数人来说，面试是一个非常紧张的过程，在这种情况下要求他们在创造性或解决问题方面表现出色并不总能带来最佳表现。以下是帮助候选人在编码任务上发挥出色的一些方法：

- 在可能的情况下，提供选择语言/工具的灵活性。
- 允许以异步方式完成工作（即完成后交回，而非实时编码）。
    - 如果你非常看重编码过程中的信号，以至于完成后的作品不能充分展示候选人的水平，可以考虑要求候选人使用 Loom 或其他类似工具录制他们完成任务的部分过程。
- 在候选人的产出中明确你所期望的内容。例如，如果你的评分标准包括他们对代码的文档说明程度，那么确保给候选人的提示中告诉他们要包含文档说明。或者如果你计划运行代码，让候选人知道你只评估正确性，还是其他因素也很重要，比如性能、负面案例等等。 候选人体验和掉落

如果候选人觉得你的编程作业有趣且容易入手，他们更有可能完成。最好的作业与你的业务相关，并且最好暴露候选人面临的日常问题类型。

不好的例子： 你是一个 Web SaaS 平台，而你让候选人做一个与手机开发相关的挑战。

好的例子： 你的公司与许多传统第三方 API 集成，你的挑战是构建一个与真实业务具有相似领域名词 / 动词的有限集成接口。

为候选人提供具有工作构建系统/测试的现有代码库，可以节省候选人自己构建的时间。 为了尊重候选人的时间，我建议为 take-home 任务设定一个明确的时间限制。时间限制的目标不是为了施加时间压力和迫使快速交付，而是确保候选人不会在挑战上投入过多时间，并感觉到挑战是一个合理的要求。为了确保候选人理解时间限制，您应该：

- 充分解释时间限制
- 确保任务能在规定的时间限制内完成

- 让候选人知道他们的提交将如何评估。如果您的评分标准会评价候选人的 README 文件，那么让候选人知道他们应该花时间编写一个 README。如果代码将在面试官面前由候选人实时运行，或者是由面试官以异步方式运行，请告知他们会考虑运行时间。如果您更关心架构决策，而不太关心运行性能，请也告诉他们，这样他们可以合理分配时间。

技术面试

无论是 take-home 编程面试的方法如何备受争议和多样化，技术面试本身也是各种各样的。总的来说，我鼓励您遵循相同的基本原则：确保您收集到与实际工作相关的信号，并对候选人本身保持尊重和体谅。 经典的技术面试，由许多大型科技公司采用，涉及某种形式的共享白板体验，候选人被要求实时解决一个技术问题。问题范围从学术问题如按某些特殊条件对数组进行排序，到高级/模糊的架构问题，如设计一个能处理 1 亿用户发布新闻更新的系统。

经典面试方法在大公司中必须有效，因为他们继续年复一年地使用它，但我不明白它们如何在初创公司中发挥作用。它们往往过于宽泛或过于狭窄，因此很难公平评分。学术问题很少与人们在工作中每天解决的问题相关。

最致命的是，它们并没有让候选人在面试环境中取得成功。毕竟，在大公司里，几乎没有工程师会把数组排序算法写成日常工作的一部分。

我在本章中概述的方法论代表了我在初创公司环境中看到和自己成功使用的另一种方法。

方法论

以下是我使用的一种方法论，来源于对高潜人才选用的经验，用于筛选和雇佣高级软件工程师候选人。按照高潜人才的精神，我称之为技术重点面试。

技术重点面试指南

要了解候选人的优势和不足以及这在您正在招聘的职位中有多重要，首先您需要确定哪些主题领域对您的职位至关重要。为此，您需要创建一个技术重点面试指南，其中包括四到八个技术领域的列表，以及每个领域内的一组示例问题、最佳实践答案和评分指南。

示例答案和评分指南的目的是确保在多个面试官和候选人之间评分的公平性和一致性。您需要区分候选人在某个领域中的差距和真正的专业知识水平，因此您的问题应设计成能够引出三种答案之一：差的、好的和优秀的。因此，它们应该适合进行相应的评分。在评分问题时，为了明显区分知识差距和真正的专业知识水平，我建议差的答案得分为 0-2，好的答案得分为 3-6，只有优秀的答案得分在 7-10 之间。

当我说差的答案时，我的意思是对问题的回答表明对所讨论的主题几乎没有经验或专业知识。好的答案表明在该主题上具备能力，甚至可能达到非常高的水平。优秀的答案不仅表现出能力，还展示了对该主题的真正理解和思考深度。例如，如果问题涉及候选人如何考虑设计单元测试套件，而他们的回答是从未考虑过，那就是 0 分，说明存在缺陷。如果他们的回答包括一些他们设计过的测试套件以及对其进行的某种正当性的论述，那就是好的答案，也许是 5 或 6 分。如果他们的回答包括测试套件设计原则的完整概述，以及每种原则的利弊以及在不同情况下如何应用它们的方法，那么您将看到真正

的专业知识，得分在 7-10 之间。 为了给候选人取得成功的最佳机会，我不建议对每个问题进行评分。相反，可以对一个主题领域进行评分。这样你可以在一个主题内尝试多个问题，寻找候选人的专业知识领域，并对该主题的综合结果进行评分。

毫无疑问，编写这些问题、示例答案和评分指南是一项艰巨的工作。好消息是，任何一个给定的问题在多个角色中都是有用的，并且可以在很长一段时间内重复使用。事实上，我鼓励你建立一个问题的中央库（以及相关的示例答案/评分指南）。当需要编写下一个技术重点面试指南时，通过能够根据需要从库中重用问题，你会发现工作变得更容易。

请参阅我个人问题库中的示例重点指南：https://ctohb.com/templates。

招聘初级员工与高级员工

对于具有一到两年编码经验的初级员工，你所寻找的品质应该与具有十年以上经验的高级员工有很大的不同。理想的初级员工应该充满好奇心，热衷于学习，并具备扎实的编程基础，能够在增量功能开发中工作。相比之下，高级员工不仅应该具备编程基础，还应该对架构、观点和最佳实践等各种工具和问题有深入的思考，并能够建立信任，不仅能够构建增量功能，还能够在新的绿地项目中拥有并做出良好的架构决策。由于这两种类型的角色提供的关键价值是如此不同，对它们的面试应该有所区别。 对于高级聘用，重点面试是深入探讨候选人的决策能力、概念理解和架构知识的关键，应给予重视。对于初级角色，这种知识深入探讨应该更短，比实际编码练习的重要性更小。

面试本身

高级软件工程师技术重点面试通常是候选人和一位主面试官之间的 60 到 90 分钟对话，最好有一位主要沉默的第二面试官在场记录笔记。根据您的重点指南的长度和要涵盖的主题数量，您可以考虑将主题分成多个重点面试。

我强调这次面试应该是对话式的；您希望了解候选人在软件工程领域最熟悉和热情的领域，以及他们从未被追究过责任的领域或历史性地委派给他人的领域。做到这一点不需要脑筋急转弯、双人编程或任何问题解决。只需问问问题即可！

开始时，以一些轻松的谈话进行面试。过一两分钟后，开始描述会议的议程/计划。让候选人知道您有一份带有面试指南的文件，并且您的目标是在接下来的 60 到 75 分钟内让候选人讨论该指南中的主题，剩下 15 分钟让他们向您提问。 前言之后，您将跳转到面试指南的第一部分。您在指南的每个部分的目标不是问每个问题。您首先要确定候选人在该主题领域的三个类别中属于哪个类别：差、好、还是出色，然后

从那里缩小得分范围。在前两个问题之后，您应该对如何分类候选人有很好的想法，然后使用后续问题进一步探索以缩小得分范围。

如果候选人完全错过或承认对某个主题不熟悉，就没有必要继续问每个问题；您已经得到了分数，可以继续进行。

另一方面，如果候选人回答第一个问题非常出色，他们很可能是该领域的真正专家，但直到他们对该主题的多个问题提供了有见地的答案，您可能不会对他们的掌握能力有信心。通常，识别专业技能比缺乏资格更需要更多的时间和询问。

当您知道已经听到足够的答案时，不要犹豫地礼貌地打断候选人的回答并转向下一个类别。您的目标是帮助候选人展示他们在您决定对该角色重要并选择在此次面试中评估的所有主题上的技能和知识。如果面试时间不够，让候选人过多地探讨一个主题会剥夺他们展示在其他主题上的能力的机会。管理面试的节奏是您的工作，而非候选人的工作。

高管面试

当候选人进入高管面试环节时，您应该已经确认他们具备您职位描述所需的技能，并且与您公司的文化相匹配。在大多数情况下，高管面试并不是为了筛选候选人，更多是为了让候选人有机会与高管会面并提问。

然而，如果候选人正在申请一个非常高级的职位，或者将直接向高管汇报工作，那么这次最后的面试可能比仅仅是候选人问答更长或更严谨。

背景调查

在进行背景调查时，您需要在面试过程的早期安排它们，以确保它们不会造成瓶颈，并且不要浪费时间对那些不会被录用的候选人进行背景调查。请记住，候选人有合理的理由可能会不愿意在整个过程中提供参考，以保护他们与参考人的关系。

时间安排

由此可见，参考核查几乎总是在面试流程的最后进行。为了避免因等待参考核查而延误一个工作机会，以下是一些建议：

- 在提供参考人资料后，立即安排与所有参考人的会议，可以并行进行。考虑到参考核查的简要性，最高效的策略可能是直接致电参考人，而无需预约。
- 考虑在完成参考核查之前提出聘用意向，但要明确告诉候选人，最终确定聘用的条件是参考结果符合预期。假设你在之前的筛选过程中做得不错，参考核查的成功率很高，所以很少因为参考面谈不合格而撤回意向聘用。
- 在进行参考面谈时，对于面试官的选择要灵活一些，不一定非得是技术人员。但是这个人需要对电子邮件高度响应，并且在日程安排上有相当的空余时间来安排参考面谈。 参考资料咨询通常应该简短并尊重参考人的时间和乐意提供帮助。大多数参考资料咨询给出的反馈从中立到非常积极。很少会收到明显的负面反馈，所以你的目标是快速区分中立和非常积极的反馈，确认面试过程中确定的任何优点/缺点，并继续进行。如果你在参考资料咨询中收到了明显的负面反馈，请非常注意并尽量获取关于批评的具体细节以反馈给招聘经理。

以下是一些参考资料咨询的示例问题：

- 你是在什么背景下与[候选人的姓名]共事的?
- 评估参考人的可信度：在你的职业生涯中，你管理过很多其他工程师吗?
- [候选人的名字]的最大优势是什么?

- 那时候[候选人的名字]的最需要改进的地方有哪些?
- 你会给他们在那份工作中的表现打几分（1-10 分）？
- 是什么让你给出这个评分?
- [候选人的名字]提到他们在那份工作中遇到了困难。你可以再详细告诉我一些吗?
- 在什么样的环境和管理风格下，[候选人的名字]会表现得最好？ 候选人中，哪一位最为成功?
- [候选人姓名]如何处理冲突?
- 如果有机会，你会重新雇佣[候选人姓名]吗?

发出录用通知

在你准备给某人提供工作机会时，你应该已经有一个强烈的观点——基于岗位描述和从你的焦点面试中得到的反馈，以决定以何种级别录用该候选人。从那里，使用预定义的职级范围，确定薪水/奖金/股权金额就相对简单了。（更多内容请参阅 1.4.2《薪酬和职级划分》。） 一旦您校准了您的报价金额，您应该决定如何呈现报价。特别是如果您的报价包括股权补偿，您应该认真考虑提供一份提供报价金额背景信息的电子表格。单独给出一定数量的股票是无法让候选人评估价值的。他们需要额外的数据点来评估您所提供的价值，包括总股本数量、股票行权价、最新公司估值等。

我在 ctohb.com/samples 上准备了一个样本候选人报价电子表格。

呈现报价

呈现报价时，您需要处于超级销售模式。理想情况下，您一直在向候选人推销，所以他们对公司和机会已经非常兴奋。无论如何，这对候选人来说是一件大事，所以确保给予它应有的重视。在解释报价的过程中，请保持积极乐观的态度，祝贺候选人，并强调您将一起度过的愉快时光和共同建设的伟大事物。同样重要的是，您要透明，并提前概述报价的所有关键点，特别是他们可能没有预料到或不习惯的事项，例如股权补偿或试用期。

我建议将报价分为三个部分：电话、电子邮件和晚餐。对于电话部分，我建议不事先安排时间而直接给候选人打电话。在此之前，您已经与候选人进行了大量的安排，所以没有必要再安排一次会议增加他们的焦虑感。或者，您可以书面告知他们您打算提供一个报价，并从那里安排时间，但这样失去了与他们在电话中得到好消息时的激动。我发现直接给对方打电话并一次性告诉他们消息更简单。 在电话中，您应该表达兴奋之情，传达报价的关键要点，并回答任何初步的问题。解释说，在电话之后，您将通过电子邮件向他们发送关于股权的书面材料，帮助提供背景知识，当然，公司还将向他们发送正式的书面报价信。最后，如果在物流上可行的话，安排与候选人共进一餐，进行更个人化、深入的对话。

入职

通常情况下，新工程师入职团队并不需要团队投入较大的资源；一个优秀的工程师最终会弄明白。然而，不采取任何行动会导致新员工体验不佳。这会延缓他们达到工作效率的时间，并且可能也会使您

更难确定您的招聘质量。换句话说，良好的入职流程使得三个目标优化：

1. **尊重员工：** 良好的入职体验有助于新员工融入您的公司和文化，并尽快提高工作效率。
2. **帮助评估招聘质量：** 良好的入职提供了对新员工和经理的结构化引导，包括明确的目标，当达成这些目标时，可以证明您为这个角色招聘得当。 它建立了你的文化：良好的入职培训强调持续改进的文化，有助于简化未来招聘的流程，并增强整体流程的可扩展性。

有许多正确的方法可以做到这一点。以下是一些相对简单且廉价的技巧和实践，我自己也使用过。请随意扩展或偏离这些想法。

童子军原则：入职培训

我鼓励您向您的经理、新员工以及在您的入职培训文件中强调，成功的入职培训是团队中的所有成员的共同责任，包括最近的新员工。根据您的招聘频率，入职培训文件往往会过时。如果新员工在材料中遇到不清楚、不正确或完全缺失的内容，明确告诉他们，您期望他们付出努力来澄清并改善下一位员工的文件。 团队入职与公司入职

对于公司中的任何新工程师，有些入职要素应该在全体新员工中保持一致。这包括高层流程、对组织文化的重视、新员工接收的文件类型，以及共享文件和设定入职里程碑的结构。你不希望前端团队有一个非常顺利的入职流程，而后端团队却一头雾水。第一印象很重要，入职是确保所有团队成员对你的组织有一个很好的第一次体验，并以一种一致的方式介绍公司价值观和团队最佳实践的机会。

这并不意味着入职的具体细节在不同团队之间是完全相同的。当有意义时，你可以有针对不同团队的不同材料，每个团队和个人新员工都应该有一个定制的入职计划和里程碑。

入职文件

使新工程师顺利入职有两个关键要素：教导他们了解你的文化和最佳实践，同时给他们一些结构和指导来进行工作。我更倾向于将这些分为两个书面文档：《工程指南》和《欢迎来到[公司名]工程团队的第一天指南》。 工程指南书籍

工程指南书籍将团队意见、最佳实践、结构要素和业务操作整合在一个文档中。它应该成为任何工程师都可以依赖的单一信息源，了解在整个工程组织中预期保持一致的选择和决策。对于在整个组织中保持统一的实践，要明确而深思熟虑。

团队变大后，团队的一部分逐渐会发展出自己独特的工作方式，这是有意义的。尽管如此，对于大多数小型/中型初创企业，比如少于 75 至 100 名开发人员的企业，遵守一套健康而一致的最佳实践能够释放出很多价值和效率。

指南书籍可以采取许多形式，但我更喜欢它以幻灯片/演示文稿的形式呈现。以下是指南书籍应该概述的一些实践例子：

软件工程

- 编程语言的选择
- 对于持续集成/持续交付的意见/要求

- 命名规范（代码中的大小写，合同中的大小写）
- 数据处理、保护、备份和安全的标准
- 使用源代码控制的意见（Git Flow，GitHub Flow）
- 测试方面的意见（种类、工具、测试的程度）
- 前端和后端认证授权的标准模式
- 网络协议标准（REST，gRPC，GraphQL 等）
- 全局要求（是否支持移动设备、响应式设计、翻译等）
- 认证框架及相关培训（例如 PCI、SOC2、GDPR）
- 其他编码细节：访问私有仓库、代码检查、静态代码分析、提交信息格式/风格。

工程流程

- 对节奏/仪式的观点（敏捷开发、看板、回顾会）
- 对技术文档/规范要求的观点
- 对如何使用工单系统的观点（什么是史诗故事？使用故事点吗？）
- 团队关注的任何指标（你们在测量周期时间吗？）
- 生产事故如何处理（PagerDuty？RCA 文档？） 新技术如何融入技术栈
- 缺陷和技术债务的流程

人员管理

- 如何进行绩效评估和个人评估/晋升的期望
- 对入职/招聘流程的贡献期望

该指南应清楚标明为一份活动文档，并设立一个明确的流程来提出、获得反馈和纳入指南的变更。例如，我采用了一种请求评论（RFC）的流程来进行更新。

欢迎来到[你公司的名字]工程，第一天指南

分发一份第一天指南是为了为新员工提供一些结构，给他们一个明确的任务清单，让他们在第一天与你的组织接触时能够了解公司文化、团队成员、流程和软件堆栈。当然，第一天指南应该提到《工程指南手册》，作为必读的第一天内容。此外，你的第一天指南还应包含以下内容：

如何获取登录/访问所需的系统的指引，包括：

- 源代码控制
- 任务管理
- 任何开发/阶段/生产日志
- 错误追踪
- 任何设计工具（Figma，Sketch）
- 文档/维基（Confluence，Notion 等）
- 内部通信（Slack，电子邮件）

- 公司硬件的信息（包括新员工是否可以选择笔记本电脑/手机），以及使用该硬件的期望
- 设置本地开发环境的说明
- 团队和公司组织结构介绍：他们的经理是谁，相关跨职能领导者，直接报告对象以及相关副总裁或高管
- 透明度的期望以及跨组织结构寻求帮助或升级问题的方式
- 技术架构的介绍
- 鼓励所有团队成员阅读的相关书籍，博客和其他书面资源

入职里程碑，又称为 90 天成绩卡

正如在招聘章节中所讨论的，招聘是非常困难的。即使最周全的招聘流程也无法达到 100%的成功率。换句话说，误聘是不可避免的。

应对潜在的不成功招聘的最佳方式首先是要有谦卑的态度，承认你的招聘过程并不完美，然后仔细考虑如何衡量新员工的成功并迅速采取措施纠正任何错误。这个过程应该事先对新员工透明，在说明期望的时候要清楚明了。经理们应该与新员工合作，确保他们的角色是相互匹配的，新员工在角色中开始感到熟悉，并且他们的表现与他们的雇佣水平相符合。在六十或九十天时，新员工和经理应该清楚地知道这些期望是否得到满足。

如果对于员工的是否成功存在争议，这是一个不好的迹象，说明这不行得通，你应该迅速考虑是否有团队中另一个更适合新员工的位置，或者双方都可能通过分道扬镳而过得更好。

成绩卡

新员工的经理有责任在新员工入职前确定和记录任何新角色的可衡量的里程碑。在第一天，经理应该与新员工一起过一遍这些里程碑，收集反馈，并合作确定这些里程碑是否公平且清晰可衡量。对于某些角色，这些里程碑可能很容易和明确，比如在支持角色中通过案例处理速度来衡量。对于其他角色，您可能需要更具创意，例如交付的功能数量或关闭的故事点。无论你选择哪些里程碑，绩效评估表应该做到以下几点：

- 在经理和新员工之间建立清晰透明的期望。
- 为新员工提供关于他们在头三个月的工作内容和衡量标准的指导。
- 提供明确的标准，以判断是否达到了他们角色的期望。

绩效评估表并不需要过长或过于复杂。关键在于，无论以何种形式呈现，在九十天后，雇员和经理可以查看绩效评估表，并就雇员的表现达成共识，确定是否为长期合适的选择。

关于九十天的时间长度，九十天通常是新员工入职的常用时间框架，但并不是硬性规定。在工程领域，学习并掌握技术、工具和产品需要较长的时间。反之，等待完整的绩效周期（如六个或十二个月）则会将一个潜在的不适合角色的员工留在岗位上太久，阻碍其获取所需的改进措施，从而导致公司浪费时间和生产力。正确的答案可能介于两者之间，具体的时间长度取决于您和您的经理。 关于处理绩效评估失败的问题

如果经过 90 天后，经理和员工都认为事情没有达到预期，或者对是否满足预期有分歧，就必须做出一些变化。这并不意味着你必须解雇新员工，但确实需要采取一些行动。在这种情况下，请考虑以下选项：

- 问题出在经理身上吗？这个人在其他团队或与不同的经理一起工作会更成功吗？
    - 如果你怀疑是这种情况，可以考虑调动到其他职位，而非解雇。
- 是否存在文化不匹配？
    - 在发现文化不匹配后，重新调整员工适应你的文化是具有挑战性且很少成功的。如果你担心在 90 天之后可能出现这种情况，最合适的选择几乎肯定是分道扬镳，而且候选人很可能和经理一样感到如释重负。
- 是缺乏经验还是技能的问题？
- 如果你雇佣的高级员工却只表现出中级水平的能力，你有试图将他们降职的选择。毕竟，如果他们无法以高级水平的表现交付工作，对其他员工来说保留这个人并付高级员工的薪水是不公平的。然而，要注意，降职是非常具有挑战性的。除非能够非常仔细地管理期望，否则降职往往会导致受伤自尊心，并最终对团队产生无益甚至有害的影响。

解雇新员工

通常情况下，如果在九十天内还不清楚一个雇佣是否能够成功，那么在一百二十或一百五十天后也不会奇迹般地变好，最好还是解雇他们。你应该像解雇其他员工一样，给予全额解聘费和尽可能多的善意。

我鼓励你对这次雇佣错误承担完全责任。如果是你招聘了他们，就承担责任；这意味着你的招聘流程并不完善。不要因此惩罚员工。初创公司行业标准的解聘费是四周工资，如果可能的话，还包括福利，并提供以你认为合适的方式帮助他们找到另一份工作。

入职时间表

入职开始于某人同意加入你的公司并签署录取信的那一刻。即使在新员工的第一天之前，你应该思考如何让他们成功。并不是每个新员工都会愿意在开始工作之前花时间了解公司或自己的职责，但根据任务或所提供的内容，很多人会自愿这样做。

我建议在候选人签署录取信的那天，向他们发送第一天指南和员工手册。如果你有公司阅读书单，现在是下单并将书籍寄给候选人或提供电子书/有声书的好时机。大多数候选人在第一天之前不太感兴趣阅读/编写代码，但了解你的企业文化或阅读高品质的商业/文化/工程书籍很少被认为是负担。你不应该要求这项活动，但通过提供选择，你可能会得到相当积极的自愿参与度。

在实际上的开始日期，候选人应该在早晨第一件事就与他们的新经理见面并签到。如果他们在到达之前没有阅读你提供的资料，就要设定期望让他们在第一天进行阅读。候选人在阅读介绍材料并设置环境/登录之后，应该安排后续时间审查 90 天绩效评估计划。这也是一个很好的时机来强调持续改进的观念，并鼓励候选人对他们的入职过程中的任何问题负责并为接下来加入团队的人员改进文档和流程做出贡献。

绩效管理

提高员工士气和促进积极的工作环境文化的关键之一是确保每个人清楚地了解自己在工作中的表现，并获得可靠的指导来提升自己在组织中的水平。任何绩效管理系统的目标都是尽可能客观公正地为员工提供透明度和结构。糟糕的绩效管理系统会导致令人不愉快或尴尬且缺乏动力的局面，而一个强大的绩效管理系统会激励你的团队，并鼓励大家一同提升水平。 差劲的绩效管理通常会导致负面结果。以下是两个例子：

个人 A 被提升了，作为结果，个人 B 感到惊讶并觉得被跳过了。如果经理在之后被 B 质疑时无法提供具体的决策理由，情况会变得更糟，导致 B 产生不信任感、被忽视的感觉和士气低落。

个人 X 在四级停滞太久，感到沮丧和士气低落，不知道怎样晋升到五级并获得相应的加薪。

你的绩效管理系统应该给每个人明确的工作状况，告诉他们需要改进的地方（以及如何改进），何时进行评估，以及如何考虑这些评估来做晋升和调薪的决定。 绩效管理和薪酬设计不应该完全由您作为技术负责人来进行。在这里，有很多可能导致公司承担法律责任的错误可以轻松避免，只要确保您的人力资源负责人在此过程中扮演重要角色。事实上，理想情况下，你的人力资源负责人应该在大部分蓝图设计中发挥作用，并仅在定义技术能力方面寻求您的帮助。无论谁主导，人力资源都是您的合作伙伴。

概述

绩效管理系统的核心是一个文件、电子表格或其他可行的文档，我将其称为能力矩阵（有时也称为影响矩阵或晋升计划），它列出了每个角色的技能和影响领域。能力矩阵提供了每个技能/影响领域在各个级别上的细化、具体化和期望。

例如，一个独立贡献者软件工程师的能力矩阵可以包括一项与编码/功能输出速度相关的项目。在一个 1 级到 5 级的系统中，该矩阵将指定对于 1 级工程师，在代码速度方面的期望是每周完成 X 个拉取请求，或者在每个迭代中能够关闭 Y 个故事点，具体要根据团队的实际情况来决定。理想情况下，每个描述要么是可以直接量化并具体说明的，要么是质量上可感知的，并由团队以一致的方式进行解释。其余级别的期望将逐渐增加，并在 5 级时达到非常高的代码速度标准。通过完整的能力矩阵，任何团队成员应该能够在每个类别中进行自我排名，并产生一组排名结果，这些结果与他们的经理或同事提供的排名非常接近。 一旦你已经写出了每个级别的描述，剩下的就是发布一个公式，将个人技能的排名总结成一个单一的工作级别。有了这个，你就拥有了一个透明、客观、可衡量的系统，任何员工都可以用来了解他们在职业表现上的表现，并确切地知道他们可以在哪些方面提高以升级。我在《绩效评估》第 101 页提供了一个样例公式来进行总结过程。

请记住，不同的角色应该根据对团队的不同贡献进行评估，并且应该有不同（虽然可能有重叠部分）的能力矩阵。特别重要的是要为管理岗位与个人贡献工程师创建一个独立的矩阵，以鼓励经理人在编码之外发展自己的技能。

创建能力矩阵

能力矩阵的细节影响到团队的每个成员，所以合理的做法是让团队参与制定这些细节。参考《迷你管理框架》第 33 页的团队决策模型部分，这绝对适合用草稿模型或者合作开发模型来做。

我推荐使用草稿模型：概述你希望在任何给定职位中看到的关键技能和影响领域，尝试填写大部分能力矩阵。然后向你的技术团队介绍这个想法，并让他们知道你希望他们对如何完善第一稿提出意见。为团队设定固定的时间，鼓励大家一起讨论矩阵，也许可以使用分组讨论各个类别。 无论你选择什么样的结构，都要明确地表述出来，并为其提供至少几个小时的保护时间，以便进行工作，并设定一个最后反馈的截止日期，以便将其整合并转化为团队的最终候选稿。

尽量设置不超过五个一般类别，以及每个类别中不超过三个领域。超过大约十五个技能/影响领域会使矩阵过于庞大，无法作为一种有效的绩效评估工具使用（并且肯定会变得过于繁琐，无法及时收集团队反馈）。

每个级别都应该有其评级系统和明确的绩效期望，或者可以与相邻级别共享描述（如果适用的话）。

我在我的网站 ctohb.com/templates 中提供了一个样本晋升计划。Codeacademy.com 也有一个很好的模板，位于 ctohb.com/competencies。

薪资和层次结构

我建议将补偿与能力矩阵等级系统挂钩，因为这样做会优先考虑两个目标：公平性和激励高绩效。

关于公平性，如果同一角色和级别的两名员工得到同样的补偿，那么这个补偿的公平性将取决于层级设置的公平性。如果团队共同为和相信能力矩阵的公平性做出了贡献，那么他们一般也会相信与该矩阵挂钩的补偿是公平的。

至于激励高绩效，将补偿与层级挂钩会在经济上激励每个人提升水平。如果能力矩阵设计得好且民主，你的团队将专注于真正有助于业务的技能/影响领域，这将使他们获得更高的级别（和更高的补偿），同时也加速你的团队的发展。

将级别转化为公平的补偿比大多数人想象的更微妙。最简单的做法是创建一个透明的电子表格，指出每个 X 级员工每年的薪资是 Y 美元，但这样严格的系统会带来一些问题：生活成本调整（也称为当地薪资标准）和非绩效导向的补偿奖金。

GitLab 在一篇很好的博文中解释了为什么他们支付当地薪资（请参见 ctohb.com/local，他们的补偿计算器也公开在 ctohb.com/gitlabcompcalc 上）。也就是说，处理当地薪资没有一个正确的方式，你应该考虑是否支付这些薪资对你的业务是否合理。如果合适，以透明和数据驱动的方式计算这些薪资标准。 将绩效水平转化为特定薪资范围，而不是确定的薪资金额，可以解决许多薪酬问题。任何一项工作都希望与市场价值相匹配，但是市场价值是如何确定的呢？一般来说，用于确定市场价值的工具和数据相对不太精确，最多只能给出 10-20%的范围。之所以不太精确，原因很简单：你公司的软件工程职位和其他公司的同一职位要求不太可能 100%一致。毕竟，你的代码库和工具也不是百分之百相同。

设立薪酬带还可以留出空间，在绩效管理系统之外增加薪资。非绩效因素的变化包括经验或者通胀调整。你还可以使用薪酬带作为一个大致替代，为生活成本调整留出空间，然后在组织制定更复杂的当地薪酬体系之前。

竞争薪资与市场价值数据

所以，你设计好了能力矩阵，并决定将等级转化为薪酬带。现在，你只需要计算你的薪酬带。在这个领域，你肯定希望你的人力资源负责人与你密切合作，以确保满足可能存在的任何监管要求。定义薪酬带应尽可能基于数据，幸运的是，通过一些现有的平台（无论是付费的还是只需要共享数据的平台），这些数据相对来说是比较直观的。像 Pave、Advanced-HR 的 Option Impact、Levels.fyi 和 Glassdoor 这样的平台可以提供丰富的数据集，可以根据公司的规模/形态/阶段进行筛选，确定特定职位的相关薪酬带。

职位名称

许多创业者会告诉你他们的组织非常扁平化，职位并不重要。在某些情况下，这可能是真的，但这是例外，而不是规范。在绝大多数公司，包括创业公司在内，职位使用存在着一致的趋势。给予职位会产生对级别和责任范围的期望。职位很容易给予，但很难取消，因此在确定工作描述或晋升时，值得考虑和慎重思考要给予的职位名称。

对于非行政职位，我首先鼓励你使用数字来确定级别，例如，Level 1、Level 2 等，通过能力矩阵（参见能力矩阵和级别，第 95 页）。一旦你知道每个级别的期望，就可以将级别与职位相对应。同时，不要害怕在职位名称后面加上数字后缀；使用像初级工程师 1 和初级工程师 2 这样的职位名称更容易和清晰，而不是发明一个新的形容词，它的意思是比初级工程师稍微有经验但不是中级水平。

工程师个人贡献职位

工程师个人贡献职位的职位名称非常简单，使用描述性的形容词来传达资历和责任范围的大小。大多数创业公司会使用以下三个主要职位：初级工程师、中级工程师和高级工程师。在高级工程师之上，通常还会使用像首席工程师、研究员和架构师这样的词汇，但它们的定义和层级往往不太一致。

高级个人贡献者通常拥有技术负责人的非正式职位。技术负责人意味着个人贡献者的一部分时间用于管理式的职责，但他们的主要责任仍然是进行工程工作。很少有人在简历或组织结构图上注明技术负责人的职位；对于更高级别的员工来说，这只是额外的职责，同时也是该级别资历的预期的一部分。如果技术负责人的主要产出是管理，而不是编码，那么他们应该进入管理轨道，承担经理的期望、职位、培训、教练等。

经理职称

管理职称比个人贡献者具有更微妙的涵义。最常见的职称是软件开发经理（SDM）或软件工程经理（SEM），有适当的资历装饰，例如中级软件工程经理或高级软件工程经理。SDM 或 SEM 通常负责一个工程团队，他们负责一个特定的功能或产品。

下一个级别通常是工程总监。总监对单个或高度相关的产品中的多个团队的表现，协调和输出负责。在大多数组织中，总监并不被期望承担战略角色。换句话说，总监不会制定基础技术方向或产品策略。

总监之上是工程副总裁（VPE）的角色。VPE 没有一个通用的实施方式。它可以是公司所有工程师的组织领导（替代 CTO），也可以是跨多个产品领域的战略技术领导。有时 VPE 向 CTO 汇报，其他时候向 CEO 汇报。然而，VPE 们共同的期望是技术非常高级、经验丰富、善于人员管理、良好的沟通者和战略思考者。

绩效评估、调查和晋升

不是所有在能力矩阵中的技能领域都能够由管理者或电子表格以定量方式进行评估，因此每个团队都需要一套绩效评估流程，可以从管理者和同事那里收集定性反馈。

挑战在于以一种能够与你的水平对齐的方式收集定性反馈。在本章中，我提出了一种方法论，可以用来收集反馈，并最终得出一个相对容易理解的评分系统，可以产生个人绩效水平。

评估周期

绩效评估需要大量的时间，可能会给人带来情感上的疲惫，对团队来说也是非常昂贵的，这些都使得管理层减少评估的频率。但员工希望能够得到更紧密的反馈环路和更多升职机会。平衡是在每六个或十二个月安排一次评估会议（更即时的反馈应在员工与管理者定期举行的一对一会议中进行）。请记住，为了让个人成长和展示成长，需要有足够的时间间隔。一般来说，六个月是一个安全的下限，能够平衡这些问题。

评估员工的选择

谁评价谁的问题并没有简单的答案。许多公司只是让经理来进行评价，仅此而已。虽然经理的反馈很有价值，但也可能会给偏见留下太多的空间，忽视同样重要的同事和直接报告者的观点。运行一个公正且全面的流程最简单（尽管稍微更昂贵）的方法是让每个员工接受多次评价，包括这些其他观点，通常称为 360 度评价。

在这里，我建议使用稻草人技术（见迷你管理框架的团队决策模型部分，第 33 页）：每个经理应该列出那些对该团队成员有足够接触的直接报告者和同事，并在最终确定名单前与员工进行对话并获得反馈。经理还应该跟踪每个员工被要求完成的评价数量，以保持请求的可管理性。

评价问题

您的问卷应与您的能力矩阵相一致，并鼓励评价者明确引用矩阵。

同一组问题可以适用于每个矩阵类别：

- 这个人在这个领域有哪些表现出色的例子？
- 这个人在这个领域有哪些需要改进的地方？你认为这个人在这个领域的表现水平如何？

请注意，从无意识偏见的角度来看，最好在要求评审人给出级别之前让其列举例子。另一种做法可能会鼓励评审人选择一个级别，然后选择一些例子来证明他们之前选择的级别。

在最后加入一些更高级/较宽泛的问题可能也是有意义的：

- 你对与这个人合作感到多么渴望和兴奋？ （等级：一点都不兴奋至非常兴奋。这个问题来自 Netflix 的 Keeper 测试 [ctohb.com/keeper]）
- 这个人目前处于 X 级别。你觉得他们准备好可以晋升到 X+1 级别了吗？
- 他还有其他哪些优点你想要突出强调的吗？

- 还有哪些方面需要改进你想要强调的吗?

