第一部分

可扩展性组织的人员配置

第 1 章　人员和领导力对扩展性的影响

孙子说：斗众如斗寡，形名是也。

这里有一个多项选择题。哪个因素对确保产品的长远扩展性最为关键？人员、过程还是技术？你的答案是人员吗？如果是，那就给你赞一个。如果不是，那就要考虑下面的事情：据我们所知，电影《机器人启示录》中所描述的事情目前还没有发生，产品也不会进行自我复制。产品所带来的价值和出现的责任都是人为的结果。因为是依靠人去完成设计、编码、配置和构建系统来运行产品的，所以在这个过程中所出现的任何缺陷都只能由人来负责。对于产品的架构，不论是成功地处理了海量交易，还是悲惨地在消费者的压力下崩溃，人都要负全部的责任。然而，作为科技工作者和工程师，当我们努力设计可扩展性方案或者解决可扩展性问题的时候，却往往忽略了人的因素。人的作用不幸地被忽略和低估。员工考评每年一次，玩的是打钩选择的游戏，评估报告草草收场。经理和领导往往对其任职的岗位没有接受过培训或者培训不足。我们将在本章解释为什么组织中的人员、结构、管理和领导对扩展性有着极大的影响。

1.1　案例方法

在本书第 2 版中，我们讲述了现实世界中真实的故事和案例，为讨论的问题注入了生命力。我们用的一些例子来自于与 AKF 公司的客户一起工作的亲身经历。其他的案例源于对公开文档的深入研究，或者采访我们感兴趣的公司中拥有一手资料的人。

第 1 版有意避开了现实世界中的案例，用虚拟的 AllScale 公司来替代。这个虚拟公司是许多客户所面对的挑战及成功经验的集合体。AllScale 公司的价值在于通过一个能力有限的虚拟公司，把用户和现实世界的成功与失败连接起来，从而帮助我们传达要教导的经验和教训。

在第 2 版中，有时候我们会在讲述故事的时候简单地介绍人物，比如苹果的史蒂夫・乔布斯，亚马逊的杰夫・贝佐斯。在一些故事里，我们会简要地概述一个公司的成功或者失败，然后分析和指出那些导致他们成功或者失败的关键因素。

1.2　为什么要讨论人

在我们看来，人对扩展性具有重要作用。如果想要确保产品可以扩展，人是最为重要的因素。好的一面是，没有人就永远没有扩展性问题；坏的一面是，如果没有人，那么将永远没有机会去研发一款需要扩展性的产品。人负责架构系统、研发或者选择软件、安装软件来运行和支持产品。人（或通过脚本）配置服务器、数据库、防火墙、路由器和其他的设备。通过启用或者停用，人决定哪部分

产品将会在密集需求的压力下成功或者失败。人来设计公司的需求，定位现在和未来的扩展性问题的过程。没有人无法开始新的项目，没有人错误就不会犯。人，人，人……

所有扩展性最成功的产品，最核心的部分是有一群人，他们做出了很多正确的决策，当然偶尔也出些昏招。在扩展性方面，忽略人的因素的作用，是一个非常大的错误，我们经常发现忽略人的因素是产品无法满足用户需求的根本原因所在。

既然人是可扩展性的心和脑，我们就应该花大力气去吸引和留住最好的人才。正如我们在第 5 章管理秘籍中谈到的，这个问题不仅仅是要找到技能最好且付得起工资的人才。更进一步，要想成功，必须有合适的人、合适的行为、合适的工作、合适的时间。

合适的人指的是此人具有合适的知识、技术和能力。把一个人在合适的时间放在一个合适的岗位上，确保其能成功地为股东创造出最大的价值，同时这也符合其职业发展方向。合适的行为指的是此人能与其他的员工融洽相处，并认同公司的文化和价值观。不良的行为如同缺乏技能一样，也是团队淘汰人的有力理由，因为不良的行为对任何团队来说都会产生恶性循环、降低士气和生产率。

也许，没有什么比硅谷的传奇人物史蒂夫·乔布斯被开除，后续又重新被雇用的故事更能说明为什么要求有合适的人、合适的行为和合适的时间。在沃尔特·艾萨科森的主力传记《史蒂夫·乔布斯》一书中有一幅画，描述了 1985 年，在史蒂夫大约 30 岁的时候，以他的孩子气、粗鲁、自私和捣乱的行为对公司进行破坏，造成苹果士气低下。显而易见，约翰·斯卡利（取代乔布斯的人）也并不是一个好的答案。乔布斯缺乏团队精神，不断地改变产品的构思，结

果造成公司的产品缺乏聚焦和连贯性。在苹果发布了改变世界的麦金塔电脑（Macintosh）后，乔布斯显然拒绝相信或者接受苹果 II 是公司继续驱动销售业绩的主要产品。他的嘲弄行为激起了产品团队的不满，所持的意见和公司普遍认同的，从错误中学习和在成功时获利的文化背道而驰。作为公司的领导，乔布斯当时的行为绝对是错误的。

让我们再来看看十年后的情况吧。1996 年，通过并购 NeXT 乔布斯被召回，担任苹果的 CEO，后续他罢免了吉尔·阿梅里奥。也许是在 NeXT 和 Pixar 工作的锻炼，使乔布斯更加成熟。他依然保持着对完美产品的疯狂追求。乱发脾气是导致他 1985 年突然被解职的原因，这次他没有重蹈覆辙。乔布斯返回苹果后，更加自律，伴着与生俱来的能力和三十岁的盛气凌人，带来了扭转公司局面的领导力。这是一个如何把领导者培训得更好的案例（本书的后续章节会进一步描述）。在 1985 年，乔布斯是一个合适的人，但遗憾的是当时他没有合适的行为。在 1996 年回归时，显然，他是个合适的人，有合适行为，并在合适的时间做了合适的工作。请注意，这里说的“合适的行为”不是说乔布斯是一个好人；事实上，艾萨科森在书中也指出乔布斯并不是一个好人。进一步说，合适的行为意味着乔布斯行为的综合结果，对苹果成长的影响基本上是正面的。

1.3　为什么组织很重要

如果人是系统扩展中最重要的因素，那么如何把人组织起来完成工作也就同样重要。要成功地设计一个组织，首先必须弄清楚组

织的产出是什么。相对于我们要达成的目标，任何一个组织都有缺点和优点。这里，我们提出几个问题，来考虑组织是如何对期望值产生正面或者负面影响的。

- ▼ 为该组织增加或减少人有多难？是否要成组地增加？还是可以单个人增加？
- ▼ 组织对制订度量生产率的指标起到促进还是阻碍的作用？
- ▼ 组织是否允许团队拥有自己的目标，并且有充足的授权和能力去实现？
- ▼ 有哪种类型的冲突？这种冲突会帮助还是会阻碍使命的完成？
- ▼ 组织对内部创新起到促进还是阻碍作用？
- ▼ 组织对于产品上线起到促进还是阻碍作用？
- ▼ 组织会增加还是减少实现单位价值的成本？
- ▼ 流程在组织内是否很容易流转？或者只是在部分组织内容易流转？

关于是否容易增加人的问题，其实，答案是显而易见的。如果组织不允许增加人手完成更多的工作，那么增加工作量将是一件极为困难的事情。好的组织支持增加少量或者大量的额外人手，允许形成新的团队或者把人加到现有的团队中去。当公司景况不佳的时候，机构也能够灵活地缩小规模。

有关指标的问题也很重要，因为组织的产出取决于规模（多少人工作）和效率（每个人或团队的产出）。取得好的产出，有时并非单指增加“更多的人”，相反，是“每个人做更多的工作”或者“以相同的规模有更多的产出”。如果不能很好地理解和掌握个人和组织的效率（度量产出、可用性、响应市场的时间、单位产出成本），我

们如何才能确定是否应该在提高生产率的工具上加大投入，还是要增加人员？如果汽车没有配备油量表和速度表，那么大多数人都不会考虑开这种车出去。同理，如果没有关键性的性能指标来帮助我们度量达成期望的结果，那么我们就不应该去管理这个机构。管理意味着度量，度量失败即管理失败。如果组织很难度量每个人的表现，那么你就无法度量产出。如果你无法度量组织的产出和工作的质量，你就无法应对突然发生和快速发展所带来的问题。

如果团队成员对是否拥有目标和足够的授权去完成任务理解得很清晰，那么就可以靠团队的能力去拥有目标，并达成目标，相反，则只能依赖其他的团队才能达成目标。拥有足够授权的团队，通常比授权不足的团队有更高的士气、较低的跳槽率和较快的市场响应速度。授权的基础是为达成目标，有可以独立做出必要决策的能力。要真正感受到自己是任务的主人，团队必须具备应有的工具和能力，并按照自己的决策去实施。授权程度最高和对目标拥有度最高的是那些跨职能而且拥有达成目标所需要的全部技能的团队。有关组织的内部冲突类型和冲突是促进还是妨碍使命达成的问题，这里将讨论组织内部的两个基本的冲突类型（情感型和认知型），以及它们与组织产出之间的关系。情感型冲突是以角色或控制为基础的冲突，经常发生在团队之间。认知型冲突常常是关于“谁”做，或者“怎么”做。在组织中，常常需要多个团队的合作才能完成一个产品，例如运维、研发、质量保证，真正的问题是关于“谁”来定义何时做完，或者“怎么”把事情做完来满足客户的需要。对于应用和数据库交互的具体方式，基础设施团队参与提供多少意见？对于定义基础架构的实施，软件团队参与提供多少意见？谁来做决策？情感

型冲突很少能提升产品的价值，相反，几乎总是延迟产品的发布和增加成本。更进一步，如果不妥善处理，就会降低员工的士气，增加员工的跳槽率，降低公司内部的创新水平。

认知型的冲突，如果处理得当，常被称为“好的冲突”。最常见的认知型冲突是关于“为什么”某件事一定发生，或者公司需要“什么”来达成期望的结果。试想一下你参与过的那些头脑风暴会议，或者真正有效率的优先级调度会议，团队确实聚集到一起，做出艰难的决策，同时对会议的结果深感满意。认知型冲突扩大了策略的可能性范围，通过结合不同的知识、技能和经验来覆盖那些相互交叉的部分，从而增加了正确决策的概率。你可能有过这样的经历，当和那些缺乏你所拥有的知识和专业经验的人在一个会议室里开会时，有人问了一个问题，进而彻底地改变了解决问题的方法。

当两种类型的冲突交织在一起的时候，可以帮助我们思考应该如何构建团队。为了减少情感类型的冲突，我们希望跨职能的团队能够拥有完整的目标及产出。这就意味着把各种必要的技能集中到团队中来，共同完成某些解决方案。同样的策略，在团队内把各种技能集中起来，提高团队的认知或者积极冲突的水平。当我们构建这类跨职能团队的时候，会发现士气提升、授权度提升、员工跳槽率降低、市场响应速度提高和创新提升。

另外的两个问题，“机构对创新起到促进还是阻碍作用?”和“机构对市场响应速度起到促进还是阻碍作用?”至少部分答案已经从前面的讨论中看出了。对问题的重要性大家已经有了共识，但是得到的答案却不太可靠。我们将在第 3 章中对这些问题进行更深入的探讨。

组织和个人的平均产出之间的关系，引出了“组织扩展成本”

的话题。在《人月神话》(The Mythical Man-Month）一书中，弗雷德里克·布鲁克斯曾经指出，在软件项目开发的过程中，有一个时间点，给项目组增加人员实际上会导致项目的进一步延迟（也就是说交付得更晚)。布鲁克斯指出延迟的原因之一是团队每增加一位新成员都会带来沟通上的额外负担。随着团队规模的增长，增加成员所带来的额外沟通负担呈线性特征。换句话说，人增加得越多，每个成员的单位沟通和协调成本也就越大。

你是否曾经有过这样的经验，每天收到数以百计的内部邮件，每周收到几十个会议邀请？如果是这样，你一定花了大量宝贵的时间去删除邮件和拒绝那些和你的工作职责不相关的会议邀请。公司、部门、团队的人越多，你就必须要花越多的时间阅读和删除邮件，参加各种会议，而不是做真正的工作。这种场景完美地解释了，为什么增加人可能降低组织内的个人产出。在前面的例子中，当人员增加时，邮件的数量增加，沟通的时间也随着相应地增加。图 1-1 描绘了工程团队试图协调和沟通的场景。表 1-1 列出随着团队规模从 1 增加到 3，整体产出增加，而个人产出却降低。在表 1-1 中，由于沟通协调而造成的个人生产率损失是 0.005，这意味着每天 8 小时中有 2.4 分钟的协调活动。这并不是很长时间，而且大多数人会直觉地认为这三个人做同一个项目，每天至少花 2.4 分钟去协调他们之间的活动，甚至还配有一位经理。这样一来，即使个人生产率降低，但是团队产出会提高。

有几种方法可以减少，但是却无法彻底消除这种负面影响。一种可能的方法是通过加强管理来限制那些没有必要的协调工作。另一种方法是通过建立自给自足的小团队来限制个人之间的交互活动。

这两种方法各有利弊，我们会在第 3 章中详细讨论。当然，还有很多其他可能的办法。任何能够提高个人产出，同时不损害创新的办法都可以考虑。

图 1-1　协调偷走了个人的生产率

“工作是否很容易地在组织内部流转？还是容易在组织的某个地方被卡住？”这个问题聚焦在组织设计与所从事工作类型的匹配度上。瀑布过程类似于流水线，工作在组织机构内的流转是否组织得像流水线那么有效？如果允许工作在团队之间流转，那么公司的流程就很合理。产品架构和流程（如敏捷）是否允许工作在同一个职能部门开始和结束？要实现这样的结构，架构的组件必须能够独立工作，团队之间只有最基本的交互。还有一点就是组织、流程和技

术必须紧密配合。面向职能采用敏捷过程的团队，在运行庞大、高度独立的架构时，可能遇到的问题如同采用瀑布流程的跨职能团队，在独立和水平扩展的产品架构上所遇到的问题一样多。

著名的亚马逊公司清楚地展示了对问题锲而不舍所带来的价值。你可能听说过亚马逊著名的“两张比萨团队”的概念。在一次外出的团建活动中，亚马逊的执行人员开始讨论团队间需要更好的沟通。创始人和 CEO 杰夫·贝佐斯加入说道：“不，沟通很可怕！”㊀贝佐斯意识到表 1-1 中所描述的与弗雷德里克·布鲁克斯在《人月神话》中所阐述的沟通成本。贝佐斯的解决办法是建立“两张比萨饼团队”规则：任何一个团队的规模不能大过两张比萨所能喂饱的人数。这条规则的含义是，沟通主要发生在团队内部，因此额外的沟通负担就大大地减少了。团队还要有足够的授权才能成功地达成目标。每个团队都要设置一个或几个 KPI 来衡量整体的成功情况。通常配置跨部门的人员来确保团队技能齐全，不必请求外援。㊁

表 1-1　团队规模扩大造成的个人生产率损失

团队规模	沟通成本	个人生产率	团队生产率
1	0	1	1
3	0.005	0.995	2.985
10	0.010	0.99	9.9
20	0.020	0.98	19.6
30	0.05	0.95	28.5

㊀ Alan Deutschman.“Inside the Mind of JeffBezos.” FastCompany.http://www.fastcompany.com/50106/inside-mind-jeff-bezos.

㊁ Stowe Boyd. “Amazon’s Two Pizza Teams Keep It Fast and Loose.” GigaOm. http://research.gigaom.com/2013/08/amazons-two-pizza-teams/.

