Part One

第一篇

布局时期

很多探讨软件开发的书籍都假设自己存在于理想的状态下：团队本来就应该非常专心且克尽职责地完成工作，并完全掌握任务的本质；他们收集得到完整的需求情报，建立设计规格，并导引软件原型（prototype）一次又一次的蜕变；他们请求使用者参与，使用者便会全力配合，而且非常认真地与开发团队共同完成深入的需求分析。所有的事情都是那么完美顺利。很不幸地，理想状态是不存在于现实世界的，在真正的软件开发项目中，您可能看不到一丝一毫属于完美顺利的景象，而且看到的恐怕只有一堆的问题。

不只是问题，还有可怕的挑战。优秀的开发团队在布局时期必须做的事情多如牛毛、广如大海，我们大略分为五个范畴：组织、竞争、顾客、设计、开发。布局时期的工作是多维的，而且必须在每一个细节中都能综合、兼顾所有预期的结果。

组织开发团队

我所谓的组织，是指集结适当的人选分别担任下列角色并参与设计：

项目管理（Program Management）——负责制定开发日程、与外界沟通、寻求技术方面的后勤支持。

软件品保（Quality Assurance）——测试与评估软件的品质。

程序开发（Development）——写程序、抓错虫。

产品管理与行销（Product Management/Marketing）——负责整个产品的形象定位，传递正确的产品信息给顾客，以及产品的上市发表、与传播界的沟通。

系统文件与使用者教育（Documentation/User Education）——负责以文字表达正确的产品使用方法。

(这里所谈到的设计，是指软件产品的设计，不是指写程序或程序设计。读者耳熟能详的“程序设计师”，译自Programmer一词，虽然这种工作有相当的设计成分，但这种设计比较不是艺术性的，而是工程性的。原作者甚至鲜少使用Programmer一词，而以Developer表达。概略地说，软件产品设计是厘定目标，事先定义出软件要做到的功能，本产品预定要满足什么需求，目标顾客是什么样的，主要的硬件或操

作系统环境等等。——译者注)

您不一定要将自己的团队成员划分成这五种角色（虽然我认为这是最有效率的做法），但务必要确定这些工作都有适当的负责人选。请注意每一种角色都有参与设计，如此每一位成员对项目都有整体性且清楚的认知，使每一位成员的目标是一致的。

如果您的开发团队无法合作无间，对于目标老是有不同的意见。那么，首先要做的是找出不团结的真正原因。

品保人员（QA）是少数民族？

如果品保人员认为他的工作是测试程序，而开发人员认为他的工作是写程序给品保人员测试，那就得小心了，这是一个警讯。这种情况会造成开发人员和品保人员之间的疏离，开发人员的优越感会使品保人员感觉自己是被歧视的少数民族，当然会影响到软件的品质。品保人员的最主要功能，是不断鉴定和评估产品的现状，是否在品质上和功能上确实遵行产品目标，而让其他的人员专心投入他的职务。

品保人员的评估工作是一项整体性、持续性软件开发活动中的一环，而不是偶尔来点缀一下。好的评估报告在

本质上应有客观的分析和衡量标准，如此才能导引软件产品符合现实的需要。这种导引的重要性是不容轻忽的。因为在开发过程中，开发人员可能因为一些偶发的小事或某种无关的灵感而不知不觉地偏离了现实的需要，暂时忘记了什么才是产品最该有的功能。品保人员的职责就是为软件的品质把关，以现实、客观而市场导向的眼光，不断地检视软件产品。

谁来设计产品？

如果项目经理、产品经理、开发人员不断争论谁有权设计产品或是各执一词，这是一个愚蠢的开发团队，只会关心自己的权威；然而，真正的权威是来自于对现实状况的精确掌握。产品设计的目的是将最好的想法列为产品的基础，每一位工作人员都应该为此努力。至于什么是最好的想法，应该在项目真正开始之前就通过实地检验。市场调查是很好的方法，不需要太多的时间或成本，就可迅速平息大家对于产品设计的争论。我们在下文中会讨论如何增进彼此之间了解，这也是解决方法之一，精确缜密的思考通常能使问题柳暗花明。

权力争夺战会迫使人们心胸狭窄，
互相竞争而不合作。

产品设计的争论是权力的争夺战，会使人们心胸狭窄，互相竞争而不合作。可能有人主张产品在这项功能上有所不足，结果焦点变成了功能而不是产品本身。一位有智能的领导人应该将争论视为组织出现问题的症状，而去发掘问题的根源，这比仲裁这项争论要有用得多。在这方面我的经验与传统所谓的智能刚好相反，我认为用职权裁定谁有产品设计权是毫无用处的，虽然改变正式的产品设计权或许有点帮助。

老实说，谈这个谁来掌权、谁来负责的主题并不有趣，而且解决这个问题非常浪费时间。组织健全的团队是适才适任，各司其职的；此外，每个人的除了各自专司的角色之外，还担任一般的角色（generic role），这一点也受到管理者的肯定，因此角色与能力之间的关系能够取得和谐。一般角色会使团队更能融为一体，互相支持。团队是一种平衡的生态，人与人之间的职权界线像是和睦邻人的关系，

于是，对每一位单独个体都有足够的尊重，整体的效能也达到最佳化。这种平衡的组织生态是自然生成的，强制性的外力干预只会造成负面的影响。如果团队成员彼此的关系无法做到亦邻亦友的和谐，领导人必须分析原因。通常如果利益分配不公平，或有多人争夺同一项利益，或是提供的报酬不够齐全，不平衡就会发生；有时候管理方面欠缺弹性也会造成资源必须用人为的方式重新分配，效果不一定会好。