评审格式

您可以使用正式的评审工具（也称为绩效或文化管理工具，如 Culture Amp 和 15Five）进行评审，也可以不使用。当然，专门的工具可以节省时间，并且能够快速为规模较大的团队提供支持。至关重要的是要保持所有个人反馈的匿名性，除了指明哪些评分来自管理层（我们将在后续过程中将这些评分单独用作与同事评审进行核对）。

结果计算

一旦评论提交完成，每个人针对每个能力矩阵类别应反映一组分数，并理想情况下将评分分为同事、直接下属和经理评分。下面是一个示例矩阵分数：

<TODO：在此处插入表格>

现在的挑战是如何将这些分数合并到最终的职位级别计算中。在此计算中有一些关键注意事项：

- 保护流程的完整性（例如，员工未与同事串通，人为地增加或降低任何人的分数） 确保公式公平且可一致计算。

确认经理对员工的影响所持的观点与同事/下属的反馈相一致。

决定矩阵中的所有类别是否权重相等或不相等。

以下是我推荐的确定级别的方法：将级别赋予在累积分数达到 66%或更高的水平。给定级别的累积分数是所有得分达到该级别或更高级别的百分比。最低级别的累积分数始终为 100%，第 2 级将为 100%减去第 1 级的投票百分比。第 3 级是 100%减去第 2 级和第 1 级的百分比，依此类推。

在上面的例子中，此人将是第 2 级。在第 2 级，他们的累积分数为 90%。根据 66%的规则，他们离第 3 级（60%）仅相差两票之远。这一点可以在辅导中用来鼓励进一步提高以获得晋升，或者用来为薪酬档位进行调整提供理由。 当你完成整体计算后，如果你成功地跟踪了经理的分数，你可以将同样的公式应用于经理的分数，以单独计算经理在累积分数和所有同行评审的累积分数之间的差异。在这里，大的差异，也就是超过一个等级的差异，需要密切关注和额外的审查，因为这意味着经理和同行对个人表现有明显不同的观点。或者，这可能表示投票/评分过程中存在某种不规则性。

结果讨论

我鼓励经理在实际的 1:1 绩效评估会议之前提供绩效评估数据，以最大限度地提高会议本身的价值。最好给个人提供数据并给他们一些时间来思考，这样当会议发生时，他们可以全身心地参与其中。

绩效评估会议的议程应该简单明了：

1. 讨论在 1:1 会议中没有预料到或通常不涵盖的任何优点或缺点。
2. 汇总一小份要在下一次评估期间努力改进的重点领域清单。许多领导者主张只有一个重点领域，但我看到一些人在给定的时间段内以多种方式成长，所以两个或三个重点领域是一个合理的上限（适用时）。

3. 确立一个时间表，使经理和员工定期关注这些重点领域，并确保在下次审查之前有进展。

薪酬调整发布

在许多公司，审查反馈和薪酬变动是在不同的时间处理的。我不认为在绩效审查会议上讨论薪酬变动是至关重要的，甚至有利的，因为这可能会分散注意力，从而影响本来可能是具有挑战性但重要的讨论。关键是在绩效审查过程之前设定对薪酬变动的决策、沟通和实施的期望，这样他们在进入会议之前就知道可以期待什么。

绩效改进计划 (PIPs)

绩效改进计划，通常称为 PIPs，是一种旨在改善员工绩效或解雇员工的工具。PIPs 的一些关键方面有：

- 您团队中的每个人都应该了解贵公司的 PIP（绩效改进计划）流程。
- 许多人上班时心里都有同样焦虑的问题：今天会不会被解雇？我表现得够好吗？
- 知道自己所在公司有一个正式的解雇员工流程，其中包括纠正机会，可以帮助减少这些焦虑。
- PIPs 应该只在真正努力解决和提高绩效不足的情况下使用，因为它们需要员工和经理共同付出大量的努力。
- 经理应该花时间仔细填写 PIP 文件，确保清楚地阐明绩效不足的问题，提供定量、明确的评估标准和结构，尽可能提供支持和指导，以帮助员工改进。
- PIPs 应该允许合理的时间段来展示改进，例如，对于个人贡献者为 30 天，对于高级员工或经理为 60 天。
- PIPs 应该始终包括完整的书面版本，不仅为了员工和经理之间的清晰沟通，还为人力资源/法务提供了相关文件，以备后续查询之用。 以下是绕过绩效改进计划（PIP）直接终止雇佣的几种情况：
- 公司政策违反、人力资源违规行为、不当的职场行为等不应该通过绩效改进计划来纠正，而应该采取零容忍的态度。
- 某些技能缺陷不适合通过绩效改进计划来纠正，例如在某个特定主题上缺乏良好判断力、与企业文化不匹配或在关键技能领域缺乏经验。这通常是针对管理层或高级职员岗位的考虑因素，因为良好的技能判断力对该职位至关重要。

调换职位

在制定绩效改进计划或终止表现不佳的员工之前，值得问一下自己是否有可能将该员工调换到公司的另一个团队或角色上，看看是否更合适。如果表现不佳是基于技能方面的，而该员工具备在不同职位上能够更好发挥的技能，那么调换团队可能非常有成效。如果表现不佳是基于企业文化方面的原因，那么你可能会发现双方都没有转部门的动力。

能力出众的讨厌鬼

出色的怪人是一个行业标准术语，用来描述那些在个人工作效率上高产，但其存在会影响周围人的士气或生产力的人。他们通常被描述为有毒的个性。

由于他们有时出色的个人工作效率，选择解雇一个出色的怪人可能会感到困难或错误。几乎普遍的建议是无论如何都要解雇他们。作为经理，只要你允许有毒行为持续存在一天，就可能会增加团队对你的怨恨。无论个人生产力有多高，都无法弥补保留有毒个体对公司文化和作为经理的信誉的打击。

解雇

你的公司应该有明确的程序来实际解雇员工。我最好的建议是：遵循这个程序。如果处理不当，这可能成为公司的重大责任。关键考虑因素包括：

文件记录： 确保您有足够的文件记录来证明解雇该人员的决定，并证明解雇是基于绩效或其他正当原因。

时间安排： 一旦决定解雇某人，尽快执行。常识是，在解雇某人之后，经理们通常更少担心是否该解雇该员工，而更关心他们是否等待太久。从机械的角度来看，无论选择星期几解雇员工，对结果并没有太大影响，但如果你能够将其安排在月初而不是月末，就可以为员工提供额外一个月的公司医疗保险福利。

见证人： 确保经理和人力资源部出席真正的解雇会议并作为见证人。会议应该非常简短而直接，关于解雇安排的大部分后续问题应由人力资源部回答。

离职程序： 在解雇之前，提前制定计划，确定何时关闭员工对公司系统的访问权限，并取回任何公司硬件设备。

赔偿： 解雇员工往往不仅是公司/管理层的失败，也是员工的失败。解雇某人不是发泄怨气或小气的地方；这个人已经在你的公司投入了时间和精力，你应该尽一切努力使其在下一份工作中取得成功，包括提供行业标准的赔偿方案（范围有些广泛，从普通员工几周的赔偿到高级执行官两到三个月的赔偿，赔偿方案通常还考虑到员工的任职时间)。

团队组成

初级人才和高级人才在影响力上的主要区别在于他们能够可靠地解决不同类型问题的一致性。随着工程师经验的增长，他们在更广泛的领域中的判断和决策能力得到改善。同样，你应该期望更资深的人才能够开发出更少缺陷、持久性更长、更能适应需求变化的解决方案。这并不是说每个人都必须是高级人才；事实上，大多数项目都很少涉及构建全新的绿地解决方案。

对于任何给定的团队来说，合适的组合应该考虑到要完成的工作类型，并因此慎重地组建团队。

资历构成

如果你的代码库是：

- 非常新的，需要大量的架构和基础契约的建立。

- 非常陈旧、维护状况差或设计得不好，被认为难以工作，简言之，是一个老旧的代码库。

那么你的团队应该更加倾向于拥有更多的高级工程师人才。

- 在需求方面有意义地变化，尤其是如果新的需求与旧的需求看起来不太相似。
- 使用需要验证的新工具、技术或模式来解决问题。
- 需要建立新的工作模式/方法，特别是在生态系统中没有明确规定鼓励健康模式的情况下。

技术专业化

在大多数初创公司的第一天，团队通常只有两三名工程师。有三个人的团队对于专业化的机会有限。有二十个技术类别要处理，但只有三个人，所以根据抽屉原理，至少有一个人，更有可能是所有三个人都会承担多种技术工作。换句话说，在公司早期，每个人都要扮演多样的技术角色。随着公司的发展，加入更多人员的团队，您和您的员工将会发现更多专业化的机会。

那么，你已经融资并且首次扩大团队，你如何决定是否需要前端工程师、后端工程师、DevOps 工程师等等呢？以下是一些一般性的指导原则： **听取团队的意见:** 目前从事工程工作的人很可能会在哪些地方存在最大的低效率和最需要帮助的方面发表意见。作为经理，你的工作就是接受这种观点，并进行未来的推测。这是团队指出在两个月内会消失的问题吗？那么雇佣某人是不合适的吗？还是问题有系统性的特点，可能长期存在?

寻找影响生产力的因素: 如果你的团队大部分是前端工程师，而你在后端可靠性方面遇到困难，那么这应该是你需要后端或者 DevOps 工程帮助的一个信号。同样的原则适用于测试、开发者体验等方面。

在规模化中专注: 在你的团队人数超过十几人之前，高概率情况是，你最好拥有一个以多数通才为主的团队。

以下是根据创业经验的团队构成的一些粗略数字:

- 团队规模 1 5
- 你的团队都是通才，最多只在前端、后端和移动端专长。
- 团队规模为 5 至 15 人。
    - 你的团队可以根据产品或者一般技能领域进行专长划分，例如后端、前端架构、前端设计、DevOps 和测试等。
    - 当你的团队扩大到 15 人以上时，你可能会考虑专门的测试和 DevOps 资源。
- 团队规模为 15 至 30 人。
    - 到了这个阶段，你应该有真正的专业化，并只招聘在软件工程的子领域具有专业知识的人。
    - 在这个阶段，工作方式的任何低效都可能会对整个团队造成很大的成本损失，所以确保你在投入人力或时间方面，确保开发人员能够完成工作、他们的工具能正常运行，以及操作流程得到简化。

- 团队规模为 30 人以上
- 在这个阶段，有很多将团队分解成较小单位以确保工作效率的方法。如果你有多个产品线，将团队成员与特定的产品线对齐是一个相当自然的组织起点。
- 在这个阶段，许多公司使用“Pods”的概念，其中每个 Pod 都有一个专注领域，并由由具备多种能力/跨职能的团队组成，能够独立执行该领域的任务。

项目维护：两支船员的理念

如果你正在发布面向终端用户的软件，你的工程团队必须在新工作和处理来自活跃客户的支持票据之间取得平衡。如果不加以控制，处理支持票据的需求可能会成为团队的一大分心，损害效率，消耗士气，并令你最优秀的人员筋疲力尽。解决这个问题有很多正确的方法；重要的是你认识到它对团队的影响，并设计出解决方案来帮助他们提高生产力，并推动出色的客户结果。

微软在这个主题上发表了一篇很棒的文章，标题为《构建高效的团队》（ctohb.com/teams），介绍了他们所称的“两支船员模式”。这个模式概述了功能船员和客户船员。

功能船员专注于未来，开发新功能。客户船员专注于现在，处理活跃客户的问题，诊断错误，并优先考虑站点健康问题。 客服团队的其他名称可能是维护团队或二级支持团队（其中一级是非技术客服人员）。

将维护工作分配给专门的团队有许多好处：

- 它允许一个专门的团队随时监控客户队列，对重要和紧急的问题进行分类和解决。
- 它使得你的特性团队能够 100%地专注于未来，不被客户支持工作分散注意力。
- 它允许在客服团队内部发展专业化，在处理问题方面建立工具和专业知识，使问题的处理成本随着时间的推移降低。 它为个人工程师，尤其是初级工程师，提供了另一条职业道路，可以在您的团队中学习并提高水平。

关于两个团队方法，我经常被问到的第一个问题是：一个人在客户团队中会待多久？有四个确定客户团队任期的方法：

* 将客户团队作为一个永久的、独立的团队或部门。

* 对专注于支持和调试的工程师，您已经发布了工作描述。请注意，对许多工程师来说，只专注于调试的工作可能听起来不太理想。对我来说，强调数据录入或会计的工作描述听起来非常不愉快，然而有很多人喜欢甚至追求这些工作。

* 不要假设仅因为您不愿意从事该工作，就没有其他人可能对前景感到兴奋。特别是，在客户团队工作可以让工程师接触大量的代码，通常提供与客户交流的机会，并且涉及较少由产品驱动的截止时间压力，这些都可能吸引合适的候选人。

- 为工程师制定一个明确讨论过的职业发展轨迹，从客户团队开始，经过一段时间（通常为 12 个月或更长时间）后转移到功能团队。

- 工程师定期在这两个团队之间轮换。上面提到的微软博客文章建议每周在这两个团队之间交换一些团队成员。
- 将客户团队定义为临时团队。这可能意味着客户团队本身不是全职存在（例如每月只有一周时间），或者团队成员在客户团队和功能团队之间不断轮换。 我建议只为团队或产品中的客户问题足够昂贵需要这样做的团队创建一个专门的、永久的客户团队。如果你在一个支持需求较高的业务中，一个专门的客户团队可以为工程效率和客户满意度提供倍增效果。

团队组织

一般来说，技术组织图的组织方式通常有两种：职能组织和产品组织。职能组织是围绕团队成员从事的工作类型组织的，比如前端工程、测试或特定的内部服务。产品组织，有时称为业务单元，是围绕特定的业务/产品焦点组织的，比如企业核心应用团队或消费者移动应用团队。团队组织方式对团队的合作、生产力和士气都有重要影响。大多数情况下，产品组织的团队，通常被称为 Pods，是正确的选择。

设计组织图时，你应该考虑你要优化的目标。对于初创公司，组织图的主要目标是确保需要紧密合作的不同人能够通过组织结构得到支持和鼓励。实现这一目标的最佳方法是将所有直接为产品的可行性和成功作出贡献的人员组织在一起，让他们对一组共同的目标负责，并发展对该产品的共同所有权的意识。

当你的团队规模较小，且只有一个产品时，如何组织的问题并不重要，因为所有成员都是在同一个产品上进行跨职能团队合作的。而当团队规模扩大到 12 人或更多时，你需要开始更加有意识地定义什么是 Pod，并找到一个易于理解并与产品现实相结合的方法，然后将团队拆分为 Pods。

Pod 的优势包括更强的团队凝聚力、自治权、责任感和整体执行速度。然而，这种方式也有权衡之处。让一组工程师长期从事一个产品的工作可能会产生知识孤岛。随着组织规模的扩大，产品团队更有可能做出优化局部的架构决策，导致重复工作或计算资源的低效使用。如果你预见到这些问题的出现并做好规划，那么当这些问题得到良好管理时，带来的痛苦远远超过独立、高绩效 Pod 所带来的好处。

管理远程团队

有足够数量的成功的 100%远程软件工程团队，这无可辩驳地表明远程组织是可行的。这并不意味着所有远程团队都成功，或者说完全建立一个远程文化并不容易。我花了将近十年时间管理远程团队。以下是适用于大多数远程管理场景的一些建议。

文档化

远程工作意味着没有人坐在你旁边回答问题，但这并不意味着问题会消失。这些问题仍然会被问到，只是现在失去了社交背景，所以问题可能要等一段时间才能得到答复。或者，现在他们会立即提出问题，引发各种通知，并分散注意力，而不是等到对方休息时问问题。拥有一套健全的内部文档，配备有效的搜索功能，可以加快答案的速度，减少一对一上下文切换的次数，从而降低完成工作的障碍。

异步工作

将远程工作变成资源而不是负担的一个很好的方法是采用异步工作方式。强大的异步文化减轻了时区不匹配的负担，减少了在远程会议中花费的时间/处理远程上下文切换的次数。（有关异步沟通和异步工作的价值，请参阅《过度沟通的好处》，第 20 页。）

面对面会面

尽管视频通话非常有用，但它无法替代团队共进餐的面对面会面。面对面会议建立的纽带通常能够持续很长一段时间，因此即使是偶尔的面对面会议也能够在数月内改善团队成员之间的社交关系质量。通常来说，一个团队每季度面对面会面一次有助于维系这些关系并减少远程社交上的挫折感。

时区重叠

对于在同一个项目上工作的团队，我总的建议是要有至少四个工作小时的重叠时间。这样可以为任何定期安排的会议留出足够的时间，也为临时对话和问题提供了机会。

创造社交机会

一般来说，人们在视频通话时的默认模式是在通话期间同时进行多项任务，然后尽快挂断通话。一旦工作对话结束，每个人都会挂断电话。这不是人们在面对面交流时的互动方式；例如，会议结束后，在返回办公桌时，你会在走廊里谈论体育。会议前后的社交时间对于人们建立关系和信任非常有价值，除非领导和企业文化积极支持，否则这种社交时间不会发生。一种经济简便的定期做法是通过提出一个轻松、围绕房间逐个进行的破冰式问题，例如你能分享一条个人和职业上的好消息吗？来支持远程社交建设，可以举办远程欢乐时光、虚拟团队晚餐和社交活动。新冠疫情后，有许多在线社交组织者会组织远程活动，从数字赌场之夜到虚拟逃脱室等各种活动。

打开摄像头

根据不同的文章，70-90%的交流是非语言的。在视频通话期间使用网络摄像头虽然不能完全弥补这70%的交流，但无疑比完全没有视频要好。现在网络摄像头如此便宜，没有理由不在公司内部获得这种额外的社交交流带宽。经常的反对意见是员工担心别人会对他们在摄像头前的形象做出评判。如果您面临这样的担忧，我鼓励您对在远程环境中希望获得尽可能高质量通信的业务理由进行思考，并对任何对他人外貌发表不雅言论的员工采取零容忍政策。此外，为人们创造适当的准备自己的空间也会有所帮助，例如允许/鼓励虚化背景，并偶尔关闭视频以处理现实事务或整理一下自己。

录制会议

记录的会议提供了一个很好的方式，使员工在不需要停工的情况下就能了解一个主题。一般来说，你会发现很少有员工或候选人反对录制视频通话的想法。录像也是一种很好的方式，可以让更多的人了解其中的内容，超出了当时的舒适范围。例如，你可能希望一个由四五个人组成的招聘团队观看应聘者的面试，但同时在房间里有五个人可能会让人感到压力。录制一对一面试的视频是一个绝佳的方式，可以确保每个人都有机会评估应聘者，而不会在面试过程中因人数过多而产生尴尬或不公平的情况。

领导责任

作为高管，你的领导责任超出了开发和发展工程团队的技术范畴。你应该努力帮助公司整体取得成功，尤其关注技术如何为成功做出贡献。这意味着要关注技术与公司其他部门之间的协作情况，促进与内部和外部团队之间的良好合作，并建立良好的技术和产品开发管理流程。

产品和设计团队

作为技术领导者，你有责任确保你的工程团队与产品和设计团队高效合作，即使这些团队向其他人汇报工作。在设计这个流程时，通常情况下，各个团队之间的工作流程不应该是互相扔东西过墙。产品、工程和设计团队之间的良好互动需要共情和理解。了解其他团队在做什么，他们面临哪些挑战，你和你的团队如何让他们的工作更轻松？下面我会简单介绍这些其他职能，并提供一些建议，如何高效地进行合作。 设计系统

设计团队理想情况下希望软件工程团队能够准确地实现他们的工作，像素完美无缺。没有任何结构或约束的情况下，实现像素完美通常是昂贵甚至不可能的；然而，一点点共享的理解和设计系统可以大大降低成本。

设计系统是一套管理可复用组件和模式的设计标准，用于大规模的设计。大公司往往会创建自己的设计系统。例如，Atlassian 公司在公开平台上提供了自己的设计系统（ctohb.com/design）。作为一个小型创业公司，设计系统的创新可能不是你成功的关键，所以你可以选择一个现成的设计系统。现在你可以选择各种丰富功能和不同美学风格的现成系统，很可能可以定制以适应你的品牌。

设计系统不仅为设计师提供了一套节省时间的规范和组件，还经常提供对各种前端语言和框架的原生支持。例如，Material Design 在 Figma（ctohb.com/figma）上发布了设计系统，同时提供了一套 JavaScript react 组件（mui.com）和 angular 组件（material.angular.io）。

通过采用像 Material（或 AntD、Chakra UI、Blueprint、Bootstrap、Semantic UI 等）这样的系统，我们不仅可以获得一套预构建的技术组件，还可以获得一个与设计师工具紧密集成的系统。通过将相同的系统与工程师和设计师的工具集成，你将确保设计师创建的内容能够与工程师可用的组件清晰地对应起来。这减少甚至消除了工程人员进行自定义样式或前端用户界面的需求，并使设计与像素级的匹配变得轻而易举。 除了在设计和工程之间保持一致性带来的效率提升外，大多数设计系统组件实现还考虑或自动解决其他设计优先事项，例如辅助功能（针对屏幕阅读器和色盲的颜色对比度管理）、符合用户界面标准，甚至原生支持深色模式。

PRD 和规格

产品需求文档（PRD），有时也称为产品规格或简称为规格，是产品开发过程中的重要组成部分。请注意，产品规格具有不同的目的，是与技术规格（请参阅技术规划和规格，第 164 页）不同的文档。有许多关于 PRD 的方法论和模板。像 lennysnewsletter.com 这样的来源经常归纳一些最常见和全面的规格。PRD 有共同的目的，就是描述问题背景以及为项目或功能提供理由。PRD 通常包含需要满足目标的要求列表。大多数 PRD 将功能的实现方式留给技术规格。

与技术规格一样，PRD 是一个不断发展的文档。随着团队对问题的了解越来越多，或者外部环境因素发生变化，这些文档可以和应该进行变更和更新以匹配情况。我鼓励您和您的团队将这些文档作为不断更新的真实来源，记录您在产品开发过程中的考虑和最终决策。

EPD

EPD 是工程产品和设计的行业术语缩写。这意味着这三个部门都对产品开发生命周期至关重要，并且需要健康地合作才能取得出色的结果。从企业的角度来看，这三个部门共同负责生产出顾客喜爱的产品，因此有一个单一的人员设定了统一的业务目标，为三个部门确定了方向是有帮助的。

产品管理与项目管理

产品管理和项目管理是具有明确含义的行业术语。产品经理负责产品的设计和创建，以及产品应满足的关键绩效指标（KPI）方面的责任。Melissa Perri 的《逃离建设陷阱》是一本深入探讨了一个优秀产品经理在组织中所扮演的角色和影响的绝妙资源。项目经理负责指导团队内部组织、管理内部沟通，并遵守路线图和最后期限。

在你的创业初期，作为首席技术官，你可能需要充当这两个角色。在你的招聘路线图的早期阶段，你应该计划拥有一个优秀的产品经理，可以分担你的一些责任。有些产品经理擅长项目管理，而其他人则对此不感兴趣，因此在项目管理方面不投入时间和精力。可以说，在创业初期，无论哪种方式都可以；在小团队中，松散的项目管理所带来的后果是很小的。然而，随着团队规模的扩大，形式化的项目管理变得更加重要，如果你的产品经理未能担任该角色，你应该考虑补充一个项目经理。

我的一般建议是，在 EPD 团队扩大到大约 20 人左右时正式委派项目管理职责。这可能意味着你正式承担这个角色，让产品经理担任，或者雇佣一个专职的项目经理。你将希望在设立项目管理流程的早期阶段参与其中，确保为项目管理而建立的机制支持你希望建立的文化，并对所有相关方持同理心。

管理经理和经理培训

业界有一个说法，即你的公司最终是由你的中层管理人员来运营的。这意味着，尽管高管制定了任何战略和流程，但最终对产出的数量和质量产生最大影响的是中层管理人员。中层管理人员雇佣个人贡献者，设定他们的日常目标，并对其质量标准负责。最优秀的中层管理人员是小型高管，专注于文化建设，打造协作团队，并努力使他们能够做出最好的工作。因此，作为高管，您应该花费大量精力来聘请、管理和培训这些中层管理人员。

培训经理的复杂主题需要一本专门的书，这需要时间去掌握的一项技能。尽管如此，我可以给您提供的两个最重要的课程是：树立自己的榜样，并建立持续的管理学习文化。从您雇佣或提升某人到管理职位的那一刻起，您应该明确表明您的期望是他们将致力于投入时间和精力来提升自己的管理技能。您会尽力让他们轻松做到这一点，将管理培训包括在他们的个人目标和发展计划中，为他们提供提升管理技能的资源，帮助他们解决管理问题，并传授您所掌握的一切知识。

管理培训并不一定要过分繁琐和昂贵。如果预算允许的话，我强烈建议为您的经理们聘请外部管理教练。内部经理每月进行书籍阅读/回顾也可以有效地启动持续发展的良性循环。

财务和预算

Translated Content: 财务和预算 你的角色的一部分很可能是承担软件工程部门（有时包括设计和产品部门）现在和未来的成本责任。作为负责任的预算管理人员，你应该了解过去在各项事务上花费了多

少，并且你的公司总体上为未来分配了多少。最重要的是，你需要制定一个计划来合理地进行预算支出。

总体来说，技术部门中最大的两个科目是人力资源（工资表）和基础设施/软件即服务（SaaS）。要注意的是，员工的实际现金成本不仅仅包括他们的薪水，还包括福利和工资税收。一个很好的经验法则是，员工的现金成本比他们的薪水高出 20%；这个百分比被称为负担率。

预算

大多数财务团队使用先进的会计和预算软件来管理公司的账目。如果没有，他们会有一张远比你作为 CTO 所需的更大、更复杂的公司范围电子表格。根据我的经验，财务部门通常不愿意让财务以外的人对核心预算系统进行更改，所以除非他们给你一些起点，否则你需要为你的部门建立一个财务模型。

考虑到你部门的成本相对可预测，并且集中在几个科目上，你制作的模型不需要非常复杂。我建议你维护一张电子表格，其中包括以下内容： 工资表 软件即服务/销售成本（CoGS）表 基础设施表 其他表（包括差旅费、硬件费用）汇总表

您可以在 ctohb.com/templates 上找到一个样本技术部门预算电子表格，大部分模型都已预先设置好。

与财务总监合作

如果你的公司像大多数初创公司一样，你的部门将是最昂贵的部门，而你的首席财务官应该对此心知肚明。以下是一些建议，这些建议对你很有帮助，也会让你的首席财务官成为你最好的朋友：

- 帮助你的首席财务官了解资金是如何被使用和将会被使用的。尽可能地避免出现意外。提前为硬件成本、差旅/会议费用和云服务费用提供指导。
- 为你的部门设置一个预算，并随时更新它以反映预测的变化。
- 定期根据财务部门提供的实际数据更新预算，并确保理解和管理预测与实际之间的差异。
- 设立一个招聘计划，并包括预计的薪水。
- 追踪 SaaS（软件即服务）账单通常是一项负担。考虑使用信用卡对账单分析工具（例如，SaaS 管理平台，简称 SMP）或聘请一位助手定期对这些费用进行分类和对账，以与你的预算保持一致。
- 财务部门通常非常关注成本归属，以区分，例如作为销售成本的成本。在你的预算中为每个项目提供一个粗略的解释，这可以在财务方面赢得朋友。