贝佐斯希望通过制订“两张比萨团队”的规则来产生高水平的创新、快速的市场响应和团队间低水平的情感冲突。他为团队设计了明确的 KPI，目的是确保目标完成的情况可度量。同时确保解决方案具有高度的可扩展性。是否需要更多的能力？为了加强产品是否要增加更多的团队？其实，每个团队的内部工作都很容易控制。通过一个简单的组织结构解答所有的关键问题。

讨论完了组织设计对扩展的重要性，现在把注意力转移到另一个问题，为什么管理和领导如此重要？

1.4　为什么管理和领导如此重要

大多数理工科（科学、技术、工程和数学）的毕业生从来没有接受过管理学和领导方面的学术或者基础性的培训。开这类课程的大学极少，除非你读管理专业或 MBA 课程。因此，对如何管理和领导这个问题，大多数的产品经理都是自学成才。他们通过观察，向同级别和更高级别的经理学习，做出自己的决策。久而久之，这些经理们开始开发自己的“工具箱”。有些经理从专业读物上学习和掌握到新的秘籍，摒弃陈规陋习。这就是多年来培养经理们通用的“从工作中学习”的方法。虽然这种办法有些好处，但不幸的是在大学和大公司的结构化的课程中，管理和领导的教育没有受到应有的重视。

管理和领导对于处在增长中的公司来说，可以倍增你的能力，从而实现业务的扩展。尽管管理和领导经常同时出现，然而实际上他们对可扩展性的影响大相径庭。同一个人经常既是领导也是经理。

在大多数的机构中，一个人往往从独立贡献者发展成为初级管理者；假以时日，又不断地被提升，去承担更多的领导责任。

总的来说，管理是与“推”（pushing）相关的活动，而领导是与“拉”（pulling）相关的活动。领导设定目的地和通往目的地的路线图。管理设法到达目的地。领导会说：“我们的系统永远都不会因为扩展性而瘫痪”；管理者的工作就是确保这种情况永远不会发生。每个公司都需要领导和管理，而且两者都要做好。表 1-2 列出了部分领导和管理的活动及其分类。

表 1-2　领导活动与管理活动比较

领导活动	管理活动
制订愿景	评估目标
定义使命	绩效考核指标衡量
设定目标	项目管理
营造文化	绩效考核
绩效考核指标选择	员工指导
激励	员工培训
制订标准	评估标准

我们常常听说“管理风格”，一个人的“管理风格”使其更像一个领导或更像一个经理。风格代表着个人对领导或管理任务的理解。如果一个人聚焦在事务处理方面，那么他就是一个经理，如果一个人更具有远见卓识，那他就是一个领导。尽管每个人都拥有独特的品格和技能，让我们有能力自信地工作，但是我们可以提高自己的领导和管理水平。理解两者之间的区别，才能区隔和发展管理与领导的能力，为股东带来利益。

前面提过，管理是与“推”相关的活动。管理把适当的任务分

配到人，并且确保这些任务可以在指定的时间内以适当的成本完成。管理对工作表现及时反馈，既反馈对良好表现的赞扬也指出需要改进的地方。管理聚焦在度量和提高，目的是为股东创造价值，例如降低成本或者以相同成本增加产出。管理要求尽早和经常地进行沟通，清楚地定位哪些进展良好，哪些需要给予帮助。在通往目的地的路上经常有障碍，管理活动也包括移除障碍或者帮助团队绕过障碍。管理对扩展性很重要，因为这关系到如何从组织中得到最大的产出，因此要降低单位产出的成本。定义应该怎么表现是管理的责任，实际表现得如何对组织、流程和系统的影响极大。

管理与人相关，需要确保合适的人，在合适的时间，以合适的行为，做合适的工作。从组织的角度看，管理要确保团结有效，包括把合适的技能和经验组合起来，才能取得成功。把管理应用到组织工作上，就是要确保项目在不超过预算的前提下，按时完成预订的任务，达到预期的交付效果。管理就是度量，度量失败即管理失败。如果疏于管理，结果必然是无法达成组织、流程和系统的可扩展性目标。如果没有管理，就没有人对要按时完成的事情负责任。

以此相对，领导是与“拉”相关的活动。如果管理是推动组织爬坡，那么领导就是选择山头，鼓励员工一起努力翻越山头。领导激励员工和组织做正确的事并好好做事。领导是描绘激动人心的愿景，并把愿景深入到员工的心里，引领他们为公司做正确的事情。领导确定使命、描绘愿景、制订路线图，帮助员工理解做什么和如何做才能为股东创造价值。最后，在向组织最高目标前进的路上，领导定义阶段性的目标和 KPI。领导对扩展性很重要，不仅是因为要设定方向（使命）和目标（愿景），而且要激励员工和组织向目标

迈进。

任何缺乏领导的努力（包括公司里为了提升可扩展性而提出的项目），即使不会因为某种挫折而彻底失败，也仅仅是靠侥幸取得成功。好的领导创造文化，聚焦打造具有高可扩展性的组织、流程和产品而取得成功。这种文化靠激励体系来确保公司能够在成本可控的情况下扩展，同时不影响用户体验和出现扩展性问题。

1.5　结论

我们前面已经强调过，人员、组织、管理和领导这几个因素对可扩展性都很重要。人是影响可扩展性最为重要的因素，没有人，就没有过程和系统。有效的组织将协助你快速达成目标，反之则产生阻碍作用。在组织里，管理和领导的作用分别是推和拉。领导鼓励员工取得更大的成绩，管理鼓励员工实现目标。

关键点

- ▼ 在可扩展性的拼图中人员是最重要的一块。
- ▼ 对可扩展的组织，需要有合适的人，在合适的时间，以合适的行为，做合适的工作。
- ▼ 组织的结构很少有对与错之分，任何的结构都有利有弊。
- ▼ 在设计组织的过程中要考虑下述几个方面：
  - 易于在现有的组织上增加新的工作单元。在组织里增加和减少人员有多么困难？你可以成组地增加人吗？你能增加独立贡献的个人吗？

- 容易度量一个时期组织和独立贡献者的工作业绩。
- 把一个目标交给一个团队有多么困难？团队是否能感觉到有足够授权去达成目标？
- 团队内部和团队之间的冲突情况如何？是否促进或阻碍实现公司的使命。
- 组织会促进还是阻碍创新以及市场响应时间？
- 组织的结构会增加还是减少单位产出的成本？
- 工作在组织内部的流转是否容易？

▼ 增加组织的人数可能增加整体产出，但是会降低个人的平均产出率。

▼ 管理是和实现目标相关的，缺乏管理可扩展性注定失败。

▼ 领导是和确定目标、描绘愿景、定义使命相关，缺乏领导对达成可扩展性目标不利。

第 2 章　可扩展性技术组织的角色

孙子说：将弱不严，教道不明，吏卒无常，陈兵纵横，曰乱。

可扩展性和可用性失败的共同原因是责任不清。本章通过回顾角色不清，责任不明的两个真实案例，讨论执行人员的角色、组织的责任以及各种独立贡献者的角色。最后，我们通过引入可以协助定义责任和有助于减少情感冲突的工具来结束本章。

本章适用于任何规模的公司。对大公司，可以作为一个清单来确保厘清与扩展性相关的技术和执行人员的角色和责任。对小公司，帮助开启定义与可扩展性相关角色的过程。对技术新手，可以作为了解技术组织如何工作的入门读物。对于经验老到的技术专家，可以帮助大家回顾组织的结构，以确认可以满足扩展性的需求。对所有的公司，本章清楚地解释了公司里的每个人对扩展性都有自己特定的作用。

2.1　失败的影响

有时，角色和责任不清意味着有些事情没人去做。比如，公司

没有安排任何人或团队去负责系统的容量规划。在这种情况下，系统的容量规划是通过比较系统未来的需要和现有的容量，拟定计划来确保系统有足够的容量来达到甚至超过需要，这包括申请购买服务器、软件调优或者增加持续层的数据存储，比如，数据库或NoSQL 服务器。

在这种情况下，缺乏团队或者个人来负责系统的容量规划，对一个快速增长的公司是灾难性的。然而这种情况引起的失败却经常发生，特别是在那些新公司里面。即使公司指定了负责系统容量规划的人或者组织，往往因为系统过去没有数据积累而无法规划。

“无人负责”问题的另一个极端是有多个组织或人被赋予了同样的目标。请注意，这和“共享一个目标”并不是同一件事情。共享目标是好的，因为“共享”内含合作之意。我们这里描述的是目标有几个主人，而且彼此之间没有清楚地说明谁该去做什么，以达成什么目标，有时候，他们并不知道其他的人或团队在做着相同的事情。在小公司里，这种问题看起来好像有些好笑，因为每个人都知道其他人在做什么。不幸的是，这种问题存在于很多大公司中，而且当它发生时，不仅浪费金钱，而且还破坏股东的价值，同时在组织之间产生长期的怨恨，降低员工的士气。造成团队士气低落的最大的原因，莫过于某个团队对某项任务负全责，因为团队成员以为其他人在负责某个部分而没有去做，结果造成整个任务的失败。多人负责的问题是本书第 1 章所描述的基于情感冲突的核心问题所在。

作为一个案例，我们假设一个组织分成工程和运维两个部门，工程部门负责研发软件，运维部门负责搭建、配置和运行数据库、系统及网络。进一步，我们假设经验不足的 CTO 最近读了一本讨论

共享目标的书，所以决定把平台可扩展性的责任交给两个部门共同承担。公司的容量规划人员确定，要满足明年业务发展的需要，团队必须把客户合同管理子系统的处理能力提高一倍。

工程和运维部门的架构师们读过本书的技术部分，他们决定分割客户合同管理子系统的数据库。这两个部门的架构师们都相信他们已经得到足够的授权进行独立决策，他们没有意识到同样的责任已经被赋予了几个人，并且没有人通知他们去一起工作。工程部门的架构师认为以交易为功能进行分割最有效（这是网站的功能，比如在电子商务平台上买一个产品和看一个产品是两个不同的功能），相反，运维的架构师认为按照客户的边界来分割数据库效果是最好的，这样把不同组别的客户分在不同的数据库里。两组架构师都做好了分割的初步计划，组建好了团队，然后分别请求对方团队来做一些协助工作。

听上去这个例子可能有点儿荒谬可笑，但是却经常发生。在最好的情况下，两个团队会停下来解决纠纷，损失的只是两组架构师的宝贵时间。不幸的是，通常情况下，团队会变得极端化，甚至会在政治斗争上浪费更多的时间。如果把这个项目在一开始就交给单个人或团队来承担，同时听取另一个团队的建议，解决方案的效果可能会更好。

2.2　定义角色

这个部分将以实例来讲解如何通过清楚地定义角色来解决基于责任的冲突。案例讲述了在典型的技术组织结构内，如何定义公司

高管团队和独立贡献者的角色。

实例中提到的高管和独立贡献者，他们的责任并不局限于某个特定的职位或组织。相反其目的是要帮助厘清公司里必要的角色，清楚地定义某些责任或聚焦的专业领域。为了让大多数的读者能够更容易地理解，我们选择定义那些公司里普遍存在的传统角色。例如，运维、基础设施、工程和质量保证（QA）在同一个团队里，每个团队专门负责一个产品线（事实上，我们强烈地推荐这种组织结构，在下一章你将会看到）。

你可能还记得，我们在前面的引言中提到过组织的设计并没有对错之分，任何的组织结构决策都有利有弊。重要的是在组织的设计中要包括全部的责任，不仅要清楚地定义谁是决策者，还要搞清楚谁负责为决策提供信息，决策和行动方案都应该通知谁，谁来负责执行决策。做出最佳决策最为关键的要素是有一个最佳的决策过程，来确保合适的人收集合适的信息，并把它提供给最终的决策者。这适用于公司里任何要做决策的人。我们会在本章的概要部分，用一套有价值的工具对最后这一点进行讨论。尽管我们无法彻底消除冲突，但是合理的组织结构可以帮助我们限制一些因为缺乏清晰度而造成的冲突。作为一个领导者你至少应该清楚在自己的组织内，每个团队的责任和期望的产出。

有关权力下放的注解

在讲组织内责任划分问题之前，我们认为讨论一下权力下放是很重要的。当定义一个组织的角色和责任的时候，你其实就是在设计一幅权力下放的蓝图。广义地说，权力下放就是授权别人

做你该做的事情。比如，由架构师或架构师团队来设计系统，就是你把架构设计工作的权力下放给那个团队。根据公司大小和团队的能力，你或许也会把做决策的权力下放给某个团队。

这里有一点非常重要，那就是你可以下放任何权力，但是必须对其结果承担所有的责任。接受你权力下放的个人或团队最多承担连带的责任，虽然你可以解雇、提升、奖励或惩罚团队，但是必须清楚自己要对最终的结果负全责。好的领导本能地明白这个道理，他们总是把赞扬留给团队，承认失败并公开地承担责任。相反，差的领导在失败时找替罪羊，在成功时抢功。

为了帮助你理解这一点，我们来做个“股东利益”的测试。假设你是一个公司的 CEO，你决定把某个业务单元交给一位总经理负责。当你告诉董事会或者股东，这个业务运营的结果和你无关时，请你想象一下股东们会有什么反应。更进一步，如果业务表现达不到预期，你觉得董事们会不会追究你全部或至少部分的责任？

这并不是说要你自己去做所有的决策。随着公司的成长和团队规模的扩大，你根本就不可能对所有的事情都去做决策。在很多情况下，事实上，你可能没有资格去做决定。例如，一个不懂技术的 CEO 或许不应该去做架构的决策，一个 200 人的工程机构的 CTO 不应该去写最重要的代码；这些高管们的工作经常需要由经理们去完成。现实告诉我们，你必须找到最好的人，然后才有可能把权力下放给他，并对这些人以最高的标准来严格要求。这也意味着你应该问最合适的问题，比如那些最为关键的项目和系统的决策是怎么做出来的？

2.3　执行人员的责任

公司的高管，对在公司中营造一种在第 1 章中提到的可扩展性的文化氛围负有最大的责任。

2.3.1　首席执行官

CEO 是公司里负责扩展性的最高官员。和公司里的其他事务一样，对于扩展性，CEO 是最终的决策者和扩展性的仲裁者。一个技术公司，好的 CEO 需要精通技术，但是这并不意味着他必须是一个技术专家或者主要的技术决策者。

很难想象一个升到 CEO 位置的人，读不懂资产负债表、损益表、现金流表。同样，除非有财务背景或者当过 CFO，否则很难理解财务制度里面的复杂逻辑。CEO 的工作就是提出合适的问题，让合适的人参与，并且协调外部的支持或建议来寻求正确的答案。

同理，在技术世界里，CEO 的工作是了解一些基础知识（相当于前面提到的财务报表），知道该问什么问题，知道去哪里和在什么时间可以得到帮助。在技术的组织机构中，我对没有做过 CTO 或 CIO，没有技术学历或工程师经历的 CEO 及其他的管理者有下面一些建议。

1. 提出问题并从答案中寻找一致性

你的部分工作就是寻找真相，只有真相才能使你做出及时和明智的决策。尽管我们不认为团队会说谎，但是对事实经常会有不同的解读，特别是那些涉及可扩展性的问题。当你对一些事情不理解，或者发现事情看起来不太对劲的时候，就要提出质疑。如果你没有

办法从不同的理解中甄别出事实，那就在答案的一致性上努力。如果你能够战胜自我或骄傲，提出一些看起来容易被忽略掉的问题，那么就会发现不仅学到了很多东西，而且也磨炼出了发现真相的重要技能。

很多成功的领导都具备“高管盘问”（executive interrogation）这个关键的能力。比尔·盖茨版本的盘问技能叫“比尔·盖茨审查”[⊖]。懂得在什么时候去调查、去哪里调查，直至找出满意的答案，这种技能不仅限于 CEO。事实上，经理和独立贡献者也应该尽早磨炼出这个技能。

2. 寻找外部的帮助

在可扩展性方面，要从朋友或专家那里寻求帮助。别指望着请他们进来，然后让他们帮你打理清楚，如果这么做，可能会产生很大的破坏性。相反，我们建议你与技术公司建立专业或个人的关系，依靠这些关系来帮助你提出正确的问题，并且当你要深究的时候，协助你评估答案。

3. 加强对可扩展性的理解

列个清单，把你自己在技术方面的弱点都罗列出来，特别是那些有问题的，然后寻求帮助使自己变得更聪明。可以向你的团队或者外部的专家提问，阅读与公司或产品的扩展性相关的互联网上的文章，参加那些为没有技术背景的人专门举办的研讨会。你可能已经通过阅读专业书籍，特别是那些普通技术类的书籍来解决这个问题。你没必要去学习程序语言、理解操作系统、搞清楚数据库是怎

⊖ 参见 Joel Spolsky.“My First BillG Review.”http://www.joelonsoftware.com/items/2006/ 06/16.html.