在任何项目开始之前，项目经理必须弄清楚团队中有哪些地方需要特别的注意，尤其是团队的变动性（team dynamics）。就像团队中的每一份子必须清楚要做什么工作、有什么资源、目标在哪里，项目经理也必须设计自己所要带领的团队该有什么样子，自己该如何“激活”这个团队。因此，我们导出了法则1。

法则 1

Establish a shared vision
建立一个共同的目标

乍看之下，这何需赘述，但这却是相当困难的事。团队中每一位成员都必须非常清楚我们要做什么、成品会是什么模样、基本的产品策略是什么、什么时候必须完成。凡是与共同目标相冲突的看法都必须转化成一致，而不是把它消音。和谐的共识是绝对必要的，否则软件不可能做得好，很多事会复杂化而难以收拾。

建立共同目标的方法从独裁式的极端到放羊式的极端，中间有无数不同的方法，但在我们讨论这些方法之前，得先说明何谓目标，毕竟目标的建立已是现今衡量领导能力的指针之一。简单地说，目标是共同的希望，对未来的事情所描绘出来的景象。

目标一词常被政治家或经济学家使用。任何一位好的领导者，都有责任为其追随者创造目标。以我的看法，领导者对其组员心理状态的掌握与认知，是目标的开始，然后领导者对于组员之间复杂的心理状态认知、修正、建立共同点，融入领导者的个人特质，使得领导者与组员、组员彼此之间的防卫界线逐渐消失。有共同的目标才能建立团队的认同感和归属感，大家会觉得我们是一个整体，这是团队气氛的心理基础。

团队的认同感和归属感，能消除个体的自利行为。

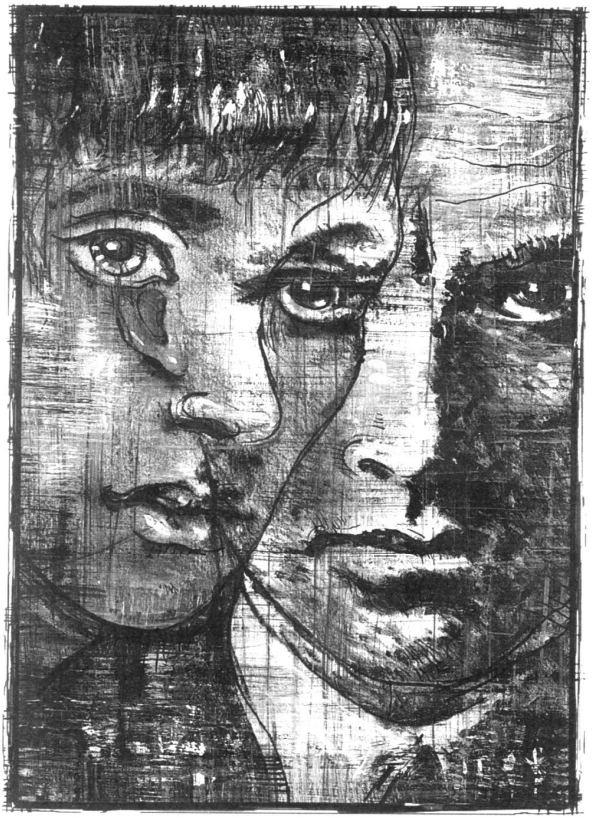
通常，共同的经验可以作为领导者的感情移入，领导者用各种不同的形式告诉追随者：我们过去是那样，我们现在是这样，而我们将来会是什么样。

不论是否有共同的历史，“成功的领导者”能在团队中营造共鸣，而“群众煽动者”则不能。当组员对领导者意见相左时，最常见的反应不是直接反对，而是：“是啊，我们也是这么认为，但我们该怎么办呢？”这时必要的努力、鼓励和妥协都可能发生。领导者应该自问：“如果换他们来领导，他们会怎么做？我该如何将他们复杂的情绪转化成行动力？”这些问题的答案能帮助领导人解决大部分的问题。

有远见的领导者会构思一个美好的前景，需要大家共同努力创造；而群众煽动者则是对现状不满，意欲去之而后快。有远见的领导者会带领不同心态的人朝共同的目标努力，有时为了长远之计而放弃近利；而群众煽动者则是

逞一时之快、急功近利的人。

领导能力与目标是来自领导者与组员之间的心理共鸣，若非如此，目标会流于虚幻，也无法带动团队迈向成功。



共同的目标

微软程序语言部门的故事

1992年的微软程序语言部门，情况实在不乐观。那年的四月，微软发表了C7。C7花了超过两年的时间开发，经历无数次的进度落后和品质恶化，似乎是一场永无休止的死亡进行曲。而微软的竞争对手已经抓住这个机会，在Windows版的C和C++上面获得很好的评价。C7在很多地方都不足以与之竞争，虽然它有精美的包装、动人的价格、优异的类函数库（class library）MFC1.0，和微软素以见长的最佳化执行码编译器(compiler)。这些优点只能使微软勉强维持不被淘汰出局，但很显然下一版非得大幅扭转劣势不可。

微软的C/C++开发团队经历了好几年的流年不利，光是项目经理就换了三次。丹尼斯·吉伯特（Denis Gilbert）是新任的资深开发部经理，而我是新任的行销兼使用者教育部经理；虽然我们都从未领导过这么庞大的团队（超过两百人），但我俩都充满斗志，企图有一番作为。杰夫·赫伯（Jeff Harbers）管理AFX小组，这个小组集结了微软的精英工程师，负责为C/C++的产品提供“所见即所得”（What You See

Is What You Get, WYSIWYG）的类函数库工具。他们的经验极为宝贵，虽然两组人马的文化差异刚开始难免造成点小摩擦。

程序语言部门是微软中最古老的部门，第一个产品就是比尔·盖茨与保罗·爱伦（Paul Allen）合写的 Basic 直译器。当微软进入操作系统与应用软件领域之后，程序语言的重要性相对降低许多。微软的重心转移之后，尽管微软的程序语言产品有占先的优势，但开发者的辅助工具方面则跟不上竞争者了。