衡量工程效率/健康程度

迄今为止，我还没有遇到一个能够持续且有用地量化实际工程成果的技术团队或领导。这包括将速度作为已完成任务估计之和来衡量。（关于技术估计的不可靠性，请参见《工作流程》第 160 页的估计部分的讨论。）

然而，我确实看到许多公司有效地衡量工程健康状况和影响工程速度的因素，即周期时间和工作时间分配。

周期时间

周期时间是从构思到实现功能所需的时间的衡量，通常可以细分为以下子阶段：

- 编码所花费的时间 开始进行代码审查的时间 代码获得批准的时间 部署代码的时间

一般来说，低周期的团队更高效，能够快速迭代、创新并给客户提供价值。有许多工具可以帮助测量周期时间，包括 LinearB（linearb.io）和 Code Climate（codeclimate.com）。LinearB 已经发布了一套基准数据，使用了来自成千上万个工程团队的数据，用于衡量与周期时间相关的指标。

工作时间分配

测量工作时间分配的想法是，虽然测量完成了多少工作很困难，但相对而言，测量团队成员花费时间在哪些类型的工作上相对容易。得到的信息在某种程度上是有用的。例如，如果一个团队大部分时间都在处理错误，那么改善软件质量并减少这部分时间的假设，将会导致更多的时间分配给开发新功能，从而在整体健康和速度上获得改善。

实际上，可以通过从票务系统获取报告或定期进行轻量级脉冲调查来以半自动化的方式测量工作时间分配。 筹款和尽职调查

一般来说，作为首席技术官，你在筹款和尽职调查方面的角色相对较小。在大多数初创公司中，首席执行官和可能还有首席财务官承担了大部分的工作。你可能在整个过程的后期参与进来，当投资者对公司进行尽职调查时。很多（但不幸的是并非全部）尽职调查中的请求实际上是关于一个良好组织的工程团队已经在日常业务中维护的信息。请关注以下内容，并准备好提供给尽职调查：

- 组织结构图
- 部门预算
- 所有工程部门已创建和维护的产品的完整描述
- 工程路线图（通常他们寻找的是短期/中期的路线图）
- 技术债务的主要领域清单，我称之为技术债务资产负债表（参见《技术债务》第 145 页）
- 高级系统架构图 软件分发和更新的全面描述，无论是作为 SaaS 还是作为版本化的桌面/移动软件系统的高级描述，包括它们的托管方式和您的安全实践 软件许可方面的信息，包括对公司代码的扫描，确认没有违反许可协议或使用未经许可的专有软件

投资者聘请第三方公司进行技术尽职调查并不罕见。这些调查可能涉及与您以及团队的其他几位高级成员进行面谈或会议。准备好讨论您的工程流程，评估团队的整体生产力，并对系统的部分代码进行演示。

我建议您在与这些审计人员交流时坦诚相待。他们已经审查过很多科技公司，他们非常了解每个工程团队都存在技术债务和一些团队自豪感更强的系统部分。投资者对您的优势和劣势了解得越多，他们就能更好地支持并促使贵公司团队不断改进劣势。

供应商管理 一般而言，作为首席技术官，你将负责寻找、与第三方技术供应商进行谈判和管理的责任。对大多数人来说，这是一项令人不愉快但必要的工作，因此往往没有太多考虑。然而，做好这个

责任可以为企业带来显著的成本节约。在这里，我将逐步为你介绍典型的 SaaS 谈判/签约过程，并提供一些提高效率和节约成本的建议。

自助注册工具

通常情况下，你或你的工程团队中的一员会提出需要某种工具，如果是你的工程师或经理，他们会要求你批准开支或者让你注册该工具。许多工具的成本微不足道，有简单的自助注册流程。我建议你与预算和财务团队合作，设定一个门槛，低于此门槛的工具可以由你的经理直接自行注册。对于超过成本门槛或没有自助注册的工具，你需要与企业供应商进行发现和谈判，这通常包括四个步骤。

企业工具

你现在面临的是一项企业销售过程。在联系供应商的销售团队之前，请确保这家公司是最适合解决你的问题的公司。在大多数情况下，我建议你创建一个电子表格，在该领域对所有供应商进行一些尽职调查，然后筛选出两到三家最佳的供应商。即使其中一家是明显最好的选择，了解该领域的情况并掌握谈判知识、筹码和 BATNA（最佳替代谈判协议）也是有益的。

销售资格认定

当您首次联系一家企业供应商时，您几乎肯定会进入所谓的销售资格认定过程。在这一步中，特别是作为技术高管，您的需求尚未与供应商对齐。您可能正在寻找价格、合同以及最快上手的途径。供应商则希望确认您是否符合他们的目标客户，并且可能至少保持几年的签约且不会快速流失。

大多数 SaaS 销售公司的前线销售代表会接听初始电话，他们的主要目标是了解您的公司并判断您是否符合他们的要求。他们可能没有太多的技术知识，通常也无法与您讨论价格。因此，您可能在这次初次会面中得不到太多价值。我的建议是要么将这些简介会议交给其他人处理，要么通过电子邮件向销售代表获取销售资格认定问题，并用足够的回答跳过简介会议。

谈判

一旦供应商确认您符合他们的目标客户类型，他们将安排第二次会议，并邀请更高级的资源参与。通常，这可能是一个销售经理或者技术销售代表。在这个阶段，您将开始得到更多关于解决方案的技术问题的答案，并对供应商授权使用的定价模型有一定的透明度。 以下是一些常规的谈判技巧：

- 请记住，销售人员的工作是向您销售产品，所以在达成双方都同意的条件方面，你们是同一团队的。你越能透明地表达对什么事情很重要，销售人员就越能制定符合您需求的销售协议。
- 不要低估除了总成本以外的其他因素，例如总合同期限（较长的合同应该享有更大的折扣），付款频率，付款条件（例如净额 30 天，净额 90 天），或者在任何一家公司发生控制权变更的情况下合同的处理方式。
- 总的来说，寻找随着你的发展而发展的合同。理想情况下，初始成本较低，并且随着工具的重度使用和提供更多价值逐渐增长。鼓励我们共同成长的思维可以帮助减少初始成本。
- 让你的财务和合规团队了解情况。如果你的首席财务官擅长谈判，就让他们处理这个过程的部分。
- 如果你的 SaaS 总预算很大，或者你要进行很多此类交易的谈判，考虑使用第三方谈判者，比如 Vendr。通常这些谈判者会按照他们为你节省的金额收费，并且他们从比你更多的合同中获得了定

价数据。 请注意月末/季末的销售目标是实实在在的，这个时候常常会出现折扣。

签署

一旦您达成了协议并交换了文件，下一步就是找出贵公司的授权签署者是谁（假设您是其中之一），将文件签署，并保持整理有序。

不要丢失或错放这些合同，它们将对未来的谈判或尽职调查很有用。请确保在您的预算或 SaaS 管理平台（SMP）中跟踪成本。

销售后

签约后，您很可能会被转交给供应商的另一位代表，他们的职位可能类似于售后支持或客户服务经理（CSM）。这些人的激励、衡量和关注点都是客户保留或产品增值销售。他们对产品非常了解，至少在技术方面有一定的知识。在接下来的时间里，他们将成为您争取新功能和解决缺陷的倡导者。

您是哪种类型的初创公司 CTO?

无论您是 CTO、充当 CTO 角色的 CEO，还是希望雇佣 CTO 的创始人，清楚了解 CTO 在初创公司中的具体职责，以及与更成熟公司中的高管和领导角色有何不同，都会很有帮助。和大多数事情一样，答案取决于背景，并会随时间变化。Calvin French-Owen 有一篇很棒的文章（ctohb.com/founder2cto），将 CTO 分解为四个原型：人才领导者、架构师、研发和市场/消费者导向。我喜欢这种分类，并在此基础上稍作调整，将其分为三类：

- 以技术为重点
- 关注人员
- 关注外部

技术关注的技术主管（也称为首席架构师）

技术关注的技术主管也可能是首席技术官办公室的负责人，领导一个内部的技术创新小组，其主要任务是提出前瞻性的战略、架构，并有时进行概念验证实施，以帮助企业未来的发展。这位技术主管将拥有较少的下属，大部分工程组织机构将向一个独立的工程副总裁汇报。在这种情况下，工程副总裁通常不向技术主管报告，而是直接向首席执行官或首席产品官（CPO）报告。

内部技术关注的技术主管也可能是首席技术过程架构师，负责建立技术工作的工具、系统和流程。 作为一个以技术为重点的内部首席技术官（CTO），如果你公司的产品具有高度技术性质（例如开发工具、API 作为服务等），你也可以充当产品经理的角色。

以人为中心的 CTO（又称为工程副总裁（VPE））

一个典型的初创公司通常不会同时设立 VPE 和 CTO 职位，因此通常由 CTO 来充当 VPE 的角色。对于创始人兼首席技术官来说，填补 VPE 角色的责任通常是最困难的，因为第一天的技术创始人的责任与以人为中心的内部技术领导者的责任不太重叠。

以人为中心的 CTO 负责制定内部技术文化、招聘流程，并监督内部流程。这位 CTO 大部分时间都在管理独立贡献者或其他技术经理。这是三个重点领域中最关键的一个。如果一个公司招聘不当，或者其技术人员管理不善、缺乏动力、缺乏集中，都可能影响生产力，甚至导致长期对组织造成不利影响的错误决策。

以外部为重点的 CTO（又称技术销售/市场负责人）

这可能是创业公司首席技术官（CTO）最不常见的职责，但在正确的时间和地点却同样重要。通常情况下，你会在为技术人员构建产品的公司中看到这样的 CTO，例如开发者工具公司。这些 CTO 会花费大量时间撰写博客文章或在会议上发表演讲。也许他们会被带到销售会议中，作为行政技术代表，以促成大笔交易的达成。需要注意的是，将技术团队构建成一个品牌不仅对产品销售有积极影响，还可以成为一种很好的招聘工具，并减少招聘的时间和成本。

对于创始人 CTO 来说，这可能是最容易接手的角色，因为他们对公司的历史背景有了解，可以最真诚、最热情地讲述公司故事，并宣传其产品和价值。

不同类型之间的过渡

理想情况下，创业公司的 CTO 应该全面熟悉这三个领域，尽管大多数人只会专注于其中一两个领域。如果你的业务需要一个你不擅长的专业知识，那么你可以考虑将这个任务交给一位同事来执行，以提高效率。特别是在初创阶段，大多数技术联合创始人/创始人 CTO 都会专注于技术。在这个阶段，通常没有太多其他工作需要完成！

随着公司的发展，这个人往往会发现自己在内部人事或外部职责方面扩展，对于他们来说，并不总是一种理想的过渡方式。承认你的专长或动力在技术方面，而人员管理却不适合你，并不是个人的失败，恰恰相反。识别自己的超能力，并在公司中规划一个能够发挥这种超能力的角色，是你提供最大价值的方式，公司应该雇佣另外一个擅长人员管理的人来填补这个角色。 如果你发现自己在一个让你不快乐的角色中陷入困境，重要的是你要向自己和 CEO 承认这种不匹配。这并不意味着放弃你作为创始人或领导者的职位，或者放弃作为一名在公司中受尊敬且具有重大影响力的人的地位。

关于各种转变的一些想法：

- 你的超能力：内部技术
  - 公司需求
    - 内部人员：聘请一个以人为中心的工程副总裁，积极将 CTO 的角色定位为技术。
    - 外部关注：如果你没有比你更擅长这个角色的人员，那就雇佣一位开发者大使，并授权他们履行这个角色。否则，从内部提拔并确保清楚了解该角色所涉及的内容。
- 你的超能力：内部人员
  - 公司需求
    - 内部技术：如果你的团队中有一位高级工程师或架构师，可以使其成为技术领导者，那是值得考虑的。否则，技术架构师应该是你招聘的首要考虑对象。
    - 外部关注：如果你还没有内部有人比你做得更好，那么请聘请一位开发者倡导者，并赋予他们履行这个角色的权力。否则，从内部提拔人员，确保他们清楚这个角色的职责。
- 你的超能力：外部关注

    - 公司需求
- 内部技术：如果团队中有一位高级工程师或架构师，你可以授权/提升他们成为技术领导者的职位，这值得考虑。否则，招聘一位技术架构师应该是你雇佣的首要任务之一。
- 内部人员：雇佣一位以人为本的工程副总裁。

在许多情况下，最终结果可能意味着雇佣一位技术联合创始人，其超能力与公司目前需求更加契合。在其他情况下，这可能意味着雇佣一位非常有能力以人为本的工程副总裁，来补充一位高度技术的技术联合创始人。

技术团队管理

作为技术领导者的产出是由你的团队的产出来衡量的，因此您负责确保整个团队的运行良好。良好运行的含义会因团队和情况而异，但是有一些普遍的模式和趋势与高绩效相关。本节旨在为每位技术领导者提供对所有技术领导者常见情况的指导。 技术文化与一般哲学

我鼓励所有领导者采用一种被称为“仆人式领导”的一般领导风格。作为一名仆人式领导者，你的主要关注点是服务团队的需求。这意味着注重赋权他人，建立透明度、沟通、协作和成长的文化。当你思考团队、文化以及每天的决策时，请自问哪个选项能使团队最好地完成工作，从而为企业发挥最大价值。

技术文化的十大支柱

1. *将团队时间用于对企业有意义的事情*

总体而言，你希望你的团队将时间花在对企业有推动作用的事情上，作为技术领导者，你的责任是创建一个环境，让你的工程师可以专注于这样做，并且能够持续地工作，减少分散注意力的因素。这种框架可能显而易见，但正确使用时，它是一个强大的决策工具。

举个例子，假设你的团队正在讨论是使用现成的后端框架还是从头开始编写。有一个微妙的利弊清单，可以讨论诸如从头开始构建带来的灵活性与使用框架进行快速部署的时间缩短等问题。在这个假设的现实中，你的企业实际上需要的是对前端进行迭代和优化客户旅程，因此每一分钟花在后端的时间，都是前进的一分钟。即使现成的解决方案足以支持前端迭代，即使你必须在十八个月后放弃框架并使用定制解决方案重新编写后端，现在快速迭代前端并找到产品市场适配度是为你的企业赢得重新编写的权利。

2. *在能使用可靠工具的地方使用，并在关键的地方进行创新*

这也可以被称为不要重复造轮子或站在巨人的肩膀上，这里的想法是在可能的情况下使用现成的组件（库、云服务、应用、软件包）。使用现成组件存在一个固有的权衡，在轻松入门和将解决方案定制以匹配你确切的问题之间。我认为，在现有实现已经存在的大多数情况下，权衡更倾向于使用现成的服务、库或应用程序。 在很少见的情况下，当您最终需要重写或大量定制依赖时，您对现成工具的经验将非常有价值，可以影响和加速自定义构建的设计。

3. *自动化推动速度*

作为团队的架构师，您的目标是确保团队将尽可能多的时间花在为业务产生价值的代码上。开发人员经常进行一些耗时的任务，这些任务对于编写代码是必要的，但本身并不增加业务价值（例如，复制生产环境、让代码在本地运行、找出如何运行测试套件、在云中为特性分支环境进行配置、追踪未被测试覆盖的错误等）。

这些类型的任务对整体生产力产生持续性的负担。当发现这些任务时，您可以通过投资来自动化这些任务来避免付出额外的负担。鼓励您的团队，在他们花费超过 30 分钟进行非生产性技术工作时进行记录，并在流程中提供一个自动化这些任务的空间，这样没有人会再浪费那个小时。

4. *频率降低困难程度* 根据“频率降低困难”的标题，马丁·福勒详述了一句格言：“如果它让你痛苦，那就多做几次”（ctohb.com/fowler）。福勒认为，任何对团队而言痛苦、容易出错或者代价高昂的过程或任务，都是该任务发展不充分的症状。如果没有你的施压，痛苦的技术任务往往是最后一个自愿承担的。结果就是，它们被忽视了，而痛苦随着时间的推移而加剧。另一方面，如果你的团队文化强调优先处理这些痛苦的任务，那么更多的努力将花在自动化、文档化和改进这些任务上，使它们最终变得不那么痛苦，甚至完全自动化。正如福勒指出的那样，更频繁地执行任务也能提供更多的反馈，并通过实践来建立技能，这些都进一步降低了任务的难度和痛苦。

对于许多团队来说，将代码发布到生产环境就是一个常常痛苦的任务的例子。发布代码可能需要数小时的时间才能实际部署，因此很少进行。然而，如果你的团队认同“如果它让你痛苦，那就多做几次”的理念，那么作为技术领导者，你将推动定期快速发布。如果你从每周发布开始，那么在第一周，团队会感到痛苦，第二周将与第一周一样痛苦，但也许工程师会注意到一个自动化部署的机会。到了第三、第四或第五次发布时，你可能已经有了全新的脚本和基础设施，以便更快地发布代码，使发布频率加倍。过一段时间后，你将能够按下一个按钮就将代码发布出去。每天多次发布被称为持续部署（见《持续部署》，第 216 页）。

5. *标准化 RFC 过程*

RFC（Request for Comment）是一种文档，概述了一个技术想法、过程或规范，并以同行评审和随后的采纳为目的编写。你熟悉的大多数协议都有相关的 RFC，比如 HTTP 的 RFC 2616 或 DNS 的 RFC 1035。同行评审和随后的采纳和标准化的理念是一个强大的工具，用于在技术决策和结果获得认可时进行“准民主”决策，它可以成为你与团队一起使用的好工具。

我建议你正式设立一个 RFC 过程，并提供一些指导，说明哪些决策应该通过 RFC 过程进行。

正式流程可能看起来像一个简单的检查表和一个包含在该文档中放置副本的模板文件，该文件还包括收集反馈/评论的方式，以及投票、最终确定和标准化文档的过程。

我的建议是充分利用现有工具，例如在源代码控制中创建一个作为 RFC 模板的 Markdown 文档。然后，新的 RFC 将是该存储库上的一个拉取请求，引入一个具有提案的新 Markdown 文件。在这个过程中，将收集投票作为对该拉取请求的批准，并在拉取请求最终合并时完成 RFC 的最终确定。或者，如果您已经设置了内部维基，您可以在那里创建一个 RFC，并使用维基的评论系统收集反馈。

您还应该明确规定哪些决策应该提交给 RFC，并设定清晰的期望。我建议将其限制在高级流程、技术意见和文化方面，并不将其用于工具选择或系统架构等方面。下面是一些适合作为 RFC 主题的好示例:

- 统一命名约定（master vs. main，black/whitelist vs. allow/denylist，各种上下文中的 camelCase vs snake_case）
- 代码格式化程序/静态代码分析工具的使用
- 会议频率/议程/文档
- 单一仓库 vs. 多个仓库
- API 设计意见/理念

我鼓励你在 RFC 模板中添加一个关于如何将 RFC 的结果制度化的部分。例如，如果 RFC 提议为团队制定一个技术意见的标准，一旦 RFC 被通过，这个意见应该被纳入你的工程指南和入职文档中，这样它就成为了当前和未来员工的法规。

6. *把速度作为目标，而不是策略*

在《好策略/坏策略》一书中，作者理查德·鲁梅尔特将好策略定义为包含三个要素的策略：对如何克服挑战的诊断，一个指导方针或总体方法，以及一套用于实施该政策的行动。策略规划了前进的道路。

相比之下，目标是对目的地的描述。我听到过无数团队告诉我，他们的策略是快速前进，却没有说明如何实现这个目标。我完全同意，工程速度和速度是一个很好的目标，但它不是一个策略。

挑战：我们是一家未受监管的初创公司，需要尽快进行迭代，以在资金耗尽之前找到产品市场适应性。

诊断：我们有机会最小化复杂性并快速推进。由于我们尚未找到产品市场适应性，我们目前建立的价值很小，因此我们需要忽略已经投入的成本，选择快速重写，因为我们还不确定长期来看我们的产品会是什么样子。

行动计划：我们将专注于招聘软件工程通才，加强好奇心和勤俭节约的文化，摒弃自我，只在长期技术规划上投入适度的努力。相反，现在我们将专注于我们推出的 MVP 产品实验的能力。

7. 参与持续教育和技术会议

如今技术会议已经无处不在，几乎每个软件工程的子专业都有一次聚集志同道合的人的机会。你的团队，如果还没有的话，可能会在某个时候要求预算参加这些会议。一个典型的预算请求可能是 1000 美元的机票和酒店费用，以及 500 至 1000 美元的会议注册费和住宿费等。总体而言，一个工程师参加一次会议的费用将在 2000 美元左右，还要考虑他们离开办公室的时间。为了学习和持续改进的精神，以及其他一系列附带好处，我建议您为此预留预算，并定期批准或系统性地批准会议请求。

如果参加会议的唯一收益就是员工对自己热衷的相关技术主题有所了解，那么这样的成本是值得的。然而，还有许多其他好处。

我鼓励您要求会议参与者在回来后提交一份总结他们所学到的关键内容的书面文件。您还应考虑赞助与您业务最相关的会议，并让您或您的团队成员主持一个特定主题的研讨会。这些研讨会和赞助提供了您公司的出色品牌机会，让您的名称和信息出现在一个非常有针对性的观众面前，这个观众很可能包含您未来想要招聘的候选人。

总之，为工程师参加会议分配预算有助于个人专业发展，并进行相对轻松的文档记录。这对团队学习也有好处，并提供了通过社交网络、赞助或主持招聘未来团队成员的绝佳机会。

8. 使用橡皮鸭调试法

您是否曾经经历过试图解决问题，在向同事解释过程中突然找到解决方法的经历？橡皮鸭调试法（ctohb.com/rdd，ctohb.com/tpp）是一种简单的实践方法，试图在不牵涉同事介入或切换上下文的情况下复制这种现象。

橡皮鸭调试是通过首先大声讲述问题的过程，也许对着桌上的真实橡皮鸭讲述。这个方法的理念是通过大声讲述问题，往往可以听到问题的缺陷，或看到解决问题的方法，这些在问题只停留在头脑中时可能没有考虑到。橡皮鸭方法可以避免打扰同事，并且能更快地获得答案。

9. 建立解释视频库

如果一张图片胜过千言万语，那么一分钟的显示桌面/集成开发环境/应用程序的视频加上你的讲解声音，能够节省$1,000 的开发人员时间。有许多工具可以轻松地录制和分享这些迷你视频，包括内置于 Slack 和 Loom 的工具。通过屏幕录制和配音，你不仅能够更容易地传达技术思想，而且可以根据自己的时间来完成。生成的视频可以被观看者自行加速，并且可以作为组织知识库的一部分进行存档和重播。

作为团队的技术负责人，我鼓励你定期制作这些迷你视频讲解，尤其是在你对平台架构进行更改的时候。在内部维基中组织所有视频，并确保它们完整且独立，但长度要简短并且直截了当；如果你在一个过长的视频的末尾才讲解关键内容，你的团队成员可能永远不会看到它。此外，尽量保持一致的方法，以便观看者知道可以期望什么。

我保证在你制作和分发这些视频时，起初的观看率可能相对较低，但随着时间的推移，这个视频库将作为参考知识的宝库提供巨大的价值，并且会有观看次数，为你和团队节省大量时间。 10. 入职培训是每个人的责任

你的团队不断地在创造部落知识，无论是如何启动服务，还是如何在代码库中使用代码模式。对于这个问题，有两种处理方法：一种是什么也不做，每天都会增加新员工知识差距的大小，另一种是积极努力将孤立的部落知识转化为可扩展的文档，以供现在和未来的员工使用。基于明显的原因，我推荐后者。这意味着每个人都应该时刻问自己，我怎样才能记录下我刚刚创建/学到/发现的东西呢？当有新员工遇到知识库中找不到的内容时，就要由他们自己去寻找答案并记录下来。

技术债务

旧金山的金门大桥是用钢铁建造的，其本身并不是黄金颜色的。这座桥经过粉刷，保持其标志性的颜色对于旧金山人来说非常重要，以至于他们会不断地重新粉刷（ctohb.com/painting）。一旦粉刷完成，流程就会立即重新开始。这种持续的投资或永久性维护形式是维持最重要和复杂系统达到预期性能的必需品，无论是金门大桥还是你团队的软件项目。只不过对于你的项目来说，维护并不需要油漆桶，而是以技术债务的形式呈现。

软件开发团队交付的每个功能都会带来一定程度上未来工作的需求，或者说是债务。这种债务可以以需要修复的错误或者提供增量客户价值的快速跟进功能的形式存在，也可以是代码中的粗糙之处，需要修正以提高可维护性、性能或安全性。即使你的团队正在休假，一定程度上的债务仍然会自然积累：发现依赖软件的安全漏洞，包件过时，工具的新版本发布，第三方 API 被废弃或更改等等。债务是不可避免的，你需要对其进行计划。

技术债务和产品生命周期

另一种理解技术债务的方式就像理解财务债务一样，比如房屋按揭。当你贷款购买一套房子时，你是在经过深思熟虑的情况下，做出接受债务的决定，明知后果（利息），以便让你能够立即得到你想要的东西（获得一套房子）。然后你会按照一定的频率和一段较长的时间来偿还这笔债务（每月支付。

技术债务也是一样的。你的创业公司可能会有意地积累这种债务，作为一种有意识的权衡的一部分，其中一部分权衡是制定一个实际的计划来偿还这笔债务。在解决技术债务时，你应该运用与偿还财务债务相同的逻辑：要么一次性偿还，因为你有额外的资源（并且没有更好的地方来利用这些资源），要么持续地偿还，或者在将来的某个时候一次性全部偿还，但可能会付出更高的总价，其中包括利息。

无论你选择如何偿还技术债务，成功偿还的关键是要认识到债务是软件工程过程中不可避免的一部分，主动偿还债务是对整体工程健康的必要投资。

定义技术债务

定义技术债务的另一种方式是指技术决策、实现或细微之处，会主动降低今天或未来业务的效率或效果。关键在于技术债务有重要的后果。你的一些代码可能在客观上丑陋或效率低下，但如果这种低效对业务没有影响，并且将来不需要修改代码以保持质量、性能或迭代，则这段代码在技术债务方面不具成本。毕竟，软件开发的目标，特别是在创业公司，不是编写完美的代码库，而是构建能够支持业务的软件。

不要害怕债务。它可以有其用途。例如，在构建尚未在市场上验证过的产品的第一版时，技术团队可能决定采用无法扩展到一百个用户以上的架构。如果这个决策使团队能够快速验证产品，并确定是否会有一百个用户使用该产品，那么这条路线可能是值得的，特别是考虑到可能需要几个版本的原型才能找到用户喜欢的版本。

技术债务至少有七种类型：

1. **架构债务**或**设计债务**会在软件设计无法满足近期或未来业务需求时产生。例如，设计使构建业务所需的功能变得过于困难，或者设计无法满足业务的用户数量或性能要求。
2. **代码债务**会在实现代码时没有注意最佳实践，导致代码难以理解和维护时积累。
3. **测试债务**会在运行不足够的自动化测试以提供团队对代码库正确性的信心时累积。
4. **基础设施债务（Infrastructure Debt）** 发生在基础设施、可观察性和支持系统不够健壮或维护不善的情况下，导致难以扩展或部署更新，或者导致运行时间不稳定和可靠性低下。
5. **文档债务（Documentation Debt）** 是由于文档不足或文档过时/不准确所导致的，这会给团队成员在项目入职过程中带来困难。

6. **技能债务（Skill Debt）** 出现在团队成员缺乏维护或更新代码及周围基础设施所需的技能时。
7. **流程债务（Process Debt）** 是团队在解决问题的方法上不一致，导致错误、延迟或成本增加。

技术债务破产

如果您的创业公司长时间忽视或忽略技术债务，它可能会成为未来进展的重大障碍。由于技术债务的存在，团队可能会不经意间花费 80%甚至 100%的时间来解决系统问题或低效率，这种状态被称为技术债务破产。

以下是团队可能处于技术债务破产状态的一些迹象：

- 你经常遇到生产停机的情况，对业务产生重大影响。
- 由于需要处理技术债务，你经常受到持续的阻力或夸大的时间表，以开发新功能。
- 团队抱怨代码库太复杂，无法顺利完成工作。
- 新功能无法发布，因为会意外破坏旧功能或引入过高的缺陷水平。

如果你发现自己陷入技术债务破产，是时候提醒相关利益相关者，设定期望值，制定整合债务的计划，并立即开始还债。

如果你已经对同级领导诚实坦率（参见“递送坏消息”，第 46 页），你应该具备必要的可信度来解释技术债务问题，并发展出一个对解决技术债务的投资回报率（ROI）有共同认识的态度。

测量债务-债务清单

与抵押贷款或汽车贷款不同，没有一个网站能够告诉你确切的技术债务金额和剩余还款额。一些形式的债务可以通过定量化进行衡量，但大多数分析是定性的。为了在规模上进行健康而负责任的债务管理，我建议进行债务清单调查。

该调查应该定期进行。每年一到四次，在不同种类的债务上进行冷静的分析，对团队的运营状况进行诚实的评估。不要独立进行调查；相反，与团队中每天在代码上工作并定期与债务互动的其他工程师合作进行。

一份调查可以简单地包括以下内容：对于下列类型的债务，请在 1 到 10 的评分中评估我们有多少，然后用几句话解释评分的理由。

利用调查结果来指导团队如何消耗精力偿还债务，并比较不同调查之间的结果，以确保债务保持在合理水平，并使团队定期解决最大的债务痛点。

偿还债务的策略

为了决定何时偿还技术债务，你首先应考虑团队花在此上面的时间的价值。关键考虑因素包括：

- 存在多少债务，根据最近的债务清单调查所示
- 它对公司日常运营的阻碍程度，例如通过故障、客户流失或缺陷率
- 它对团队实施新项目的能力造成了多大的伤害
- 偿还债务将会有多么困难

如果你没有陷入技术债务破产，并且你的目标是保持健康水平的债务，我建议将团队时间的 10-20%分配给非功能工程方面的投资（例如偿还债务、探索新的模式/概念验证、改善开发者体验等）。你当前债务影响越严重，偿还债务的努力越大，你应该分配给团队时间的百分比也就越高。

及时支付

处理技术债务最常见的方法是按需进行支付，这意味着债务将作为业务驱动的项目的一部分进行支付。通常情况下，团队会在计划会议中为选定的冲刺添加与故事相关的技术债务任务。这种方法的开销低，计划工作量少，因此往往能够取得良好的效果。但要注意一些潜在的问题：

- 由于及时支付对整个团队来说不太可见，可能导致对技术债务的系统性投资不足。确保在进行及时支付时，向团队诚实透明地说明你预计有多少团队时间将用于处理债务。
- 在一个冲刺中加入技术债务可能意味着对冲刺目标来说，投资技术债务是次要的，如果团队时间不够，可能会被削减掉。
- 在一个冲刺中解决技术债务可能被认为是减慢冲刺进度或引发延迟，而不是对速度和整体系统健康的投资。 定期偿还

定期偿还类似于支付车贷或按揭贷款。团队在固定的时间间隔内腾出空间偿还债务（例如，每个迭代一天，每个月几天，或每个季度几周）。谷歌曾允许工程师占用他们 20%的时间来做任何他们想做的事情，包括偿还债务或创新新的项目和工具。这里的想法是相同的：作为经理，你明确地为团队腾出时间，并鼓励他们对用于进行工程的工具和流程进行投资。

例如，Shape Up 方法（参见技术流程，第 157 页）在六周循环后描述了一个两周的冷却期，或者是八周周期的 25%用于进行技术投资。请记住，25%并不是一个神奇的数字；正确的百分比将取决于团队的债务清单。