么工作的或者理解如何实现“并发的碰撞检测”。相反，你需要的是能够更好地提出和评估问题。扩展性是一个业务问题，但是要解决就必须熟悉相关的技术情况。很有可能，CEO 会把权力下放给团队里的几个人，包括 CFO、独立业务部门的负责人（总经理）和负责工程技术的高管（本书指 CTO 或者 CIO）。

2.3.2　首席财务官

一般来说，CEO 会把预算权下放给 CFO。系统容量规划是成功的可扩展系统中非常大的部分，前面的例子已经提过系统容量规划不力所带来的影响，系统容量规划的结果一般都会通知预算部门。预算办公室责任中的一个关键任务，就是要确保团队和公司能有足够的资金来扩展平台、产品和系统。要有足够的预算，公司才能通过增加或者减少服务器，雇佣更多称职的工程师和运维人员来扩展系统达到预期的需求。在公司的业务还没有那么大规模的情况下，不应该过度预算盲目地扩大规模，那样会稀释公司短期的净收入，收效甚微。适时地采购和部署系统及解决方案会优化公司的净收入和现金流。

CFO 也不太可能有技术背景，但是可以像 CEO 一样，通过构建适合的网络，提出正确合理的问题。CFO 可能需要问清楚，在制订预算的过程中是否考虑过其他的可扩展性方案，在确定预算方案的时候，做过什么样的权衡安排？其目的是要确保团队考虑得多一些。不太好的答案是“这是唯一可能的预算”，其实这不太可能，只有一种可能的事情极少。例如，如果公司面临降低成本的挑战，CFO 就要决定旧服务器和网络设备是否能继续使用，因为折旧会对

账面有较大的影响。然而事实上，旧设备通常效率较低而且容易出故障，会造成整体持有成本的大幅上升。比较好的回答是“我们评估了所有的可能性，这个方案用相对较低的成本来水平扩展，为未来的扩展搭好一个框架，使我们可以持续扩展。”

2.3.3　业务部门的负责人、总经理、产品线负责人

负责公司或部门盈亏的人，如业务部门的负责人等，要对平台、产品和系统的业务增长做好预测。在中小公司里，业务部门的负责人很有可能是 CEO，其权力和责任会下放给一些员工。然而，需求预测对确定扩展计划至关重要，这可以避免在公司的实际业务没有发生之前，安排过大的预算。

需求指的是系统承受用户并发请求的数量，我们经常会碰到业务负责人无法对需求进行预测的情况，这是绝对不能容忍的不负责任的行为。如果没有专人根据业务的发展进行需求预测，那么对可扩展性预测的责任就会被转移到技术部门，它们对业务需求的预测能力远没有业务部门的强。预测或许不准，特别是在公司发展的初期，但是开启这个过程至关重要，预测的准确度会随着时间的推移而逐渐成熟。最后，和公司里的其他高管一样，业务负责人有义务帮助营造可扩展性的文化氛围。高管要在技术部门里向同级或者下级提出合适的问题，确保那些技术伙伴们能够得到足够的资金，有效地协助有扩展性问题的业务部门。

2.3.4　首席技术官 / 首席信息官

CEO 是公司负责可扩展的最高官员。首席技术官主要负责技

术、流程和组织的可扩展性。对以技术为主要产品的公司，特别是互联网公司，常常把这位高管称为 CTO。在对产品的研发和交付中只负责技术支持作用的公司里，常常称其为首席信息官 CIO。如果公司里只有一位 CTO，那么这个人一般就聚焦在产品的科技上。本书混用 CTO 和 CIO 这两个词来指公司的技术高管。这位高管很可能有最强的背景和能力，在业务发展需要的时候，可以确保平台和系统有效地扩展。

对平台可扩展性的成功，CEO 责无旁贷，CTO 从 CEO 那里接过并与 CEO 共同分担这份责任。如果平台无法扩展将会导致 CTO 和部分组织，甚至 CEO 被解职。

CTO 或 CIO 必须要有公司的整体技术愿景，而且这个愿景要包括可扩展性。CTO 除了负责设定愿景中积极的、可度量的、可达成的目标外，同时还负责为团队配置合适的人员以确保其能完成可扩展性的相关使命。CTO/CIO 的责任还包括发展可扩展性的企业文化和过程，以确保公司走在用户需求的前面。

当公司的规模扩大的时候，CTO 或者 CIO 需要把某些有关可扩展性决策的责任下放。当然，如前所述，权力下放也绝不会解除高管确保按时、按预算做好可扩展性工作的责任。另外，在高速发展的公司里，可扩展性对公司的生死存亡至关重要，CTO 绝不应该把制订可扩展性发展愿景的权力下放。在这件事上，没有什么比“身先士卒”更为重要，而且制订愿景并不需要专深的技术。尽管我们所见到的最好的 CTO 有从独立贡献者到系统分析员或项目经理等不同的技术背景，但是我们也曾经见到过没有技术背景的成功 CTO。对没有技术背景的 CTO 或者 CIO 而言，他们必须要有技术的敏锐

性，同时具备较好的语言能力，了解技术领域内诸如时间、成本和质量之间的重要关系平衡。让一个初出茅庐的 CTO 来领导技术团队，就像让一个不会游泳的人从船上跳到湖里一样；假如那个人会游泳，你可能很欣慰，但更大的可能是，你要给自己再找个新的伙伴一起划船了。对初来乍到的 CTO 同等重要的是赢得技术人员的信任，没有手下的信任，CTO 无法有效地领导。

同样重要的是，CTO 要有一些业务感觉，但不幸的是，这和找一个有电气工程博士学位的首席市场官一样难，这类人存在，但是很难找到。大多数的科技工作者，他们在本科和研究生课程中，不学习商业、财务或者市场方面的知识。尽管 CTO 不必是一位资本市场的专家（那是 CFO 的工作），最好还是需要理解公司业务运作的基本情况。例如，CTO 应该能分析和理解损益表、资产负债表和现金流表之间的关系。在一个像互联网公司这样的以技术为中心的公司，CTO 通常拥有全公司最大的预算。这样一位高管经常负担着非常大的投资任务，购买电信运营商的服务、数据中心的租约、软件的许可权，所以缺乏对财务计划的了解会给公司带来很大的风险，如挪用资金、多付给供应商、无计划、没有预见的支出（定期软件许可费用的调整）。CTO 应当有市场营销的基础知识，至少读过社区学院或公司资助的专门课程。这并不是要求 CTO 是这些领域的专家，而是希望 CTO 对这些方面要有基本的了解，从而为可扩展性做好商业安排，同时可以有效地在商业界沟通。本书稍后会讨论这些方面的内容。

2.4　独立贡献者的责任

这里会对在大多数公司成长和扩展时需要独立贡献者所扮演的角色进行描述。这些角色包括总体架构、软件工程、系统运维、基础工程和质量管理。

我们的目标不是要严格定义组织或功能的边界。如前所述，通过功能组织是设计组织的一种方法。当应用瀑布方法开发的时候，尤为有效。我们将会在第 3 章中讨论，当公司聚焦在产品开发的时候，我们希望组织与公司的产品生产与开发的过程相匹配。

2.4.1　架构师的责任

架构师的责任是确保系统的设计和架构可以随着业务的发展而扩展。这里我们清楚地指出设计和实施之间是有差别的。架构师需要在业务需要发生之前就想好，远在业务部门的预测超过平台的容量之前，就已经对如何扩展系统深思熟虑了。例如，架构师可能已经开发了一个可扩展的数据存取层（DAL）或数据存取目标（DAO），当用户需求在任何一个方面增加的时候，允许通过不同的数据结构，从多个物理数据库存取数据。真正的实施或许只是一个数据库，通过对 DAL 或者 DAO 做的一些修改及迁移脚本的开发来提高效率。当需求发生时，用几周而不是几个月的时间，更多的数据库就可以被部署到生产环境。架构师也负责制定代码设计和系统实施的技术标准。

架构师负责设计系统并确保其设计能解决任何的扩展问题。在第二部分中我们会介绍一个架构团队应该采纳的关键过程，它可以

帮助架构师定位跨越所有技术领域的扩展性问题。

架构师也可以负责信息技术的管制、标准和过程，通过第 13 章讨论的架构审查委员会（ARB）和联合架构设计（JAD）执行这些标准。首席技术官会要求架构师履行他们的这些职责。有一些大的公司可能会组织流程 – 工程联合团队，负责对流程的定义和标准的执行。

公司给予那些聚焦在技术架构设计的独立贡献者不同的职衔，包括软件架构师、系统架构师、扩展性架构师、企业架构师和敏捷架构师。不管什么职衔，这些角色基本上都会聚焦在一两个领域。首先是软件架构师，主要关注如何设计和架构软件。这些架构师聚焦在不同的领域，例如，面向服务的架构框架或者代码模板为研发人员提供指导。其次是系统架构师，主要负责解决软件在硬件配置和支持方面的问题。这些架构师聚焦在去除单一的失败点，发现出错点和做好容量规划方面。从客户的角度看，系统的可扩展性是和系统架构最为相关的。本节的后半部分，会更加深入地讨论有关独立贡献者这一重要的角色。

2.4.2　工程师的责任

工程师是战斗在第一线的，真枪实弹的基层人员。工程师是可扩展性使命的首席实施官和系统的首席调优官。工程师遵循公司的架构标准，根据架构进行具体设计，并且最后完成代码的实现。工程师团队是最有可能真正了解系统局限性的仅有的几个团队，他们也是发现未来可用性问题的关键人员。

2.4.3　DevOps 的责任

DevOps 是“Development”（开发）和“Operations”（运维）两个词的合并和缩写，近来常用这个词描述软件研发团队和技术运维团队之间交互和合作的现象。这个词的出现代表着多年来大家所熟知的事实得到了认可，就是系统和服务既需要软件也需要硬件。由此软件研发和系统管理混合成 DevOps。

在 SaaS 和 Web 2.0 时代，DevOps 通常负责配置、运行和监控生产系统。这个团队也应该是本书后续将要讨论的敏捷开发的一部分。即使不想采用敏捷的组织结构或开发策略，至少也应该考虑让 DevOps 的人存在于工程团队中，以帮助研发人员更好地理解如何把研发好的应用配置到生产系统。DevOps 的人员既知道系统日常运行的情况，也了解系统资源的使用情况。只有他们有足够资格发现系统的瓶颈，并且在系统设计的时候时刻想着系统配置。

通常，DevOps 的人员负责研发和测试环境，包括研发和配置脚本、监控、日志和其他系统工具。DevOps 的人员负责输出报表展示一个阶段可用性的发展趋势，分析出问题的根源并给予纠正，确定各类问题的平均解决时间和平均恢复时间。

不管团队的构成情况怎么样，都是 DevOps 负责监控、报告应用与系统的健康情况和服务质量，在解决可用性问题的时候起着关键性的作用。这个部门所采用的管理问题和解决问题的流程会成为其他流程的入口信息，以帮助在系统发生大规模灾难前，确定扩展性问题之所在。运维所收集的数据，对进行系统容量规划、解决系统性及反复发生的可扩展性的问题，有令人难以置信的宝贵价值。架构和工程团队严重依赖 DevOps 来帮他们确定在什么时间解决什

么问题。在第二部分第 8 章和第 13 章我们会详细讨论这些流程。

2.4.4　基础设施工程师的责任

负责基础设施的工程师技能独特，在敏捷团队中甚至不必每天露面，但是却横跨很多个敏捷团队。有些大的企业，为了研发端对端的解决方案，把这部分事情放在敏捷团队中。但是在大多数的中小企业中，这部分人集中在基础设施部门中来支撑多个工程团队。这样的团队包括 DBA 数据库工程师、网络工程师、系统工程师和存储工程师。常常由他们来确定使用什么系统，什么时候购买，什么时候报废。不论基础设施人员团队的大小，其主要责任都包括设计共性资源的架构（像网络和传送系统），定义全局的存储架构，确定关系型或非关系型数据库的解决方案。

对使用云的公司来说，基础设施团队经常负责管理虚拟服务器、网络和信息安全。基础设施工程师也有助于寻找系统、网络和数据库的容量限制点，支持和帮助其他团队确定解决可扩展性相关问题的合适方法。

2.4.5　质量保证工程师的责任

在理想情况下，质量保证工程师主要是指那些负责应用测试，确保测试结果和公司期望的结果一致的工程师，在可扩展性测试过程中也同样起着重要的作用。新产品和功能会改变系统、平台或者产品的需求特性，最常见的是增加新功能，从而产生对系统资源的额外需求。最为理想的做法是我们掌握新产品的需求的特点，确保新功能和新应用不会给生产环境带来大的冲击。质量保证的专业人

员需要了解和掌握其他的变更情况，以确保应用能够及时通过可扩展性相关的测试。质量保证一定要聚焦自动化的测试，而不是手工测试。不仅仅生产系统要可以扩展，而且当添加新功能的时候，测试的过程也要可以高效扩展。

2.4.6　系统容量规划师的责任

有系统容量规划责任的人可以在任何团队工作，但是他们需要能够取得最新的系统、产品和平台的性能数据，系统容量规划是高效扩展和成本控制的关键。如果做得好，对那些很容易做水平扩展的系统可以适时地买进设备，对无法进行水平扩展的系统可以安排紧急购买大型设备，清查系统的可扩展性问题，分优先级逐步予以解决。

你或许注意到了，我们在描述采购大型系统的时候用了紧急两个字，很多的公司把这种按比例放大的方法当成有效的策略。正如我们将从第 19 章到第 23 章要讨论的那样，如果你的扩展策略是依靠更快和更大的硬件设备，那么该解决方案将不具备可扩展性，你的可扩展性取决于供应商。如果说换更大更快的硬件能够扩展系统，那么购买更大更快的车是否可以使你跑得更快？在还没有赚那么多钱之前，你只能和收入水平类似的人开得一样快。可扩展能力与更大更快的系统以及应用服务器的新版本没有关系。

缺乏角色和技能会怎么样

eBay 的故事

你是否曾想过，如果请一位木匠来家里修理水管，会是怎么

样的一种情况？马斯洛的锤子理论预见了这个假设的结局，很有可能，木匠会拎着锤子来完成工作。你很可能以前就听说过马斯洛的锤子理论，通常和另一句一起出现，“如果你手里只有一把锤子，那么所有你看到的都是钉子”。