在1992年，大部分的微软高层主管都希望重新振作这个部门，也许是被忽略得太久了吧，我无意批评他们，但我猜多少也是微软扩张得太快太广所致，无论如何，这使得我和丹尼斯肩负着更重大的责任，更强烈地希望把事情做好。

对我们的压力与期望不只来自微软高层，还有来自专业杂志的负面评价和使用者的责难，甚至微软内部同仁也轻视这个部门。大家都说我们缺乏目标，是一群笨蛋，要不然就笑我们 C7品质不良，完成日期老是拖延。就连比尔·盖茨的电子邮件也会带上一句——程序语言部门是全微软最笨的一群人。这一切

不只是没面子而已，我们实在该做点什么。

回想过去这段时日的发展和本部门堆积如山的工作，我们（主要是丹尼斯）发现本部门一直被以下的几个错误所蒙蔽：

- 我们无法完成任务，因为我们没有足够的能力。
- 即使我们每件事都如期完成，也一样会失去市场。我们没办法叫媒体不批评，或叫使用者不要抱怨。
- 我们的人力素质和数量可以做一个项目，最多两个。

跨出重要的第一步

丹尼斯征询过所有的经理人的意见，要求他们将各项项目作个评比，发现每个人都认为Caviar 是最重要的项目。Caviar 就是Visual C++ for Windows 3.1的计划代号。我们的结论是只要将Caviar做得好，又能在适当时机推出，就是打赢了一场重要战役，甚至会带来最后的胜利。第二重要的项目是Barrauda，基本上就是Caviar 的Windows NT 版。

我们必须立刻加强这两个项目，幸运的话两个都能完成，但若做不到，至少要集中火力在 Caviar。我们把所有的资源都集中在 Caviar，期望它赢得Windows 3.1的市场胜利。

于是我们把手上大大小小的项目逐一列出，要大家票选自己认为最重要的项目，以期形成初步的共识。但丹尼斯并未加强这个共识，他甚至并未再邀请组员发表意见，因为他已做了决定。我们明白Caviar一定要赢，虽然希望渺茫，大家都愿意尽可能贡献自己的力量。

丹尼斯召来了全体组员，向大家说明我们要如何打败竞争对手。我们必须将不重要的项目放弃，并且重新整编改组。

这是一个好的开始，但只不过是开始而已，其他重要的条件诸如团体共识、决断能力、清楚的目标、组员心理上的满足感等都还差得远。丹尼斯注意到整个团队都为无法打败竞争对手而感到愤怒而沮丧。他自己也有一股驱动力，想把这一切好好整顿。他感受到只要有人把胜利的希望带进团队，全体组员都会发誓：“ 我们要赢！ ”

Caviar 会成为 Windows 3.1 的 Visual C++。

丹尼斯的决心点燃了整个团队，Caviar非赢不可，虽然大家对于部分项目必须牺牲放弃不免情绪反弹。显然丹尼斯已经创造出成功的必要条件，但他还缺少

充分条件。

问题接踵而至

这群人从来不曾形成团队。我们也没有跨部门的协调组织，而且当初所期望的充分授权管理方式(consensus-style management)还处于很幼稚可笑的阶段。我们没有跨版本的产品计划，也没有详尽的技术规划，过去也从来没有成功的经验作为开发程序的参考。几乎每个人都高估了我们如期完成的能力，即使对于我们刚刚设立的新目标也是如此。但无论如何，弃车保帅的策略和改组行动确实为部门注入了新的动力，但结果还未见分晓，我们可不知道是否能一举奏效。

几个月过去了，事实告诉我Caviar实在不太可能如期完成，事实也告诉我非得如期完成不可。我们已决定在 1993年的Software Development ’93 West展览上发布 Visual C++ 1.0，同时铺货在各种通路上，要让竞争者来不及反应。这是我们成功的关键，也是我们士气和信心的关键，最后的期限是1993年 2月22日。

然而，这时的开发工作只能说是一团糟。我们仍不断添加零星的功能；好的一面是我们还在不断提

升系统效能，坏的一面是几乎无法将模块建构成完整的产品，甚至无法每天产生一次完整的执行档。开发人员把软件品保当成是黑箱作业，每天丢一份雏型给品保就继续埋头苦干了，根本不管品保人员为时数周的测试。

大约在推出前的四个月，我受到了赫伯的严重警告，他很坦诚地告诉我，如果我真要做好这个产品，必须改变一些做法。他使我了解到除了我自己，没有人能阻碍我做该做的事。我明知道该做些什么，却往往因为某些因素（大多是个人的因素）而退缩不前。我现在是行销经理，虽然过去在微软的 R&D部门曾经领导过15个项目，但从未加入过 C++ 的团队，如果我再不好好表现，我大概就是第三个卷铺盖走人的经理了。

赫伯的忠告使我明白我太消沉了。我可以清楚看到事情是如何一塌糊涂，我有责任改变它，而不是逃避它。赫伯向来以毫不遮掩、实话实说闻名，而且总是能够一语中的。我终于被他的话点醒了。

于是我和丹尼斯讨论了眼前的困境。我们其实无法确定该怎么做才对，但都明白非得有一番彻底的

改变不可。当天下午我们便召开了一次小组会议，这件事本身就极不寻常，抓这么多人过来只为宣布一件小事？是的，我们要在组员的心理上制造一个转折点。

我们为了求胜已经尽可能减轻不必要的负担，而我们仍然在市场上失利，在同类产品评比中输给对手，我们已经受够了失败的滋味——总是在进度严重落后的情况之下做出平庸的产品——而我们明明知道自己可以做得更好。只要我们同心协力，我们一定可以在期限内做出最优良的产品；没有人能阻挡我们，只除了我们自己（当然也有些不可知的因素）。我愈思考眼前的处境，以及它对微软公司、对我们的意义，就愈觉得斗志高昂。