持续偿还

根据团队负债的昂贵程度，您可能希望将更多资源用于整体系统质量，而不仅仅是定期策略所允许的。这看起来像是在一个双班组的情况下，拥有一个专门的团队，我称之为客户团队（参见项目维护：双班组理念，第 113 页），他们作为日常工作和目标的一部分来偿还技术债务。 确保任何主要目标是内部效率的团队（如技术债务团队或客户团队）都有清晰且可衡量的工作目标非常重要。例如，如果您的债务清单将测试债务排在最高债务类别，则测量缺陷率和代码覆盖率，并要求客户团队对改进这些指标负责。如果您的基础设施债务最大，则应关注正常运行时间和恢复时间指标。

技术债务的沟通

非技术领导者并不期望他们的技术团队做到完美。但是他们确实期望高绩效和始终满足期望。

当涉及到债务时，这意味着清晰地传达您的债务管理策略，并提前诚实地沟通债务可能妨碍业务目标的时间，以及还债策略，使其不再成为障碍。

技术路线图

时间框架

我发现将技术规划划分为三个时间框架很有用，有时被标为短期、中期和长期，或者被称为第一、二、三视野（horizon one, two, and three）。每个时间框架应由不同的过程管理，并且通常由不同的利益相关者拥有。

短期/第一视野

您的短期规划是您的团队当前正在努力完成的工作。这包括正在进行中的功能、积极处理的缺陷、技术债务或紧急工作项目。有关管理短期规划的更多细节，请参阅技术流程部分的工作流程，第 157 页。

中期/第二视野

如果您是团队上唯一的技术经理，那么您将负责中长期路线图的规划。如果您有一些主管或高级经理向您汇报工作，您可能会与他们合作制定中期路线图。中期路线图不仅对您自己的规划和组织起到了重要作用，而且还可以作为一个工具，与其他部门/利益相关者沟通工程团队的工作。

通常，中期路线图用电子表格的形式实施，每个团队或个人作为一行，列是时间段通常以周或迭代为单位，表格的内容是高层次的工作项或工作领域。制定路线图的目的不是准确预测任何特定任务的完成时间；这样做需要准确而精确的工作估计，但这种估计最多只是拟稳定的（即使以周为单位），并且不能保证准确性。相反，路线图的理念是概述操作顺序并为团队确定方向。

在工程进展过程中，您可以并且应该根据实际情况更新任何给定活动的持续时间。更新给定任务的周数是评估是否继续投资项目的一个良好时机，还可以向外部利益相关者更新当前完成预计时间。最后，路线图对于追踪主要计划所需时间以及评估团队在高层次上投资时间的情况非常有帮助。

长期/第三视野

注：只提供翻译的部分文本，请忽略原文。 作为团队的领导者，你有责任关注团队的长期健康和生产力。你应该花时间设计这些目标，并制作一个经过深思熟虑、清晰明确的文件（或幻灯片，视频，维基文章等），向团队解释这些目标。一旦你设定了初步目标，在组织中改变战略目标会引起动荡，因此不频繁地重新审视目标是很重要的。频繁改变方向同样会让团队感到困惑和失去动力。我鼓励你每个季度向整个工程团队以及其他高级领导层提供关于长期计划进展的更新。

一些长期计划的例子：

技术架构债务

- 从已废弃的框架迁移到持续维护的框架
- 从一个托管环境迁移到另一个（例如，迁移到 Kubernetes）

编程语言债务

- 合并编程语言的使用
- 从旧版本转移到新版本的语言（从 Python 2 到 Python 3，或从.NET 4 到.NET 5+）

平台/架构采纳

- 多个团队采用或迁移至内部服务的新版本
- 迁移到/离开无服务器环境
- 采用新的范式（例如，服务器端渲染，边缘计算）

招聘计划

- 扩大或重组团队
- 招聘专家或建立新的技术部门

时间表沟通

与我合作过的销售、市场、产品或支持领导人都对技术流程和技术路线图的透明度表示赞赏。相反，我曾与一些公司的领导人交谈，他们形容他们的技术团队就像一个黑洞。毋庸置疑，您不希望被称为黑洞。不成为黑洞很简单；它看起来就像在您的组织中建立一个定期提供透明度给其他领导人的流程。理想情况下，您还可以通过一个论坛或机制来接纳其他部门的意见，并将其纳入路线图流程中。您还可以通过您的流程向利益相关者回馈他们的需求在开发过程中的进展，并管理何时完成的期望。

技术流程

康威定律指出，设计系统的组织受限于产生与组织沟通结构相似的设计。换句话说，团队的组织结构以及团队内部和团队间的工作流程将对产品产生重大影响。在信息孤岛中工作的团队很难生产出与其他团队设计完美集成的产品，因此作为这些沟通结构的监督者和最终架构师，你需要确保这些结构满足你正在开发的产品的需求。

工作流程

技术工作是一个高度微妙的问题，涉及成千上万个微小的决策，这些决策将影响事物最终的互操作和行为。为了在组织中保持生产力，你需要一套标准和指导原则，以确保日常技术决策在广泛一致的基础上进行，从而对团队可管理。这意味着你需要确实制定这些标准，对团队进行培训，并建立日常流程来强制执行和修改这些标准。

团队在决定要构建什么以及如何完成工作方面所遵循的模式称为工作流程。最流行的五种工作流程模式如下：

- 敏捷（Agile）
- 敏捷开发（SCRUM）
- 看板（Kanban）
- 瀑布模型（Waterfall）
- 形而上（Shape Up）

这些方法模式有整本书的介绍，我最喜欢的是 Jeff Sutherland 的《SCRUM：在半个时间内完成双倍的工作》。

这些方法有各自的优点和缺点，此章节将讨论一些基本的优势和劣势；然而，在现实世界中，这些流程的差异相较于管理者对所选择的流程的实施方式的影响微不足道。作为技术领导者，你的工作是选择一个流程，并确保其得到良好的实施和迭代。

一个好的开发流程应当遵守以下关于软件开发的真理：

- 没有人能够完美地预测完成任何给定工程任务所需的时间。
- 工程很少是一条直线；建造 X 功能可能需要在建造 X 之前解决问题 Y。
- 完美的规格是不存在的；在构建技术时总会存在缺口和需要发现的事物。

一般来说，工作流程的目标是确保团队组织得井然有序，并以可接受的速度交付工作。有些工作流程甚至以迂回的方式试图量化工程团队的速度，从而向非技术相关的利益相关者报告速度随时间的变化。

瀑布流

最古老的工作流程可以追溯到 1950 年代，那就是瀑布流（参见 ctohb.com/waterfall）。瀑布模型将项目活动分解为顺序步骤，每个步骤依赖于并在前一步骤完成后开始。在软件工程中，它看起来可能是先有产品愿景，然后进行产品概念设计，再进行产品设计，然后进行软件开发，最后进行测试、部署和维护。对瀑布流最常见的批评是该结构僵化、不灵活，并且不促进迭代开发。

敏捷/Scrum

敏捷和 Scrum 过程是比瀑布模型更细致和规定性的方法论。有许多出色的资源详细介绍了这些细微差别，比如 Sutherland 的《Scrum》、Mike Cohn 的《敏捷估算与计划》、James Shore 和 Shane Warders 的《敏捷开发艺术》。

关键要意识到这些过程是指导方针，而不是圣经。要让你的工程团队发挥最佳效果，先从一个过程开始，并观察它在你特定的人员群体和技术挑战类型上的表现如何。有些团队的工作更适合估算和故事点估算，而有些团队面临更加模糊的维护项目，估算几乎是不可能的。留意一下从过程中是否真正为工程团队增加了价值，或者它只是个每个人都害怕的冗长会议。

不要犹豫地跳过对团队来说不明显有利的仪式。例如，我发现 Scrum 对计划扑克的要求对大多数团队来说效率低下。

Shape Up

Shape Up 是由 Basecamp 公司规范化并在 basecamp.com/shapeup 上发布的方法论。Shape Up 的核心周期为六周，比 SCRUM 所倡导的短程冲刺时间要长得多。这个周期采用了固定时间和可变范围的方式。其想法是，更长的时间段提供了更多的空间来制定清晰的计划（规格）并在项目上做出良好的工作。与其他模型相比，Shape Up 对估算的重视程度要少得多，而正如我即将讨论的那样，这对于工程团队来说是一件好事。

工程估算

根据 Google 的定义，准确性在技术上是指测量、计算或规格说明结果与正确值或标准一致的程度。换句话说，准确性是整体正确性的指示。当你将飞镖扔向飞镖靶，并瞄准靶心时，准确的一组扔镖就是一组趋向于中心的扔镖。

与此相反，精确性在技术上被定义为测量、计算或规格说明的精细度，特别是通过所给出的数字来表示。换句话说，精确性表示了一种精确程度。当扔飞镖时，如果你的所有扔镖，无论瞄准点在哪里，都紧密地聚集在一起，那就可以说它是一个精确的聚集。

正如这个描述所帮助你形象化的，某物可以准确而不精确（环绕或接近靶心的一个大范围的扔镖聚集，但没有命中靶心），可以精确而不准确（一个紧密的扔镖聚集但未命中靶心），当然也可以准确且精确（一个紧密的扔镖聚集并命中靶心）。 你应该期望并要求团队对软件开发任务的完成给出准确但不一定精确的估计，并对其负责。如果今天是一个月的第一天，团队给出的合理指导是，“我们将在本月发布这个功能。”如果团队表示，“我们将在 23 日发布这个功能。”他们更有可能错过这个截止日期。

如果你按照周、月或季度来计划工作和资源分配，就不需要尝试为每个任务估计小时数或天数。随着时间的推移，注意你的估计是否真正给你希望得到的规划能力。如果不能，不要通过继续这个过程来惩罚团队，更不要在绩效评审中将其作为一个贡献因素。相反，调整估计，让它们对你有所帮助而不是伤害你。改变你的期望，并且不是对于错过估计作出反应，而是对于团队在努力满足估计时面临的挑战作出反应。

关于估计的最后一点说明是：不要将错过估计与总产出/速度不佳混为一谈。有些团队效率很高，产出很多，但仍然会错过估计。速度是更重要的指标，一个产出高但估计不完美的团队不应受到惩罚。相反，经常错过估计且难以交付新价值的团队是表现不佳的，需要做出改变。

消耗图

SCRUM 消耗图显示团队对估计的进展，是衡量迭代效率的重要工具。然而，估计是不完美的，由于各种原因，消耗图可能显示为一条平直线，甚至消耗过多。这可能是因为团队确实没有取得进展，也可能是估计问题或数据收集不准确导致的。 一个持续上升的燃尽图表，尽管团队完成工作并付出了努力，但会削弱士气，未能实现预期的效益。如果有简单的调整可以帮助您更好地捕捉数据并修复图表，请进行更改。但是，如果您发现特定的输出测量方法仍然不起作用，就放弃它吧。承认你估计这些特定类型的故事与这个团队的方法不够精确，并转而采用其他监控和改进绩效的方法是可以的。

根据我的经验，只有很小一部分的团队能够成功使用燃尽图表，所以如果这种技术对您的工程团队没什么帮助，不要灰心。

选择工作流程

我认为选择哪种工作流程并不是工程团队最终成功和速度的重要因素。关键因素在于您是否关注团队的工作流程，并不断迭代工作流程本身，确保您的模式增加价值，并与您的团队以及团队所面临的问题类型相匹配。

话虽如此，这里有一个大致的模型，用于考虑哪种类型的工作流程可能是一个更好的起点：对于理解良好的工作（即具体、尚未开始的任务，且易于解释），可以通过更细致或规范的计划过程更容易管

理，并产生更多的效益。换句话说，如果工作模糊且难以估计，使用看板比使用 Scrum 更好管理。

很容易理解并且在 SCRUM 中表现良好的故事通常包括：

- Greenfield，即不依赖于可能是旧系统或难以处理的外部模块的全新代码
- 不依赖于新的模式/工具/技术，而是依靠现有（平庸的）技术堆栈
- 易于将故事拆分为较小的任务，对团队来说很熟悉之前的工作

相反，如果你的工作是：

- Brownfield，或者受到技术债务的严重阻碍且没有明确的降低债务/重构途径，那么你可能更适合使用看板方法。
- 定期更改或承担不可预测的优先事项
- 需要采用新的工具和模式，可能在最初的几个实施中引入意想不到的成本
- 分配给一个全新的团队，这个团队没有过去合作的历史，也没有做过这类项目的经验

冷却/创新冲刺

在使用类似冲刺的定期节奏时，团队面临的主要危险是期望冲刺结束，功能将被发布，团队可以立即转向下一组功能。由于技术债务的累积和产品功能的迭代需求，这种持续是无法持续下去的。必须不断或定期留出时间来偿还债务。

定期偿还债务的常见做法是冷却冲刺的概念。有时被称为技术债务冲刺或创新冲刺，其思想是相同的：给团队时间整理他们的数字工作空间，进行一些代码清理工作，确保他们和代码都处于良好的状态，以便以高速度进行工作。正如《偿还债务的策略》第 150 页所讨论的，合理地将总开发时间的 5%至 20%用于冷却工作。

如果您正在进行两周冲刺，这可能意味着每四到五个冲刺中有一个冷却期。

技术规划和规范

当与工程师讨论编写技术规范的过程时，他们经常问我，你怎么有时间写规范呢？通常我会反问，你怎么没有时间不写技术规范呢？我的回答中隐含的意思是，在开始构建之前花时间思考你要构建的内容是一个能节省时间的方法。

软件工程本质上是一个创造性的过程，意味着我们每次不是在做同样的事情，并且有多种方法来完成每一个特定的任务。一个优秀的规划过程认识到提前思考一个故事是有价值的，但同时也平衡了对预先规划的重视，因为真正了解一个功能的唯一途径是实际构建它。

一个优秀的技术规划过程可以实现几个目标：

- 减少功能中的重新工作量
- 寻找实现相同功能的较少工作量的方法
- 降低重要但不可见于业务的考虑因素被遗忘的可能性，例如错误处理/负面案例、测试、日志记录、监控、分析、安全性、可扩展性、发布计划和技术债务偿还

- 增加多个人/团队共同合作的工作以兼容方式完成的机会
- 为将来的维护、改进或扩展提供有价值的关于特定功能建设方式的文档
- 保持团队对于无法逆转/昂贵的技术考虑（例如工具和架构）以及易于遗忘的关键细节方面的思考和一致性
- 证明足够轻量化，能以合理的时间完成，并且不会迫使在次要细节上做出不重要或缺乏足够前期信息的决策

技术规范领导者

我建议您为需要计划的任何项目指定一个主导者：一个负责制作技术规范并通过您的批准工作流程的人。

这并不意味着他们是唯一的贡献者。相反，如果在计划阶段其他团队成员可用并具有有用的知识，他们可以和应该提供贡献。

计划可以是同步的（即，每个人都在同一时间段内在一个房间里）或异步的。我建议尽可能进行异步计划，因为在计划中的大部分工作都涉及研究（例如阅读产品文档、阅读代码、制作原型/概念验证、评估工具和 API 等等），这些可以独立完成。

计划时间

您的技术规划流程应该在初始实施中节省您的时间。它还应通过最小化技术债务并留下可以加速未来改进的文档来节省时间。 规划投入的时间不正确，无法满足这些目标，要么因为时间太短不足以节省时间/生成良好的文档，要么因为时间过长而无法通过节省时间来弥补成本。

对于正确的时间量，没有普遍适用的公式，但我可以给出一个经验法则：为估计项目需要花费的每周工作时间安排一天的技术规划时间。一般来说，这将导致半天到三天的规划时间段。如果你的项目所需的工作时间少于两天，那么它可能只需要很少的规划工作，并且风险较低。相反，如果你的项目预计需要超过三周的实际开发时间，你可能无法一次高效地规划如此大的项目，应该考虑分解它。

如果你的团队拒绝投入时间进行规划，你可能过分追求结果而忽视了过程。确保工程过程为企业产生良好结果的方式不是更加严苛地推动团队，而是建立一个健康的过程，使其能够产生良好的结果。你不会通过让结构工程团队加班来加快设计桥梁的速度。你会确保他们拥有最好的桥梁设计工具，并提供有关桥梁跨度的最佳信息。软件工程也一样。但我们不是使用 CAD 软件或实际土壤/岩石测量数据，而是使用产品规格、设计流程和软件工具。

相反，过于冗长的规划过程，团队成员坚持在一开始就获得每一个细节，可能是一个严重的文化问题的指标，团队成员被错误的恐惧所困扰。有效的规划不能消除风险，但是提前思考重要的高层决议可以将风险最小化。一个过于关注细节的团队可能害怕犯错或者不愿意对他们的工作进行迭代，这是过于注重结果的管理的表现。个人不应该因为合理的错误或规划失误而受到惩罚。如果技术规范一开始并不完美，那没关系；你应该期望你的团队在实施过程中发现错误或间隙，并在发现这些问题时更新规范。

规范编写中的原型设计

写技术规范时，通常会有多个选项可以实现目标，而且没有明显的理由去选择其中一种。或者你会发现关于某个选项的有效性还存在未知因素，这使得决策变得模棱两可。如果可能的话，尤其是如果可以高效地完成，我建议你给工程团队留出空间来原型设计其中一种或多种解决方案，以获取数据以便在规划中做出更好的决策。花半天时间用新工具建造一个玩具，以验证该工具是否能够事先达到预期结果，这将是非常值得的。

技术规范内容

拥有一个团队在开始撰写技术规范时使用的模板是加快工作速度和确保重要主题不被忽视的好方法。我建议你的模板主要是一系列的标题和要涵盖的主题区域，也许还带有一些指导或提醒给技术规范的作者。我在 ctohb.com/templates 中包含了一个示例技术规范。

在深入内容之前，先来简单提一下：技术文档可能会有些沉闷严肃。如果适合你的公司文化，我鼓励在适当的时候加入一些轻松的元素，不要分散注意力。一个很好的例子是在文档顶部插入一个聪明的模因，与规范的主题相关。根据我的经验，只需要一个经理/领导人在规范中使用一次模因，就能鼓励团队中的其他人（也就是打开闸门）添加他们自己的模因。

个人建议在模板中包含以下组成部分：

- 提醒该文件实际上是一个模板，作者在开始写作之前应该复制一份（这是一个容易犯的错误！）
- 如何考虑规范/参考公司规范指南和批准流程的指导
- 为项目解释商业原理的背景部分
- 该项目具有特别突出的技术风险领域（例如，涉及敏感的个人身份信息，或涉及以前未使用的工具/架构）
- 对任何不明确的术语提供词汇表/定义
- 明确规范旨在实现的业务目标/与先前设定的目标（如季度关键绩效指标或目标）
- 解决方案架构概述（文档的主要部分）
- 特别讨论技术债务，包括为什么或为什么不解决任何所需/相关的债务
- 数据建模，包括对数据库或数据管道的必要更新
- 内部和外部报告或分析和测量要求

提醒：请仅返回已翻译的内容，不包括原始文本。

- 测试
- 部署
- 功能切换/标志
- 对整体系统可靠性或灾难恢复的影响
- 安全性和隐私
- 可交付的里程碑

技术规范批准

为了确保项目之间和团队成员之间的一致性和协调，您必须确保团队成员阅读和参与彼此的规划过程。 我建议在规范被视为完整之前，对规范进行轻量级的批准过程。

审查目标

您的技术规范审查目标应包括以下内容：

- 确保所有团队/项目在技术方向上保持一致并持续建设。
- 对团队进行重要数据概念/技术合同的审查和教育。
- 确保对问题的普遍而一致的理解。
- 减少错过重要边界情况或其他非业务可见需求的机会。

审查流程

我对轻量级审查流程的建议是在文档中进行异步对话，然后进行同步冲突解决会议（有关冲突解决会议的更多信息，请参见《会议和时间管理》第 28 页）。技术规范的作者应该在项目的关键要素上取得一定的进展后，与其他了解上下文的工程师共享文档。其他人可以在自己的时间里阅读文档并留下评论和问题。其中许多问题可以由主要作者快速和异步地解决，但有些问题可能具有争议性或高度细致，需要更高带宽的沟通。

为了结束这个过程，作者应安排一次只有那些已经阅读文档并提前做出贡献的人参加的会议。这次会议的目的是审查悬而未决的问题和冲突，并做出解决。会议的目的不是让作者简单地向一个厌烦或不感兴趣的听众大声朗读规格说明。如果有需要进一步了解和解决的问题，那么在会议之外进行。解决问题后，记录谁对规范进行了贡献（以便未来读者知道应该向他们提出进一步问题），然后考虑批准该文档。

规范审查会议中的领导者

技术负责人或经理不必成为所有技术文档的批准者。我鼓励您建立一个团队整体感到安全并且不依赖您提供技术支持或指导的文化。刚开始时，当部门规模较小时，您应该参与大部分或全部规范，但这种方法不具备可扩展性。一旦您雇佣了其他高级个人贡献者、架构师或经理，就要授权他们成为领导审查者，并听从他们的意见，让他们做他们的工作。如果发现高级成员在这些审查中没有很好地引导团队，不要在公开场合咄咄逼人。请私下与他们讨论并进行纠正。 技术规格作为文档

您的技术团队现在花时间创建思考周到的文档，涵盖您如何构建产品，因此团队应该因此而犯的错误会更少。技术规格帮助您的另一个方式是为未来需要增强或修改已完成工作的工程师提供有用的资源。我建议您创建一个井井有条且可搜索的目录（例如内部的 wiki，如 Confluence 或 Notion，或者类似 Google Drive 的文档存储），并且您的团队务必确保将所有规格文档添加到目录中。在代码注释中链接或引用规格也可能是有帮助的，以解释为什么要这样实现某个功能。

开发者体验（DX）

DevOps 工具公司 Harness （harness.io）将开发者体验（DX）定义为开发人员在工作过程中产生的整体互动和感受。它类似于用户体验（UX）的定义，只不过在这种情况下，主要用户是软件工程师。

开发者体验可能并不总是在仪表板上进行衡量，但是当设计不良时，团队会明白，并且他们可能会对此大声抱怨。糟糕的开发者体验可能会使工程师的整个下午都被拖延，例如，尝试启动微服务进行测试时抛出了一个难以理解的回溯，并且服务的维护人员正在休假，因此中层工程师将花费数小时来构建可靠的构建-执行-测试循环。 将这种低效性乘以团队中所有工程师的数量，再乘以公司中存在的各种类型的仓库、服务和项目，很快就会导致直接时间上失去数月的生产力。再加上为解决问题而花费的额外上下文切换时间，糟糕的开发体验很快就会成为当未被解决时，能够拖垮一个高效工程团队的领域之一。

拥有出色的开发者体验需要以下两个前提条件：

1.具备简单易用、高度可靠和可重现的环境和依赖链工具； 2.为事务处理提供文档和一致性的实践方法。

庆幸的是，现在有许多可用的工具和生态系统可以帮助满足第一个要求。大多数编程语言都有具有标准化工具用于依赖管理和可重现环境的生态系统。你需要识别并使用它们（例如：npm，pipfile 等）。其中许多系统会生成一个名为锁定文件的文件。

锁文件不是用来并发管理以避免死锁的；它被设计用来锁定特定的依赖关系图实例。您应该提交这些锁文件，并确保其他开发人员和任何构建系统使用它们。锁定文件有助于确保团队中的每个人都安装了相同的依赖关系集。

如果您选择的编程语言没有提供这些工具，那么您需要自己构建可重现性，例如使用 docker 容器、makefiles 或类似的工具。

良好开发体验和差劲的开发体验之间的差距通常只需要对熟悉代码库的人投入二三十分钟的前期努力。

确保基本构建命令在全新安装中正常运行，并将这些命令记录在本地的 README 文件中并不会花费很长时间。

作为 CTO，您的一个机会是确保您公司的不同代码库和存储库之间使用一致的构建命令。无论是否是 docker-compose up 还是 yarn run，任何开发人员都应该能够 git clone 任何存储库，然后能够直接运行第一个想到的构建和运行软件的命令。

开发者体验优先级

无法在产品路线图上列出的事项往往很难进行优先排序。

幸运的是，开发者体验（DX）很少需要投入大量时间来进行路线图上的分类。在公司初创阶段，我更喜欢遵循童子军准则，即使让代码库（或者开发者体验）变得比你找到它时更好。每当开发者遇到构建、运行或测试问题时，他们有责任修复、记录或以其他方式确保下一个使用该代码的人更容易操作。

当系统开始变得更大时，将所有内容一起本地运行以进行功能测试可能会变得越来越繁琐。这时候，值得考虑在路线图上更正式地投资于开发者体验，甚至可以通过专职人员来确保工具能正常运作，开发者不会因为与系统作斗争而浪费大量时间而不是编写有成效的代码。

简单的开发者体验收益

这里有一些简单的方法可以提升软件工程团队的 DX（开发者体验）：

- 在项目中添加一个 README 文件，里面包含了运行代码库的指南，最好是一个命令行一键安装依赖并构建和运行代码的步骤。
- 强制要求所有的代码都要遵循一套严格的代码规范，这套规范在公司内所有项目中都保持一致。如果代码规范检查不通过，构建过程应该失败。如果所有开发者的 IDE 都进行了自动代码规范检查的配置，构建过程中由于代码规范问题导致的失败应该会很少。
- 尽可能把代码规范的配置文件上传到源代码管理工具中（比如设置 VSCode 的 settings.json 文件，路径为 ctohb.com/vscode）。
- 投入时间来确保本地测试数据可以从头开始在本地数据库中建立。通常情况下，一个快速的数据生成器或种子数据脚本可以解决很多开发者的问题。最好的情况是，当系统演进时，种子数据可以很容易地扩充以添加额外的边界案例或使用案例，从而使基本的测试数据尽可能全面/代表性。
- 制定一个计划，了解如何在需要时模拟或实际调用本地的依赖服务以测试多个服务之间的交互。理想情况下，通过良好的契约和面向领域的设计，这种需求会很少，但当需要时，应该很容易实现。 改变开发者体验的工具

在 2022 年，Stripe 这家被估值超过 100 亿美元的金融科技独角兽公司决定，其现有的编程工具 Flow 成本过高。它占用了太多内存，导致电脑卡顿，并且与开发者集成开发环境的兼容性差。

TypeScript 和 Flow 一样，是一个建立在 JavaScript 之上的类型语言。TypeScript 的应用范围比 Flow 更广，因此解决了 Stripe 团队在使用 Flow 时遇到的许多问题，随着时间推移，使用 Flow 变得越来越痛苦。很明显，TypeScript 相比 Flow 在开发者体验方面有了显著改进。唯一的问题是，如何将数百万行代码从一种语言转换为另一种语言?

结果证明，答案是一个由工程师团队进行的为期十八个月的项目，准备进行一次大规模合并提交，一次性更新整个代码库。在 2022 年 3 月 6 日星期日，Stripe 的超级合并完成，而在 3 月 7 日星期一，团队回到工作岗位开始使用新的编程语言。一位开发者形容这个变化是他在 Stripe 工作期间最大的开发者生产力提升。

这个教训告诉我们，如果糟糕的开发者体验带来的痛苦足够严重，那么几乎没有太高的成本或者无法改进的项目。你的团队的规模可能远小于 Stripe，并且你可能没有处理数百万行代码，但是同样的计算也适用：如果你的团队在开发者体验方面遇到了阻碍，你必须投入必要的开发时间和精力来改进，以恢复效率。

另一个问题团队经常面临的是过于频繁地更换工具。在某些技术生态系统中（尤其是 JavaScript 领域），似乎每个月都会推出一些新的、耀眼的工具，可能会为团队提供生产力的提升。我鼓励你在采用新工具时要有纪律性，确保你已经花时间真正理解存在的问题，认真评估新工具，看它是否满足了你所有的需求，而不仅仅是看看耀眼的头条，并根据情况做出决策。有关我在这方面推荐的流程的更多信息，请参阅《Implementing Internal Technology Radar》第 204 页。

技术架构

作为技术领导者，你的一个重要责任是在架构和工具方面做出明智的决策。良好的架构能够将你选择的工具和模式的优势与你组织现在和可预见的未来的需求相匹配。这需要理解每种选择所固有的优势、弱点和权衡取舍。我在本书的这一部分的目标是让你对各个领域的选择的整体情况有所了解，并帮助你认识到不同策略所涉及的一般性权衡。

在与团队讨论工具和工具选择时，有一点需要牢记：工程师在工具选择上可能会情绪化。工具会被评价为好和坏，人们有个人的喜好、厌恶和偏见。作为领导者和决策者，我强烈警告你不要在讨论工具时采用这种说法。这不仅可能让团队成员感到被冷落，如果你贬低他们个人最喜欢的工具，而且也是不生产性的，会分散注意力，无法实现找到解决问题的好方案的目标。有些个别的工具确实设计得很糟糕，并被更好的替代品所遮盖。

多数情况下，更细致入微的评估会揭示出某个特定工具并不是固有地糟糕，而是适合或不适合某个特定公司或项目。不要让过去试图使用不适合解决某个问题的工具的不良经历，阻止你或你的团队在另一个时候使用它，因为它可能更加适合。

架构模式

有许多优秀的资源深入探讨各种建筑模式；其中我最喜欢的是马丁·福勒的《企业应用架构模式》。在本章中，我将提供一些关键词汇的摘要，以便在其他地方深入探讨这些主题时具有上下文。

领域驱动设计

领域驱动设计（DDD）是一种软件开发方法，它专注于理解和建模问题领域，以设计更好的软件解决方案。 DDD 的核心概念包括：

领域模型： 在技术系统中将业务概念表示为对象的方式；

共通语言： 公司内部使用的一种通用且一致的词汇和语言，以减少混淆；

有界上下文： 领域模型适用的边界，也是使用共通语言的范围。

高层级模式

当有人使用术语技术架构时，通常是指代码的执行方式以及信息在系统中的传递方式。架构的描述通常涉及服务、单体应用或消息传输等术语。这与编码模式相对，编码模式经常出现诸如面向对象、函数式编程或依赖注入等词汇。编码模式有时也被称为代码架构，并在《编码模式》的第 188 页进行了讨论。

技术架构中具有最大影响的决策是代码是作为单体应用还是一组服务（通常称为微服务）运行。我将从描述每种模式的外观开始，然后提供一些关于它们之间权衡的指导。

单体应用架构

单体应用架构是指所有代码作为一个单一进程来执行的模式，在该模式中，信息完全在内存中的系统部分之间通过模拟的简单函数调用进行传递。如果你曾经坐下来在一个下午内构建一个简单的应用程

序，那么很可能它属于单体应用的范畴。单体应用的形状和大小各异，从非常小型的项目到包含数百万行代码的重大项目。

构建成功的单体应用的关键是通过使用领域驱动设计仔细设计应用程序中的数据流动。你可以很容易地衡量这一点；你希望确保当开发人员需要更改应用程序的功能时，他们清楚地知道他们应该在单体应用中的哪个部分工作。他们只需要在一个明显和明确定义的区域内更改代码，以实现他们的目标。每增加一个需要更改的代码区域来满足功能要求，都会增加额外的复杂性或错误的机会，并且通常会降低开发速度。 单体应用的关键特性：

- 代码作为一个单一单位部署。
- 代码在一个源代码仓库中管理。
- 部署的代码以一个单一单位进行上下扩展。
- 信息通过内存在系统的不同部分之间传递，通常使用函数调用。
- 领域驱动设计和清晰的信息流设计不被系统所强制执行，这留给工程师来进行良好的设计。

服务导向架构(Soa)/微服务

面向服务的架构（SOA）这一短语起源于 1990 年代，并用于指代一些相当具体的技术选择。如今，该短语被用于更广泛地描述信息在系统各部分之间通过网络传递的系统。使用 SOA 的主要权衡是，相对于单体架构，它可能非常复杂，需要团队做大量的设置和深思熟虑的设计，以确保其益处超过增加的复杂性。