故事得从 2001 年我们的合伙人汤姆·科文加入 eBay 说起。第一天，汤姆作为负责基础设施的副总裁，走进了运维中心，随便指着挂在墙上显示数据的一些大屏幕问道：“这些是做什么用的？”

一位运维操作员转过身来，看了一下汤姆，然后转回身继续做他的事情。“那是我们的控制器”，运维操作员回答道。

汤姆又问道：“什么是控制器？”

“接受来自于用户的请求，然后分配到网站的服务器上”，运维操作员回答道。

“哦，是负载均衡设备”，汤姆说。

“我不知道负载均衡是什么玩意”，运维操作员说。

这件事看起来有点儿怪，但却是对 2001 年发生的事件的准确描述。对话的细节或许无意中有些许改变，但是对话的要点确实相差不多。在那个时候，网上交易才只有几年的历史，并不是每个人都能够理解系统基础设施方面的基本概念，当然这在今天已经是司空见惯的事情了。

汤姆所指的以硬件为基础的负载均衡器，当时（2001 年）已经在市场上出现了好几年的时间。这些设备不仅仅用在网站上，也用在各类 CS 架构的系统上，服务于公司的内部员工。我们来讨论一下这个故事说明的问题。

汤姆决定寻找 eBay 当时使用的更多的“控制器”，结果发现这些设备已经使用了好几年，在负载均衡之前，eBay 使用 DNS（域名服务）轮询的技术来控制网络的流量。（DNS 轮询技术我们至今仍然还能偶尔在一些客户那里看到，这可不是开玩笑。）此外，原来负责选择解决方案的人，是没有多少基础设施经验的软件工程师。

汤姆对软件和硬件负载均衡都有所了解。例如，他知道硬件负载均衡每秒能处理更多的请求，而且比它的软件表弟（“控制器”）的故障率更低。因为软件负载均衡是按每台设备或每张许可证收取的，所以成本更高，不过如果把吞吐量也考虑在内，每笔交易的费用一般会比硬件的低。如果再把整体可用性（较低的设备故障率）考虑进去，那么在选择负载均衡解决方案的时候，硬件负载均衡器明显胜出。

汤姆经常表现出他对基础设施架构的奇妙偏好，其实他的洞察力并没有什么神奇魔法。这些洞察力来源于 20 年来在基础设施、运维和解决方案架构的生产第一线。积累起来的知识实属弥足珍贵，来之不易。汤姆重新审查了与软件负载均衡相关的事件以及许可证的费用，结果很快发现，如果把负载均衡换成硬件，不仅可以提高可用性，而且还能降低成本。

这里所学到的是 eBay 缺乏关键的技能，即深而广的基础设施经验和知识。这并不是说 eBay 在汤姆加入前没有特别好的基础设施人才，实际上，eBay 确实有这类人才。问题是在做关键的负载均衡决策的时候考虑的范围不够广泛，没让这类人才有机会参与决策，以避免错误。对 eBay 平台系统容量的管理乏人问津，显

而易见，这是个可扩展性相关的错误。

汤姆的决策最后稍微降低了 eBay 每笔交易的成本，提高了系统可用性。尽管结果并不是惊天动地，但是股东（成本降低）和客户（可用性提高）最终都受益了。这正如我们说的，积少成多，集腋成裘。

2.5　RASCI 工具

我们有许多客户在使用一种简单的工具，来清楚地定义项目中的角色。每当我们进入一间需要做可扩展性项目的公司，我们就用该工具来定义谁该干什么，去除浪费的资源，确保全面覆盖扩展性相关的所有需求。尽管这只是一个过程，但因为本章是论述角色和责任的，我们觉得很有必要介绍这个工具。

我们用的这个工具叫 RASCI，是一套用来确定责任的表格，RASCI 是指负责、批准、支持、咨询和知情。

R：负责（Responsible）对项目或者任务的完成负责的人。

A：批准（Accountable）项目关键决策的批准人。

S：支持（Supportive）为项目完成提供资源的人。

C：咨询（Consulted）为项目提供数据或者信息的人。

I：知情（Informed）需要了解项目相关情况的人。

RASCI 可以用在矩阵当中，每个活动或者任务标在 Y 轴上，每个独立贡献者或者组织的名字标在 X 轴上。活动（Y 轴）和组织（X 轴）的交叉将会有（R、A、S、C、I）中的一个字母，如果交叉

处什么都没有，那么相关的独立贡献者或者组织就不是这个任务的一部分。

在理想情况下，一个任务会有一个 R 和一个 A。这个工具可以帮我们解决本章前面提到的那个问题，多个组织或个人认为他们对任何工作负责。这样可以明确只有一个人或者组织对任务的成败负责，坚守“职责清楚、奖惩分明”的原则。一种更为温和的说法是分给几个人负责的项目等于没有人负责。

这并不是说其他的人不允许为项目或者任务提供建议。RASCI 模型明确允许并执行对顾问、公司内部或外部可以为任务增加价值的人的使用。A 不应该批准 R 的方案，直至 R 已经就方案的正确性咨询了所有相关的人。当然，如果公司的文化好，R 不仅可以寻求人们的帮助，而且会让那些被咨询的人感觉到自己有价值，其价值已经被考虑到决策支持的过程中。

只要你愿意加，觉得有价值，或者对完成项目来说是必需的，你可以增加多个 C、S 和 I，但是，同时要注意不要过度知会。记住本书前面提到的关于陷入邮件和沟通泥沼中的事例。新的公司往往假设决策应该让每个人都参与或者知会，这种信息的发布机制是没有可扩展性的，结果是大家花时间读邮件，而不是去做他们应该做的，能为股东产生价值的事情。

表 2-1 是一个部分完成了的 RASCI 矩阵样例。根据我们关于不同角色的讨论，让我们来看看如何完成这个 RASCI 矩阵。

早些时候，我们曾指出，CEO 必须对可扩展的文化氛围负责，这是公司可扩展的 DNA。从理论上说，尽管让 CEO 把责任下放给公司的其他人，在客观上是可行的，就像你在关于领导的一章中读

到的，这个高管必须践行公司和平台可扩展文化的价值。因为我们在讨论公司如何行动才能实现可扩展性，即使 CEO 授权，也还是必须要对可扩展性负责。因此，我们把 R 放在 CEO 列和可扩展性任务行。很明显，CEO 对董事会负责，因为营造可扩展性的文化氛围和整体的文化氛围有关，所以我们把董事会标成 A。

表 2-1　RASCI 矩阵

	CEO	业务经理	CTO	CFO	架构师	工程师	运维工程师	信息安全	质量保证工程师	董事
扩展性文化	R									A
扩展性的技术愿景	A	C	R	C	S	S	S	S	S	I
产品扩展设计			A		R					
软件扩展实施			A			R	S			
硬件扩展实施			A				S	R		
数据库扩展实施			A				S	R		
扩展性验证			A						R	

对可扩展文化任务而言，谁是 S？应该知会谁？应该咨询谁？在寻找答案的过程中，任何情况都允许有支持者 S 存在，也可以在寻找解决方案的时候有顾问 C 参与。因为 C 和 S 的参与所以才会有结果，因此一般上没有必要包括 I，那个需要沟通决策和执行结果的人。

我们填了可扩展性的技术愿景一行。如前所指，CTO 负责制订产品、平台和系统可扩展性的愿景。CTO 的老板是 CEO，所以 CEO 要负责审批决策和路线。请注意，在决策中 R 的老板不一定非得是 A。R 完全可能代替某个和他不相干的人去完成某个任务。在这种情况下，假如 CTO 为 CEO 工作，那么由 CEO 以外的人去批准可扩展性愿景或者计划的可能性极小。

CTO 可扩展性愿景的顾问包括那些需要依赖 CTO 来实现生产系统可用性或公司运营后台系统的人。这些人需要被咨询，原因是 CTO 搭建和运维的系统是业务部门的生命线，后台系统是 CFO 工作必须依赖的心脏。

我们曾经指出 CTO 的组织（架构、工程、运维、基础服务和质量保证团队）全部都是愿景的支持者，其中一个或几个部门也可能是咨询者。CTO 的技术背景越差，就越需要依赖其团队来制订可扩展性的愿景。在表 2-1 中，我们假设 CTO 有很丰富的技术经验，但是实际情况并不总是这样。CTO 可能也需要从外部引入资源协助，以确定可扩展性的愿景和计划。这种外部的帮助可以是顾问服务公司，或者技术顾问和管制委员会，他们提供与公司董事会一样的技术管制和监督建议。

最后指出需要把可扩展性愿景知会董事会。这或许是董事会的一个会议纪要，或者是围绕着现有平台能支撑什么规模的业务所做的讨论，公司需要什么样的投入来满足来年可扩展性的目标。

矩阵的部分空白已经填好了。和矩阵相关的一个要点是我们已经把任务分割以避免 R 部分的重叠。例如，基础设施团队的责任已经从软件开发和架构设计团队中分割出来。这样保证责任清楚，符

合奖罚分明、职责清晰的思路。然而，这样做的结果是组织向纵向条块化发展，与我们长期的发展方向不符。你要把组织调整成这样吗？要设计出最好的解决方案，就需要不同的团队在一起工作，这一点很重要。矩阵型组织的团队，不但可以避免团队内部存在的围绕功能或组织的责任而产生的独立心态，而且可以从 RASCI 中获益。既要有单一责任的组织，又要确保合作。通过 C 特性的使用来落实 RASCI。

建议你多花点儿时间来完成表 2-1 中的余下部分，直到掌握 RASCI 模型为止。这是一个非常有效的工具，它可以清楚地定义角色和责任，帮助我们去掉重复的工作，减少会引起士气低落的不幸的争斗和发现失踪的任务。

有关角色和责任的最后一个注解：不应该有任何一个团队，把角色和责任作为无法完成工作的障碍。公司里的每个人的主要责任都是为客户和股东创造价值。当然会有这种可能，当员工履行职责时，完成的任务超过了定义好的责任范围。如果能够帮助公司完成使命，那么员工可以，也应该自愿跨越边界去完成工作。重要的是，当这种情况发生时，他们应当和领导一起去找出到底无人负责的地带在哪里，领导应该承诺在未来纠正这些问题。

2.6　结论

定义清晰的角色是领导和经理的责任，不管是独立贡献者还是组织都需要角色的清晰性。本章通过一些案例讲解了怎样清楚地定义角色从而帮助组织实现高可用性的使命。这些例子中提到的组织

是众多组织结构中的一部分，根据个人、组织和他们的角色可以产生很多不同的结构。你的组织现实的解决方案或许和这些结构相比差别很大，角色的定义要和公司的文化和需要一致。当你加强角色清晰度的时候，要注意避免责任的重叠，这会造成无效的努力和价值损毁的冲突。在公司内部，我们通过清晰的角色来确保可扩展性，当然也可以应用在公司业务的其他方面。

我们引进了一套叫做 RASCI 的工具，来协助公司内部的组织定义清晰的角色和责任。你也可以尽情地使用 RASCI 工具，定义自己组织的角色和项目内部的角色。RASCI 的应用可以去除重复的工作，使你的组织更加有效、更有力、可扩展性更好。

关键点

- ▼ 角色清晰对可扩展任务的成功极为关键。
- ▼ 责任重叠会造成资源浪费和价值损毁的冲突。
- ▼ 无人负责形成真空地带无法完成扩展的任务。
- ▼ CEO 是公司里负责扩展性的最高长官。
- ▼ CTO 或者 CIO 是公司里负责技术扩展的最高长官。
- ▼ RASCI 是一个可以帮助减少责任重叠，产生清晰角色的工具，RASCI 以矩阵方式出现。
  - R 代表负责，指决定要做什么事的人，而且负责任务的实施。
  - A 代表批准，指在决策过程中批准任务并验收任务结果的人。
  - S 代表支持，指为完成任务而提供服务的任何人。
  - C 代表咨询，指在决策前和关于任务完成情况接受咨询的人。
  - I 代表知情，指需要通知决策和任务执行结果的人。

第 3 章　组织的设置

孙子说：将凡治众如治寡，分数是也。

本章将讨论组织结构的重要性及对扩展性的影响。我们将讨论组织的两个关键属性：规模和结构。这两个属性将对我们掌握组织的运作机制，以及当团队调整时，哪些地方容易出问题大有益处。

3.1　组织对可扩展性的影响

组织中一些最重要的因素会影响沟通、效率、质量、标准和责任。让我们来分析一下组织是如何影响这些因素的，以及为什么这些因素对扩展性很重要。通过这些分析，建立组织和可扩展性之间的因果关系。

沟通和协调，对任何需要多人努力才能完成的任务来说都是必要的。如果对架构设计的沟通失败、对产生故障的原因和结果沟通失败、对客户投诉的来源沟通失败、对生产上线后产品的预期值变化沟通失败，所有这些失败的后果都有可能是灾难性的。想象一下，在一个 50 人的团队里，如果责任或者组织结构层次不清晰，同事间

很可能不清楚彼此都在做什么。如果不需要去协调变更，那么这种情况还过得去。在无组织的情况下，有问题该去问谁？该如何协调变更？怎么知道你的变更是否和别人的冲突？在大多数情况下，这种沟通不畅会导致一些小麻烦，比如你要给每个人发封邮件去寻找答案。可是时间久了，工程师们都学乖了，他们开始忽略这些邮件。忽略邮件就意味着无法找出答案，更糟糕的是，变更的代码无法合并到代码库，导致关键功能无法发挥作用。错误的根源到底是因为工程师不是一个超级明星，还是因为组织的设置导致无法清楚地沟通和有效地合作？

当组织有利于工作的时候，效率就会得到提高；反之，当出现不必要的结构层次且需要大量交流才能完成工作的时候，效率就会降低。在敏捷开发的过程中，产品负责人往往和工程师坐在一起，确保高效而快速地回答有关产品的问题。如果研发工程师需要澄清产品的功能点，那么他有两种选择，一是猜测该怎么做，二是等产品经理回答。在等待的这段时间里，要不就去做另外一个项目，要不就去做些无价值的事情，比如玩游戏。大量的等待时间可能会造成团队无法完成承诺的任务，把一些不必要的事情带入到未来的迭代中去，所以会延迟或者降低潜在的投资回报。

安排产品负责任人和工程师坐在一起可以使有关的问题很快得到解答从而提高效率。坐在一起是一把双刃剑，它也可能降低效率。首先，完成项目的成本提高了。其次，当资源紧张的时候，资源向面向客户的短期项目倾斜，结果牺牲了长期的稳定性项目。季度目标可能会在短期内实现，但是因为缺乏资源而造成的技术负债增加，结果造成系统故障而中断服务，搁浅新产品的

上线计划。

可以通过标准化来提高组织的效率。一个不注重代码、文档、规范和配置标准的制订、发布和应用的组织，研发效率和质量必然低下，生产中出现严重问题的风险很大。要很好地理解这一点，我们可以考虑一个完全矩阵化的组织，该组织只有少数几个工程师与产品经理、项目经理和业务负责人在一起工作。如果不采用共同的标准，那么团队很容易在什么是最佳实践的问题上出现严重分歧。因为研发团队追求能有更大的产出，所以不知不觉地忽略了要按照标准注释代码，但这是以牺牲代码未来的可维护性为代价的。为了避免这些问题，大的机构会帮助工程师理解发布指南、原则和共同规范的价值。