我们有士气、充满活力，一切成功团队所需的条件我们都有了，只除了共同的目标。我个人有个想法：我们不但要追上日程，也要做出最伟大的产品Visual C++ 1.0，要令所有的人都刮目相看，尤其是那些一向对我们避之惟恐不及的营销人员。我们要扭转战局，让对手俯首称臣。

我的求胜心很强，但矛盾的是，我却不敢告诉

我的同事。没有人阻止我说出心中的想法，倒也没有人要求我说，这种时候个人的感觉似乎已经不重要。我害怕自己被嘲笑：“他是谁啊？新来的嘛，这搞行销的家伙想教我们写软件？”

然而我再也无法沉默，再不做点事情我在微软就混不下去了。那天下午我终于告诉全体组员我的感受，也问问他们的感受：“我受够了挫败，你们也是吧？作为全微软最差劲的项目经理，我恨透了，外界的批评令我觉得丢脸，你们也是吧？Visual C++是最伟大、最值得骄傲的产品，你们难道不这么认为吗？我知道我们可以如期完成它，而且用它来给对手一个迎头痛击，不只我一个人这么想吧？我们将创造微软的明日世界，我们会大有作为的，不是吗？”

结果，几乎每个人都有类似的感受，纷纷发言为我补充。我终于整合了共同的目标：我们要准时完成 Visual C++ 1.0，没有什么事情比它更重要。所有的资源都投注在这个清楚明白的目标；不必要的枝节可以牺牲，其他的项目（即使是NT版的Visual C++）都可以暂缓，全部的人、钱、机器，都押这一注，而且得立即行动！

会后我们紧接着要求每一个工作小组至少提出五项具体务实的建议，只要对 2月22日完成Visual C++ 有帮助的想法，请大家务必提出来。我获得了许多宝贵的建议，本书的第三篇“推出日期”中，就有许多很棒的观念是来自这次的建议。而此时此刻，大家的心理上和情绪上都有完成目标的强烈的决心，因此会主动告诉管理者该做什么，然后确实去做。而我和丹尼斯则负责提供工作环境，凡是对目标有帮助的，我们都全力支持，甚至包括一些背离传统却实际可行的构想，这就是员工领导的管理风格(management-led style)吧。

我个人觉得，在任何项目的执行过程中总有些“关键时刻”，使大家在心理上和情绪上因此凝聚成对目标的共识，对共同的目标产生共鸣，对事情的优先级形成清楚的认知。我无意夸口Visual C++的团队有多伟大或领导多卓越，我的目的是指出在大家一片士气低落时促成大家态度转变的关键因素。

首先，有人察觉到了团队发生问题，并勇敢地指出来。赫伯以他多年丰富的经验，独特的真知灼见，看出了我们在胡乱挣扎；虽然我们的做法基本上是正

确的，但却不够。赫伯没有任何义务帮助我们，但他仍然不吝直言。

赫伯在这件事情上，完全克服了人性的弱点。一般人很少会主动帮助并未求助的人，尤其是软件开发这种智能财产的工作。赫伯首先必须对自己的看法有足够的信心，并且在心理上要冒着很大的风险；然后，赫伯确信自己的行动对事情有所助益；最后，他必须不介意是否失去我的友谊。为了帮助整个团队，他必须冒着得罪我或整个团队的风险。

他的忠言刺伤了我的自尊心，激起了我的防卫心理，并且刚开始时不断和他争辩。他说我搞砸了一切，我应该彻底改变做法。我用尽自己的脑力和心力反击他的“帮助”。而他则以关怀和冷静的事实击退我的不理性。直到凌晨四点，我还特别记得他冷冷地说：“别管我的语气，别管是谁告诉你这些，忘掉你的骄傲，只要你听得进去我的话。”

渐渐地，我想我可以接受他的劝告了，我的情绪平息，思考也变得更有建设性。我开始觉得不必因为接受他的建议就感到自己不如他，我太骄傲了。想通了这一点，我的创造力和旺盛的活力又恢复了。

这个事件的教训是，我们应该坦诚将事实告诉那些当局者迷的人，我们应该告诉他们问题在哪里、他们的潜力在哪里。如果你的心胸够开阔，当你犯错时就会有人及时纠正你，甚至提供更有用的建议。如果你的忠告被曲解成太骄傲或自以为什么都知道，或者“你是谁？凭什么批评我？”，别管他们的防卫心态，直接告诉他们的情绪正在混淆事实。如果他们说：“什么，你以为只有你懂软件？”你只要在被拒绝前简单告诉他们，你只不过提供一些想法，又没有恶意，何不试着了解呢？

单纯的沟通的效果可能会昙花一现，你必须真的以彼此的情谊为赌注（有时甚至冒着失去工作的危险）来讲出实话，如果不是这样，可能很难突破对方的心防，而打从心底接受你的建议。

这种实话实说需要高度的技巧与智能，但却是建立共同目标的基础之一。经过不断发掘出最真实的情况，才是鼓舞团队士气的金钥匙。

然而，真话也得有人听才行。事实是最残酷，实话最难听，但不论是你自己发现或别人告诉你，你终究得接受它；缺乏自信心或安全感的人比较难接受

事实，而抗拒事实或情绪反应会使人丧失创造力，或是发生错误的直觉。

这是两个阶段的过程，你不只要学会接受事实，还得学着传达真实的信息。传播事实的最佳媒介是情感，这是你用之不竭的，多付出你的关怀和好意，让事实被正面地接受。即使你被拒绝，你也可以达到鼓舞对方的目的。