微服务是面向服务架构的一个子集，每个服务都像其名称所预示的那样非常小。有一些系统实现包含数千个微服务，每个微服务只有几行代码。也就是说，您不需要有数千个微服务来体验面向服务架构的好处。在适当的情况下，将系统分解为四个或五个较小的服务，可以极大地改善代码健康情况。

你可能听说过，微服务是唯一好的架构模式；这是不真实的。这种观念源于许多单体架构设计不良或没有得到足够的关注和技术债投资，无法释放出生产力的潜力。同时，认为所有微服务架构都很愉快的工作也是不真实的。由于各种原因，许多微服务实现也无法实现预期的好处。

SOA 或微服务系统的关键特点是：

- 不同的服务可以独立部署和扩展。
- 代码由单一代码库或多个代码库来管理。
- 信息通过网络在系统的不同部分之间传递，通常使用 HTTP、RPC（远程过程调用）或队列系统。
- 数据合同必须经过有意设计和深思熟虑，因为合同被实现为 API，并通过网络进行通信。

在面向服务的体系结构和单体架构之间进行选择

一般来说，单体架构比面向服务的体系结构更容易设置，并且需要更少的技术后勤工作来管理。因此，在绝大多数情况下，单体架构是首选。如果团队在设计单体架构时非常有纪律和深思熟虑，它可以随着团队的发展而扩展。然而，并非所有人都是如此。对于许多团队/项目来说，单体架构缺乏强制合同、无法作为独立组件进行扩展以及缺乏强制关注点分离，这些将成为生产力的障碍。

如果你发现自己面对一个难以管理的单体架构，这并不意味着你的工程师做得不好。软件工程的本质就是需求的变化和系统的演化。维护单体架构可能意味着，有时要投入大量资源来更新系统设计以跟上发展，如果一个团队未能做出这样的投入，单体架构的复杂性将成为生产力的障碍。 有一些情况下，转向面向服务的架构明显是正确的选择：

您的服务具有需要独立扩展的元素。例如，某个功能消耗大量的 CPU 资源，您不希望它干扰其他功能，或者您更倾向于只针对该功能进行成本效益更高的独立扩展。

您正在开发需要公开自己独立 API 并具有自己独立数据领域的功能，与主系统分开。特别是如果这个 API 是为外部客户提供服务的，那么将这个功能作为自己的服务是显而易见的明智选择。

由于某种原因，您需要在应用程序中使用另一种编程语言。一个很好的例子可能是，因为 Python 有一个强大且高质量的框架来解决某种类型的问题，但您的应用程序的其余部分使用 Java。在内存中桥接这两种语言是可行的，但效率低下。更简单的选择是通过 API 桥接它们，使它们自然地作为独立服务托管。

部署您的整体架构过于昂贵、缓慢或有风险。在这种情况下，您可以通过将新代码作为独立服务部署来提高额外的生产力并减少部署时间。只要确保新服务独立于整体架构运行，并且不会创建新的部署依赖关系。

面向服务的架构的源代码控制：单一仓库和多个仓库

管理单体架构的源代码相对简单，因为它存在于单一仓库中，使用单一的构建系统。一旦您开始将代码拆分为不同的包、项目和服务，您将面临一个决策：您是在单一代码仓库中管理多个服务，还是创建多个仓库？这种权衡被称为单一仓库（monorepo）与多个仓库（manyrepo）之间的选择。

如果您选择将多个服务作为单一仓库进行管理，您可能希望寻找一个工作区管理解决方案（例如，JavaScript 生态系统中的 yarn workspaces），以便独立构建各个项目。以下是单一仓库和多个仓库方法之间的一些基本区别：

**单一仓库的优缺点 TODO：将图表放在这里

这样可以轻松确保每个服务或包的依赖关系都是最新版本。 很多 CI 系统本身不支持在单个仓库中使用多个包，因此你需要手动构建一个支持多包的工具来实现这一点。

将所有的代码放在一个仓库中可以提高可发现性，让开发人员更容易找到他们正在寻找的模块或参考。集成开发环境对这种搜索方式有良好的支持。

与此相反的是 Manyrepo

需要使用一个带有版本控制的中央包管理器。虽然这本身并不是一件坏事，但在同时开发项目和其依赖关系时可能会增加很多额外的工作量。

与 CI/CD 流水线系统（如 Bitbucket pipelines、GitHub actions 等）可以很好地集成。

我一般的建议是保持简单。对于中小规模项目来说，使用单体存储库（Monorepo）设置和维护更加简单。转向多存储库（Manyrepo）意味着愿意投资工具来确保开发人员能够顺利使用多存储库；这是一

个重大成本。对于一个小型创业公司来说，这个成本可能不值得。但是如果你的公司快速发展或者拥有 50 个以上的开发人员，并且单体存储库变得难以控制，而且你拥有一个专门的内部平台或者 DevOps 团队可以帮助你使多存储库易于使用，那么转向多存储库模式可能是正确的选择。

分布式单体应用

分布式单体应用是指部署的多个服务彼此间缺乏足够的独立性和隔离性，因此无法独立部署。需要明确的是，这是最糟糕的情况。开发人员不能仅仅前往某个服务并在隔离环境中工作，而是需要思考该服务对其他服务的影响。不仅如此，他们还可能需要在多个服务中进行更改，并在特定顺序中协调部署，以确保发布期间的兼容性。这种开发和部署复杂性削弱了微服务系统的主要好处。

如果你注意到团队陷入这种模式，或者抱怨在服务之间协调发布，这就是一个警示信号，需要仔细观察并考虑解决一些技术债务，以恢复独立可部署的服务。这种技术债务通常出现在合同、API 设计和系统中数据处理的方式。

编写可读性好的代码

在专业环境中，任何给定的代码行的主要受众不是计算机，而是将来可能需要阅读该代码进行进一步开发的开发人员。这是编程的黄金法则：工程师编写的代码应具有与任何其他人的代码相同的可读性。

语言和生态系统的选择

根据编程的黄金法则，你选择的语言应使你的团队能够编写高度可读且易于维护的代码。通常，优秀的工程师可以在任何语言中实现这一点；然而，有些语言比其他语言更容易并持续实现。选择语言或生态系统时需要考虑以下一些因素：

- 熟悉该语言的人才库有多大，尤其是对你的创业公司感兴趣的人才库有多大?
- 是否有现有实现可以用作起点?
- 是否有特定的性能或扩展要求？某些语言在特定类型任务上比其他语言快得多。Haskell 在字符串操作上以效率低下而闻名，而 C 语言在大多数任务上非常快速，尽管有其他语言在某些问题上接近或超过了 C 语言的速度，同时还提供了更容易和友好的编码环境。
- 是否有特定的框架在某种语言中可能是一个很好的起点？例如，React Native 是一种强大的跨平台移动语言，需要使用 JavaScript 或 TypeScript。

在企业环境中，我推荐使用具有静态类型系统的语言，例如 Golang、TypeScript、Rust 等，这样编译器可以更多地帮助确保代码的正确性，这些约束条件可以对其他开发人员可见，也可以避免在运行时遇到这类问题。你应该努力实现一个本地开发环境，在代码执行之前工具可以找到错误，这被称为编译时检查。修复编译时检查通常比修复运行时问题更快更便宜，而且由于自动化的特性，它比运行时检查更有能力可靠地发现问题。

代码风格与格式化

在任何广泛使用的语言中，都会有一个已发布的代码格式化标准（例如 Python 中的 PEP8）或者一个可配置的工具，可以强制执行特定的代码风格和格式化（例如 JavaScript 中的 ESLint 或 Prettier，或

者 C#中的 ReSharper）。大多数这些工具都非常擅长确保代码在样式上是一致的，无论是谁编写的。为了确保你的代码库可读性强，没有理由不使用其中的一个工具，并确保你的整个代码库都按照相同的规则格式化。你可以根据团队和个人的喜好选择使用哪些规则，但一定要保持一致，生成可读的结果。

我建议你为开发人员使用的集成开发环境（IDE）准备一组配置选项或指令，以在保存文件时自动格式化代码。然后，在你的持续集成系统中，确保所有新代码都被正确地格式化。 小心：在没有自动格式化的情况下，在持续集成系统中强制执行代码风格会让工程师感到非常沮丧，因此请确保在第一天就培训每个人正确设置他们的集成开发环境，以避免在持续集成中出现一致的错误和浪费时间。

静态代码分析

现代静态代码分析能够识别和警示一系列常见的代码问题，包括安全漏洞、错误和风格不一致等。这些工具价格相对较低，并且能与常用的持续集成系统和开发者集成开发环境无缝集成。通过在各种项目和编程语言上使用这些工具的经验，噪音比很低，输出对于提高生产力和软件质量有正面影响。在软件项目的早期阶段，你应该开始整合静态代码分析。我鼓励你使用与你选择的编程语言相关的工具，如 ESLint 用于 JavaScript，以及通用的分析平台，如 SonarCloud、Codebeat、Scrutinizer-CI、Code Climate 或 Cloudacity。

Greenfield vs. Brownfield

Greenfield 软件开发指的是在新环境中进行开发工作，几乎没有现有的遗留代码，并可以自由选择工具、模式和架构。这样做的明显优势是能够在工作中深思熟虑地选择合适的架构和工具，并且不会受到现有技术债务的干扰。微妙的缺点是，在有这么多选择和如此少的约束的情况下，糟糕决策的风险更高。对于新项目来说，通常还需要付出相当大的投入成本，这些成本经常被低估，例如设置测试、构建系统、静态代码分析等等。 棕地软件开发是相对于绿地开发而言的，它指的是与现有遗留系统一起工作。这些权衡基本上是相反的：无论好坏，你都会被前人做出的高层决策所束缚。

棕地开发中最大的风险是"非此地不建"综合症。"非此地不建"是指个人倾向于回避对自己没有创造的事物负责或给予足够关注。在棕地软件开发中，这可能导致对现有工作的理解系统性不足，从而导致对现有系统的增补或修改时产生挫败感和低效率。我强烈建议管理者为团队在任何修改现有系统之前读取和理解现有系统提供显式空间。前期的理解时间将因减少惊喜和提高后续工作速度而回报。

编码模式

对于许多程序员来说，编写何种风格的代码是一场宗教讨论。我在本章的目的是简要介绍编码模式中最常见的短语的含义，并为每个实践提供更广泛的资源。

如果你遇到了一场情感激烈的关于这个问题的讨论，请记住，存在许多成功的公司使用每种模式。一切都是一种权衡。一个糟糕的程序员可以用任何工具搞砸，反之，一个优秀的程序员将找到一种方法来用次优的工具找到可读的解决方案。

面向对象编程 (OOP)

面向对象编程 (OOP) 是一种设计代码的方法论，通过模拟现实世界中的名词和动词来进行。一个典型的例子是将两个人的互动建模成两个人物对象，人物的任何动作，比如说说话，都会成为这些对象的函数。许多语言本身就是面向对象的，比如 Python、Ruby 和 C#。有些语言，比如 JavaScript 或 C++，是面向对象可选的（支持一定程度上的面向对象和函数式风格），还有些则完全不同。

纯洁性

纯洁的代码没有外部依赖或副作用。换句话说，给定相同的输入，纯洁的代码总会产生相同的输出。纯洁代码的优势在于易于测试，并且不需要外部设置或模拟。纯洁的代码也更易于阅读和理解，因为它不需要阅读其他代码才能理解其功能。一个简单的纯洁代码的例子是一个将两个数字相加的函数；给定任何两个输入数字，求和函数总是产生相同的输出。

有些代码本质上是不纯的，比如与外部世界交互的代码，如文件系统、网络或数据库。对于大多数其他情况，我们可以以纯洁的方式对业务逻辑进行建模。在可能的情况下，我鼓励您和您的团队编写纯洁的代码。

函数式编程

为了符合描述编码模式的词类模型，函数式编程将动词（函数）作为系统的第一类部分。大多数函数式编程从非常小的功能部分开始，然后将它们组合在一起以创建更复杂的系统。当使用得当时，函数式代码的好处在于它更加纯净，因此更容易阅读、推理和隔离测试。甚至存在学术上的函数式代码示例，可以进行形式化的推理，也就是说可以产生数学证明代码运行正确性的证明。

函数式编程的不足之处是会产生非常冗长且难以阅读的代码。例如，当组合多个函数时，重要的是考虑被组合的函数数量以及每个函数在组合链中的行为是否明显。

最糟糕的情况是：想象一下连续十个函数的函数链，每个函数的名称对你来说毫无意义（例如，a(b(c(d(e(f(g(h(i(j(input))))))))))）。唯一更糟糕的情况是，这些字母函数的定义在代码库的十个不同文件中，或者更糟糕的是来自不同的导入库。

极限编程和测试驱动开发（TDD）

极限编程是一种开发方法论，类似于敏捷或 SCRUM。它可以用来指代肯特·贝克（Kent Beck）所著书籍《极限编程简介》中描述的正式方法论，或者更非正式地指称该方法论所倡导的一些编码实践。该短语的非正式用法描述了该方法论中测试实践，特别是测试驱动开发的理念。

测试驱动开发（TDD）是一种在编写功能软件之前编写测试的过程，而不是先编写功能代码再编写测试。行为驱动开发（BDD）和验收测试驱动开发（ATDD）是类似的实践。

依赖注入

依赖注入是一种模式，其中特定对象、模块或代码块的服务依赖项是通过传递而不是实例化来获得的。例如，数据对象可以通过在配置文件中查找连接字符串并创建数据库客户端来实例化自己的数据库连接。或者，一个父代码块可以创建数据库服务，然后将单个数据库服务传递给每个数据对象的实例。

依赖注入的主要优势在于减少了服务与其依赖之间的耦合，有效地为它们之间增加了一个文档化的接口。这个接口允许在测试环境中使用其他接口实现，例如模拟服务。 清洁地进行依赖注入存在一些微妙之处。我鼓励您采用常用且经过深思熟虑的框架或模式，适用于您所使用的编程语言。

领域驱动设计

领域驱动设计这个术语源自 Eric Evans 的著作《Domain-Driven Design》，该书于 2003 年出版。其核心思想是创建一个模型，不论是面向对象设计中的对象还是数据库架构中的模式，用以模拟您的业务领域中的名词。这看起来可能很简单和直观；然而，在复杂的业务领域中，代码很容易以不一致的方式来模拟领域，或者以一种阻碍团队理解的方式来模拟领域。特别是在面对更大、更复杂的问题时，我始终坚持让团队一起坐下来，达成一致的方式来对问题进行建模，使用统一的术语来指代整个系统中的相同概念。

API 合约

应用程序接口（API）与法律合同类似。

这是在实施之前事先设计、调整和达成一致的，双方都期望对方遵守合约以达到预期的结果。当您设计和实施 API 时，您向 API 的使用者承诺它将以某种方式工作。就像法律合约一样，您可能对 API 的功能有特定的想法，但如果对方对细微之处有不同解释，您可能无法实现目标。API 的细节真的很重要，作为技术负责人，您的角色是确保团队以一致、高效的方式设计和构建 API。

尽管如此，构建高质量的 API 是一项出乎意料的复杂任务。它需要考虑许多因素：设计接口、实现处理逻辑/数据的代码、测试功能、构建文档、处理版本控制/变更管理、及时更新文档以适应 API 的变化，并使开发人员与 API 交互变得简单。做好这些事情可能意味着在构建开发人员喜欢的 API 和妨碍实施并减慢重要项目上线速度的 API 之间的区别。作为领导者，您有两个主要手段来确保处理这些事情得当：治理和架构。

API 设计治理

构建 API 涉及许多决策。好的 API 与糟糕的 API 之间的区别在于这些决策的一致性、可预测性和正确性。作为技术负责人，您的职责是确保在您的组织中，您有一套结构来帮助开发人员构建彼此一致、可预测且正确的 API，以使用适合具体问题的常见模式。

实现这些目标需要一种形式的治理系统。这可以是一套清晰记录的准则和标准，也可以是一组负责定期审查和批准所有 API 的人员。团队越大，您需要在流程和治理上投入更多的时间和精力，以保持高标准。

API 架构

在实际应用中，你可能会遇到两种主要类型的 API：基于 HTTP 和非基于 HTTP。与任何工具一样，HTTP 有其权衡，不适用于每个任务，因此如果您的业务需求要求超低延迟、超高吞吐量/低开销或实时流应用，您可能会寻找超出 HTTP 的解决方案。下面我将讨论一些 HTTP API 类型，并简要介绍一些非基于 HTTP 的 API。

基于 HTTP 的 API

如果您正在构建 Web 或移动应用，甚至大部分系统后端，很有可能您主要处理的是 HTTP API。

XML 和 SOAP API 在 2000 年代初，最常见的 API 模式是基于 XML 的简单对象访问协议（SOAP）。SOAP 和其他基于 XML 的 API 样式已经在 2020 年代的初创公司中过时了，但它们在遗留系统中仍然很常见，特别是在技术发展缓慢的大型公司中。你不应该构建新的 SOAP 或基于 XML 的 API。

REST

REST（表现层状态转移）是一个通用短语，描述了使用 JSON 通过 HTTP 作为 API 的方式。REST 有时会使用一种称为 HATEOAS 的模式进行增强，该模式提供了对 REST API 内容/负载的更正式的标准集。在没有 HATEOAS 的情况下（这并不常见），REST 不包括关于如何建模 JSON 数据的正式或品牌指导。REST API 通常将单个名词建模为一个端点，并使用 HTTP 动词（GET，PUT，POST，DELETE 等）来确定对名词的操作。例如，GET/users 将列出用户，POST/users 将创建新用户，DELETE/users/123 将删除 ID 为 123 的用户。

REST 很可能是你会遇到的最常见的 API 形式。REST 具有广泛而强大的工具生态系统，几乎每个工程师都熟悉它。

GraphQL GraphQL 类似于 REST，它也是通过 HTTP 传送 JSON 数据；但是，它并不依赖于 HTTP 动词。在 GraphQL 中，几乎所有的操作都是 POST 请求，并且它使用了结构化的查询和变更模式。

我觉得 GraphQL 就像是带有类型和自我记录模式的 REST。因此，GraphQL 的 API 往往会自动生成文档和模式浏览器。由于 GraphQL 的模式系统，它还允许从多个服务中组合多个模式，形成一个更大、更强大和更复杂的数据图，有时被称为联合模式。Apollo 公司提供了用于管理和扩展图形的高级解决方案。

使用图形来建模公司数据以及强制设计模式所带来的好处是不容忽视的。然而，没有哪个系统是没有代价的。由于 GraphQL 没有使用传统的 HTTP 动词，它与 Web 堆栈的某些元素不兼容。GET 请求的缓存和开发工具仍在不断改进以更好地支持 GraphQL 请求。如果这些缺点对您的业务并不重要，我强烈建议您访问 apollographql.com 并考虑在您的 API 中使用 GraphQL，尤其是针对内部使用案例。

非 HTTP API

一般来说，对于传统的同步请求/响应式 API（也称为远程过程调用或 RPC），由于其普及性，您会希望使用 HTTP API。然而，有一些 API 模式，尤其是用于异步操作的模式，无法简单地映射到 HTTP，因此有常用的替代实现。 排队系统

一个排队系统通过维护一个收件箱（或多个收件箱）来接收消息，并提供一个接口供消费者读取消息并具有特定的保证。

一个典型的排队系统可以保证消息的顺序（先进先出，FIFO；或后进先出，LIFO），同时可以提供至少一次或最多一次的投递。大多数云平台都有托管的队列实现，比如 AWS Simple Queue Service (SQS) 或 Google Cloud Task queues。

排队系统通常具有“显式”调用的概念，即当发布者创建一条消息时，它会明确指定请求应该如何处理或执行。与之相反，大多数发布-订阅系统支持“隐式”执行。这意味着发布者不一定预先知道哪个系统将处理该消息，只知道发布-订阅系统将会传递它。

“发布-订阅（pub/sub）模式”

发布-订阅模式，简称 pub/sub，允许设计一个系统，在该系统中，消息可以由多个来源创建，并通过不同的模式传递给多个订阅者。发布者-订阅者关系可以建模为一对一（直接）、一对多（扇出）、多对一（扇入）和多对多。各种 pub/sub 实现可以提供消息被传递到所有订阅者、至少一个订阅者、至少一次等保证。类似于队列，有现成的解决方案，如 RabbitMQ，也有易于扩展的云托管选项，如 Amazon 简单通知服务（SNS）或 Google Cloud Pub/Sub。

pub/sub 模式及其提供的保证非常强大。然而，其中的权衡是实现需要一些小心和注意细节，以实现宣传的保证。例如，实现订阅者需要密切注意消息确认语义，并仔细管理主题订阅，以确保正确的消息发送到正确的位置。

如果你在使用队列、pub/sub 或 HTTP API 之间犹豫不决，我的一般建议是保持简单，选择同步的 HTTP API。你犹豫于使用哪种实现说明异步系统提供的保证对你的实现并不是关键，在这种情况下，增加的复杂性可能对你的初创项目来说不值得。

任务系统

任务（或定时任务）是一种后端 API，很少由发布者或客户端触发，而是由某种形式的计时器触发。常见的例子包括每夜清理数据任务或每周发送电子邮件摘要/通知。一些任务的最佳实践：

- 使用由他人维护的任务系统，不要自行构建。
- 在选择任务系统时（或者如有必要自行构建），确保它具备以下功能：
  - 每次任务执行时都有日志记录；
  - 允许配置失败任务的重试；
  - 在任务失败时提供通知。很常见的情况是工程师设置了一个定时任务，在第一天它能正常运行，但到了第十五天失败了，直到第三十天才有人注意到；
  - 提供查看任务和任务状态的界面；
  - 允许将任务配置存储为代码或配置文件并放入源代码控制系统中；
  - 允许根据需要在内部环境/私有网络/安全组内运行任务以访问其他内部系统的 API/资源；
  - 与您的秘密管理系统集成；
  - 允许在开发和生产环境中轻松设置任务，并在每个环境中进行简单测试。

文档

对于你的 API 来说，拥有全面、清晰和最新的文档与你的构建和维护方式一样重要。一些优秀 API 文档的关键特点有：

- 始终与实现保持最新
- 记录所有可能的输入及其类型，记录所有可能的错误

- 其他工程师能够轻松阅读和浏览

使用包含 API 文档生成功能的系统构建你的 API 是一个明智的选择。如果不这样做，将几乎不可能保持这些目标的一致性。如果你正在构建一个 REST API，我强烈建议你使用 OpenAPI 设计你的 API（一个描述你的 API 的 YAML 或 JSON 文档）。在大多数编程语言中，都有用于使用 OpenAPI 规范来自动生成控制器/路由或生成测试工具，以确保实现的 API 符合规范的 SDK。此外，还有一些在线工具，如 stoplight.io 和 readme.com，可以解析 OpenAPI 文档并生成美观易读的文档，并提供便于导航的功能。 如果您正在使用 GraphQL，GraphQL Playground 或 Apollo Studio 浏览器可以作为对广泛类型文档的替代品。我建议您仍然构建一个单独的 API 文档页面，可以使用 readme.com 之类的工具或手动创建，作为入门指南。内置的 GraphQL 文档缺少关于身份验证工作方式的描述，并且在解释 API 中数据之间的关系方面做得不好。

这些是您需要在其他地方填补的空白。

使用 OpenAPI 或 GraphQL 的另一个好处是，生成的 API 规范不仅适用于文档生成器和测试框架，还适用于开发者 IDE，如 Insomnia 或 Postman。这些 IDE 使开发人员能够快速与 API 交互，验证功能而无需编写代码。正式规范还可以与代码生成工具一起使用，以确保代码中的类型一致性。

幂等性

当多次发出相同的请求与仅发出一次请求具有相同的效果时，称 API 请求具有幂等性。幂等性是构建健壮系统和避免数据损坏的重要概念。与所有事物一样，幂等性为系统提供了有用的保证，但它也带来了一些成本：实现幂等性会增加后端系统的复杂性。 在 REST API 中，普遍假设除了 POST 之外的每个 HTTP 动词都应该是幂等的。举个例子，GET 请求根据定义应该始终返回相同的结果（除非底层数据发生变化）。通常情况下，PUT 请求用于修改现有对象，因此自然应该是幂等的。然而，在大多数系统中，多次调用 POST 请求表示要创建多个对象的意图。

幂等性键

在 REST 中的 HTTP POST 请求以及 GraphQL 突变 API 中，标准/规范并没有提供幂等性。如果你希望客户端能够重试这些类型的请求并保持幂等性行为，你应该实现幂等性键模式。幂等性键是一个由客户端提供的任意字符串（可以是 HTTP 头部，也可以是 GraphQL 的输入变量），后端使用它来去重传入的请求。这要求后端存储幂等性键，并且存储使用该键的请求的响应，以便稍后提供给客户端。

请注意，实现幂等性键并不简单，因为它需要额外的数据库写入，以及在捕获请求响应和处理同时到达的重复请求时处理并发/锁定问题。如果幂等性对于你的应用程序很重要，比如处理财务交易，我建议你采用后端 API 实现，它提供了一个强大的开箱即用的幂等性系统，而不是从零开始构建它。

数据与分析

大多数初创公司在业务中使用至少三种不同类型的数据：

- 交易数据
- 分析商业智能数据

- 行为数据

每一种数据类型都会有不同的数量、读写模式和需要不同的工具来可视化和获得见解。

关于大数据的说明。作为初创公司，很有可能你并不需要将数据架构设计成无限规模（或者 Web 规模）。通常的现成数据库配备合理数量的硬件和合理的数据模型设计，完全能够处理数千万行和数百 GB 数据而具备可接受的性能。大多数大数据解决方案，例如数据流水线或数据仓库设备，都需要额外的设置复杂性、延迟和成本，并且它们很可能对你的初创公司来说过于复杂。为简单起见，只有在你能够有说服力地证明常规数据库（例如 PostgreSQL）无法完成工作时，才应考虑采用大数据解决方案。换句话说，不要过早地对数据库架构进行优化。

交易数据

交易数据是驱动您的应用程序本身的数据，通常是您的主要 NoSQL 或 SQL 数据库。交易数据需要非常低的延迟和高可用性，并且总体大小与其他形式的数据相比较小。我建议您选择现成的 SQL 或 NoSQL 解决方案，最好是像 MongoDB Atlas 或 Google Cloud SQL 这样为您托管的解决方案。在您的生产数据库中，以下是一些有用的功能：

- 一键式的时间点还原
- 定期备份并一键式还原
- 用于负载分担的只读副本
- 多区域复制和托管以提高可用性
- 基于事件的审计日志
- 自动磁盘扩展/收缩
- 连接/IP 级别安全
- 资源（CPU、RAM、磁盘、网络）监控和警报
- 一键式扩展/缩减 CPU 和 RAM
- 慢查询监控

分析型商业智能数据

商业智能（BI）数据是用于深入了解您的用户行为的数据，通常来自您的交易数据。初始阶段，您通常可以直接在事务型数据库上运行商业智能查询。随着数据量和查询复杂性的增加，这变得更加有问题，因为它给需要高可用性和低延迟的系统增加了额外的负载。解决方案通常是查询事务型数据库的只读副本，或者通过数据管道将数据复制/转换到另一个数据存储系统。

构建数据管道和数据仓库是一本完整的书，并且技术总是在不断发展。我只能给出一些建议： 考虑寻找企业数据解决方案，例如 Snowflake、Databricks 或 Google BigQuery，用作主要的商业智能数据仓库。这些工具是改变游戏规则的。特别是无服务器的仓库（如 BigQuery、Aurora），设置起来非常简单，无论数据规模大小，延迟都相对稳定，并且对于初中期的创业公司来说具有高性价比。

在现代，创业公司不需要建立和托管复杂的数据管道架构了。ELT（抽取、加载、转换）和 ETL（抽取、转换、加载）工具现在可以完全在企业数据库数据湖/数据仓库中运行，而像 dbt 之类的工具提供

了可重复性、可测试性和管道即代码的能力，使得运行数据管道更加可控。

考虑使用托管或云原生的数据可视化解决方案，如 Looker、Domo 或 Preset。

确保工程和产品团队与负责数据和商业智能的团队成员密切合作。在产品流程的早期引入数据的角度将大大减少后续的麻烦，遵循“量取两次，创建数据模式一次”的思维方式。

行为数据

行为数据，也被称为行为分析事件，描述了用户如何使用您的应用程序。行为数据通常具有较高的数据量，模式有些有限，并且最适合与功能强大的可视化软件结合使用。

总体而言，您希望从应用程序获取行为数据并发送到多个源。这带来了一个路由问题：您只能有一个数据源（您的应用程序），但您希望事件发送到多个地方。几乎普遍采用的解决方案是 Twilio 的 Segment 平台，尽管也有一些叫做客户数据平台（CDP）的新兴替代品，如 RudderStack。CDP 可以从您的应用程序接收数据，然后将其发送到数据仓库以及您想要的许多其他 SaaS 平台。

行为数据与应用程序生成的交易数据之间的一个重要区别是其精确度。大多数行为数据是有损失的，用户可能有广告拦截器，请求可能会丢失，或者防火墙可能会阻止连接。有很多原因可能导致事件无法从客户设备传输到您的 CDP。这并不意味着行为数据没有用处，但要意识到其有损失性质，以此来确定数据的期望，并限制使用查询的用例。如果您需要精确的数字，请从您的 BI 平台和交易数据中获取。

架构设计的一般提示和最佳实践

让我们用一些建议来总结这一节，帮助您设计您的架构。

将业务逻辑放在后端

在构建应用程序时，常常会面临一个选择：逻辑应该放在客户端（如 Web 浏览器、移动设备、物理硬件）还是某种形式的后端服务器上。对于某些类型的逻辑，比如与身份验证、值计算、防作弊/篡改机制相关的逻辑，这是一个明确的要求。对于其他大部分逻辑来说，出于以下原因，将其放在后端仍然是一个好主意：

后端的逻辑通常比客户端更容易进行测试，因此您可以更有信心地确认后端上的业务逻辑的正确性。

在后端上放置更多逻辑意味着客户端会更轻巧，并且意味着您可以为多个平台生成客户端，这些客户端可以利用单一的逻辑源，减少代码复制。

在后端上的逻辑无法被客户端篡改或修改。

将服务外部化

从后端到其他后端，或从后端到前端的 API 应该被视为可以被第三方使用的通用 API。这将迫使您保持一些良好的设计习惯，包括确保接口本身清晰易懂（面向域设计），使用合理的身份验证机制和适当的高级所有权抽象在数据设计中。而且，万一你将来希望将服务外部化，这样做的道路将更加简短。

尽量少使用编程语言

每种编程语言都有与之相关的构建系统、依赖管理系统、编程最佳实践和接口。您的团队应该花费相当多的精力来确保您主要的语言和生态系统与本地开发人员、测试环境和生产环境的集成良好，并且在这些环境中运作良好。

在您的栈中添加任何额外的语言时，您需要复制所有这些工作，并且无法在运行时之间共享代码。在允许栈中添加其他语言之前，您应该能够构建一个强大且牢不可破的论点，证明新语言的好处超过了新语言带来的运营和维护负担。否则，您最好不要添加它。

工具

软件工程的工具生态系统和模式不断发展和变化。你不可避免地会被自己或团队成员引诱，改变你进行工程的某些方面，比如采用新的库、框架、语言或模式。采用这些变化往往很快导致一个脱节的拼凑架构。相反，忽视所有变化会导致一个陈旧的代码库，随着时间的推移，效率会变低，新来的人才很难进行工作。正确的方法是正式化改变技术栈的过程，并提供一些防护措施来激励团队对工具变化进行好奇和深思熟虑。