举另一个例子：假如你的公司有一条架构原则，任何一个服务都必须部署和运行在多个服务器上。有个团队无视这个标准，发布了一个既不能水平扩展，可用性又令人难以忍受的解决方案。你认为这样的事不会发生？仔细想一下，你就会发现我们的团队，一直在犯这样的错误。支持这一偏离原则行为的最常见的借口是，这个服务对系统无关紧要。如果真的是这样，那么为什么要浪费时间来研发它？如此说来，团队没能有效地使用研发资源。

如前所述，一个团队如果不能养成遵守规范和标准的习惯，本质上是组织容忍了研发质量标准的降低。举个简单的例子，某组织有一个可靠的单元测试框架和流程，但是由于严重地依赖于团队的组成和规模，以至于无法实际应用。如果一个团队漠视上级组织关于所有功能都必须进行单元测试的要求，很可能，这一决策将导致代码质量降低，增加数个大大小小的缺陷。高缺陷率又进一步导致

系统服务中断，引起可用性相关的问题。代码错误和生产系统问题的增加，会削弱本应该投入在新功能编码，或者像分库分表这样的扩展性项目上的工程资源。当资源稀缺时，很难通过延迟短期的面向客户的功能来换取长期的可扩展性任务。

长角

内部代码长角（Longhorn），对外称为 Vista 的微软操作系统，就是在一个机构中典型的标准和质量失控的例子。虽然最终 Vista 成功地发布，成为排名第二被广泛使用的操作系统，但它还是成了微软及其客户的痛。Vista 和前一代操作系统 XP，成为该公司历史上发布间隔最长的产品。

在 2005 年 9 月 23 日，《华尔街日报》的头条新闻上，微软联合总裁吉姆·阿尔钦承认他曾经告诉过比尔·盖茨“长角项目不会成功”。阿尔钦把研发描述成“飞机撞地坠毁”，原因是功能的引入和集成杂乱无章，毫无计划。⊖

微软征召了高级执行人员阿米塔·斯里瓦斯塔瓦来拯救濒临末日的产品。斯里瓦斯塔瓦安排架构师团队来统筹规划操作系统，制订了研发流程以确保高水平的代码质量。尽管这一改变招致很多研发人员的批评，却挽回了失败的产品。

归属权也会影响系统的可扩展性和可用性。当很多人在同一套代码上做研发的时候，如果代码各部分的归属权不清楚，那么就没有人会感觉到自己对此有责任。当这种情况发生的时候，没有人愿

⊖ Robert A Guth.“Battling Google, Microsoft Changes How It Builds Software.”*Wall Street Journal*, September 23, 2005. http://online.wsj.com.

意多做一步以确保其他人遵循标准，开发需要的功能，同时维持预期的高质量。这样我们就看到了前面曾经提到的应用的可扩展性问题，这些问题来源于低效率地使用工程资源，产生越来越多生产系统的问题和不流畅的沟通。

从扩展性的角度来观察，我们所关心的组织已经有了一个清晰的基础，现在，是时候来理解有关组织的两个基本决定因素，即规模和结构。

3.2　团队规模

试想有一个两个人的团队，两个人都了解彼此的怪癖，总是知道对方在干什么，永远也不会忘记相互沟通。听起来很完美吧？假设他们没有足够的研发力量按时完成像分割数据库这类的可扩展性项目；无法灵活地把任务交给另一个团队，因为每个人可能都有另一位所不掌握的知识；他们可能有自己的代码标准，和其他两人团队的标准不同。很明显，团队的大小各有利弊。关键是平衡团队的规模为组织寻找最优的方案。

寻找最优方案的重点在于为组织寻找最佳的团队规模，并不存在一个适合所有团队的魔术般的数字。当为组织确定最佳规模的时候，要考虑很多因素。如果硬要给出一个关于最佳团队规模的直接答案，我们希望能给出一个范围以满足一些特定的需要。即使范围再广，也总会有些意外。团队规模的下限是 6 人，上限是 15 人。下限意味着团队不少于 6 个工程师，因为如果少于 6 人，就没有必要把他们分入一个单独的团队。上限指的是一个团队不要超过 15 人，

15 人的规模开始阻碍经理的管理能力，团队成员之间的沟通能力也开始出现问题。上面给出的团队规模范围总会有些意外，重要的是，当你对标自己的组织、人员和目标的时候，考虑下面这些因素。可能还记得第 1 章中的案例，亚马逊把上限叫做“两张比萨饼准则”，即任何一个团队的规模不能大过两张比萨饼所能喂饱的人数。

当确定团队规模的时候，首先要考虑的因素是经理的经验。本章后半部分会专门对经理的责任进行讨论，对目前的讨论，我们先假设经理来自于一线的基层，他们的责任包括三个方面：确保工程师通过自我管理或经理指导，能够在有价值的项目上高产出；确保传达人力资源类似工资、福利或者分配方面的信息，并完成日常的行政管理工作；确保掌握目前进行中的项目和问题的状态，在经理的同级之间以及向上级管理层同步项目的信息。

一个刚从工程师职位提拔的初级经理或许会发现，即使管理仅有 6 个工程师的小团队，行政和项目管理任务也会用掉一整天，很少有时间来处理其他的事情，因为对他而言这些任务很新，相比级别更高的经理，需要很多的时间和精力来处理。比起反复做过多遍的工作，新任务通常要求花费较多的时间和精力。因此，人的经验是确定最优团队规模的一个关键因素。

第二个要考虑的因素是团队的任职时间。任职时间长、经验丰富的团队，并不需要太多的管理，内部沟通的成本也较低。在公司时间的长短和经验多寡对工程团队尤为重要。在职时间长的员工通常需要较少的行政管理（例如，签医保合同、改正工资单中的错误、为某个任务寻找帮助）。与此类似，有经验的工程师不太需要额外的协助来弄清楚规范、设计、标准、框架或者技术问题。

当然，个体之间存在差别，必须考虑团队整体的经验程度。如果一个团队高、中、低搭配平衡，那么中等规模的团队也可以有效地运转。通过比较发现，如果一个团队都是高级工程师，比如在基础设施项目上，很可能团队的规模再扩大一倍也不是问题，因为沟通的成本比较低，而且不会为那些普通的工程任务分散精力。当确定团队的规模时，应当考虑所有这些问题，厘清团队规模要多大才能保持高效率，不至于因为规模太大，负担过多而影响生产率。

如前所述，每个公司对经理应该负责的任务有不同的期望。基层经理的管理职责包括：

- ▼ 确保工程师们在有价值的项目上产出高
- ▼ 确保完成行政管理工作
- ▼ 确保掌握项目和问题的进展状态

显然，我们会给经理们安排很多的管理职责，这包括每周与工程师 1 对 1 的会谈，自己写代码实现一些功能，审阅规范，管理项目，审查设计，协调别人或者亲自进行代码审查，制订标准和确保遵循标准，指导新人，表扬和鼓励好的行为，总结个人表现和业绩。经理处理的任务越多，团队就要越小，以确保经理可以完成所有安排的任务。例如，对 10 个工程师的团队，如果需要为每个工程师每周安排一次 1 对 1 的一小时会谈，就会占用经理 40 小时工作时间的 25%。虽然这个数字可以调整，比如会谈短些，工作时间长些，但是道理是一样的。对一个大团队来说，和每个工程师对话，一定会占用管理者大部分的时间。作为一个管理者和领导，需要不停地和团队中的每个人沟通。显然，当确定组织的最佳团队规模时，任务的数量和完成这些任务所需要的工作量是两个主要因素。对管理层

来说，调查一线经理每周在每个任务上所花费的时间是一个有趣而且具有启发作用的事情。一个经理每周的时间很容易被那些看起来十分快的任务所填满，比如，前面提到的 1 对 1 会谈。

前面提到的三个因素，即管理经验、团队在职时间和经理的责任，都是限制团队规模的约束条件。限制团队规模的目的是减少经理的负担，使团队创造价值的时间最大化。

与前面的三个因素不同，最后一个因素是业务的需要，其目的是加大团队的规模。业务负责人和产品经理想总是想要增加收入、击败竞争对手并增加客户数量。为此，他们经常需要研发更多、更复杂的功能。保持团队规模小的一个主要问题是大的项目需要非常多的研发迭代时间。其结果是项目需要更长的时间才能交付给客户。第二个问题是增加工程师的数量需要相应地增加支持人员的数量，同时增加管理人员。把工程类的经理称为支持人员，或许会冒犯他们，事实上，这正是管理所应该做的事情，即支持团队完成项目。团队越大，每个工程师所需要的经理数量就越小。

熟悉《人月神话》的人都知道这个概念，要想加快项目的进度，可以把项目进一步分解，但是这种分解有一个限度。即使考虑到了这个因素，有一点还是非常清楚的，那就是：团队的规模越大，项目交付的速度就越快，这样团队就可以承接更大的项目。

3.2.1　警告的信号

让我们把注意力从影响团队规模的因素，转向当团队规模不正确的时候出现的信号。沟通不畅、生产率低下、士气低落都是团队规模太大的信号。不畅的沟通可能有多种形式，包括工程师们缺席

会议、不回邮件、错过规范变更或者多个人问同样的问题。

生产率低下可能是团队规模太大的另外一个信号，如果经理、架构师、高级工程师没有足够的时间来指导初级工程师，那么，这些新成员将不会很快上手。如果没有人指导、引领、答疑，初级工程师要比正常情况下折腾得更久。相反，高级工程师们忙于回答太多初级工程师的问题，以至于无法完成自己的工作，也会造成生产效率的降低。生产率低下的一些信号包括错过发布日期、较差的功能点、推迟新功能研发。功能点和场景点是度量功能的两个不同的标准化方法。功能点从用户角度出发，而场景点从工程师角度出发。工程师在本性上对他们认为能够完成的任务过于乐观。如果他们推托一些以前做过的工作，这种无奈的反应或许就是生产率下滑的一个明显信号。

前面讨论了由于缺乏支持而导致的沟通不畅和生产率低下两个问题，团队规模过大的第三个信号是士气低落。通常，当一个朝气蓬勃的团队开始出现士气低落的情况时，这种不满情绪就清楚地说明出问题。尽管士气低落可能有很多原因，但是团队规模因素不应被忽略。那么我们怎么才能确定导致士气低落的真正原因呢？这和定位错误类似，从最后的变更开始查起。最近团队的规模增加了吗？士气低落表现在各种行为上，比如上班迟到，更多的时间待在游戏房里，在会议上争辩，在高管会议上不寻常地推托。问题的原因很直接：作为一个工程师，如果他感觉到缺乏支持，被排挤在沟通圈子之外，无法取得任务的成功，那就会情绪低落。很多工程师都喜欢挑战，即使是那些仅仅要搞清楚问题的来龙去脉，就要花好多天的难题。当工程师意识到没有办法解决问题的时候，他就会陷

入失望之中。这一点对初级工程师来说更是如此，所以要密切留意团队里初级工程师的行为。

相反，如果团队规模太小，要注意的信号包括不满意的业务合作伙伴、微观管理的经理、过度劳累的团队成员。在这种情况下，最麻烦的信号或许是业务合作伙伴，例如，产品经理或业务拓展人员花很多时间缠着研发经理抱怨，他们需要更多的产品交付。太小的团队无法快速研发和交付规模较大的功能。心怀不满的业务负责人，并不直接抱怨工程师和技术负责人，而是把力气花在增加预算聘请更多工程师上。

正常情况下有效率的经理如果趋向于微观管理将是一个令人担忧的迹象。或许是因为团队太小，经理像在团队成员的头顶上飞来飞去的直升机一样，针对他们的决策做事后评论，要求对项目状态更新这件事本身的进度进行说明。如果情况属实，那么就说明应该给这个经理增加一些任务了，通过扩大经理的聚焦点来解除他对团队的持续关注。在这种情况下可以安排的特别任务包括担任某个标准委员会的主席，负责一个跟踪 bug 的新工具评估活动，建立一个旨在指导工程师的导师计划。

确定一个团队是否太小的第三个迹象是，看是否有工作过度的员工。大多数的团队都会被正在从事的产品研发工作所鼓舞，相信公司的使命。他们希望成功，希望参与能够发挥作用的任何活动。这也包括接受太多的任务并试图在计划的时间内完成。如果某个团队的成员离开公司的时间越来越晚，或周末经常加班，那么你就要看看这个团队是否缺人。这种工作过度的情况对大多数的初创公司而言是可以预见的，甚至是必要的，但是如果持续几个月，最终会

消耗团队的干劲，从而导致员工流失、士气低落、质量不佳。最好能留意并分析工作时间的记录，尽早纠正问题，而不是等到大多数工程师递交辞呈时才意识到这个问题。

对此处归纳的种种现象置之不理将导致灾难性的后果。如果对这些现象失察，那么不可避免地会给公司带来人员流失、延迟交付研发的产品等问题。

3.2.2　扩大或者缩小团队

给小团队增加工程师，不能说轻而易举，但是至少也直截了当。比较困难的是拆分一个增长很大的团队。如果拆分得不好，会带来不良后果，比如，搞不清楚代码的所有权，沟通不畅，在新经理管理下工作压力增大。每个团队和机构都不同，没有完美的、标准的、放之于四海而皆准的办法。相反，当对组织进行改革的时候，必须考虑一些因素，以减少影响并使团队成员快速恢复生产状态。

当要拆分一个团队（包括在分解代码）的时候，必须要考虑一些事情，比如，谁出任新经理？每个团队成员的参与度有多高？与业务负责人的关系怎么处理？

首先要根据代码和工作来聚焦。如第三部分会详细讨论的，最佳方案是根据故障域来拆分团队和代码，通过隔离服务来限制故障所带来的影响。

以前一个团队拥有的代码需要分给两个或多个团队。对工程研发团队而言，这种分解通常围绕代码反复思考。旧的团队或许拥有一个应用的管理部分的所有服务，比如账户的建立、登录、收款和报表。再强调一下，这种情况没有标准的分解方法，但有一种可能

的解决方案是把服务分成两组或多组，一组负责账户建立和登录，另外一组负责收款和报表。当你深入到代码里去的时候，很有可能接触并决策分配那些基类。在这种情况下，我们想把总的拥有权分给一个团队，甚至一个工程师；通过在代码库中设置预警来通知其他团队，如果那个文件或者类发生任何变更，都会通知每个人，让他们意识到有人在代码中做了改动。

下一个要考虑的问题是新经理的身份。这是一个外聘或内选的机会。外聘的可以带来新思路和经验，内选的熟悉人员和流程。因为这些选择各有利弊，所以不要仓促做出决策。深思熟虑后再决策绝对正确，但是久拖不决可能带来同样多的问题。未知的压力可能使员工士气低落而引发不安。如果候选人来自外部，要尽快地做出公开的决策。在内选和外聘之间摇摆不定、拖延选择过程将增大团队的压力。

拆分团队的三大要素中最后的一个，是对业务部门会有什么影响。如果在工程团队、质量保证、产品管理和业务团队之间有一对一的关系，那么很明显，当团队拆分后这些关系就会发生变化。应该在拆分决策前，与受到影响的负责人讨论清楚这些变化。合作团队将同时被拆分，或者重新分配每个人的工作，以便在新的阵容中彼此合作。组织机构调整有许多的可能性，但最重要的考虑是，开诚布公地在工程技术团队内外进行讨论。

到目前为止，我们已经讨论了与团队规模太大或太小相关的警告迹象，也分析了拆分团队时需要考虑的因素。从本节中可以得到的收获和学到的经验是团队的规模，以及调整规模对组织的士气和生产率所带来的巨大影响。重要的是我们要进一步认识到，团队规模是确定组织对应用扩展有效性影响的主要决定因素。