我们终能如期完成 C++ 的开发，而且它确实是一件伟大的作品；微软的行销部门也全力支持我们。我们终于打败竞争者，虽然压力和威胁还在，但我们毕竟扭转劣势，赢了这一仗。

往后的日子里，我们逐渐建立了版本控制的规则，我们可以每隔固定的期间发表新的版本，以及共享版本（请参阅法则3：建立开发多版本的技术规划），但最艰苦又最令人难忘的、团队精神最炽热的，还是那第1版的经验。

法则 2

Get their heads into the game

使大家主动投入

如果每个人都在认真思考如何使团队更有效率，这个团队自然就比较容易表现出高效率，反之亦然。听起来又是老生常谈，但事实上，让每个人的思考都动起来是一项管理上的重大成就，这需要有人用智能和努力营造出这样的环境，才能让它发生。

这种全体一致为整体设想的思维，其重要性绝不致被高估，但其困难性却常常被忽视。在团体思维之外，每一个人都有自己个人的想法，不能说不对，却常常是问题发生的根源。

每个人都有自己的思想、价值观、信念，也许不会表达出来，当然也不是每件事都非得说出来不可，但是团队中大部分的事都值得花时间沟通，特别是有人的想法非常敏锐时（请参阅附录）。每个人的想法都值得注意倾听，而且其所隐含的信念都值得花时间了解，尤其是有流言、断章取义或误会发生的时候。

人们的行为来自信念，人会为行为改变信念，也会为信念改变行为。

创造力的可贵

在一个需要创造力的环境中，好点子是愈多愈

好。你只要鼓励大家有这种共识，很容易造成深远的影响。假设大家在讨论一个问题时遇到瓶颈，没有好点子，你的做法是，规定每个人至少提出一个或两个好点子，然后你就会发现气氛动起来了。

有好点子是令人愉快的事，尤其当两个看似无关的点子，凑在一起突然变成一个崭新的、更复杂更丰富的好主意时，大脑中的愉悦中枢会受到刺激，所以创造力总是伴随着喜悦。

有人倾听或了解自己的想法也是令人愉快的事。最棒的是你的好点子被大家广为使用或接受，这本身就是心理上的很大的快慰与满足。想出好点子是内在的脑部活动，但当别人恭贺你或肯定你的点子，你会禁不住开心地微笑。

好点子是具有感染性的。真正的好点子应该像病毒般广为散播，每个听到的人都会感受到创造性思考的快乐。

好点子具有相乘的效果。当好点子打破某些错误的假设时，会有更多更好的点子被激发出来。

创造性思考能培养你的洞察力和头脑敏锐度，以及对新想法的成本、价值等的评估能力。

创造性思考只有好处没有坏处。虽然培养某种特殊的创造力不是件容易的事，但“想出好点子”本身很单纯，你多用它就会磨亮打光，一旦时机到来，创造力就会像核反应一样，绵延不断地增生扩大。

为什么组员会怠于思考或是不敢说出想法？

- 因为他们认为没有人会重视他的想法。
- 因为他们认为该由别人告诉他该做什么事。
- 因为他们认为这样做没有什么好处，只会使老板皱眉头罢了。
- 管理者只管发号施令而已。

如果人们的想法被采纳，他们就会愿意思考和表达想法。如果他们的新点子能够被团体接受，就表示值得实施，团体的力量会更强化好的想法，这就叫做“授权共决”(empowered consensus)。或许有更适切的名词表示这个抽象的观念，我选用这个名词是因为它表达出决策形成的两个重要过程：主动思考与尝试表达。

法则 3

Create a multi-release technology plan

建立开发多版本的技术规划

组员如果乐于参与我所谓的“技术规划”(technology plan)，会对未来更有信心。授权共决是所有信赖的基础，技术规划是它的结果之一。在民主式的授权共决中每个人都有投票权，参与技术规划的决策。更重要的是，每个人的想法都受到重视，被认真地评核。技术规划是团队行为的基础，在最理想的情况下，它包含了全体组员最好的想法。

技术规划是组员对团队工作的契约，也是团队的宪章。一年大约修改一到两次，以适时反映每一位组员的意愿，这么做会让组员更有安全感，更愿意信赖彼此和未来。技术规划勾勒出了大家信赖的未来的蓝图，因此，假定有一位组员对目前分配到的工作觉得不太喜欢，没关系，下一次改版时他可以挑个比较有兴趣的工作；或者有一位组员不喜欢现在的产品方向，可以暂时退出，等下次技术规划修改时再加入。

技术规划对于日程的掌握也有帮助：开发人员总是想把一些了不起的功能加入，常常和进度不能两全其美。如果有技术规划，开发人员会信赖未来，他有的是表现机会，不必急于一时。另一方面，开发人员很清楚该在适当的时间做适当的事，既不会负荷过重，工作品质也更能掌握。

当每个人都很明白在这个产品中自己该做什么，事情就很容易被精确掌握。其中的关键就在这种“割爱”的艺术。

一个技术规划的实例

有一次我受邀担任一个软件开发团队的顾问(不是微软)，协助他们建立技术规划。他们的资深项目经理出身该公司主要的技术部门，决定集中力量在一个五年计划(即使两三年的时间对科技来说就已经是全面改观了，我一般不建议定这么长的计划日程)。当时，他们正在讨论各种新技术能为工作带来的好处。

在讨论中有一个主题很有意思，是他们年久失修的关键程序。这个程序庞大而脆弱，有很多错虫，跑得很慢，其中有一部分的程序代码甚至已经超过10年。组员中没有一个喜欢它，很多地方根本不符合现代的设计原则，他们甚至不愿去碰这个大麻烦。