实施内部技术雷达

Thoughtworks，一家总部位于旧金山的领先软件咨询公司，发布了一款名为“Technology Radar”的工具（ctohb.com/radar），用于评估 Thoughtworks 每年看到的数百个项目。他们将新工具、新技术、新模式和新语言（他们称之为"blips"）根据在现实世界中的有效性分为四个类别。

这些类别分别是"保留"，"评估"，"试用"和"采用"。

如果你从未读过 Thoughtworks Radar，我强烈推荐它作为一个对当前情况有了解的指南，同时也可以为你自己团队的流程提供灵感。

在保持工程师的积极性和代码库的相关性与工具变动之间取得平衡的方法是跟随 Thoughtworks 的做法，制定一个内部技术雷达。与 Thoughtworks 的普适性评估不同，我的方法是使用相同的四个级别来评估 blips 在我们组织中的适用性和有效性。具体来说：

1. 有人提议使用一种新的工具、技术、平台或语言（blip）。最初，该提议被分类为"评估"。提出者必须在技术文档中说明新的 blip 对已被业务选择的项目（或创新冲刺阶段的实验见 Cooldown/ 创新冲刺，第 163 页）的实质性益处。然后，如果获得批准，该 blip 会进入试用阶段。
2. 开发人员在项目中使用这种新的 blip，无论是由业务选择的还是在创新冲刺阶段选择的。项目结束时，作者会撰写一份后续文档，描述他们对该 blip 的体验，包括优点和缺点，以及该 blip 如何与公司的其他工具生态系统协作。
3. 基于试验结果，团队整体上将采取以下两种行动：要么采纳该提示并解锁给其他团队成员使用，无需额外的仪式；要么将其保留，并需要进行新的试验和评估以再次使用。如果商业项目中的试验失败，建议团队仔细考虑是否移除该提示以避免未来的维护问题。

在大多数情况下，我发现当一个提示试验失败时，它通常在早期就失败了，项目负责工程师不会将该提示包含在最终交付的实施中。

无趣技术

“无趣技术”是丹·麦金利在 ctohb.com/boring 上提出的一个词汇。关键思想是你的团队的工作是提供支持业务的功能，而大多数情况下并不依赖于使用新潮的工具。事实上，使用一些非无趣的东西通常会有许多隐藏的成本，只有当你的团队完全意识到这些成本并认为收益更大时，才应采用新的工具。如“无趣技术”所描述的，总成本=维护成本/速度收益。一些需要考虑的隐藏成本包括：

- 不完整、不准确或不成熟的文档
- 周围工具/技术生态系统不完善，包括 SDK 和与其他工具的集成
- 更高的可能遇到缺陷或功能/特性缺失
- 团队成员需要额外的培训成本来适应新工具
- 需要投入额外精力来保持工具或软件包的最新状态，修补安全漏洞等等。

工具成本

现代初创公司在 SaaS 上通常会花费很多钱。你的公司很可能也不例外，所以当你发现在 B 轮融资前，你将花费整个员工数量甚至更多的工具和基础设施上时，不要感到惊讶。

预算

有一些已发布的针对各公司阶段 SaaS 和工具支出的基准，以公司收入或总支出的百分比计算。

没有一个确切的基准，但典型的 SaaS 销售成本（COGS）似乎在收入的 10%至 30%之间。

了解你的支出并关注成本增长。很容易不经意间让几台机器在 AWS 上运行，并增加你每年的云托管费用 10,000 美元。大多数云平台都具备内置的预算功能，因此没有不使用它们的借口。如果你使用基础设施即代码，可以很容易地设置一个模块，使每个新部署的云系统在同一时间自动应用一个云预算，用于监控和警报该特定系统的成本。

随着时间推移，SaaS 成本通常会增长，不管是因为基础设施的扩展，还是因为你发现了一个新的 SaaS 供应商可以节省团队时间。我建议不要将成本作为避免采用典型 SaaS 工具（每月成本在数百美元范围内）的理由。相反，我建议将定期增长纳入你的 SaaS 成本预测中。

追踪 (Tracking)

您应该追踪您的组织在工程工具上的支出，包括集成开发环境（IDEs）、软件即服务（SaaS）和基础架构（云平台）。您可以使用电子表格手动进行追踪，或者使用软件即服务管理平台（SMP）。这些解决方案可以从不同的供应商，如 BetterCloud、Zluri 和 Vendr 获得，并且可以与您的信用卡或银行进行关联，自动对现金支出进行分类。

DevOps

维基百科将 DevOps 定义为将软件开发和 IT 运维相结合的一套实践方法。它旨在缩短系统开发生命周期，并以高质量的软件提供持续交付。

对我而言，这个定义的关键是 DevOps 旨在缩短软件开发生命周期，换句话说，DevOps 是团队整体生产力的推动者。如果您还没有具体考虑过 DevOps，那么您可能在一定程度上优先级低或未投资足够的 DevOps。这不仅仅是我的观点；在技术行业中，高质量的 DevOps 被广泛认可为整体工程速度的关键驱动因素。

四个关键指标（DORA）

2022 年 Thoughtworks 技术雷达（ctohb.com/techradar）中评分最高的标志是四个关键指标。这些指标由谷歌云内部团队 DORA（DevOps 研究与评估）描述，该指标体系来自一个长达七年以上的研究计划，对结果及其对技术、流程、文化和定量结果的影响进行了验证。这四个指标如下：

交付时间（Lead Time）： 代码从提交到在生产环境运行所需的时间

部署频率（Deployment Frequency）： 代码发布到生产环境/最终用户的频率

恢复平均时间（MTTR）： 在事故/缺陷发生后，恢复服务所需的时间

变更失败百分比（Change Fail Percentage）： 需要热修复、回滚、补丁等的生产发布所占的百分比

这些指标一起量化了团队能自信地部署软件的能力。在这四个指标上得分高需要在自动化、DevOps、测试和文化方面做投资。正如 Thoughtworks 所指出的，从这些指标中获得价值不一定需要高度详细的仪表盘、指标或仪器设备。DORA 发布了一个快速检查调查（ctohb.com/dora），您的团队可以参与，以了解它在宏观水平上的进展情况。还有很多工具，入门门槛相对较低，可以提供足够的数据质量来指导您的进展，比如 LinearB 或 Code Climate。

下面的 DevOps 子章节介绍了一些改善这些度量指标的概念、学科和重点领域。

可复现性

部署代码是一项非常微妙的活动，需要极高的精确度。配置文件中的一个错误字符可能导致服务启动失败。更糟糕的是，对于大多数工程师来说，调试 DevOps 问题是缓慢痛苦的，而且识别和修复那个错误字符可能意味着在调试和修复中可能丢失几个小时的时间。我们都是人类；这些错误是不可避免的。由于在 DevOps 中它们非常昂贵，我们必须建立系统来将人为错误的机会最小化。在 DevOps 中降低人为错误的频率和影响的关键组成部分就是可复现性的概念。

可复现性意味着我们有能力将某个活动以低成本和保证与第一次操作完全相同的方式再次进行。在 DevOps 中实现可复现性需要自动化和工具支持。可以说，在提高可复现性和加快开发时间方面，DevOps 工具中最重要的工具是容器化。其次是基础设施即代码（IaC）的概念。由于这些技术如此基础，我会花一些时间在这里介绍每个技术的概念。

容器化

在 DevOps 环境中解释容器在其中的角色最常见的方式是考虑到其名字的来源：货物集装箱。在货物集装箱标准化之前，如果你想要将货物跨越海洋运输，你需要将货物打包成各种形式，包括将其放在托盘上、存放在箱子或桶中，或者简单地用布包裹起来。使用这些不同方式打包的货物需要进行装卸

操作，而这种操作效率低下且容易出错，主要是因为没有一种起重机或手推车可以有效地移动所有的货物。

将这种杂乱的方法与部署一个标准化的货物集装箱进行比较，在这个集装箱中，船只和港口操作员可以使用统一的形式，使用标准设备和运输商，以及一个单一的、灵活的所有货物的包装形式。从历史上看，使用标准化的货物集装箱开启了一个范式转变，将全球运输成本大幅降低了。在一个标准化的容器中打包软件，可以在任何系统上以相同的方式运行，这提供了类似的能力和效率的进步。

你最常接触到容器的方式是通过一个名为 Docker 的软件系统。Docker 提供了一种声明式编程语言，让你在一个名为 Dockerfile 的文件中描述系统的设置，例如需要安装哪些程序，文件放在哪里，需要存在哪些依赖关系。然后你将该文件构建为一个容器镜像，该镜像提供了你的 Dockerfile 指定的整个文件系统的表示。这个镜像可以被移动到并在任何其他具有兼容 Docker 容器运行时的机器上运行，保证每次启动时都处于一个隔离的环境中，具有相同的文件和数据。

容器管理最佳实践

一次构建，处处运行地设计容器

在持续集成（CI）中构建容器，并确保它可以在不同的环境（生产环境、开发环境等）中运行。通过使用一个单一镜像，您可以确保代码和设置完全相同，在从开发到生产的过程中不会发生变化。

为了实现容器的处处运行，将环境之间的差异提取到运行时容器环境变量中。

这些环境变量可以包含保密信息和配置，如连接字符串或主机名。或者，您还可以在镜像中实现一个入口脚本，用于从中央密钥存储库（例如 Amazon 或 Google Secret Manager，HashiCorp Vault 等）下载所需的配置和保密信息，然后再调用您的应用程序。

运行时保密信息/配置下载策略的另一个好处是它可以在本地开发中重复使用，避免开发人员手动获取保密信息或要求其他开发人员发送保密文件的需求。

在持续集成中构建镜像

为了追求可复现性，我鼓励您使用自动化工具构建镜像，最好是与持续集成过程相结合。这样可以确保镜像的构建过程本身具有可重复性。

使用托管式仓库

一旦您开始构建容器镜像并进行迁移，您肯定想要对构建的镜像进行组织管理。我建议给每个镜像打上一个基于源代码控制的唯一值，可能还包括一个时间戳（例如，该镜像构建的提交的 git 哈希值），并将镜像托管于一个镜像仓库中。Dockerhub 提供了一个私有仓库产品，而主要云平台也都提供了托管式镜像仓库。

许多托管式仓库还会提供与其镜像仓库相关联的漏洞扫描和其他安全功能。

保持镜像尺寸尽可能小

较小的 Docker 图像在 CI 上传速度更快，在应用服务器下载速度更快，并且启动速度更快。就操作角度而言，上传 50MB 图像和 5GB 镜像的差异可能是启动新应用服务器的时间从五秒到五分钟的差异。这意味着在部署时间、回滚时间等方面增加了五分钟。这可能看起来不多，但特别是在修复紧急情况或管理数百个应用服务器时，这些延迟会累积并对业务产生真正的影响。

Dockerfile 最佳实践

Dockerfile 中的每一行或命令都会生成一个被称为层的东西，实际上就是整个图像硬盘的快照。后续层存储层间的增量。容器镜像是这些层的集合和组合。

因此，通过保持每个单独层的尺寸较小，您可以最小化容器的总图像大小，并且可以通过确保每个命令在移动到下一个命令之前清理任何不必要的数据来最小化层的大小。 另一种将图像大小保持在合理范围内的技术是使用多阶段构建。多阶段构建在此处稍微复杂一些，你可以在 Docker 的文章中详细了解它，文章位于 ctohb.com/docker。

容器编排

现在，你已经有了可复制的图像，并将其管理在托管的注册表中，其大小也相对较小。接下来，你需要在生产环境中运行和管理这些图像。管理工作包括：

- 在机器上下载和运行容器
- 在容器/机器和其他服务之间设置安全网络连接
- 配置服务发现/DNS 简化服务
- 容器的配置和秘密管理
- 根据负载自动调整服务的扩容和缩容
- 容器管理有两种通用方法：托管式和自管理式。

托管式容器管理

除非您的要求很特殊或者规模非常庞大，否则选择一个托管解决方案将为您提供最高的投资回报率，因为它会为您处理管理生产容器的大部分工作。这些解决方案通常被批评的一个公平观点是，它们往往比自我管理的选项要昂贵得多，提供的功能更少、限制更多。换句话说，您将获得更少的开销和更少的复杂性，对于大多数初创公司来说这是一个非常值得权衡的选择。大多数小团队缺乏有效地自主托管的专业知识，因此自主托管要么需要现有团队成员投入大量时间，要么迫使您早期雇佣一位昂贵的 DevOps 专家。为了避免这些问题，每个月多花 1000 美元很可能会带来非常好的投资回报。

一些常见的托管式容器平台包括 Heroku、Google App Engine、Elastic Beanstalk 和 Google Cloud Run。Vercel 是另一个流行的托管后端解决方案，但它不像这里描述的那样运行容器。

自主管理/ Kubernetes

对于容器而言，最受欢迎的自主管理解决方案是 Kubernetes，通常简称为 K8s。Kubernetes 是一个非常强大和灵活的系统，因此也更加复杂。它的学习曲线很陡峭，但如果你需要自主管理容器，其带来的收益和回报是值得的。

如果你考虑采用这种方法，我强烈建议不要在工作中学习 Kubernetes。特别是对于一个团队领导者来说，一方面任务太多，另一方面没有足够的经验难以胜任。相反，我建议购买一本关于 Kubernetes 的书籍，在投入专业项目之前，花一两个星期的时间阅读它并搭建自己的沙盒环境以尽快熟悉。此外，寻找一个对 Kubernetes 有很好理解的顾问或导师，可以加速你学习这个工具。

点击式运维 vs 基础设施即代码（IaC）

点击式运维是指使用用户界面配置云基础设施的过程，而不是使用提供的 API。随着基础设施的增长，你的系统中细节和细微之处的数量很快就会超出你使用点击式运维来复制的能力。点击式运维对于原型或概念验证是可行的，但是当需要构建生产环境和镜像开发环境的时候，使用点击式运维将很快导致相当大的 frustration 和成本，同时还会限制功能。点击式运维的替代方案被称为基础设施即代码（IaC）。

有几种工具和框架可以用来定义基础设施即代码（IaC）。领先的工具是 HashiCorp 的 Terraform。Terraform 使用 HashiCorp 配置语言（HCL），一种声明性的配置语法，允许工程师定义他们想要的资源以及如何配置这些资源。Terraform 代码可以和其他代码一样，使用源代码控制和同行评审的方式进行管理。一旦得到批准，Terraform 可以生成基础设施变更计划，并将这些计划应用于您选择的云服务提供商。我无法强调使用 Terraform 有多么容易、强大和易于维护，以及通过从 ClickOps 迁移到 IaC 可能获得多少投资回报。

持续集成

持续集成是自动化将新代码并入项目的过程。这可能包括对新代码进行静态分析、运行测试、构建代码以及生成任何所需的构建产物（如容器镜像）。大多数创业公司使用托管的持续集成平台，如 GitLab Runner、GitHub Actions、Bitbucket Pipelines、Jenkins 或 CircleCI 来进行持续集成活动。

一些持续集成的最佳实践：

- 确保团队了解持续集成系统，并且能够轻松添加新要求、更新系统，并在出现问题时进行故障排除，以防止构建失败。
- 确保构建过程一致且可确定。不可靠或不稳定的构建会极大地降低生产力并浪费时间。
- 尽量减少构建时间。对大多数团队来说，CI 的一个好目标是在十五分钟内完成。
- 学习您的 CI 工具在保持构建快速方面的功能，包括构建缓存、构建产物和并行运行作业。
- 构建可以变得复杂。在可能的情况下，尽量保持您用于 CI 的代码的可重用性。在构建流水线之间重复使用代码。
- 在构建过程中一致地访问密钥。要么依赖云密钥管理器，要么在必要情况下构建环境密钥，并尽量避免混合使用它们。应该有一种明显且一致的方式来处理配置和密钥。

持续部署

在项目的早期阶段，部署新代码相对简单。此时，代码或架构复杂性并不多。然而，在不久之后，管理依赖、依赖服务、CDN、防火墙、构建产物、构建配置、密钥等需求导致了复杂的部署流程。随着这些需求的积累，很容易忽视自动化，只是依赖专门且高度信任的个人作为发布经理。存在着无数团

队遵循这种模式，我向您保证，其中大多数发布经理都提前在日历上圈出了发布日期，并对这些日子必定带来的压力、长时间工作和沮丧感感到畏惧。

幸运的是，出现了抵御错误和压力过度集中的疗法，即每天发布，甚至每小时发布！这从技术文化的十大支柱之一（见第 138 页）逻辑地得出结论，频繁的发布将迫使您的发布经理和团队自动化部署的难点。通过足够的迭代，发布可以完全自动化，并且通过足够的测试，您对新变更有足够的信心，可以触发每个代码变更集的新发布，即持续部署的方式。

除了通过自动化来消除发布过程的复杂性之外，更频繁地发布意味着每个发布的代码量更小。较小的代码更改更容易让其他开发人员进行审查。简而言之，较小的变更仅仅因为它们更小，意味着更少的缺陷机会。

自动化的发布过程往往意味着在生产环境中改变或从问题中恢复的能力得到改善。这也被称为恢复的平均时间（MTTR）。

总之，自动化发布意味着代码更快地发布（减少交付时间*），意味着您可以更频繁地部署（增加部署频率），并提高 MTTR。这是 DORA 的四个关键指标中的三个之一（请参见第 208 页）！

在我曾经直接或咨询的公司中，我见过至少一打团队投入精力部分或完全过渡到持续部署。这并非总是一条直线的道路，不会在一夜之间发生，通常会遇到有充分理由的反对意见。然而，在每一种情况下，当团队回顾投入的时间，无论是三周、三个月还是两年后，无论从文化还是整体输出和速度指标上看，差异都无疑是颇具变革性的。

feature 分支环境

一个特性分支环境是一个托管环境和基础设施的集合，公司内部可以使用，运行特定分支的代码。特性分支环境非常有用，可以验证代码在类似生产环境的设置中是否正常工作，并且使公司的团队成员能够使用或测试变更，而不必自己构建和运行代码。不要低估人的懒惰；每多一步让他人测试和验证代码的操作，他们就会更少频繁进行测试。检出代码、安装依赖和启动服务器比访问自动生成的特性分支 URL 要麻烦得多，因此特性分支会被更多地使用。

特性分支环境还解决了只有一个暂存环境的团队遇到的争用问题。我倡导给每个分支都提供一个独立的暂存环境。

特性分支环境的一些考虑因素：

- 自动化是关键；手动设置特性分支环境几乎总是不切实际的。
- 在可能的情况下，使用像 Vercel 或 Firebase 这样的系统，将特性分支环境作为一级功能来降低设置和维护成本。
- 仔细考虑如何处理特性分支环境中的数据。你需要回答以下问题： 请帮助我翻译：
  - 每个后端功能分支环境都有一个新的数据库吗？我的建议是是。
  - 功能分支环境是否使用生产数据？我的建议是不要。相反，使用一个种子脚本生成与生产环境类似数量的数据。当需要进行调试时，将生产数据复制、清洗和还原给开发人员是值得的。
  - 功能分支环境如何处理数据库架构更改/迁移？

- 在面向服务的架构中，功能分支环境中的服务发现是如何工作的？通常情况下，您可以拥有一组通用的服务，并仅部署测试中的服务的功能分支版本。我建议您为每个服务都提供一个始终运行的集成环境，并让所有功能分支引用其他服务的集成环境。每个集成环境的更新应该是一个生产更新，但您也应该允许开发人员将功能分支代码临时部署到集成环境中进行跨服务集成测试。

看完这个关注点列表，您可能会对设置和维护功能分支的负担感到难以忍受。确实，一个恰当的功能分支设置并不便宜，但它提供的价值在于改善您的测试能力，并减少验证不同类型软件更改所需的后勤开销。

管理 DNS

了解域名系统（DNS）的工作原理，并知道如何管理 DNS 及其对您的公司的安全影响，是初创公司首席技术官常常要承担的关键任务。如果您还不熟悉 DNS 的基本工作原理和不同的记录类型，我建议您花几分钟浏览维基百科，对这个主题有一个基本了解。

您还应该了解 DNS 在您的组织中是如何用于电子邮件的。特别是，要熟悉发件人策略框架（SPF）记录和 DKIM/DMARC。

您还应该使用基础设施即代码（IaC）设置您的 DNS 记录。我见过很多公司，DNS 完全由一个高管负责管理，只允许通过双重身份验证来更新区域记录，而当这个人度假时，没有任何备选机制来管理网站。

一个更好的解决方案是使用 Terraform 设置 DNS（它与所有主要的 DNS 提供商都有集成），然后使用源代码控制来管理 DNS 记录，使开发人员能够以一种负责任的方式添加新的记录，不依赖于任何一个个人。

解耦代码和功能的发布（功能切换）。 在高层面上，特性开关是一种可以在不改变实际代码的情况下改变系统行为的开关。我强烈推崇使用特性开关，特别是因为它们允许您的团队在代码和功能交付上有独立的流程和时间表。Pete Hodgson 在 ctohb.com/hodgson 上对特性开关有一个很好且深入的解释。

四个关键指标鼓励尽可能频繁地发布代码。这样做对于您的工程流程的健康带来许多积极的益处。然而，一个自然的担忧是，当代码完成后，您的业务可能还没准备好让某个特定功能立即上线。有很多原因会导致您的开发和发布计划在这方面不同步，比如需要与营销激活时间协调、创作客户支持文档、等待监管批准、暂停内部沟通等。特性开关使得你的工程团队能够专注于尽快、可靠地进行交付，而将功能的启用协调问题委托给可能由其他团队负责的带外过程。

系统监控：APMs 和 RUMs

应用性能监控（APM）和真实用户监控（RUM）是帮助团队了解应用在生产环境中的性能，并识别或防止用户面临的故障的两种类型的工具。

APM 工具通常位于或与您的应用程序一起部署在生产环境中，并提供资源使用情况、请求延迟和吞吐量的分析和见解，以后端的角度来看。RUM 是一种外部工具，它扮演用户并从前端的角度或实际用户

的角度提供延迟的分析。

如果您必须选择一个（这些工具通常非常昂贵，所以您可能被迫选择一个），请选择那个更可能对您的应用程序造成问题的盲点。如果您有大量的用户，并且每个小错误或边缘情况都被实时投诉淹没，那么 APM 监控后端负载可能比向您的用户产生冗余警报的 RUM 更有价值。然而，对于大多数初创企业来说，无论是种子或成长阶段，您的应用程序使用可能是不一致的，特别是涵盖您的边缘情况，这种情况下 RUM 可能比大部分空闲后端上的 APM 更有价值。

在这个领域中一些常见的工具有 New Relic、Datadog 和 Akamai mPulse。

测试

假设，与测试软件代码不同，您的工作是成为一座市政桥梁的检查员。桥梁已经建好，现在轮到您进行检查并决定是否允许其对公众开放。您会检查桥上的每一个螺栓和铆钉吗？这样做可能会让您对桥梁的安全性有很高的信心，但同时也会花费很长时间，阻碍市长计划下周开放桥梁。

另一个合理的策略是精确地决定桥梁哪些部分是满足指定安全系数所必需的，并对其进行测试/检查。也许只需要检查每隔一个铆钉，以及所有的电缆和结构混凝土。 同样地，在软件工程测试中，目标不是为了覆盖率而覆盖率，而是为了提供对软件是否按照预期工作的信心。

有效的软件测试并不总是要求百分之百的代码覆盖率。一个良好的软件测试套件的标准是能够让你的团队确信，当构建成功并且所有测试都通过时，软件已经可以发布给最终用户。这可能意味着百分之百的代码覆盖率，也可能意味着百分之三十的代码覆盖率。准确的数字由你来决定和监控，如果你发现测试无法提供之前的信心，那么这个数字可能会随时间变化（反之亦然，如果你发现你过度投入，即花费了大量资源在测试套件上，但错误仍然频繁出现）。

测试/质量保证团队

根据团队的规模，你可能没有专门的测试团队，只有一个测试团队负责一个或多个测试类型，或者许多测试团队负责各种软件测试。无论是谁在进行测试，都要认识到软件测试是一个复杂的过程，细微的差别很重要。要有效地测试软件，测试人员（无论是他们自己编写的代码还是接手的代码）必须深入理解代码/软件应该做什么。你的角色是让团队能够共情。为此，确保团队共享目标/关键绩效指标，确保你的流程中开发人员和测试人员之间有牢固而持续的沟通，同时监控团队之间拥有良好且富有成效的关系。

测试质量

在探究不同软件测试范式的具体内容之前，值得思考的是软件测试的目的，以及什么样的测试是好的，相反地，什么样的测试是糟糕的。

定义糟糕的测试很简单。糟糕的测试对团队来说成本比收益高。一些常见的糟糕测试的特点包括：

- 维护和修复测试针对合法代码变更所需的时间比通过发现的错误节省的痛苦更多。
- 测试具有很高的误报率或不一致性，这会降低持续集成的速度，让开发人员感到沮丧，同时因重新运行伪失败而带来上下文切换的成本。

- 测试考虑不周，实际上只验证了代码做错的事情。
- 测试套件检测代码是否正确执行且误报率低，但其复杂难懂，只有编写测试的人能够添加新测试，其他工程师看到后会头痛不已。

该测试无法增加对待评估代码可交付给最终用户的信心。

有了这样的背景，就可以相对容易地看出良好的测试应具备的特性。良好的测试应该具备以下特点：

- 在本地和共享 CI 环境下运行简便快捷
- 能够稳定可靠地运行，并产生相同的结果
- 容易扩充
- 易于重构或更新以适应潜在的逻辑变化
- 与底层代码模式适当地耦合，以在适当的层次上进行测试以捕捉重要的错误
- 容易让任何工程师理解和操作

评估测试方法的最佳方式之一也是最明显的方式：通过简单的情感分析询问团队对测试的感受。结果往往非常二元，要么测试是团队依赖的安全源，并自然地增强它们，因为它们显然是一个净增值；要么团队被动地或甚至主动地讨厌他们的测试，因为它们对生产力产生负面影响，而没有足够明显的价值。

应该测试什么

您的团队应该在增加测试的同时，以尽量减少在面对有机系统增长时对测试的长期维护负担的方式来增加对系统正确运行的信心。一个常见的减轻这种痛苦的方法是测试公共接口。公共接口应该经过深思熟虑，并在一段时间内保持相对稳定。

它们也是软件使用者实际体验到的顺畅路径。

因此，对公共接口的测试在时间上应该变化很少，并且能够提供对代码中重要部分是否正确工作的信心。

您的软件的公共接口可能会因项目而异。对于许多项目，它可能是一个实际的基于 HTTP 的 API；对于某些项目，它可能是一个库或内部服务中的一组函数/类。对于其他项目，它可能是一个用户界面。

测试类型比较

软件测试可以分为以下几个类别/范式:

- 单元测试
- 集成测试
- 端到端测试
- 手动测试
- 半自动化测试

区分这些测试类型的属性如下：

代码规划： 在实际编写代码时，需要多少事先考虑才能确保它在这种测试范式下可测试。

测试范围： 每个测试可以同时评估多少内容。

变更粒度检测： 测试可能检测到并导致失败的代码更改的大小或类型。

运行成本： 运行测试的速度或成本，在时间或金钱方面。

增加工作量： 添加额外覆盖所需的工作量。 安装工作量：设置一个有效的测试套件需要多少工作量。

下面的图表总结了这五种范例与这些度量标准的映射关系。请注意，没有一种完美的测试范例；每种范例都存在权衡，我鼓励你仔细考虑哪种权衡对你的公司和代码库来说是合理的。通常正确的答案是将不同的测试范例相结合，在应用程序中以最大程度增加价值的地方使用每种类型的测试。

TODO：从书中插入表格

单元测试

单元测试通常是提到软件测试时首先想到的。它广泛地被教学和收录在教材中，并且在现实世界中通常会付出最大的努力。鉴于这一切，你可能认为单元测试是最好的测试类型，尽管我认为情况并非总是如此。让我们首先清楚地定义单元测试，以便将其与其他测试范例区分开来。

单元测试完全在内存中的一台机器上运行，在评估的代码下共享内存空间，不需要任何网络连接或对外部服务的依赖。大多数单元测试运行非常快速，一次只测试很少的代码量，并且相对容易开始。单元测试通常也与你的代码契约紧密耦合，通常需要在代码-受测下执行而普通可用的外部依赖项。

单元测试所做的主要权衡和其主要缺点是它们与受测代码紧密耦合。很容易使单元测试与实际的函数调用和内部数据对象紧密交织在一起。这种深层依赖意味着对代码的任何重构，即使是简单和良性的更改，也需要相当大量的单元测试更新。创建模拟和测试数据夹具通常也需要相当大量的代码来创建和维护以使单元测试运行，为单元测试框架增加了意外的成本。

集成测试

集成测试放松了单元测试中的内存和零依赖约束。结果，集成测试运行较慢，并与代码-受测在更高级别上进行集成。当它们在与受测代码不同的进程中运行时，它们通常在执行外部暴露的合同，例如 API。这意味着不改变 API 行为的内部重构往往不会被集成测试注意到，这使得这些测试整体上不那么脆弱，但也不太可能检测到较小的副作用。 此外，集成测试要求创建较少或没有模拟对象，因此实现的代码可以更少。

端到端测试

端到端（e2e）测试以与最终用户相同的方式运行代码。对于后端代码，端到端测试和集成测试可能以相同的方式工作，通过外部协议运行代码。对于面向用户的前端代码，端到端测试通常涉及机械化客户端界面，通常是一个网页浏览器或移动电话。我不建议任何技术领导人编写自己的客户端机械化代

码，因为这是一个特别棘手的问题，下载工具如 Selenium、Cypress 和 Puppeteer 可以帮助解决。对于移动端，也有像 HeadSpin 和 Detox 这样的工具。

端到端测试在现实世界中的主要权衡是可靠性。至少在撰写时，可靠的网页端到端测试仍然有些难以捉摸；浏览器渲染的特性意味着很容易发生竞态条件。构建一个可靠的网页端到端测试套件需要非常小心、注重细节和维护。

然而，回报是相当可观的，使用 e2e 测试套件可以实现非常高的测试覆盖率和用户流程功能的高度可信度。有一些公司，包括 testim.io 和 rainforestqa.com，正在探索使用人工智能和机器学习来解决这个问题。这些解决方案使用模糊视觉测试，而不是仅仅依赖 CSS 选择器的存在或缺失，以提高测试的可靠性。希望到您阅读此文的时候，技术已经更加先进一些，端到端测试的价值主张将比撰写本文时更强大。

视觉回归测试

视觉回归测试是一种相对较新的范式，旨在通过对渲染可视化效果进行差异比较来检测用户界面应用程序中的缺陷。有一些框架可以在不同的粒度上完成这个任务，从捕获整个页面的屏幕截图到渲染单个组件，并生成差异以检测缺陷。显而易见的缺点是，对任何被测试的可视组件进行有意的更改都需要修改测试。

幸运的是，这些测试框架通常可以简单而轻松地复现一组正确的可视效果进行比较，这也带来了另一个潜在问题：通过简单的工具来覆盖测试目标，很容易产生错误的虚假负面结果，意外接受了实际上是错误的视觉差异。

手动测试

手动测试，顾名思义，由人类而非机器代码来运行。对于人类测试代码，我们可以进一步将这个类别分为专业化和非专业化的测试人员。

专业手动测试

专业手动测试是一个内部手动测试团队。与传统的内部团队相比，例如利益一致和长期的工作关系等优点，内部团队的价值在于该团队可以在您的产品上建立专业知识，并深入了解您的用户。这使得团队可以发现一个未经培训或不熟悉的测试人员可能会忽略的缺陷。一个高质量的测试团队不仅可以提供捕捉软件缺陷的资源，而且还可以对产品层面提供有价值的反馈，识别设计上的不一致之处，并对某个功能的工作方式提出引人深思的问题。