事项检查表

最佳团队规模的事项检查表：

1. 确定经理的经验水平。

2. 计算每个工程师在公司的工作年限。

3. 询问每个工程师在行业内的工作年限。

4. 根据经理的责任估计其总工作量。

a. 调查经理在不同的任务上花多少时间。

b. 列出你期望经理完成的核心管理职能。

5. 寻找因团队规模太小而无法发挥作用，所以心存不满的业务伙伴和经理。

6. 发现因团队规模太大而造成的生产率低下、沟通不畅和士气低落的迹象。

团队拆分的事项检查表：

1. 确定如何分拆代码：

a. 按照服务拆分。

b. 尽可能把基类平均分割，每个基类由一个人负责。

c. 在代码库中加入提醒，确保当相关代码被修改时每个人都知晓。

2. 确定新的经理。

a. 考虑内选还是外聘。

b. 设定一个紧迫的时间来完成决策。

3. 分析团队和其他部门之间的相互关系。

a. 与其他部门的负责人讨论团队拆分计划。

b. 协调本团队的拆分与其他团队之间的关系，确保平滑过渡。

c. 通过联合通知的方式同时向全体员工公布消息。

3.3　组织结构

组织结构指的是一个组织内部团队之间相互关系的布局。它包括把员工分配到不同的部门和团队，也包括为了发号施令而安排的层次结构。各公司有不同的组织结构，功能型和矩阵型两种基本结构已经使用多年，一种新型的敏捷型结构最近也开始流行。本章将对两种久经考验的组织结构的利弊进行分析，然后介绍近来出现的新型结构，以帮助你选择最适合的组织结构。最常见的情况是，设计一种混合的结构来最好地满足公司的需要。本节将给出每种组织结构的基本定义，并概述每种结构的优缺点，提出如何进行选择的建议。这里要掌握的最重要的内容是如何选择某种形式的组织结构，组织结构又如何随着公司的成熟逐渐演化。

3.3.1　职能型组织

军队和工业界曾经使用过职能型结构的组织形式。如图 3-1 所示，该形式的结构按照主要目的或职能来设置部门。该策略常被称作竖井式方法，就像玉米或谷物按照粮食的不同等级分入不同的粮仓一样，人被编入不同的组别。在技术组织内，按照职能的不同分成不同的部门，如工程部、质量保证部、运维部、项目管理部等。同时，每个部门都有自己相应的管理层级结构。各个部门都有自己的负责人，比如工程部门的副总裁。每个部门都有类似的职责组织结构。向工程副总汇报的经常是其他的工程经理，如工程总监。向工程总监汇报的是高级工程经理，向高级工程经理汇报的是工程经理，以此类推。这种层级结构是通用的，工程经理汇报关系如此，

质量保证经理的汇报关系亦如此。

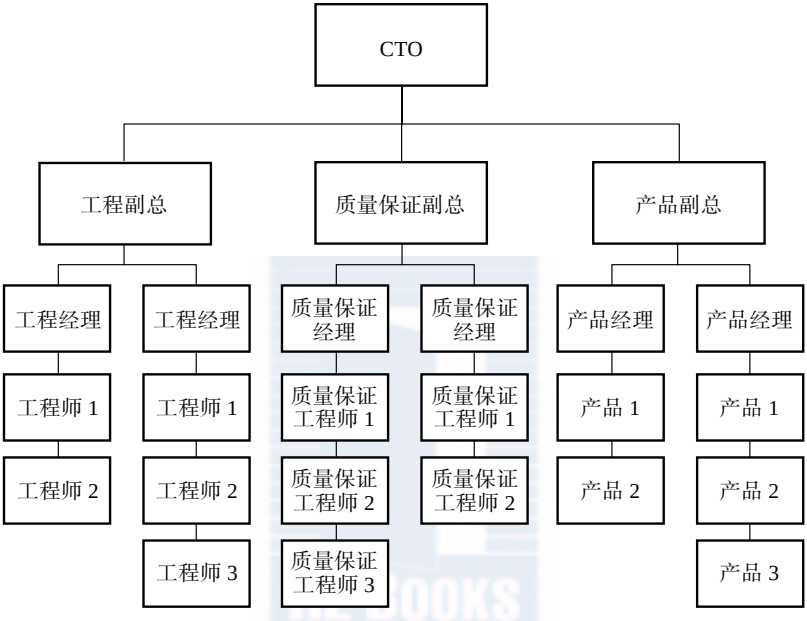

图 3-1　职能型组织图

职能或竖井式方法的组织结构有很多好处。所有的经理几乎都是按照层级提升的，即使他们的业绩表现并不抢眼，他至少清楚如何按部就班地工作。除非时代久远发生了很大的变化，否则根本就没有必要花费时间，向深谙于此的老板们解释这种安排的奥秘和细节。员工在专业性方面也容易取得一致，比如工程师与工程师在一起工作。在这种结构下，同伴们一般会相邻而坐，可以很容易地回答涉及技术方面的问题，整个机构都是按照专业分工来组织的。

用运动来作类比，职能型的组织结构就像在高尔夫球场练球。

高尔夫球手们想要练好和打好高尔夫球，就要和其他高尔夫球手，甚至教练混在一起，切磋技艺。记住这个类比，我们将利用它来对职能型结构和矩阵型结构进行分析和对比。

除了在管理和工作伙伴方面的同质性和共同性外，职能型结构还有其他的优点，这包括职责明确、容易分配任务、更好地遵循标准规范。因为组织结构极其清楚，所以几乎每个人，即使是新入职的，也能很快地掌握谁负责哪个团队和项目。这种简洁性使工作的分配极为容易。在一个采用瀑布型开发方法的职能型组织中，研发的责任显然落在工程团队上。因为软件工程师都向工程部门的负责人汇报，质量保证工程师均向质量保证部门的负责人汇报，所以非常容易制订、发布、遵循和执行标准。所有这些因素解释了为什么军队和工业界长久以来一直以职能型的组织结构为标准。

职能型或直线式结构的问题包括缺乏单一的项目负责人和跨职能部门的沟通效果不佳。项目很少只发生在一个职能部门内。大多数的软件研发项目，特别是通过网络交付的服务，比如，软件即服务（SaaS），总是需要不同技能的人来共同完成任务。即使是研发一个简单的功能，也必须要有产品经理起草产品设计说明书，软件工程师设计和实现代码，质量保证团队完成测试，运维工程师部署到生产系统。项目的整体责任不会全部落在管理层级中某个人的头上，而是逐级上升，直到最后落在技术总负责人的头上，他对产品经理、工程、质量保证和运维人员都负有责任。当产品经理不向 CTO 汇报的时候，这种责任转移现象甚至会更加夸张。显然，直接由 CTO 或者技术副总裁负责项目整体的成败会有严重的问题。如果这么做，当项目出现问题的时候，每个职能部门的负责人就会把项目延迟或

者经费超支之类的问题和责任推到其他的部门。

在职能型组织中，事实已经证明跨部门的沟通出奇难。例如，软件工程师想告诉质量保证工程师，必须进行某个测试以验证功能是否正常。为此，软件工程师很可能要浪费宝贵的时间，在质量保证的管理层级中上下求索，直到找到负责分配测试任务的经理，然后请求他指派一个人来完成这项测试。工程师很可能会依赖现有的设计规范文档，通过流程来传递这些信息。可以想象，在研发工程师和测试工程师之间，写一个 20 页的测试规范比面对面的对话要带来多少额外的沟通负担。

职能型组织所面对的另外一个挑战是团队之间的冲突。如果这些团队被赋予交付和支持产品或服务的任务，而这些任务又需要跨部门的合作才能完成，团队之间的冲突就不可避免。“这个团队没能按时完成任务”和“那个团队交付的产品存在错误”，这些都是那些按职能型结构组织起来的团队在合作时经常听到的抱怨。极难听到对职能型团队的挑战或者贡献的感激和赞美。对一个工程师来说，个人认同和社会认同相关联，被视为工程师“部落”的财产。工程师们想要有归属感和被同伴接受。其他不同的人（质量保证、产品管理，甚至技术运维）经常被当成圈外人士，而不被信任，甚至有的时候成为公开的敌对目标。我们见证过由认为相互不同的个人组成的团队之间的冲突，这种冲突在最极端的情况下甚至发展为歧视。

有一个经典的“蓝色眼睛和棕色眼睛”实验，实验说明个体很容易转向攻击那些被视为与自己不同的个体。1968 年，马丁・路德・金遇刺身亡后，艾奥瓦州的一个小学的三年级老师做了一个实验，在实验中，她把班里的同学分成蓝色眼睛和棕色眼睛两个组。

她编造了一些“科学”证据，证明眼睛的颜色决定智力的水平。这位老师告诉一组人说他们是优等的，给他们额外的课间休息时间，让他们坐在教室的前边。相反，不允许另外一组用饮水设备喝水。结果，教师发现那些“优等生”变得傲慢，喜欢发号施令，对同班的“劣等生”不友好不客气。接着，她改变了自己的解释，告诉另外一组的学生，他们才是优等生。爱丽奥特报告说，新的优等生去嘲讽另外一组，就像他们曾经被烦扰的情况一样，只是没有前面那组那么凶恶。

冲突

实战经验和学术研究一致认为，冲突有好坏之分。好的冲突（认知型冲突）是健康的争辩，是团队关于该做什么或者为什么要做而发生的冲突，它涉及更大的视角和更多的经验。坏的冲突（情感型冲突）基于角色，经常涉及怎么做或者谁该做。当然，并不是所有涉及角色的冲突都是坏的，纠结不清的基于角色的讨论往往被认为是政治性或者领地性的，如果处理不当会对组织产生不良影响。

好的或认知型的冲突有助于团队扩大行动的可能范围。不同的见解和经验凑在一起有机会从多角度出发解决难题。头脑风暴会议以及组织适当的事故分析会，都是可控的认知冲突，目的是要产生一套优越的替代方案和行动。团队的规范已经充分地显示出对发展认知型冲突所起到的正面作用。情商高的负责人也会帮助在团队内部产生正向的认知型冲突。但是，如果认知型冲突得不到解决，就会演变成情感型（坏的）冲突。

坏的或者情感型冲突会带来物理和组织的创伤。在物理上，它可以耗尽我们的精力，因为交感神经系统（下丘脑激发，与涉及“打架或逃跑”综合征属于同一个系统）释放压力荷尔蒙，例如皮脂醇、肾上腺激素、去甲肾上腺素。结果，血压上升和心跳加快。随着时间的推移，不断的情感冲突让我们感到疲惫。在组织上，研究表明冲突会造成组织分裂，结果会在战术和战略上限制选择。争斗关闭了我们思维的选择空间，意味着结果可能不是最佳的。

冲突为什么如此重要呢？通过了解冲突的来源和结果，作为领导可以推动健康的争辩，尽快结束有害的情感型冲突。我们的工作就是要建立一个健康的环境，以利于股东利益的最大化。通过营造开放、关爱、尊重的文化氛围，把认知型冲突最大化，情感型冲突最小化。通过设置明确的角色和责任，限制情感型冲突的根源。通过吸纳各类人才在技能和视野上互补，最小化集体思维的机会，最大化策略的选择范围，鼓励机构快速成长。

棕色眼睛和蓝色眼睛

1968 年 4 月 4 日，在发表了《我曾经到达山顶》（I’ve Been to the Mountaintop）的演讲两天后，马丁·路德·金在田纳西州的孟菲斯被暗杀。第二天早上，在艾奥瓦州赖斯维尔（Riceville），简·爱丽奥特，一个小学三年级的老师，问她的学生对黑人了解多少？接着问学生是否想要体验一下有色人种在美国当时的感受。爱丽奥特是想做个体验实验，不过这个颜色是眼睛的颜色而不是皮肤的颜色。

首先，她把棕色眼睛的学生分配到“优等”组。爱丽奥特在报告中提到，起初有些学生对这种分配有些抵触，为了能把实验做下去，她谎称黑色素造成棕色眼睛，与较高的智商和较强的学习能力有关系。接着，那些被归到“优等”组的孩子变得傲慢，喜欢发号施令，对比他们差的“劣等”同学不友好、不客气。爱丽奥特给了棕色眼睛孩子特别的权利，比如午餐时的第二份，可以去新的攀爬架，给5分钟的额外课间休息时间。棕色眼睛的孩子坐在教室的前部。爱丽奥特不允许棕色眼睛和蓝色眼睛的孩子共用一个饮水机。

棕色眼睛的孩子在学术上的表现有所提高，而蓝色眼睛的却孩子变得驯服和屈从。有一个棕色眼睛的学生问爱丽奥特她有蓝色眼睛怎么会成为教师。另外一个棕色眼睛的学生回答道：“如果她没有蓝色眼睛，她可能早就成了校长或者学区总监了。”

下个周一，爱丽奥特把实验反过来，让蓝色眼睛的孩子进入“优等”组。蓝色孩子用类似的方式来嘲讽棕色眼睛的孩子，爱丽奥特在报告中说，他们的行为没有棕色眼睛的孩子表现出的那么强烈。下个周三，爱丽奥特停止了试验，让孩子们把他们学到的东西写下来。关于这个实验的反思发表在当地的报纸上，随后美联社做了转载。爱丽奥特为此有点儿声名狼藉，包括在约翰尼·卡尔森主持的《今夜秀》节目上的露面。对实验的反应好坏参半，既有因为她利用孩子做实验而激起的公愤，也有受邀前往白宫参加儿童与青年大会的荣耀。㊀

㊀ Stephen G. Bloom. “Lesson of a Lifetime.” *Smithsonian Magazine*, September 2005. http:// www.smithsonianmag.com/history/lesson-of-a-lifetime-72754306/?no-ist=&page=1. Accessed June 24, 2014.