我想每个公司或多或少都有这类的程序：程序逻辑纠结交错、缺乏弹性、冥顽不灵，简直令人头疼。这种程序最后会变得无法维护，更别提加入新的功能

了，结果是大大限制了开发团队创造优秀软件的潜力。

我把这种情况称之为“软件开发的恶性循环”，你不可能在推出下一个版本的同时，彻底翻修旧程序。谁能承担得起重新经历一次软件开发的整个过程，只为了将程序改写，而这段期间很可能降低或中断客户服务，在程序还没有变得更有弹性而能搭配新功能之前，客户可能得等上好几个版本的时间。

比恶性循环更深一层的问题是：为什么会有这种怪兽程序？是技术总监失职，没有好好监控软件的品质？项目经理坐视程序败坏至此，竟不能防微杜渐？在你没有弄清楚真正的病灶并矫正它以前，就算重新修缮了现在的麻烦程序，同样的问题还是会一再发生。不要在损坏的地基上重新修筑倒塌过的房子，那是白费力气！

在这一组开发人马中有一群新成员，新的资深经理带着一群新的工程师，负责这段伤脑筋的程序代码；当然，这也为这个团队和这个程序带来新的活力和希望。他们打算重新架构这段程序，以便融入更好的技术，提高产品的使用价值。

这个小组经过一番研究之后，倾向外购一项全

新的技术，作为未来开发的基石。当他们向决策当局提出这项建议时，反应是两极化的：赞同的人认为即使外购也可能有兼容性的风险，但旧程序实在太难清理不如放弃；反对的人则认为团队有能力重整旧程序，只要管理当局给他们机会。

项目经理也搞得不知如何是好。这位资深经理在情感上倾向放手让属下好好努力，去清理他们数年来不断奋斗的怪兽程序，但她又实在不放心这群属下是否真的准备好迎接这项重量级挑战，尤其是想到他们过去不怎么样的记录。她不认为这是原先管理者的“错”，虽然是他们的疏忽以致程序腐化到这般田地。

一般而言，如果管理当局对技术是压榨而非培养，优秀的人会离开，苟且的人会留下，最后组织和技术一起走下坡。

在本实例中，有不少的新人（包括那位项目经理）觉得这套旧技术实在活得太久，该被淘汰了。也许大家所缺乏的只是管理当局的承诺，包括技术生涯规划、创造力的突破和员工福利等等。这位项目经理希望她的努力可以改变这种状况。

最后，她决定做一次技术规划，然后冒着被开

除的危险充分授权给开发团队，果然不只使得怪兽程序浴火重生，而且只花了一次改版的周期时间，也扫清了许多地雷暗礁。成功的背后，她也失去了几位主张使用外购新技术的人员。

下一次她修订技术规划的时候，她不只是让开发小组参与，还设法训练主管们也能参加并了解这个决策，让负责准时完工的技术人员梦想和希望有机会呈现在技术规划之中。她也鼓励其他的主管们参加这些讨论，这样做事的时候会比较容易沟通，也更不会错失任何好主意。

有很多方法可以制定技术规划，但最重要的原则只有一个。优良的技术规划有很多好处，也许“割爱”是最重要的。技术规划会让您知道将会走到哪里，更容易掌握该做什么，采取什么方法，如何就会更接近目标。而且，惟有完善的多版本技术规划，才能帮助您准确详估工作量。

修订技术规划

另一个我所观察到的技术规划实例也是加强共识的最佳典范。由开发小组和项目经理中派人组成的技术观察员，花了一两个月的时间起草大致的技术计

划（请参考法则5：刺探敌情）；然后这份草案被全组人分别传阅、加注意见、讨论激辩，最后被交给负责执行的开发人员（大约80位）。

同一时间，这份草案也在高阶主管之间传阅，他们会参照许多其他的技术规划后，定出多版本的产品计划。他们通常不会修改技术规划，因为他们知道这是所有组员的心血结晶，也是得来不易的团队共识。然而，他们发现了一个问题：这份规划看起来像是一大串互不相干的产品特色（feature），这样蔓杂的目标怎么能激励开发人员呢？怎么可能向全世界描述他们有这么样的产品——有几十个不显眼特色的产品？

然后经理们归纳出来，这些产品特色分属于五大类型：策略、竞争、顾客满足、投资，以及他们称作“典范性”（paradigmatic）的功能诉求。典范性的功能是指有计划地放在n个连续版本中的产品特色，最终目的是改变使用者的工作方式。这套分类当然不是说典范性的功能不属于竞争或策略，或是策略性产品特色与顾客满足感无关，这么划分只是为了比较精确地描述。

“策略性”产品特色与最基本的环境和限制有关。

譬如说是本产品支持什么操作系统？采用何种对象模式（object model)？产品需要什么等级的中央处理器(CPU)？支持哪种程序语言？此外，策略性产品特色也涵盖了公司发展其他产品线的全面性策略。例如公司卖的打印机，就必须写驱动程序去支持。“竞争性”产品特色基本上是为了对抗竞争者而设计的，特别是竞争者有而我们没有的功能。虽然不必每一项竞争者的功能都要放进来，但倘若其中有特别巧妙或特别吸引媒体注意的功能，就值得投资。这一类的产品特色是数目最少的。

“顾客满足性”产品特色基本上是大家已经耳熟能详，且顾客大都需要的。多去听听顾客的声音，产品特色就会更能搔到痒处。这部分的产品特色数目最多，但开发成本相对较低。

“投资性”产品特色是技术方面的前瞻性投资，其效益可能在未来的一或数个版本中都未必能明显看出。毋庸置疑，投资性产品特色是未来大放异彩的潜力，它能带来未来版本中的典范性产品特色。这个项目的重点是使开发人员善用每一分钱的投资。