一个出色的内部手动测试团队可以大幅提高整体软件和产品质量。

专业手动测试团队应该为产品功能创建详细的测试计划，并以易于检索/重复的方式存储这些计划，最好使用像 TestRail 这样的工具，允许创建完整的手动测试计划测试套件，可以由团队手动按需重新运行。这样的工具的另一个好处是与其他开发人员和产品工具的集成，例如链接 TestRail 运行到 Jira 优秀，显示针对特定功能发布运行了多少个手动和回归测试。这不仅有助于发布检查清单，而且还有助于回顾任何已发布的缺陷，使您可以回顾在发布任何给定功能之前运行了哪些手动测试，并添加额外的手动测试以捕捉通过的任何缺陷。

非专业手动测试

非专业化手动测试通常被称为众包测试。有几个专有平台用于获取测试人员，例如 Rainforest QA、Pay4Bugs、99Tests 和 Testlio。这些平台的定价模式通常基于提交的经过验证的缺陷数量而有所不同。根据您的产品性质和所寻求优化的缺陷类型，众包测试可以是一种非常具有成本效益和低投入的方式来提高产品质量。

半自动化测试

半自动化测试是一种相对较新的软件测试类别。这些测试由非技术性员工（例如您的专业化手动测试团队）创建，然后在完全自动化或监督环境中运行。

这些测试的主要缺点是可靠性。由于它们是由非技术性员工创建的，它们可能不像完全自动化测试那样精确，从而更容易产生误报和漏报。尽管如此，这个领域正快速发展，每年都有新的公司和工具推出，例如 Rainforest QA 和 Testim，它们都有改进可靠性和降低总体成本的策略。

源代码控制

大多数公司在源代码管理方面使用的行业标准是 Git。除非你的组织有一个非常有说服力的理由使用其他工具，否则大多数现有和未来的团队成员都已经习惯了 Git 的基础知识。如果不使用 Git，可能会给他们施加不必要的学习曲线，强迫他们学习你选择的替代工具。

目前有三个主要的云 Git 托管平台：GitLab、GitHub 和 Bitbucket。这三个平台在市场上占据了绝大部分份额，因此在决定偏离这些标准之前需要仔细考虑。

Git 的学习曲线有一种有趣的形状。大多数人都会达到一个平台，他们对 Git 有基本的了解，可以应对大多数正常测试场景。但有时候会出现问题，比如开发人员丢失了提交记录或在合并时出现问题。当这种情况发生时，他们没有更进一步学习 Git 的曲线，会导致沮丧和减速。作为团队负责人，我鼓励你投入精力去攀爬曲线的后半部分。在命令行上专门使用 Git 来了解实际发生的事情。学习关于 reflog、交互式 rebase、bisect 和各种内置合并策略的知识。掌握了这些知识，你可以绕过整个一类影响工作效率的问题，并培训你的团队成为 Git 专家。

同行评审

通常，专家建议对所有代码更改实施一个强大的同行评审流程。在我写这篇文章时（2023 年初），对这个建议正在出现一股挑战的风潮，或者至少对其进行细化，我将在下一部分中讨论这个问题。大多数同行评审都是通过拉取请求、代码审查或合并请求来完成的，根据你的代码托管解决方案可能有所不同。以下是一些建议，可以使代码评审变得高效和有效。 有几种工具和框架可以用来定义基础设施即代码（IaC）。其中领先的是 HashiCorp 的 Terraform。Terraform 使用 HashiCorp 配置语言（HCL），一种声明性配置语法，让工程师可以定义他们想要的资源以及如何配置这些资源。Terraform 代码应该像其他代码一样进行管理，使用源代码控制和同行评审的方法。一旦获得批准，Terraform 可以生成基础设施更改计划，并使用您选择的云服务提供商应用这些计划。我无法强调地足够，Terraform 易于使用、强大且易于维护，并且从手动操作转向基础设施即代码可以使您获得更多的投资回报。

持续集成

持续集成是自动化将新代码加入项目的过程。这可能包括对新代码进行静态分析、运行测试、构建代码和生成所需的构建产物（如容器镜像）。大多数初创公司使用托管的持续集成平台，如 GitLab runners、GitHub actions、Bitbucket pipelines、Jenkins 或 CircleCI 来进行持续集成活动。

一些持续集成的最佳实践包括：

- 确保团队了解持续集成系统，并能够轻松添加、更新新需求，并在出现问题时进行故障排除，以防止构建失败。
- 确保构建是一致和确定性的。不可靠或容易失败的构建会极大地降低生产力和浪费时间。
- 尽量保持构建时间的缩短。对于大多数团队来说，持续集成的目标是不超过 15 分钟。
- 学习你的持续集成工具的能力，以帮助加快构建速度，包括构建缓存、构建产物和并行运行作业。
- 构建可以变得很复杂。尽量使你的 CI 代码“干燥”。在可能的情况下，在构建流水线之间重用代码。
- 在构建中访问秘密时要保持一致性。要么依赖于云秘密管理器，要么在必要时使用构建环境秘密，并尽量不混合使用它们。应该有一种明显和一致的处理配置和秘密的方法。

持续部署

在项目的早期阶段，部署新代码相对简单。那时，代码或架构复杂性不大。然而，随着对依赖项、相关服务、CDN、防火墙、构建产物、构建配置、秘密等需求的累积，部署流程变得复杂起来。随着这些要求的增加，很容易忽视自动化，仅仅依赖于专门负责可信任的个人作为发布经理。有无数团队遵循这种模式，我保证其中大多数发布经理事先在日历上圈出了发布日期，并对那些必然会带来压力、长时间工作和沮丧的日子感到痛苦。

幸运的是，有一种可以治愈这种易出错的集中压力的方法，那就是每天发布，甚至每小时发布！这是从技术文化的十大支柱之一（见第 138 页）的逻辑上推导出来的，频率降低了困难，更频繁的发布将迫使你的发布经理和团队自动化部署的难点。通过足够的迭代，发布可以完全自动化，并且通过充分的测试来对新变更充满信心，你可以达到对每个代码变更集触发新发布的程度，这被称为持续部署。除了通过自动化消除发布流程的复杂性外，更频繁地发布意味着每次发布的代码量更小。较小的代码变更更容易供其他开发人员审查。较小的变更仅仅因为它们较小，为产生缺陷提供了较少的机会。

自动化的发布流程还意味着提高了回滚更改或从生产中恢复的能力。这也可以衡量为故障恢复的平均时间（MTTR）。

总结起来，自动化发布意味着代码更快地发布（减少提前时间），意味着你可以更频繁地部署（增加部署频率），并且提高了 MTTR。这是四项关键的 DORA 指标中的三项（参见第 208 页），只需一个举措！

在我曾经直接或间接与公司合作过的公司中，我见过至少十几个团队努力投资于部分或完全转向持续部署。这并不总是一条直线的旅程，不会一蹴而就，经常会有很有理由的反对意见。然而，在任何情

况下，当团队回顾投资的时间，不管是三周、三个月还是两年后，差异对于团队的文化和整体产出速度以及每个指标来说，简直是从根本上的变革。

特性分支环境

特性分支环境是一个托管环境和基础设施，仅在内部公司提供，运行特定分支的代码。特性分支环境非常有用，可以验证代码在类似生产环境的设置中是否正常工作，并使公司的团队成员能够使用或测试变更，而无需自己构建和运行代码。不要低估人类的懒惰；每多一步测试和验证代码的工作，意味着他们进行测试的频率就会减少。检出代码、安装依赖和启动服务器比访问自动生成的特性分支 URL 更耗费精力，因此特性分支会被更频繁地使用。

特性分支环境也解决了只有一个暂存环境的团队遇到的竞争问题。我建议给每个分支都提供自己的暂存环境。

特性分支环境的一些考虑因素包括：

- 自动化是关键；手动设置特性分支环境几乎总是不现实的。
- 在可能的情况下，使用像 Vercel 或 Firebase 这样的系统，这些系统将特性分支环境作为一流功能，以最小化设置和维护成本。
- 仔细考虑如何处理特性分支环境中的数据。您需要回答以下问题： 每个后端分支环境都会有一个新的数据库吗？我的建议是是。 分支环境是否使用生产数据？我的建议是否定的。相反，使用一个种子脚本来生成与生产环境类似数量的数据。当需要进行调试时，值得创建一个将生产数据复制、清理和恢复给开发人员使用的过程。 如何处理数据库模式变更/迁移在分支环境中？ 在面向服务的架构的分支环境中，如何进行服务发现？通常，你可以通过只部署正在测试的服务的分支版本来共用一组常见的服务。我建议为每个服务创建一个始终运行的集成环境，并让所有分支环境引用其他服务的集成环境。每个集成环境的更新都应该是一个生产更新，但你也应该允许开发人员将分支代码暂时部署到集成环境中进行跨服务的集成测试。

看完这些问题，你可能会对建立和维护分支环境的负担感到困扰。确实，一个合适的分支环境设置并不廉价，但它提供的价值在于改善您的测试能力，减少验证不同类型的软件变更所需的后勤工作。

管理 DNS 了解域名系统（DNS）的工作原理，以及了解如何管理 DNS 及其对公司的安全影响，这是一项至关重要的工作，通常由初创企业的 CTO 负责。如果您对 DNS 的工作原理和不同的记录类型还不熟悉，我建议您花几分钟在维基百科上浏览一下，以对该主题有所了解。

您还应该了解 DNS 在贵公司的电子邮件中的使用方式。特别是要熟悉发件人策略框架（SPF）记录和 DKIM/DMARC。

您还应该使用基础设施即代码（IaC）来设置您的 DNS 记录。我见过许多公司，其中 DNS 仅由一个高级管理人员管理，只允许使用双因素身份验证更新区域记录，当这个人度假时，就没有后备机制来管理网站。

更好的解决方案是使用与所有主要 DNS 提供商集成的 Terraform 来设置 DNS，然后使用源代码控制管理 DNS 记录，使开发人员能够以负责任的方式添加新记录，而不依赖于任何一个人。

将代码交付与功能交付分离（功能切换） 在较高的层面上，功能切换是一种允许您在不更改实际代码的情况下改变系统行为的开关。我强烈推荐使用功能切换，特别是因为它们允许您的团队在发布代码和发布功能方面具有单独的流程和时间表。Pete Hodgson 在 ctohb.com/hodgson 上对功能切换进行了精彩而深入的解释。

四个关键指标鼓励尽可能频繁地发布代码。这样做对您的工程流程健康有许多积极的益处。然而，一个自然的担忧是，您的业务可能还没有准备好在代码完成后立即上线某个特定功能。您的开发和发布时间表之间可能有许多原因导致不同步，比如与营销动作的时间协调、创建客户支持文档、等待监管批准、暂停内部沟通等。功能切换使您的工程团队能够专注于尽快、可靠地发布，并将功能启用的协调问题委托给其他团队拥有的异步流程。

系统监控：APM 和 RUM

应用性能监控(APM)和真实用户监控(RUM)是两种帮助团队了解应用在生产环境中的性能并识别或预防面向用户的故障的工具。

APM 工具通常位于或与您的应用程序一起在生产环境中，并从后端的角度提供资源使用情况、请求延迟和吞吐量的分析和洞察。RUM 是一个外部工具，它假装成一个用户，并提供来自前端或真实用户的延迟分析。 如果你必须选择一个（而这些工具通常非常昂贵，所以你可能被迫选择其中一个），选择那个更有可能覆盖你的应用中可能出现的盲区的工具。如果你有大量的用户，并且你因为每个小错误或边界情况而被实时投诉淹没，那么一个监控后端负载的应用性能管理（APM）可能比生成多余警报的实时用户反馈（RUM）更有价值。然而，对于处于种子期或增长期的大多数初创公司来说，你可能会遇到应用使用不一致，尤其是涵盖边界情况，这种情况下 RUM 可能比主要闲置的后端上的 APM 更有价值。

在这个领域中一些常见的工具有 New Relic、Datadog 和 Akamai mPulse。

测试

想象一下，如果你的工作不是测试软件代码，而是检查市政桥梁的检查员。桥梁已经建好，现在你的工作是检查它并决定是否允许公众通行。你是否要检查桥上的每一个螺栓和铆钉？这样做可能会给你高度的安全保障，但也会花费很长时间，并干扰市长下周开放桥梁的计划。

另一个合理的策略是确定桥的哪些要素是满足指定安全系数所必需的，并对其进行测试/检查。也许只需要检查每隔一个铆钉，再加上所有的电缆和结构混凝土。 同样，在软件工程测试中，目标不是为了达到覆盖率而覆盖率，而是为了提供对软件是否按照预期进行操作的信心。

有效的软件测试并不总是要达到 100%的代码覆盖率。一个好的软件测试套件的标准是让您的团队确信，当构建成功且所有测试通过时，软件可以发布给最终用户使用。这可能意味着 100%的代码覆盖率，也可能是 30%的代码覆盖率。确切的数字由您来确定和监测，如果发现测试无法提供相同的信心，这个努力的程度随时间可能会发生变化（相反地，如果发现你过度投入，即使投入大量资源进行测试，bug 仍然经常出现）。

测试/质量保证团队

根据您的团队规模，您可能没有专门的测试团队，一个测试团队负责一种或多种类型的测试，或者多个测试团队负责各种类型的软件测试。无论谁来进行测试，都要认识到软件测试是一个复杂的过程，细微之处很重要。为了有效地测试软件，无论是测试人员编写代码还是收到代码，他们都必须深入理解代码/软件应该做什么。您的角色是为您的团队建立起相互理解的环境。为此，请确保团队共享目标/关键绩效指标，您的流程在开发人员和测试人员之间具有严密而持续的沟通，并确保团队拥有健康而高效的关系。

测试质量 在深入研究不同软件测试范式的具体细节之前，值得思考一下软件测试的目的，以及什么样的测试被视为好的，相反地，什么样的测试被视为不好的。

定义不好的测试很简单。不好的测试对团队而言成本高于收益。一些常见的不好测试特征包括：

- 修复合法代码变动所需的测试维护时间比通过发现错误节省的时间更长。
- 测试具有很高的误报率或不一致性，这会导致 CI 变慢，给开发人员带来沮丧和上下文切换成本，不得不重新运行虚假失败的测试。
- 测试设计思路不良，直接验证了错误的代码行为。 代码的测试套件确保代码能够正确运行且误报率低，但是这套测试过于复杂和难以理解，只有编写测试的人才能添加新的测试用例，其他工程师看了会头痛。

该测试无法使人们对待评估的代码准备好将其交付给最终用户的情况充满信心。

有了这个背景，就相对容易看出好的测试应该具备哪些属性。好的测试应具备以下特点：

- 在本地和共享的 CI 环境中易于快速运行
- 能够可靠地运行并始终产生相同的结果
- 便于扩充 易于重构或更新以适应底层逻辑变化

与底层代码模式适当耦合，以适当的测试高度（可以这么说）来捕捉重要的故障

易于任何工程师理解和处理

评估测试方法的最佳方式之一也是最明显的方式：通过简单的情感分析询问团队对测试的感受。结果往往是非常二元的，要么测试是团队依赖的安全源，并且自然而然地增加了价值；要么团队对测试持消极态度，甚至积极讨厌测试，因为它们会消耗生产力，但没有足够明显的价值。

测试内容

您的团队应该以一种增加系统正常运行信心的方式来添加测试，同时尽量减少在有机系统增长的情况下对长期测试维护的额外负担。一个常见的模式是测试公共接口，这样可以减少测试的痛点。公共接口应经过深思熟虑，并且在时间上相对稳定。

公共接口也是软件使用者实际体验的快乐路径。

随着时间的推移，公共接口的测试应该变化较少，并且能够提供对代码中重要部分是否正常工作的信心。

您的软件的公共接口可能因项目而异。对于许多项目，它可能是一个真实的基于 HTTP 的 API；对于一些项目，它可能是一组库或内部服务中的函数/类。对于其他项目，它可能是一个用户界面。

测试类型比较

软件测试可以分为以下几个类别/范式：

- 单元测试
- 集成测试
- 端到端测试
- 手动测试
- 半自动化测试

区分这些测试类型的特征如下：

- **代码规划：** 在实际编写代码时，需要多少先见之明来确保其在这种测试范式下可测试。
- **测试范围：** 每个测试可以同时评估多少内容。
- **变更粒度检测：** 测试很可能检测到并引起失败的代码更改大小或类型。
- **运行成本：** 运行测试的速度或成本有多快或昂贵，可以用时间或金钱来衡量。
- **添加工作量：** 添加额外覆盖率所需的工作量有多大。 **设置工作量:** 设置一个有效的测试套件所需的工作量有多大。

下面的表格总结了这五种范式如何与这些度量指标相对应。请注意，并没有一个完美的测试范式；每种范式都有其权衡，我鼓励你仔细考虑哪些权衡对你的公司和代码库而言是有意义的。正确的答案通常是各种测试范式的混合，在应用程序中使用每种类型的测试可以为其增添最大的价值。

待办事项：从书中插入表格

单元测试

单元测试通常是当谈论软件测试时首先想到的事情。它广泛地在学校教授并包含在教科书中，并且通常在实际世界中付出了最大的努力。考虑到这一切，你可能会认为单元测试是最好的测试类型，尽管我会说这并不总是正确的。让我们首先明确定义单元测试，以便将其与其他测试范例区分开来。

单元测试完全在内存中的计算机上运行，在与待评估代码共享的内存空间中，不需要任何网络连接或对外部服务的依赖。大多数单元测试运行非常快速，每次测试的代码量很小，并且相对容易上手。单元测试通常也与代码合同紧密耦合，并且通常需要以模拟的形式编写新代码，以使被测试代码在没有通常可用的外部依赖项的情况下执行。

单元测试的关键权衡和主要缺点是它们与待测试代码紧密耦合。很容易使单元测试与实际函数调用和内部数据对象紧密交织在一起。这种深层依赖意味着任何对代码的重构，即使是简单和良性的更改，

也需要相当多的单元测试更新。创建模拟对象和测试数据的夹具还需要相当多的代码来创建和维护，以使单元测试能够运行，为单元测试框架增加了意外的成本。

集成测试

集成测试放松了单元测试的内存和零依赖约束。因此，集成测试倾向于运行较慢并与待测试代码在更高的级别上集成。当它们在与代码不同的进程中运行时，它们通常正在执行一个外部公开的接口（例如 API）。这意味着不会改变 API 行为的内部重构通常不会被集成测试察觉到，这使得这些测试整体上更加稳定，但也更不可能检测到较小的副作用。 此外，集成测试需要创建较少或没有模拟对象，因此可以实现更少的代码。

端到端测试

端到端（e2e）测试以与最终用户相同的方式对代码进行测试。对于后端代码，端到端测试和集成测试可能以相同的方式工作，通过外部合同运行代码。对于面向用户的前端代码，端到端测试通常涉及使用客户端界面，通常是一个网络浏览器或手机。我不鼓励任何技术负责人自己编写客户端机械化代码，这是一个特别复杂的问题，可以使用可下载的工具（如 Selenium，Cypress 和 Puppeteer）来解决。对于移动端，也有像 HeadSpin 和 Detox 这样的工具。

端到端测试在现实世界中的关键折衷是可靠性。至少在撰写本文时，可靠的网络端到端测试仍然有些难以捉摸；浏览器呈现方式的特性意味着可能很容易出现偶发的竞争条件。构建一个可靠的网络端到端测试套件需要极大的注意、细节和维护。

然而，回报是相当可观的，通过 e2e 测试套件可以实现非常高的测试覆盖率和对用户界面流程功能的高度信心。有几家公司正在探索使用人工智能和机器学习来解决这个问题，包括 testim.io 和 rainforestqa.com。这些解决方案使用模糊的视觉测试而不是依赖于 CSS 选择器的存在或缺失，以提高测试的可靠性。希望在您阅读本文时，这方面的技术已经有了进一步的发展，端到端测试的价值主张将比本文撰写时更加强大。

视觉回归测试

视觉回归测试是一种相对较新的范式，旨在通过对呈现的视觉元素进行差异分析来检测用户界面应用程序中的缺陷。有各种框架可以在不同的级别上进行这种测试，从捕获整个页面的屏幕截图到渲染单个组件，并生成差异以检测缺陷。显而易见的缺点是，对任何被测试的可视化组件进行有意的更改都需要修改测试。

幸运的是，这些测试框架通常使得重新生成正确视觉元素集合进行比较变得简单和无痛，但这也带来了另一个陷阱：由于可以轻松地覆盖测试目标，很容易产生错误的负面结果，无意中接受了一个实际上是错误的视觉差异。

手动测试

手动测试，顾名思义，由人类而非机器代码执行。对于人类来测试代码，我们可以将此类别进一步细分为专业和非专业测试人员。

专业手动测试

专业手动测试是一支内部手动测试团队。除了内部团队的传统优势，如契合的激励机制和长期的工作关系，内部团队的价值在于团队可以对您的产品建立专业知识并深入了解您的用户。这使得团队能够发现非训练有素或不熟悉的测试人员可能会忽视的缺陷。高质量的测试团队不仅可以作为捕捉软件缺陷的资源，还可以为产品级别提供有价值的反馈，识别设计上的不一致之处，并对某些事物的工作方式提出引人注目的问题。

出色的内部手动测试团队可以大幅提高软件和产品的整体质量。

专业手动测试团队应该为产品功能创建详细的测试计划，并以易于检索和重复的方式存储这些计划，最好使用像 TestRail 这样的工具，允许创建包含手动测试计划的完整测试套件，团队可以随时手动运行这些测试。像这样的工具的另一个好处是与其他开发人员和产品工具的集成，例如将 TestRail 运行与 Jira 任务关联，以显示为给定功能的发布运行了多少个手动和回归测试。这不仅作为发布检查清单的有价值项目，还有助于对任何发布的缺陷进行回顾，使您能够重新审视在发布任何给定功能之前运行了哪些手动测试，并添加额外的手动测试以捕捉任何可能存在的缺陷。

非专业手动测试

非特定手动测试通常称为众包测试。有几个专有平台用于招募测试人员，例如 Rainforest QA、Pay4Bugs、99Tests 和 Testlio。这些平台的定价模型通常基于提交的经验证的错误数量而有所不同。根据产品的性质和您希望优化的缺陷类型，众包测试可能是一种非常具有成本效益和低投入的提高产品质量的方式。

半自动化测试是一种相对较新的软件测试类别。这些测试由非技术人员创建，可能是您的专门手动测试团队，然后在完全自动化或受监督的环境中运行。

这些测试的主要缺陷是可靠性。因为它们由非技术人员创建，可能不像完全自动化测试那样精确，从而更容易产生错误的结果。尽管如此，这个领域正在快速发展，每年都有新的公司和工具推出，例如 Rainforest QA 和 Testim，以改善可靠性并降低总体成本。

源控制 绝大多数公司在源代码管理方面所使用的行业标准是 Git。除非你的组织有非常充分的理由使用其他工具，否则大多数现有和未来的团队成员都已经了解 Git 的基础知识。如果不使用 Git，你可能会给他们增加一个不必要的学习曲线，迫使他们学习你选择的替代工具。

目前主要的云端 Git 托管平台有 GitLab、GitHub 和 Bitbucket。这三个平台占据了市场的大部分份额，因此在决定与这些标准有所偏离之前，应该仔细考虑。

Git 有一个有趣的学习曲线。大多数人在学习初期能够达到一定的水平，这对于大部分正常测试场景来说足够了。然而，有时会出现问题，开发人员可能会丢失提交记录或在合并时出错。当这种情况发生时，如果他们没有掌握 Git 学习曲线的剩余部分，就会感到挫败和减速。作为团队负责人，我鼓励你投入精力去攀登这条曲线的后半段。在命令行中专门使用 Git，以熟悉实际发生的情况。了解 reflog、交互式变基、二分查找和各种内置合并策略等内容。掌握这些知识后，你可以避免一整类损害生产力的问题，并培养你的团队成为 Git 专家。

同行评审

一般来说，专家们建议对所有代码变更实施强大的同行评审流程。（截至我在 2023 年初撰写本文时，有一股趋势正在质疑这一建议，或者至少对其进行细化，我将在下一节中讨论。）大多数同行评审通过在代码托管平台上提出拉取请求、代码审查或合并请求来完成。以下是一些建议，帮助保持代码评审的高效性和生产力：

专业手动测试

专业手动测试是一个内部手动测试团队。除了传统团队内部的好处，比如利益一致以及长期的工作关系，内部团队的价值在于团队可以在您的产品上建立专业知识并深入了解您的用户。这使得团队能够发现一个未经训练或不熟悉的测试人员可能会忽略的缺陷。一个高质量的测试团队不仅可以作为捕捉软件缺陷的资源，还可以在产品层面提供有价值的反馈，发现设计上的不一致，并就某件事情的工作方式提出引人思考的问题。

一个出色的内部手动测试团队可以大大提高整体软件和产品质量。

专业手动测试团队应该为产品功能创建详细的测试计划，并以一种易于检索/重复的方式存储这些计划，最好使用类似 TestRail 的工具，该工具可以创建完整的手动测试计划测试套件，团队可以按需手动重新运行这些测试计划。使用这样的工具的另一个好处是与其他开发人员和产品工具的集成，例如，将 TestRail 的运行连接到 Jira epic，以显示在特定功能的发布中运行了多少手动和回归测试。这不仅作为一个发布检查清单项有价值，而且还有助于进行任何发布缺陷的回顾，使您能够重新查看在发布任何给定功能之前运行的手动测试，以及添加额外的手动测试来捕捉任何通过的缺陷。

非专业手动测试

非专业化的手动测试通常被称为众包测试。有几个专有平台用于找到测试人员，如 Rainforest QA、Pay4Bugs、99Tests 和 Testlio。这些平台的定价模型通常基于提交的经过验证的缺陷数量而有所不同。根据产品的性质和您希望优化的缺陷类型，众包测试可以是一种非常具有成本效益和低投入的方式来提高产品质量。

半自动化测试

半自动化测试是一种相对较新的软件测试类别。这些测试是由非技术人员创建的，可能是您专业的手动测试团队，然后在完全自动化或监控环境中运行。

这些测试的主要缺点是可靠性。因为它们由非技术人员创建，所以可能没有完全自动化测试那么精确，容易出现虚假阳性和虚假阴性。尽管如此，这个领域正在快速发展，每年都有新的公司和工具推出，如 Rainforest QA 和 Testim，以改善可靠性并降低总体成本。

源代码控制

大多数公司在源代码管理方面使用的行业标准是 Git。除非你的组织有非常充分的理由使用其他工具，否则大多数现有和未来的团队成员已经掌握了 Git 的基本知识。如果不使用 Git，你可能会给他们增加一个不必要的学习曲线，强迫他们学习你所选择的替代工具。

目前有三个主要的云端 Git 托管平台：GitLab、GitHub 和 Bitbucket。这三个平台占据了市场的大部分份额，所以在违背这些标准之前应该慎重考虑。

Git 的学习曲线有一个有趣的形状。大多数人在达到一个基本的水平后就会停滞不前，这个水平对于大多数正常情况下的测试场景已经足够。然而，有时候会出现问题，比如开发人员丢失了一个提交或者在合并时出现了混乱。当这种情况发生时，如果他们没有继续学习 Git 的其他知识，就会感到沮丧并导致工作的放缓。作为团队负责人，我鼓励你投入时间去学习 Git 的后半部分知识。在命令行上专门使用 Git，了解实际发生的情况。学习 reflog、交互式变基、二分法查找以及各种内建的合并策略。掌握了这些知识，你就能避免一类极具影响生产力的问题，并且还可以培训你的团队成为 Git 的专家。

同行评审

一般来说，专家们建议对所有代码更改都实施一个强大的同行评审过程。（截至我撰写本文的 2023 年初，有越来越多的人对这一建议提出质疑，或者至少提供了一些细微差别，下一节将进行讨论）。大多数同行评审是通过所谓的 pull request（PR）、代码审查或合并请求来完成的。以下是一些建议，以保持代码评审的高效和高产： 请帮忙翻译：

- 将代码审查保持简短！设置最大的代码审查限制，例如十个文件和 200 行。其他内容应该拆分成多个堆叠/递增的审查。 （堆叠审查是基于或依赖于之前审查的代码审查。完成后，逐个合并审查以形成完整的更改。）
- 与团队明确并融入文化中的建立代码审查目标。代码审查不是为了风格或琐碎语义；这是您自动代码格式化器/代码检查器和静态分析的任务。代码审查的目的是确保清晰性，识别架构问题，标记缺陷和偏离的模式，注意边界情况，并确保遵守业务规则。
- 要求作者使审查员的工作更轻松。作者应包括对更改的描述，相关需求和工单的链接，并使用像 loom.com 这样的工具进行代码和代码预期工作的视频演示。
- 鼓励代码审查的作者在请求他人审查之前自我审查。作者的一些有针对性的评论可以节省很多时间。
- 预留专门的审查时间/窗口，以最小化干扰。 船、展示、询问

常见的智慧是，在将每处代码更改交付给客户之前，应由两个人进行审核。就像其他事情一样，存在权衡。手动代码审查并不是免费的，也不能保证软件质量。鉴于手动代码审查是有成本的，因此值得考虑何时这种成本提供的回报最高，并将代码审查作为最高回报投资的工具。这个通用的观念是由 Rouan Wilsenach 在 2021 年的一篇名为"Ship/Show/Ask"（ctohb.com/ssa）的博文中广为流传。

让我们来研究一下代码审查的成本。代码审查需要两个人，分别称为作者和审核者，经历多次上下文切换。一个常见的异步代码模式可能如下所示：

上下文切换 1： 作者停止在项目 1 上编码，设置代码审查，并标记审核者。作者开始在项目 2 上工作。 **上下文切换 #2：**审查者收到通知后，停止对项目 3 的工作，开始审查项目 1。审查者向作者提供反馈，然后恢复对项目 3 的工作。

上下文切换 #3：作者收到有关项目 1 的反馈通知后，停止对项目 2 的工作，处理审查者的评论。然后作者恢复对项目 2 的工作。

上下文切换 #4：审查者停止对项目 3 的工作，最好的情况是审查者对项目 1 中的更改满意并批准代码审查。然后审查者恢复对项目 3 的工作。最糟糕的情况是，作者和审查者必须多次重复上下文切换 #3 和 #4。

上下文切换 #5：作者收到批准通知后，停止对项目 2 的工作，合并项目 1，然后恢复对项目 2 的工作。

有方法可以减少这些上下文切换，但它们都涉及权衡。一个常见的替代方法是将所有代码审查作为同步的一对一编程练习进行；然而，这种策略将上下文切换换成了同步会议时间，这仍然会降低生产效率。无论如何，人工代码审查都是昂贵的。 我建议的替代方案是按照工作风险等级和代码审查的预期收益进行分类。一个示例分类系统：

不需要批准的琐碎更改：

- 复制/翻译更新
- 较小的用户界面更改，最好提交变更的视觉证据
- 仅用于测试的更改
- 明确表示目前尚未使用或在功能切换后被禁用的新代码
- 没有任何客户或用户能够访问的代码（例如，隐藏页面） 次要的更改最少审核或事后审核
- 代码更改伴随着测试，并涉及对现有模式和功能的增加
- 代码的实际使用有限或没有（例如，一个未部署的产品）
- 可以通过可靠的测试证明正确的重构