职能型组织的好处包括经理和工作伙伴的同质性、责任简单清晰、有标准可依。不利的地方包括没有单一的项目负责人和沟通不顺畅。考虑到这些利弊，当其同质性的好处超过整体协调和所有权所带来的问题的时候，可以考虑采用职能型组织结构。比如，采用瀑布式研发的组织，经常能从按职能划分的组织结构中获益。因为该结构恰好与瀑布型方法中固有的阶段控制相匹配。

3.3.2　矩阵型组织

在 20 世纪 70 年代，组织行为专家和经理们开始重新考虑组织的结构。如前所述，尽管职能型组织结构有一些不可否认的优点，但仍然存在弊端。为了克服这些问题，公司甚至军事机构开始试验不同的组织结构。从中演化出来的第二种基本的形式就是矩阵型组织结构。

矩阵型组织结构的主要概念是层级的两个维度。与职能型组织相反，在职能型组织的每个团队有一个经理，每个团队的成员向一个老板报告，而矩阵型组织则包括至少两个管理维度，每个团队的成员或许有两个或多个老板。这两个老板，每人都有不同的管理责任，例如，一位（团队负责人）负责处理行政任务和审查，而另外一位（项目经理）处理安排任务和跟踪项目状态。如图 3-2 所示，传统的职能组织旁边增加了项目管理团队。

图 3-2 的右边和职能型组织很相似。区别主要在左边，即项目管理所在的部分。注意项目管理部（PMO）中背景颜色较暗的项目经理和其他团队的成员。项目经理 1、工程师 1、工程师 2、质量保证工程师 1、质量保证工程师 2、产品经理 1 和产品经理 2 都是淡灰

色背景。这些淡灰色的成员通过矩阵方式组成了一个项目团队来一起工作。项目经理可能对任务分配和项目进度负有责任。在更大和更复杂的组织里，许多来自于不同团队的人可以属于同一个项目团队。

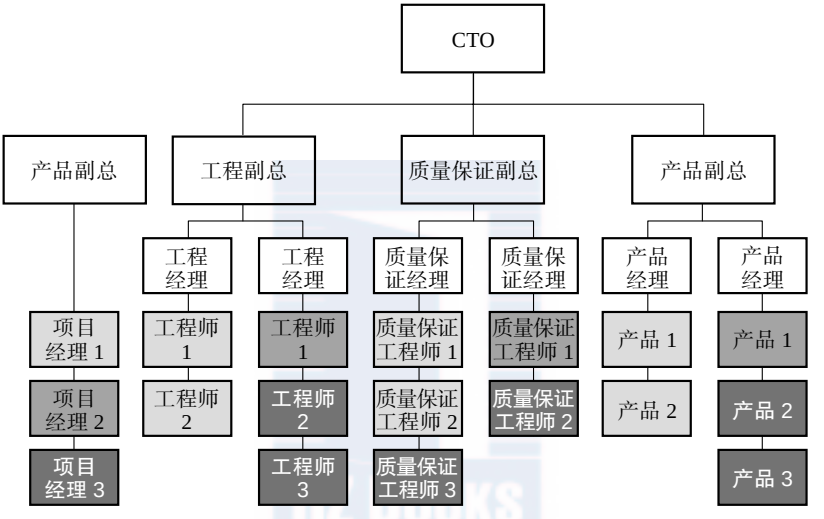

图 3-2　矩阵型组织图

继续讨论负责实施新收款功能的项目团队，我们开始意识到这种矩阵型组织所带来的好处。职能型组织有三个主要问题：没有项目责任人，跨团队沟通不畅，部门间存在情感型冲突。而在矩阵型组织里，项目团队负责解决所有的问题。现在我们有了第一线负责收款项目的项目经理 1。这个项目团队将会每周甚至更频繁地开会，以及采用电子邮件频繁往来，从而解决了职能型组织所面临的沟通问题。如果软件工程师想和质量保证工程师沟通某个测试是否应该包括在其测试桩中，那么就可以通过发邮件或在下次的团队例会中

提出来。因此，这种结构避免了在不同的管理层间寻找合适的人的困难。

我们再回到用于讨论职能型组织的高尔夫球的例子中。高尔夫球员想要更佳的表现，为此，他和其他的高尔夫球员甚至教练在一起练球。这个例子主要为类比职能型组织的团队，他们希望能把某个特定的功能发挥得更好，所以就和其他也从事类似活动的人在一起做同样的事情。按照运动教练的说法，这种专一性对于发展肌肉的记忆和基本的技能成效显著，但是要真正变得擅长，运动员必须交叉培训。按照这个概念，高尔夫球员应该时常离开高尔夫球场去练习其他的肌肉，比如参加举重培训或者跑步。这种交叉培训的方法和矩阵型组织类似，在不替代高尔夫或工程的基本训练的前提下，通过给予其他的任务比如跑步或项目管理来加强。

对那些已经经过交叉培训过的个人，你或许会问：“交叉培训是否会阻碍运动员成绩的提高?”事实上，如果你是一个高尔夫球手，也许听说过千万别去玩垒球，因为这会对你的高尔夫挥杆动作带来严重的影响。我们会在研究矩阵型组织弊端的时候讨论这个概念。

如果你已经解决或者至少通过实施矩阵型组织已经大幅度地改进了职能型组织的弊端，无疑这种提高是有代价的。事实上，在缓和项目责任和沟通问题的时候，我们也引入了其他的问题，这主要涉及多个老板和个人精力分散的问题。向两个或多个人汇报，矩阵型组织可以非常复杂，需要一个人参与多个团队，收到每个老板各自的指令，所以团队的成员会感觉到更大的压力。工程师陷在工程经理和项目经理之间，工程经理让她按照标准写代码，项目经理坚持必须按时完成任务，工程师面临着压力和忧虑，担心自己的表现

不能让经理们满意。另外，和其他团队一样，项目管理团队也需要一些额外的会议或者邮件来沟通。这种额外沟通并不能取代工程师必须参加的工程经理召集的团队会议，所以用掉了其基本代码工作中的一些时间。

正如你看到的，当我们解决了一些问题的时候，矩阵型组织结构又引入了新的问题。其实不必太吃惊，因为这是现实的情况，很少能够解决一个问题而不引发另外的一些问题。下一个问题，假如保持矩阵型组织和职能型组织结构的优点，“是否有更好的办法?”答案是“有”，这就是“敏捷型组织”。

3.3.3　敏捷型组织

研发独一无二的面向客户的 SaaS 解决方案，或者企业级软件是一个复杂的过程，需要跨部门整合多种技能的合作。在职能型组织里，要交付 SaaS 产品，冲突和沟通的问题在所难免。矩阵型组织通过协调不同技能的人进入项目组解决了这些问题，但同时又催生出其他的问题，比如独立贡献者向多个有不同优先级的经理汇报。SaaS 的独特要求，敏捷开发方法的出现，还有职能型和矩阵型组织弊端的存在，使被称为敏捷型的新组织结构应运而生。

2001 年 2 月 17 日，17 位软件研发人员，代表着采用各种以文档驱动的软件研发方法的实践者，齐聚在犹他州的雪鸟镇，共同讨论一种轻量级的研发方法。会上所有的与会代表签署并发表了《敏捷软件开发宣言》[⊖]。与某些评论相反，这个宣言并不排他，而是通

⊖ Kent Beck et al. “Manifesto for Agile Software Development.” Agile Alliance, 2001. Retrieved June 11, 2014.

过给出的 12 条原则试图恢复“方法论”一词的可信度，它概括了如何在深度聚焦客户的前提下开发软件。其中一条原则就是聚焦团队搭建，“最好的架构、需求和设计源于自组织的团队。”这个自我管治、自我组织的团队使人们脑洞大开，感到形成一种全新的组织结构的可能性，它不是以角色为基础，而是专注于满足客户的需求。

中央托管运维商业应用并不是什么新事物，实际上起源于 20 世纪 60 年代以分时为基础的主机系统。到了 20 世纪 90 年代，伴随着互联网的快速扩张，企业家们在市场营销时把自己包装成应用服务提供商（ASP）。这些公司运营和管理着其他公司的应用，每个客户有自己的应用实例。据说其价值是降低了客户的成本，因为 ASP 在运营和管理某个应用方面技能极佳。21 世纪初，又有了另外一个发展，也就是软件即服务（SaaS）的出现。据推测，这个词首先出现在 2001 年 2 月，软件和信息产业协会（SIIA）电子商务分会内部发表的“策略的背景：软件即服务”[⊖]一文中。像技术世界的很多其他事情一样，SaaS 的定义仍存在着争议，但是大多数人都同意它包括了一个预约定价模式，客户根据用量付费，而不是一个协商好的许可费用。这些应用的架构通常是支持多客户的，这就意味着几个客户共享同一个软件的实例。

随着从提供软件向提供服务方向发展，技术人员开始思考如何做个好的服务者而不是软件开发者。伴随着这一演进，人们开始思

⊖ *Strategic Backgrounder: Software as a Service*. Washington, DC: Software & Information Industry Association, February 28, 2001. http://www.slideshare.net/Shelly38/software- as-a-service-strategic-backgrounder. Accessed April 21, 2015.

考这些交付服务的质量和可靠性。从传统意义上说，当我们提到服务的时候，在脑海中出现的往往是家庭服务，如水、卫生和电力。我们对这些服务的质量和可依赖性有很高的期望值。每次打开水龙头时，我们都期待着干净、适合饮用的水喷涌而出。打开电灯开关时，我们期待着电流流过，伴随着很小的波动。为什么我们要期望软件服务也同样要高质量和高可靠呢？随着客户对 SaaS 期望的上升，技术公司开始尝试通过提供更可靠的服务来应对。但是，传统的组织结构不停地阻碍着它们达到服务标准，结果在职能团队与缓慢的交付之间产生了更多的冲突。

形成敏捷组织概念的最后一个环节，是认识到技术团队的组织结构对软件的质量、扩展性和可靠性都有非常重要的影响。本书的作者通过与客户几年来的接触得出了这个结论。作为技术工作者，当开始提供咨询服务的时候，我们肯定会把精力聚焦在技术和架构上。然而，并不是每个技术问题都应该通过技术手段来解决。当反复验证时会发现，这一问题始终会回到组织结构上，例如团队之间的冲突，一个人汇报给多个经理，而这些经理并不能理解每个人的任务优先级。事实上是人开发了技术，因此，人对过程至关重要。这使我们理解了真正的可扩展系统需要架构、组织和流程的协调一致。这一顿悟的高潮就是在 2009 年出版本书第 1 版。我们当然不想宣称自己是认识到组织结构重要性的第一人，也不想暗示这本书对宣传这一思想影响最大。因为本书的两位作者过去都在高科技公司里担任过负责运维的高管，对这一关键思想都不甚了解，这足以说明，从 2000 年到 2005 年这段时间里，技术社区对该思想缺乏深入的了解。

这三个因素所造成的结果是，技术公司开始试验组织结构的各种排列组合的可能性，试图提高所提供的软件服务的质量和可靠性水平。在职能型和矩阵型组织中，很有可能会出现不同的新组织结构的变种。为了简化，我们把跨职能同时符合服务架构的组织，标示为敏捷型组织。如图 3-3 所示，在这种敏捷型组织中，团队完全是自主管理和自给自足的。在研发的整个生命周期中，从形成概念，到研发，再到生产系统中的服务支持，团队负有全部的责任。跨职能部门的总监、副总裁以及敏捷型团队替代了工程副总裁这样的常规管理角色。

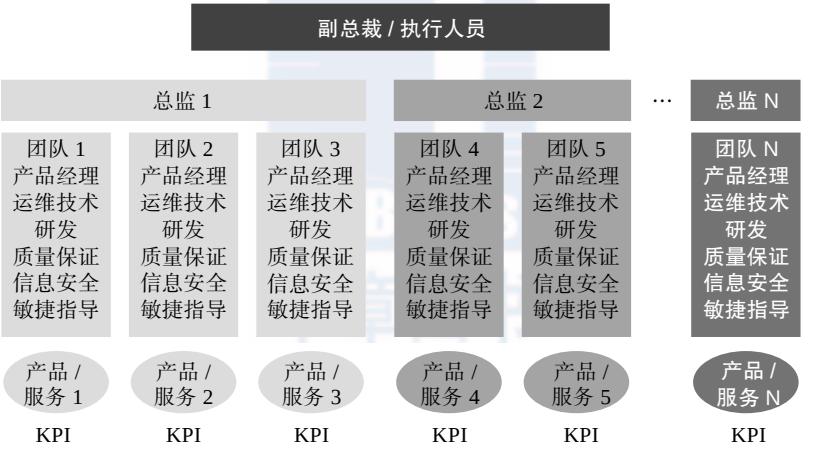

图 3-3　敏捷组织图

第三部分将要讨论如何把系统分割成微小的服务，然后再构建成大的系统。目前，用一个电子商务系统的简单实例来说明这个问题，该系统可以分割成包括诸如搜索、浏览和结算的用户服务。在这种情况下，采用敏捷型组织结构的公司将有三个团队，每个团队

分别负责一个服务。这些团队会有全职人员来管理、研发、测试、配置和支持负责的服务。图 3-4 描述了这个示例，其中每个敏捷团队都拥有一个用户服务，并且包括了需要的所有技能。

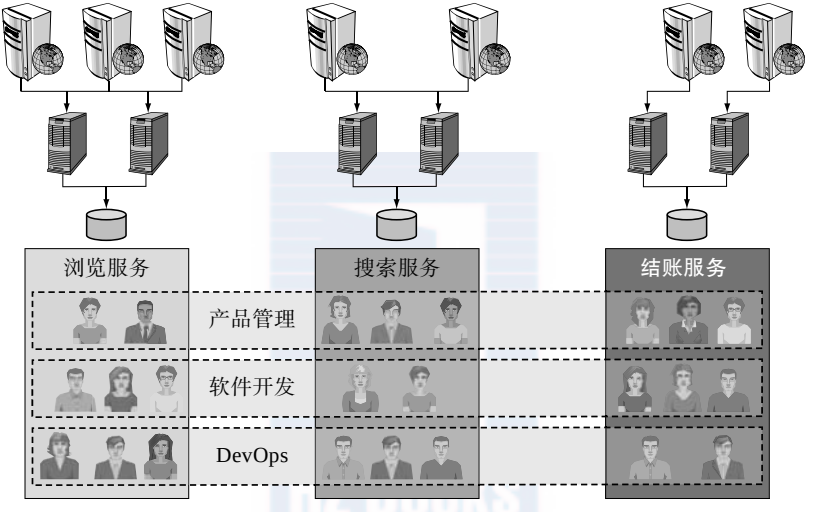

图 3-4　敏捷组织与架构匹配

1. 革新的理论

在深入了解敏捷型结构的实际应用之前，让我们先来探讨一下敏捷型组织为什么能够有效地降低情感型冲突和提高团队表现水平背后的理论基础。首先需要理解如何度量一个团队的表现。在实际的度量中，我们希望看到团队所分担的服务的质量及可用性水平得到提高。在学术研究中，我们可以用“创新”一词来表示团队有增加值的产出。创新一直被定义成一个包含有效表现的标准。长期以来我们一直在寻找合适的答案来回答“什么因素能帮助团队提升创

新的水平?”通过大量定性和定量的基本研究，以及与外部研究的结合，我们找到了一些能推动创新的因素。在此，我们将对这些因素进行逐一探讨。

如前所述，冲突可以是建设性的，也可以是破坏性的。认知型冲突可以博采众长，头脑风暴就是认知型冲突的例子，团队聚在一起，希望能想出比个人更优秀的方案。我们每个人都参加过至少一次头脑风暴，往往有意想不到的产出。这种会议一般都由经理先定好议程，确保彼此都认识，确定一些有关时间的限制和彼此尊重的基本规则等等。接着就是一个 60 ~ 90 分钟的过程，大家在彼此想法的基础上制订方案。不需要每个人都同意，但是以相互尊重的方式来交换意见，从而移除影响问题解决的障碍，使方案得到讨论。这种合作方式会带来创造性和创新的想法，这是单打独斗的个人所无法完成的。在会议结束时，每个人都非常高兴能够参加，认为自己付出的时间很值得，甚至希望那周所有的其他会议也都这么顺畅。

与认知型冲突相对的是情感型冲突，这就是所谓的基于角色的、坏的或破坏性的冲突，总是围绕着“谁来做”或“怎么做”这些问题。情感型冲突给团队的成员带来身体上和情绪上的压力。认知型冲突可以提升团队的创造力，情感型冲突则削弱团队的创造力。

情感型和认知型冲突不是影响创新的唯一因素。回想一下头脑风暴的场景，环顾会场，参会的人代表了各种不同的背景。或许有一个人来自于工程部门，而另一个来自于产品管理部门。有些人刚出校门不久，而另外那些已经有数十年的工作经验。这种经验上的差别既会带来感情型冲突，也会引起认知型冲突。有时来自不同背景的人容易产生严重分歧，因为他们从差别巨大的视角来看待和处

理问题。如果总是这样，那么我们就要为团队配置背景相似的人员，但是员工的动态不容易掌握。经验的差异也会促进思路的多元化，形成的解决方案远远超越个体所能提供的。所以经验上的多元化加剧了情感型和认知型两种冲突。对领导而言，提升团队创造力的关键在于减少经验差别对情感部分的影响，最大化经验差别对认知部分的影响。