“典范性”产品特色可以改变使用者的工作方式。

基本上这是整体开发人员梦寐以求的目标，在每一次改版中放进刻意设计的巧妙功能，逐渐引导使用者。通常这是行销部门持续性对外宣传的产品特色，它甚至能改变整个游戏的竞争规则。我们可以说：如果典范性产品特色成功的话，没有人能跟你竞争——除非他也有这项特色。

在本例中，项目经理将技术规划中所有的特色加以整理，分别归入这五大类，并决定在这五大类中分别要投入多少的资源、占整体投资的比重，并且做出相对应的技术需求分析，当然要符合内部资源和技术的分配。这一部分我就留给读者当成家庭作业，您自己思考一下，以您的条件、产业竞争型态等因素来看，五大类产品特色中，其所占资源比例应该如何分配呢？

然后，整个开发团队在接下来连续大约五天的时间，每天开会约两个小时，逐步讨论出下一版产品的技术规划的细节，整个计划就此拍板定案。这五天是大伙儿发表意见的最后机会（当然啦，事实上计划永远是可以变更的，这样做是为了强化计划的效力）。因为计划的制定是由下而上，不会有任何人跌破眼镜；因为管理阶层负责综合各方意见而形成完整的、

有系统的产品特色，组织目标会被大家接受；整个过程开放而民主，每位成员都获得充分的授权，最后的争议也会最小，很能共识凝聚，热忱也会很强烈。没错，整个团队会深刻感受到共同的目标。

这是我所见过最理想的规划过程。我写本书的时候，他们的产品已经上市了，真的是很棒的产品呢！

每个人对技术都应该有一份使命感，都有追求进步的内在欲望，然而新点子可能因为缺乏适当的分析或时机不成熟而被排除于计划之外。团队成员必须明白真正的工作内涵，并且能够理解有时候新点子虽好，但总得经过慎密的分析并证明实际可行又有足够的效益才行，而好的想法若能被团队审核过关、形成共识，最终还是会实现的。如此可以减少未经深思熟虑的提案，对整个产品的稳定性颇有助益。

最后，将产品特色分为策略性、竞争性、顾客满足性、投资性、典范性这五大类的用意是，给经理人一个管理上的指针，便于监督各项资源的运用以及诊断问题。比方说，如果失去了技术上的领先地位，管理者就应该加强典范性的投资，而相对降低其他各类的投资比重。

法则 4

Don't flip the bozo bit

别做笨蛋

我再说一次，软件是智能财产。必须运用智能，才能得到软件产品。若能用更快的速度结合更多的智能，软件的智能财产价值就愈高。这是显而易见的事实。曾经有人问我：“在软件产业中最重要的事情是什么？”

我毫不犹豫地回答：“让大家思考。”

信不信由你，大部分的人都不愿意思考。他们认为自己乐于思考，但事实上并非如此。保持脑袋空空很容易，在微软我们把这种人叫作 bozo，意思是笨蛋。永远没有人会注意笨蛋的所作所为，即使他真的有贡献，他也不会有任何份量。笨蛋当然是不可信任的，你对笨蛋惟一的期望是但愿他不要搞砸事情。

然而每个人都有可能是笨蛋。你自己反省看看，是不是知道自己在做什么，是不是觉得自己一点能力都没有？小心，笨蛋可能就是你。

在我的部门里，这种德行是不允许的。我要每一个人都全心全力地投入，每个人都得有贡献，每一个人都可以侃侃而谈我们的产品——如何在市场上竞争、何时出新版本等等，而且每个人对产品的看法都一致，不会众说纷云。

判断一个人是否在思考，最简单的指针是看他是否专

心倾听别人的看法，并且立即给予直指核心的响应。面对优于自己的看法，我们必须平息一开始的那种竞争心理或防卫反应，别人当然是经过一番智力淬炼才能想得出这些，我们应该公正地评判这些新的、可能很有价值的信息。

别让自己成为笨蛋

懂得思考的人当听到诸如批评或别人比较优秀等不顺耳的话时，会把自高自大的心理干扰过滤掉，并从沟通中接收正确的信息。他们能够避免下面两种心理现象：

第一种心理现象是防卫心理，让接受信息的人无法忍受别人的批评。在创造智能财产的工作中需要很多的情绪和创意投入（像是自己心血结晶的宝贝孩子，有一份特殊的情感），别人对产品或制程的意见往往会听起来像是讽刺。懂得思考的人在三思之后，会将自我的主观意识排除，然后接受信息的真实内涵，不过这种人并不多见。

将自己的意见强行加诸于他人者，
其实是笨蛋。

不懂思考的人不但不会请求别人赐教，反而在别人好意提供信息时过度防卫，而导致正面的冲突，所以无法对信息作出正确的判断，对事情毫无建设性。如果这种现象不断地发生，信息接受者会有两种可能的反应：一是这项信息确实很重要，二是这个笨蛋在将自己的意见强行加诸他人。结果呢？当然是后者。

第二种心理现象是相互排斥，发生的可能性更高。如果向别人提出建议，但对方却因为恐惧或其他负面情绪而排拒，久而久之，这位好意的建议人就会认定对方是无法沟通的笨蛋。

团队中有人无法沟通是极危险的事情，这会导致团体中人际关系的恶性循环，对团体能力造成莫大的伤害，而且几乎无法弥补。而且一旦管理者认为某人是无法沟通的笨蛋，团队中的其他人都会跟着这么认为。