重要的更改需要仔细的事先审核

- 涉及新工具、框架、模式或架构的任何内容
- 重大的新特性 任何涉及敏感数据、个人身份信息或对安全状况可能产生影响的事项

虽然我相信这个系统可以提高整体团队效率，但我承认它并不适用于每个人。许多合规制度（如 PCI 或 SOC 2）要求进行 100%的人工代码审核。在这种情况下，你能做的最好办法是遵守规定，可能还可以划出一些不受合规框架约束的产品或功能领域，以尝试更细致和高效的流程。

分支模型

处理源代码控制的分支有很多方法，但行业整体上正越来越倾向于主干开发的概念。由于这似乎是目前最有效和常用的模式，我们在这里讨论的就是这个模式。如果你认真考虑使用其他模式，你可以在网上找到大量资源，讨论其他方法的方法论和最佳实践。

有许多博客文章提供了有帮助的图形，介绍了 git 分支模型，比如 Reviewpad 的这篇文章：ctohb.com/branching。如果以下描述对你来说不清楚，请参考这些文章及其相关图形。 在传统的分支模型中，有两个长寿命分支，一个是主分支，一个是开发分支，通常被称为 GitFlow。工作是基于开发分支进行的，以功能分支的形式进行，然后常常派生到另一个发布分支，针对特定的发布版本进行操作，最终合并回主分支。然后，修补漏洞是从主分支开始进行的，而进一步的开发则从开发分支开始。这种系统为每个变更到达生产环境都需要至少四个分支，并且需要同时维护多个分支。由于这些原因和其他因素，GitFlow 已经在很大程度上不再流行，也不再被认为是最佳实践。

基于主干开发和稍微复杂一些的 GitHub Flow 是源代码管理模型，旨在最小化分支数量和持续时间。GitHub Flow 和主干开发的具体实现会有所不同，但它们的共同之处在于只有一个分支，其名称会有所变化，但并不重要。在这里，我们称之为生产。生产分支始终是可部署的。实际上，我建议您设置自动化程序，以便将生产中的每个提交实际部署到生产环境。工作可以在以生产为基础的功能分支上进行，在功能分支中进行审核，并在准备就绪时进行合并。就是这样，一个长期存在的分支和许多短期存在（最好是小型）的功能分支。

为了使这种模型发挥良好作用，您需要满足以下几个前提条件：

- 持续集成通过运行健壮的测试套件，以确保功能分支的合并是安全的。
- 文化和实施使用功能开关，以便分支可以快速合并，然后在有利于业务的时间之后部署/启用功能。
- 强大的生产监控，用于检测变化。
- 快速部署能力，在生产环境中进行代码更改时，实现零停机时间。同样地，能够快速撤销个别更改以应对事故。
- 一种对于小型和短期功能分支保持纪律性的文化。如果功能分支变得庞大、持续时间长且笨重，那么 GitHub 流工作模型将失去其效率和简单性。正如在 3.3.5 功能分支环境中所讨论的那样，小的提交、小的分支和小的拉取请求是提高生产力的关键驱动因素。

长期使用与短期使用的分支

保持团队之间的分支和合并系统平稳运作的关键是保持分支的短期使用。几乎所有与代码合并相关的问题都源于代码分支开放时间过长或分支中包含的差异过大（ctohb.com/diffs）。一般来说，一个短期使用的分支应该仅开放几天，最多不超过两周。 记住，一个功能不一定要在一个分支中实现。例如，你可以先创建一个只包含测试的初始分支，进行审查和合并，然后再创建一个包含实现的分支。或者，你可以先构建一个与主应用程序无关的实现，进行审查和合并，然后再构建连接和测试的后续分支。

通过思考和实践，大多数实施都可以分解为可以独立合并的部分。这是一个在你的指导下团队可以随着时间发展的技能。

保持分支生命周期短暂的好处：

- 限制了其他分支的新代码合并到主干的时间，从而减少了代码冲突的范围。较小的分支本身也有较小的冲突风险。
- 保持功能分支代码相对较小，使审阅者更容易阅读，同时减少了代码破坏的范围。
- 在审查中鼓励更快的反馈，并在实施功能的过程中更早地进行修正。
- 鼓励团队建立可靠的持续集成系统。频繁的合并将突出构建/测试环境的不足，如果系统不可靠，会引起困扰并促进对这些系统的改进。

生产升级

升级器是一种工具，接收事件并管理呼叫轮换，如果呼叫轮换未接收到响应，则向相关人员发送电话通知。PagerDuty 可能是其中最流行的工具之一。

实施升级器

在你实施扶梯并设置轮班之前，请确保团队中的工程师已经选择加入轮班，并且每个人都知道并理解创建例外情况的期望（例如，在假期期间与其他人交换电话值班窗口）。

你还需要确保已经有足够的文件记录，并且每个人都知道收到电话时应该采取的标准程序。一些建立这些程序的考虑因素有：

- 注意收件人在哪里应该发布接收电话的确认信息（可能在扶梯工具本身或专门用于处理升级的共享群组聊天中）。
- 提供易于访问的手册，用于诊断特定类型的问题。
- 确定是否以及在哪里设置任何网站停机通知（例如，需要更新公司状态页面）。
- 决定何时以及多久发布关于调查进展、影响估计和恢复估计的更新信息。
- 确定一旦事件关闭后应该做什么，安排根本原因分析，并确保特定事件不再发生。

在实施扶梯和设置轮班之前，请确保团队中的工程师已经选择加入轮班，并且每个人都知道并理解创建例外情况的期望（例如，在假期期间与其他人交换电话值班窗口）。

你还需要确保已经有充分的文件记录，并且每个人都知道收到电话时应该采取的标准程序。建立这些程序时要注意以下几点：

- 记录收件人应该在哪里确认接收到电话（可能是在扶梯工具本身或专门用于处理升级的共享群组聊天中）。
- 提供便于查看用于诊断特定问题的操作手册。
- 决定是否以及在哪里设置网站停机通知（例如，需要更新公司状态页面）。
- 决定何时以及多频发布调查进展、影响评估和恢复预计的更新信息。
- 确定在事件关闭后应该做什么，安排根本原因分析，并确保不再发生类似事件。

根本原因分析（RCA）练习

每当系统出现对用户产生可衡量影响的问题时，你的团队都应该进行一定程度的根本原因分析（RCA）。RCA 的目标是了解系统在何处出现了故障，导致重大缺陷出现并影响了生产和最终用户。

要明确的是，根本原因分析*不能*用于确定故障或归咎责任。这需要在 RCA 过程的每个环节都得到保证，并融入到团队的文化之中。RCA 针对的是系统中允许出现故障的系统性问题（而不是人为错误）。

如果没有这种安全性和团队成员坦诚反馈和文档记录的意愿，你将错过改进系统的关键机会。

RCA 文件

您的团队应该为每个根本原因分析（RCA）制作一种形式的文档。根据您的系统出现问题的频率以及问题的性质，您可能希望为 RCAs 创建一个分类系统，低影响的事件采用较轻量级的 RCA 流程，而高影响的事件采用较重的 RCA 流程。值得注意的是，对于高影响的事件进行彻底的 RCA 是一项昂贵的工作，需要花费相当多的时间和思考，并且可能对于微不足道的缺陷来说过于繁重。

也就是说，对于大多数公司来说，在这个领域开销过多并确保更高的可靠性是更好的选择。您应该首先对所有事务进行彻底的 RCA，并在您对您的团队将面临的问题类型的景观和影响有了充分的了解后，才过渡到分层的 RCA 系统。

对于值得进行全面思考分析的问题，这是一个可以帮助您开始并向团队提出正确问题的模板：ctohb.com/rca。这是一个很好的做法，实际上也是大多数合规框架所要求的，为每个事件创建一个类似的新文档，并将它们组织在一个内部公司文档存储库中以备日后参考。

RCA 会议与时间表

一旦您解决了事故，尽快指定一个合适的人担任 RCA（根本原因分析）的负责人。负责人应该克隆模板，并开始填写有关事故的相关数据，并开始探索事故的五个为什么（ctohb.com/5whys）。

他们应该完成 RCA 的初稿，并将其分发给相关同行，在团队中安排一次会议，探索并尝试改进分析和未来防范措施。

与会者应提前阅读 RCA 的草案，并准备好探讨事故的具体细节，并对未来的防范措施进行构思。

选择 RCA 主要作者

RCA 的负责人不一定是应对事故的人。理想的 RCA 负责人应该对涉及的系统非常熟悉，并能提出深思熟虑的问题，了解工具和流程的失败之处，并提出改进的想法。 请注意，我们并不会将犯了人为错误的任何人推出圈外。如果符合之前的条件，那个人可能会成为 RCA 行动的负责人，但仅凭他们的错误本身，并不能使他们成为适合领导 RCA 的人选。他们应该参与并借此机会学习。再次强调，他们不会因为错误而受到处罚，而是作为流程的一部分。撰写 RCA 不是一种惩罚，而是系统维护的重要责任和组成部分。

安排 RCA 修复工作

一个良好的 RCA 流程通常会识别出许多工作项目，以改进系统并减少未来的事故发生可能性。下一个自然的问题是：我们现在做这些工作吗？对于参与其中的工程师来说，答案很可能是肯定的；但对于

关心截止日期和路线图的经理来说，答案就不那么明确了。

对于这个问题，并没有一个正确的答案，但以下是一些一般性的指导：

永不浪费任何好机会。在事故和 RCA 会议期间，解决问题的动力将达到高峰，高度激励的工程师通常效率最高。人们往往低估团队在系统可靠性问题上的总体成本，并因此对可靠性的改进进行次要排序。生产事故发生的事实应该提醒您和您的团队，这些投资对于限制干扰、使团队专注于有生产力的特性工作和保持持续高速是至关重要的。 许多问题的努力程度可能会有很大的差异。一些典型的问题可能是增加更多的日志记录或者更改我们的 CI 提供商的设置，以确保具有构建失败的 PR 不能合并。这些类型的琐碎的问题在后勤中的维护和调整成本要比当时处理它们更高，所以就直接解决吧。它们很少会做错，而且如果产生了负面后果，可以很容易地撤销。

对于高努力的修复步骤，我鼓励你将其进行筛选，并经过常规的计划流程。通常情况下，通过时间和计划，高努力的修复步骤可以简化。换句话说，第一天找到的解决问题的正确方法可能不是最理想的解决方案，只有通过技术审查的常规步骤，才能得出一个更好、可能成本更低的解决方案。

IT

这里我指的是 IT，信息技术，即用于日常业务的内部公司工具和技术。这与您的公司为客户产品构建的技术形成对比。

IT 通常包括公司硬件（台式机、笔记本电脑和手机）、VPN、电子邮件、防病毒和监控软件等工具。作为现代世界中的初创公司，无论您是面对面团队还是远程团队，只要您做出一些明智的决策，就不需要在 IT 上花费太多时间或资本。 一些关键决策将帮助您尽量减少小型科技公司的 IT 成本：

使用基于云的公司邮件、数据和文档系统。大多数初创公司都使用 Google Workspace，但如果您的团队成员（以及未来的潜在员工）更习惯使用其他系统，可以选择其他系统。在这个阶段，建立自己的内部邮件服务器、文档存储、数据访问和网络等系统没有任何好处。

在早期阶段，除非受合规系统的要求，不要要求员工使用公司硬件。在小规模采购阶段（尤其是在产品市场适应阶段之前），配置和管理公司硬件是一项不容小觑的工作（并且成本高！），几乎没有产生实际的重要或频繁的好处。

虽然承认可能有些痛苦，但确保产品和 IT 系统的安全是一项相当大的任务，对于年轻的初创公司来说，在早期进行详尽的工作是不切实际的。我鼓励您务实地将重点放在保护系统免受最可能的违规或数据泄露来源：员工的人为错误。相对于攻击者通过漏洞中间人侵入您的数据或云基础设施，更有可能是您的工程团队忘记在 API 之前设置身份验证，或者有人在咖啡店不小心将笔记本电脑忘在那里。

即使按照最佳实践来降低 IT 工作量，仍然会有一些无法避免的 IT 任务，主要涉及激活和停用员工账户以及为员工重置密码等。我鼓励您为其他同事（例如人力资源部门）撰写文档并进行培训，教他们如何完成这些任务，以免其经常打扰您或工程团队。

安全和合规

在本章中，我将为初创企业提供安全和合规的简要概述。你可以并且应该在这些主题上努力寻找比本书更深入的资源。

认证安全术语

特别是在安全方面，使用准确和精确的语言非常重要。以下是一些经常被误用的术语的定义：

认证，或者认证（AuthN）： 验证用户或客户是否是他们所声称的身份。你的登录系统执行用户认证。

Please note that translating technical terms and acronyms may vary depending on the context. It is always best to consult additional resources or a professional translator for accurate translations in the specific field. 授权（Authorization）或 AuthZ：验证用户或客户是否有权执行其尝试的操作。您的基于角色的访问控制（RBAC）或权限系统用于进行授权。

2FA 或 MFA：双因素身份验证和多因素身份验证是使用多种凭证对服务进行身份验证的过程。通常使用密码（作为第一因素）和某种所有权证明，如电子邮件的一次性密码（证明您拥有该电子邮件）、短信（证明您拥有手机号码）或定时一次性密码（TOPT）（证明您拥有设备/通行证）。请注意，由于 SIM 卡攻击的普及，攻击者有能力拦截或重定向短信，因此通常不建议使用短信作为第二因素。

初创企业的安全性

初创企业通常受到资源限制的影响。因此，安全状况和合规性通常是待办事项列表中优先级较低的事项，因为它们不太可能对业务构成生存威胁，而其他紧迫问题可能更为重要。如果没有用户或收入，黑客能够窃取什么？

考虑安全性也可能会影响生产力或导致高昂的开支，特别是如果您的任务是保护已存在的系统。但是，如果您从一开始就开始考虑安全性，您就有机会在开始阶段做出明智的决策，以在最小的额外成本下创建强大的安全性。 一些不会花费太多的方式来在创业初期加强安全：

- 在员工入职和培训材料中，将安全设立为团队思维的首要任务。
- 让所有工程师完成入职和定期的基础安全培训，例如 OWASP 十大安全风险或一些以游戏形式的安全培训，每月只需要花费几分钟的时间，以保持安全的重要性。
- 在任何涉及身份验证或权限控制的地方，依靠经过验证和保持良好维护的工具。
- 不要浪费时间自己建立登录页面；在 2023 年，真的没有理由这样做。像 Auth0、SuperTokens 和 AWS Cognito 这样的工具提供了安全的用户注册、登录、社交账号登录、密码找回管理、邮件身份验证、两步验证和会话管理。其中一些工具还提供了强大的权限控制系统。处理认证是一个庞大的项目；它非常复杂，出错的代价很高。创业公司没有必要解决这个问题。
- 不要对 IT 安全懒散。无论您使用 Dropbox、Box、Google Drive、SharePoint 等，花几分钟来设置策略以避免人为错误，例如将默认共享权限设置为仅限内部人员。设立定期的数据共享报告，并指定一名员工每季度审查对任何特别敏感的文件或电子表格的权限设置。

- 使用企业级密码管理解决方案，例如 1Password，并确保所有员工为重要工具使用强密码。同样地，尽可能经常使用单点登录（SSO），并确保您的 SSO 提供商配置了高安全性（至少需要多重身份验证）。
- 不要在您的代码库中包含机密信息。利用安全的秘密管理器，如 Google Cloud Secret Manager 或 AWS Secret Manager，并在代码中提交秘密的名称/位置，并在生产环境中将该名称解析为值，可以在引导时使用 Berglas 或 Whisper 等工具，或者直接使用秘密管理器 API 在运行时进行。

合规性

无论是因为您所在的行业、公司规模还是客户的性质，大多数初创公司都需要符合至少一个正式的合规性框架。如果您的用户在欧洲，那么您需要符合 GDPR。如果您在收集用户数据，则了解 CCPA 是明智的。如果您与企业客户合作，人们会要求您提供 SOC 2 或 ISO 27001 的认证。在医疗保健领域，有 HIPAA 规定，如果您从事支付业务，您可能听说过 PCI DSS。 对于初创公司来说，要符合这些框架中的任何一项或全部，成本可能过高难以接受。以下是保持合规性和预计成本的一些建议：

- 不要在最后一刻才去获取合规证书。准备和完成像 PCI DSS 或 SOC 2 这样的审计是一个漫长的过程，对于大多数初创公司来说，需要六到十二个月的时间。早期开始并保持合规性比晚些时候开始并进行重做更加便宜。
- 尽量使用自动化来执行或提供合规性的证据。有一批专注于自动化这些合规性框架的 SaaS 公司，比如 Vanta、Tugboat Logic、Secureframe、Laika 和 Drata 等，它们的服务可以大大减少获得认证的时间和总成本。
- 如果你很幸运有一个正式的合规性人员或部门，请充分利用这一关系。你在与合规性部门分享计划时越主动，越早地纳入他们的反馈意见，保持合规性的成本和压力就越小。

结论：衡量成功

你已经搭建起了一个出色的招聘流程，团队很满意，你像专业人士一样进行冲刺，你的架构能够应对业务日益增长的需求。这感觉很不错，但是你如何知道这已经足够了呢？作为技术领导者或首席技术官，如何衡量自己的成功和绩效？

从首席财务官的角度来定义这个角色的伟大之处的一种方式可能是：首席技术官能以多大效率使用研发预算，并将其转化为工程和产品产出？

或者，我们也可以从首席执行官的角度来看待：首席技术官领导的团队能多快地完成特定的业务目标？

另外，考虑到有效担任这一角色对于人员领导力的重要性，我们可以从人文主义的视角来看待：你的团队是否在做出最佳工作？毕竟，一位优秀的首席技术官的使命是建立一个组织文化，使个人工程师能够在技术领域做出最佳工作并实现不可能。

或者，与其试图定义一个单一的目标，也许对于首席技术官的伟大之处最好的定义是所有首席技术官可能每天运用的技能之和。也许伟大意味着当你将架构、绩效管理、供应商管理、高层领导、文化贡献、公众传播、指导、和 DevOps 放入公式中，并获得一个大于 42 的数字。

在我们尽力而为的情况下，似乎伟大的领导，甚至伟大的*技术*领导力也是我们无法精确量化或衡量的东西。聪明的人们会争论对伟大的共同描述，但我们都会同意，这个角色是多样化且不断变化的，需要不断学习和适应。

在工程领导中，有很少的普遍真理，但其中之一是成为一位优秀的工程领导者是一个永无止境的自我提升、发现和成长之旅。走上这条道路需要谦卑、愿意犯错误，最重要的是要保持好奇心和学习的愿望。

我希望这本手册对你在领导之旅中面临的挑战有所帮助。这本手册涵盖了多年来我自己所面临的许多挑战，也包括我有幸与许多优秀领导者互动时所面临的挑战。

我已经尽力为您提供一些应对这些挑战的结构，虽然每种情况都是独特的，最终您选择的路径是您自己设计的，结果是您自己要承担的。

在人生的某个时刻，人们会问您：你会给年轻时的自己什么建议？这本手册就是我对这个问题的回答。

希望它能帮助你在建设强大技术、激发动力的团队、成功经营业务以及最重要的是在世界上做一些有意义的事情的旅程中。

书籍参考

《完成获取事务》David Allen 著

- 《极限编程》Kent Beck 著
- 《工作法则！》- 拉斯洛·博克
- 《如何赢得朋友和影响他人》- 戴尔·卡耐基
- 《敏捷估算与规划》- 麦克·科恩
- 《从优秀到卓越》- 吉姆·柯林斯
- 《高效人士的七个习惯》- 斯蒂芬·柯维
- 领域驱动设计 - 埃里克·埃文斯
- 企业应用架构模式 - 马丁·福勒
- 高产出管理 - 安迪·格罗夫
- 扩大规模 - 弗恩·哈尼什
- 关于艰难的事情的困境 - 本·霍洛维茨
- 对变革的免疫力 - 罗伯特·凯根
- 《凤凰计划》Gene Kim
- 《团队的五大失效行为》Patrick Lencioni

- 《管理人类》Michael Lopp
- 《团队的团队》Stanley McChrystal
- 《摆脱建筑陷阱》Melissa Perri
- 好的权威——乔纳森·雷蒙德
- 好战略/糟糕战略——理查德·鲁梅尔特
- 激进坦诚——金·斯科特
- 敏捷开发艺术——詹姆斯·肖尔和沙恩·沃登
- Scrum：一半的时间完成两倍的工作——杰夫·萨瑟兰
- *Extreme Ownership* by Jocko Willink and Leif Babin

数字参考资料

VSCode 的设置文件 (ctohb.com/vscode): https://code.visualstudio.com/docs/getstarted/settings

Gmail 的关键 (ctohb.com/keytogmail): https://techcrunch.com/2010/03/14/key-to-gmail

粪便伞 (ctohb.com/umbrella): https://medium.com/@rajsarkar/word-of-the-month-shit-umbrella-33e3182a0f1b **Liar's Paradox** (ctohb.com/liarsparadox): https://en.wikipedia.org/wiki/Liar_paradox

Gitlab Compensation Calculator (ctohb.com/gitlabcompcalc): https://about.gitlab.com/handbook/total-rewards/compensation/

Five Whys (ctohb.com/5whys): https://en.wikipedia.org/wiki/Five_whys

Take the DORA DevOps Quick Check (ctohb.com/dora): https://www.devops-research.com/quickcheck.html#questions

Netflix Keeper Test (ctohb.com/keeper): https://empowerment.ee/wp-content/uploads/2021/05/NETFLIX-%E2%80%93-THE-KEEPER- TEST.pdf

说谎者悖论 (ctohb.com/liarsparadox): https://en.wikipedia.org/wiki/Liar_paradox

Gitlab 薪酬计算器 (ctohb.com/gitlabcompcalc): https://about.gitlab.com/handbook/total-rewards/compensation/

五个为什么 (ctohb.com/5whys): https://en.wikipedia.org/wiki/Five_whys

进行 DORA DevOps 快速检查 (ctohb.com/dora): https://www.devops-research.com/quickcheck.html#questions

Netflix Keeper 测试 (ctohb.com/keeper): https://empowerment.ee/wp-content/uploads/2021/05/NETFLIX---THE-KEEPER-TEST.pdf

为什么 GitLab 支付本地费率 (ctohb.com/local): https://about.gitlab.com/blog/2019/02/28/why-we-pay-local-rates/

什么是 Topgrading 面试流程？ (ctohb.com/interview): https://www.greenhouse.io/blog/what-is-the-topgrading-interview-process

Topgrading (ctohb.com/topgrading): https://en.wikipedia.org/wiki/Topgrading

涂抹大桥 (ctohb.com/painting): https://www.goldengate.org/bridge/bridge-maintenance/painting-the-bridge/

勇于领导: e BRAVING 清单 (ctohb.com/braving): https://brenebrown.com/resources/the-braving-inventory/

根本原因分析模板（ctohb.com/rca）：https://docs.google.com/document/d/1GuRZgDpMVg_Qf3sR7r8tZqRx6Re0oBrwIcnj-ekgJ60/edit#

Ship Small Diffs（ctohb.com/diffs）：https://blog.skyliner.io/ship-small-diffs-741308bec0d1

GitHub Flow，基于主干的开发和代码审查（ctohb.com/branching）：https://reviewpad.com/blog/github-flow-trunk-based-development-and-code-reviews

特性切换（又名特性标志）（ctohb.com/hodgson）：https://martinfowler.com/articles/feature-toggles.html

多阶段构建（ctohb.com/docker）：https://docs.docker.com/build/building/multi-stage/

选择无趣的技术（ctohb.com/boring）：https://boringtechnology.club/

技术雷达（ctohb.com/radar）：https://www.thoughtworks.com/en-us/radar

瀑布模型（ctohb.com/waterfall）：https://en.wikipedia.org/wiki/Waterfall_model#History

`**责任型程序员**（ctohb.com/tpp）：https://en.wikipedia.org/wiki/e_Pragmatic_Programmer

橡皮鸭调试法（ctohb.com/rdd）：https://en.wikipedia.org/wiki/Rubber_duck_debugging

频率降低难度（ctohb.com/fowler）：https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

Figma Material Design（ctohb.com/figma）：https://www.figma.com/@materialdesign

如何使用 Atlassian 设计系统进行设计（ctohb.com/design）：https://atlassian.design/get-started/design **Building Productive Teams** (ctohb.com/teams): https://docs.microsoft.com/en-us/DevOps/plan/building-productive-teams

建立高效团队：(ctohb.com/teams)：https://docs.microsoft.com/zh-cn/DevOps/plan/building-productive-teams

GitHub 报酬计算器：(ctohb.com/calc)：https://about.gitlab.com/handbook/total-rewards/compensation/

Codeacademy 工程能力：(ctohb.com/competencies)：https://github.com/Codecademy/engineering-competencies

多任务的神话：(ctohb.com/myth)：https://blog.codinghorror.com/the-multi-tasking-myth

缩写真的很烦：Elon Musk：(ctohb.com/acronyms)：https://gist.github.com/klaaspieter/12cd68f54bb71a3940eae5cdd4ea1764 **How We Work Without Meetings at Levels** (ctohb.com/async): https://medium.com/levelshealth/ how-we-work-without-meetings-at-levels-a6a525e21aa5

如何在 Levels 工作而无需开会（ctohb.com/async）：https://medium.com/levelshealth/how-we-work-without-meetings-at-levels-a6a525e21aa5

在 Slack 中有礼貌地合作：考虑这些 Slack 礼仪提示（ctohb.com/slack）：https://slack.com/blog/collaboration/etiquette-tips-in-slack

如何有效地使用跳级会议（ctohb.com/skip）：https://www.managementcenter.org/resources/skip-level-meeting-toolkit/

从创始人到 CTO（ctohb.com/founder2cto）：https://calv.info/founder-cto

如何优先考虑开发者体验并提高产出（ctohb.com/dx）：https://www.harness.io/blog/developer-experience

术语表

敏捷仪式：敏捷仪式是开发团队在开发过程的不同阶段聚集在一起的会议，用于讨论未来工作的计划、沟通正在进行的工作或者回顾和反思过去的工作。

求职者追踪系统 (ATS)：求职者追踪系统 (ATS) 是用于招聘人员和雇主在招聘和聘用过程中追踪候选人的软件。

童子军法则：让事物留下比你发现时更好的状态。在技术团队中，每当你在一个代码区域工作时，始终要做出即使是小的改进，例如改善测试、文档或者提高清晰度、可读性或可维护性。

布朗菲尔德开发：与绿地开发相对，指的是利用现有的遗留系统进行开发，通常受到技术债务的重大影响。你被束缚在过去所做的高层决策中，并且在进行重大更改时有限的灵活性。

商业智能（BI）：商业智能是一种软件，它摄取业务数据并以用户友好的形式呈现，如报告、仪表板、图表和图形。

点击运维（ClickOps）：点击运维是指人们通过点击云服务商网站上的不同菜单选项，选择和配置正确的自动化计算基础设施的容易出错且耗时的过程。

容器化：容器化是指将包含运行应用程序所需所有元素的软件打包到容器环境中。这使组织能够从任何地方（包括私有数据中心、公共云甚至个人笔记本电脑）运行应用程序。

上下文切换：从一项任务转到另一项任务的过程。对于工程师来说，这意味着放下正在处理的问题，开始处理另一个问题。切换的行为通常耗时且效率不如一次只处理一个问题。

直接汇报：直接汇报是指下属直接向组织架构中位于他们之上的人报告工作，通常是经理、主管或团队领导。

工程产品与设计（EPD）：将传统上的三个独立部门（设计、产品和工程）合并为一个部门的概念。首字母缩写的顺序经常会改变为 EDP、PDE。

Greenfield 解决方案：Greenfield 软件开发指在一个全新的环境中进行开发工作，几乎没有预先存在的旧代码，并且可以自由选择工具、模式和架构。

Horizon One/Two/Three：每个 Horizon 代表不同的时间范围。Horizon One 通常是短期（几天/几周），Horizon Two 是中期（几个月），Horizon Three 是长期（几年）。通常用作规划工具，以确保满足每个 Horizon 的需求。

幂等性：如果对一个技术操作进行多次执行不会改变输出结果，则该操作被称为幂等操作。 **Kaizen**: Kaizen is the philosophy of continuously improving all processes in an organization.

改善活动 (Kaizen)：改善活动是组织中不断改进所有流程的理念。

关键业绩指标 (KPI)：KPI 是衡量进展朝着预期结果的关键可量化指标。有时被称为输入指标。

目标和关键成果 (OKRs)：目标和关键成果是一种设定目标的框架，起源于上世纪 70 年代的英特尔，用于个人、团队和组织来定义可测量的目标并跟踪其结果。

鸽笼原理 (Pigeonhole principle)：描述的是存在固定数量的结果和更多次的尝试的情况。如果你抛掷硬币三次，那么至少会出现两次正面或反面。

产品需求文档 (PRD)：PRD 是一个将产品需求的背景、参考资料、理由和表达集中在一处的文档。 **Reproducibility**: The ability to reproduce a given outcome on demand.

可重复性: 在需要时能够重现特定结果的能力。通常在软件开发中，它以用户按下按钮 X，然后应用程序崩溃的形式呈现。如果按下按钮是导致应用程序崩溃的完整可靠描述，则这是一个可重现的崩溃，并且有已知的重现步骤。

评论请求（RFC）: 这是一份旨在收集反馈并最终成为长期参考资料的文件，概述了一个想法、哲学、提案或方法论。

根本原因分析 (RCA): 一种方法，通常是一份文件，旨在深入了解事件发生的真正原因。通常用作调查工具，然后用作软件项目中发生事故原因的参考文件。

SaaS 管理平台 (SMP): SMP 提供了一个单一的位置，用于审查、管理、优化和治理组织中使用的 SaaS 工具。

面向服务的架构 (SOA): “面向服务的架构”一词起源于 1990 年代，用来指称一些相当具体的技术选择。如今，该词用于更广泛地描述系统中信息在网络上的传递的一种方式。

早会：（又称每日 Scrum）作为 Scrum/敏捷仪式的一部分而进行的常规会议。一般意在促进团队内部的沟通、更新、冲突解决和决策制定，通常时间不超过 30 分钟。

稻草人模型：是指在数据不完整的情况下快速组合起来的首稿建议。通常作为团队的起点提案，加快收集反馈和解决问题的进程。

同步/异步工作文化：异步工作文化的理念是鼓励异步进行沟通，即不需要沟通的双方同时参与。例如，书面文件促进了异步沟通，作者在读者阅读文件时无需同时在场。异步文化弱化会议的重要性，并强调书面或记录的音频/视频文档。

关于作者

扎克·戈德伯格（Zach Goldberg）以计算机科学和工程专业以最高荣誉毕业于宾夕法尼亚大学。他曾担任六家创业公司的首席技术官，其中包括 WiFast、Sticks and Brains、AutoLotto、Trellis Technologies、GrowFlow（2022 年被 Dama Financial 并购）和 Towards Equilibrium Inc，在腾讯担任企业家驻场和在 Google 担任副产品经理。2022 年 Dama 收购了 GrowFlow 后，扎克坐下来将自己的所有经验倾囊相授，并将其写入了这本书中。了解更多关于扎克的工作，请访问 zachgoldberg.com。

关于出版商

WorldChangers Media 是由 Bryna Haynes 于 2021 年创立的精品出版公司，专注于具有影响力的思想。我们知道伟大的书籍改变人生，推翻过时的范式，并建立起运动。我们致力于提供下一代思想领袖所创作的高质量、具有改变力的非虚构作品。

想与我们一起撰写并出版您的思想领袖图书吗？了解更多信息，请访问 <www.WorldChangers.Media>。