影响创新力的另外一种类型是关系的多样性，可以通过度量团队里的个人与其他的人以及专业领域联系的多寡来衡量。关系的差异对创新很重要，因为所有的项目都会遇到障碍。关系良好，差异大的团队可以更早地发现项目中的障碍和问题。我们可能都经历过这样的情形，团队里有一个和其他人背景完全不同的人，在以前待过的团队里还有些老朋友，他们可以为潜在的问题提供一些清晰的建议。例如，当你在做一个 IT 项目时，需要为一个制造厂安装一套新的软件系统。团队中有人曾经在该厂做过暑期实习，他及早地提醒团队，其中有一条生产线只能在每周的某几天关闭，以安装软件。团队里这种类型的关系差异会使项目获益匪浅。当团队碰到障碍的时候，人脉最广的团队最容易找到外部资源绕过障碍。这种资源可能是上级管理层所提供的支持，甚至可以是额外的质量保证工程师，以帮助完成软件的测试。

还有一个与提升创造力有关且广泛引用的因素是团队的授权感。如果一个团队感觉到被授权实现某个目标，那么这个目标就很有可能会达成。举一个在军队里常见到的有趣反例，降低授权会导致完成任务的动力不足。降低一个团队或个人的授权感，并让他完成任务达成目标，将极大地考验其坚毅勇敢的内在品质。达成目标的一

个窍门是移动目标线。

设想你将要参加一个竞争性很强的军队选拔课程，例如，在军官候选学校选择和培训入伍后的士兵成为军官，让其在战场上领导士兵。要达到这个水平，你必须先成为表现优异的士兵，在审查时拿到最高分。还要在未来几个月甚至几年的课程中竞争，通过一系列心理、身体、精神的测试。保持身体强健、头脑灵活，有信心克服面前的任何障碍来完成任务。

早晨，在太阳升起前醒来，穿上训练服、系好鞋带、出去列队。日程表上列出的是通过跑步做体能测试。作为一个不错的跑步选手，或许你会觉得该测试轻而易举。可是教官给你的指令中没有明确终点在哪里。你跑上几千米然后往回转，跑向起点线大多数人认为终点线就是起点线。然而，当到达终点的时候，你转过身再沿着另一条路线跑，跑了几千米后，转身往回跑向起点，你认为这次该是终点线了。别急，教练也跑到了起点，然后带领大家向另外一条路跑去。

这时候，参加跑步的候选人开始崩溃、认输。不知道终点和目标在哪里使人们变得灰心沮丧，怀疑自己的能力。与此相反，如果个人或者团队相信自己被授权，并配备了完成任务所有需要的足够资源，他们的创造力就得到了提升。再回来看一下军官候选学校的例子，如果士兵们相信自己是被授权来锻炼体魄，拥有跑步所需要的适当的衣服和装备，非常清楚地理解了他们的目标，那么他们完成任务并实现目标的可能性就很大。

在创造力模型当中，还需要了解一个和组织结构直接相关的因素，这就是组织边界，它指的是不同团队的个人。有些边界很窄，例如类似的团队之间，有些边界则很宽，比如那些非常不同的团队

（产品经理和系统管理员）。在创造力模型中，跨越组织机构边界的合作必然发生，结果增加了情感型冲突的机会。因此，团队为达成目标要进行合作，这种合作必须跨过的边界越多，其创造力就会降得越低。早些时候我们曾就其结果讨论过。工程师的个人认同感与他在工程师“部落”的归属感密不可分。我们希望有归属感并被同伴接纳。其他那些不同的人员（质量保证、产品管理或者甚至是技术运维人员）经常会被当作不应当信任的外来者。据猜测，形成“人对人是狼㊀”这种情况的原因是生存策略导致的，外来者被当成是争夺稀缺资源和充满敌意的竞争者而不受信任。对外来者的不信任感形成了很强的内部小团体㊁。我们发现很多团体都存在着这类情感上对外来者的疏远感和不信任感㊂。

图 3-5 描述了完整的团队创造力理论模型。在模型中，关系差异、授权感和认知型冲突都会提升创造力，而情感型冲突会降低创造力。经验的差异对认知型和情感型冲突均有加剧的作用，组织边界增大加剧情感型的冲突。现在，依靠这个完整和详细的模型，我们可以清楚地解释为什么职能型和矩阵型组织会降低创造力，而敏捷型组织可以提升创造力。

㊀ “Man is a wolf to [his fellow] man.”

㊁ Christian Welzel, Ronald Inglehart, and Hans-Dieter Klingemann. *The Theory of Human Development: A Cross-Cultural Analysis*. Irvine, CA: University of California–Irvine, Center for the Study of Democracy, 2002. http://escholarship.org/uc/item/47j4m34g. Accessed June 24, 2014.

㊂ Peter M. Gardner. “Symmetric Respect and Memorate Knowledge: The Structure and Ecology of Individualistic Culture.” *Southwestern Journal of Anthropology* 1966;22:389–415.

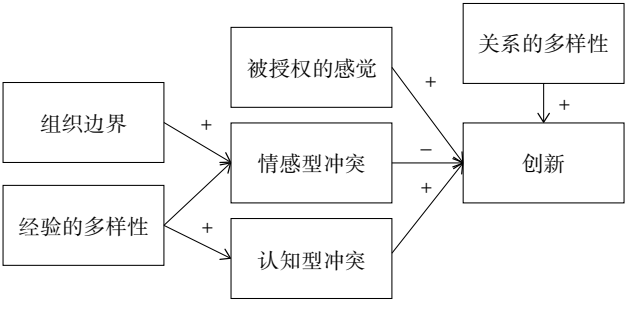

图 3-5　创新模型的理论

在一个职能型组织里，个体被按照其技能组织起来，几乎所有的项目都需要跨团队的协调。特别是对 SaaS 这样的服务提供商，其责任不仅限于研发和测试软件，还包括由公司的技术团队来托管、运维和管理。对那些打包好的商业软件，由于软件由客户自己安装和支持，所以部分的责任就和客户共同承担。在今天更流行的 SaaS 模式中，全部的责任就归属于公司的技术团队。如图 3-6 所示，这会给团队带来情感型冲突。机构的边界增加了情感型冲突，导致创造力的降低。

在矩阵型组织中，每个人都会有多个经理，而且每个经理往往会有不同的优先级，这就会出现“移动的目标”的情况。试图取悦两个领导经常会导致目标不清，降低团队的授权感。

敏捷型组织不存在这两个问题。通过打破组织的边界，解决职能型结构组织的问题，通过团队授权，减少矩阵型组织所面对的问题。

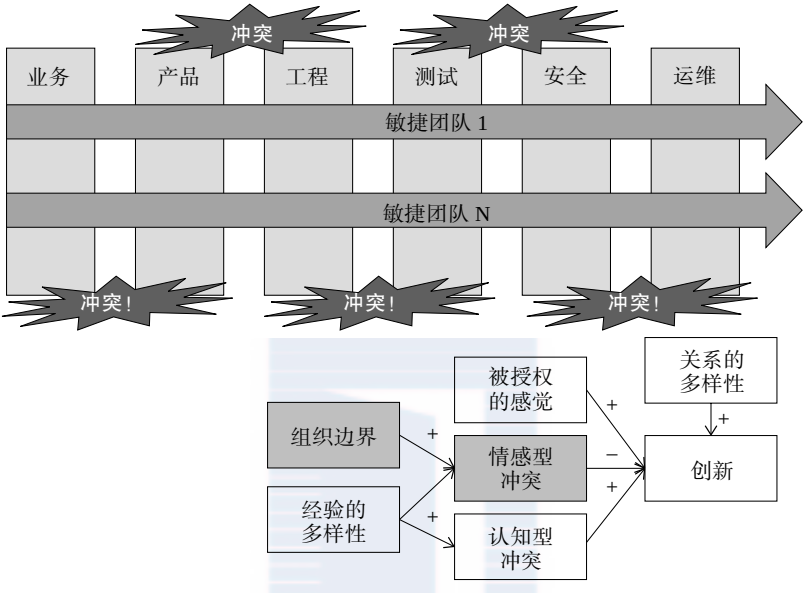

图 3-6　职能型组织增加情感型冲突

2. 敏捷型组织的优势

敏捷型组织的主要好处是提升了团队的创造力。在一个提供 SaaS 服务的公司里，往往通过快速响应市场对功能的需求、更大数量的产品和更高的可用性来度量创造力。团队往往在使用敏捷型组织结构后才体会到它的好处。正如所期望的那样，通过改善诸如冲突、授权和组织边界来实现创造力的提升。

团队按照服务组织起来，自主管理并让人员跨越职能部门，结果是大幅度地降低了情感型冲突。团队里的成员共享一个目标，大家荣辱与共，不再需要争辩谁来负责或谁应该完成某个任务。每个人都要对其提供的高质量、高可用性并能满足商业目标的服务负责任。

敏捷型组织在提高团队的创造力方面作用极大，然而这种结构也有一些劣势。下面就讨论这些劣势。

3. 敏捷型组织的劣势

我们听到的主要抱怨不是来自于工程师、产品经理或任何的成员，而是来自于管理层。当改变了一个组织的结构的时候，如果去除了一些传统的管理角色，诸如“工程副总裁”。当然，也可以保留工程副总裁的位置，但事实上那个人是在做技术召集人或者带领跨职能部门的敏捷团队。去除高级管理层的职位，有时使经理或者总监感到心里不安。

敏捷型结构的另外一个劣势是，为了按照计划的方式正常发挥作用，团队需要按照架构设计中面向用户的服务重新组织。如果用高质量的功能和高可用性的服务来度量，当敏捷型结构的团队在代码所有权和责任上出现重叠的时候，团队将无法自治，会导致较低的创新力。当然，这并不意味着敏捷型团队，无法提供通用的核心服务给其他的团队。通常这样的团队会以类似开源的模式为其他敏捷团队提供资源库或者服务。

生产系统中与服务相关的问题，经常要求软件开发、DevOps、质量保证、甚至产品经理值班，这经常被人们认为是敏捷型组织的一个弱点。在我们看来，这其实是该组织结构的好处，因为它为团队提供了闭环的反馈。如果团队的成员经常在半夜两点被叫起来解决生产系统的问题，他们会很快地意识到要研发高质量的服务，以便晚上能睡个好觉。短期或许是痛苦的，但是对长期而言，这是提升团队表现的好办法。

没有一个组织结构是完美的，我们认为对那些正在挣扎着试图

解决不良的服务交付、大量的冲突、员工自我驱动力不强和整体上缺乏创新的公司来说，敏捷型组织模式是个理想的选择。

Spotify 组织结构

在现实世界里，可以在 Spotify 公司找到一个面向服务、跨越职能的团队。它的组织结构不是按照职能型组织的，而是由小的敏捷团队，公司内部叫做“小组”（squad）的单位组成的。这些自给自足的团队按照业务部门提供的要交付的任务和服务进行组织。下面这些描述来自于汉利科·科内博和安德斯·爱瓦森发表于 2012 年 10 月的论文《部落、小组、章节和公会，发展中的敏捷 @Spotify》（Scaling Agile@ Spotify with Tribes, Squads, Chapters & Guilds）。

小组和 Scrum 团队类似，感觉像迷你型的初创公司。小组里的人坐在一起，小组里包括了设计、研发、测试、发布和生产需要的所有技能和工具。他们是自组织型的团队，自己决定采用什么工作方法，一些采用 Scrum 的迭代管理，一些采用看板管理，一些采用混合管理。每个小组根据所支持的服务都有一个长期的使命。

小组里有一个专职的产品主人，负责综合考虑产品的业务价值和技术潜力，确定工作的优先级。小组里也有一个敏捷教练负责帮助成员发现障碍，鼓励他们持续优化过程。小组里每个人都知道本小组的长期使命，并了解与使命相关的背后故事。

部落（tribe）是工作在相关领域的小组的集合，部落可以被当成是“孵化器”。部落有很大的自由度和自治权。每个部落有个负责人，他负责为部落里的小组提供最好的栖息地。部落里的小组都坐在同一个办公室里，一般座位彼此紧挨着，旁边有休息区可

以促进小组之间的合作。每个部落不多于 100 人。

章节（Chapter）由在同一领域里，工作上有相似的技能，属于同一部落的一小群人组成。每个章节定期开会讨论专业领域的问题或者是特定的挑战。章节的负责人作为经理管理这些成员，负有传统的责任，诸如人员成长和工资。然而，章节的负责人也是小组里的一员，参与日常工作，与每个人保持联系。

公会（guild）是一个更加有机和广泛的“兴趣共同体”，是一组想要分享知识、工具、代码和经验的人。章节总是在部落本地，公会通常跨越整个组织。例如，网络技术公会、测试公会和敏捷教练公会。

3.4　结论

本章重点讨论了可以影响组织结构的因素，解释了这些因素是如何对应用或网络服务的扩展性起到关键性作用的。把组织结构和扩展性关接起来，就像招聘合适的人并把他们放在合适的岗位上一样，围绕着他们建立一个支持性的组织结构也是非常重要的。本章还讨论了决定一个组织的两个关键因素：团队的规模和结构。

对团队来说，规模确实重要，太小的团队无法完成大的项目，太大的团队生产效率低、士气低落。在确定最佳团队规模的时候，必须要考虑四大因素，包括管理的经验、在职的时间、管理的任务以及业务的需要。要监控各种各样的警示信号来确定团队规模是否太大或者太小。不流畅的沟通、降低的生产率和低落的士气都是团队太大的表现。而满腹怨气的业务伙伴、进行微观管理的经理、过

劳的团队成员都是团队太小的明显表现。团队规模加大是直截了当的事情，但是拆分团队却是很麻烦的事情。当拆分团队的时候，要考虑的问题包括如何拆分代码库、谁做新的经理、每个团队成员的参与度有多高、如何改变与业务伙伴的关系。

本章讨论了职能型、矩阵型和敏捷型三种团队的组织方式。职能型结构是最原始的组织结构形式，基本上根据员工的基本职能来组织，例如工程部门和质量保证部门。职能型结构的好处包括管理的同质性、责任简单清晰、容易分配任务和很好地遵循标准。职能型结构的弊端包括缺乏单一项目负责人和不顺畅的跨职能部门沟通。矩阵型组织开始时更像是职能型结构，但是增加了第二个维度，包括一个新的管理结构。通常项目经理作为第二个维度。矩阵型组织的优点是可以解决项目负责人和沟通的问题，其弊端包括存在多个老板、个人主要工作焦点分散。最后，敏捷型组织提高团队的创新力，这一点可以通过市场响应速度、新功能的质量以及服务的可用性几个方面来衡量。

关键点

- ▼ 组织的结构既可以阻碍也可以促进团队的产出能力和对扩展应用的支持。
- ▼ 团队的规模和结构是组织的两大关键属性。
- ▼ 太小的团队没有足够的能力来满足业务发展的需要。
- ▼ 太大的团队会造成生产率的下降和士气的低落。
- ▼ 两个传统的组织结构是职能型和矩阵型。
- ▼ 职能型结构的优势是管理的同质性、责任简单清晰、容易分

配任务和很好地遵循标准。

- 矩阵型组织的优势是项目的负责人明确，跨部门的沟通有所改善。
- 敏捷型结构，特别是按照服务和架构组织的，会提高团队的创新力，这可以通过市场响应速度、高质量的新功能和高可用性的服务来衡量。