对这个问题的药方是加强每个人正面的沟通能力，能够虚心接受别人的意见。如果你正试着传出信息，而对方似乎无法接受，那么换一种比较轻松的方式做做看，至少，试着向对方解释他的封闭令你难过。相反地，如果有个家伙不断灌输你“差劲的”想法或是“恶意”的“攻击”，放下自尊心彻底反省，是不是原始本能的防卫心理蒙蔽了你的判断力。如果你能在团体中实践这个准则，当你不小心犯错时就会有人及时纠正你，团队才能在和谐中进步。

死亡进行曲

在大多数的软件开发项目中，一开始多半都在做前一个项目的收尾工作。如果你的项目是以这种方式展开序幕

（很不幸几乎所有的软件项目都是这样的开场），这个过程被比喻为“死亡进行曲”（death march）。

为了赶时间，产生了太多没人能懂的程序代码。

当前一个项目拖得太久（也许是为了更前面的一个项目而晚了几个月才真正开始进行），也许因为项目经理忽略了该注意的地方，或是因为客户的强烈抱怨你食言而肥，或是前一个项目的到期压力太大而影响了软件的品质，或是虽然前一个项目还算顺利但人员筋疲力尽想暂时歇会儿，这一切的理由都埋下了延迟和品质不良的种子。为了如期完成软件，工程师明知其中有不少有错虫，明知程序写得多松散，甚至没把握程序能够正常执行，为了赶时间只好牺牲自尊心，放弃对“我的作品”理想的执着，他们无法以自己的作品为傲，以通过软件争霸战的残酷考验而自豪。在符合进度且产出稳定的开发团队中，最后每个人都会是团队的英雄，牺牲自己无私奉献终能完成任务，事实也确是如此，软件开发是多少人绞尽脑汁的成果啊！

然而现在他们认为自己应该得到一些适当的休息，有适当的奖励或充电的机会，做些自己有兴趣的事情，或是玩玩他们心爱的计算机。

重点是，他们无论是用爬的、用滚的，还是被鞭子赶的方式，毕竟如期完成了软件，也许不是那么漂亮完美，总是达成了目标。任何有开发时间限制的软件，到最后工程师大概都是除日程之外什么都顾不了，这是非常巨大的压力。然后紧接着又一个项目开始，不知道又要投入多少脑力，这造成各种有形无形的员工的反弹，其中最可怕的是那种江郎才尽（burn-out）的感觉，这将是团队最迫切也最严重的危机。

江郎才尽

软件工程师那种“江郎才尽”的感觉，就好像巴拿马运河的建筑工人染上疟疾一般，一发不可收拾。“江郎才尽”是你再也无法承受压力的感觉，那是一种极度的疲惫和沮丧，只有软件从业人员会染上，其症状包括：

- 确信这个软件正在榨干所有的人的精力。
- 觉得这个项目管理简直乱得无可救药。

- 一想到要出下一个版本，就觉得头晕想吐。
- 对于任何企图解决问题的作法抱着愤世嫉俗的态度。
- 完全无法沟通。
- 对计算机失去兴趣。

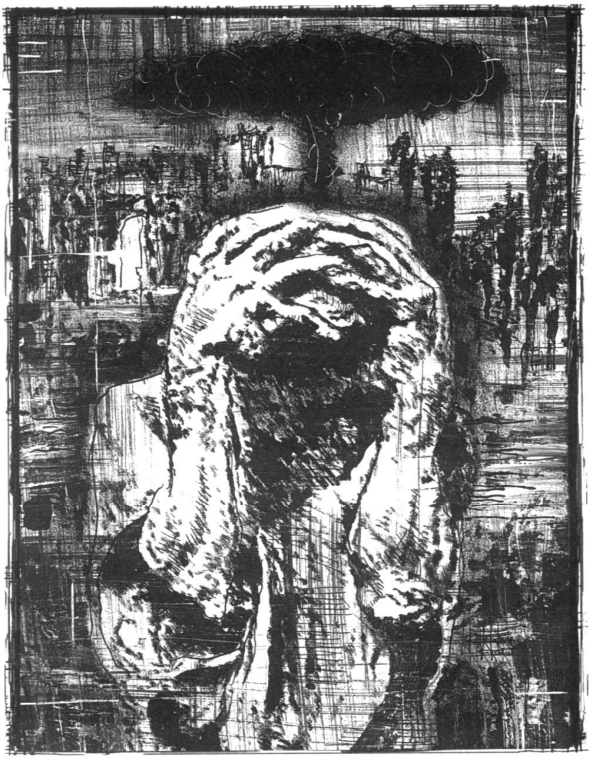
工程师染上这种“病”时，不论《PC周刊》(PC Week)或《信息世界》(Infoworld)(或是 Dr. Dobb’s和Midnight Engineering)等杂志看都不想看，觉得科幻小说荒诞可笑，虚拟实境(Virtual Reality)不过是人工智能的游戏，新版的 MFC手册懒得去翻，甚至对最新款的计算机毫无兴趣。基本上，就是软件工程师那种做出最好的软件的狂热，已经消耗殆尽或是转移到别处，只剩下颓废。

其实管理者也可能发生这种“病”，这是必须特别防范的，因为发生在管理者身上的症候特别容易传染到整个团队。

对一位软件开发人员而言，计算机的狂热是他最重要的动力泉源，对某些人而言这是一种终极的自我实现。就像是笔之于诗人，颜料之于画家，程序编译器是软件开发人员心灵之所系，用以发挥才情的工具。

当热情燃尽时，将自己的智能倾注于软件开发的那种无怨无悔，也就随之化为一堆肮脏的灰烬。

我和每一位软件从业人员一样，非常害怕染上这种综合症。如果不加以防范，这就像是艺术家的陨落一般，会使软件从业人员的职业生命骤然消失。



江郎才尽的感觉

法则 5

Use scouts

刺探敌情

在开始下一个项目之前，先刺探敌情吧！

“侦察员”（scout）是军队中先派出去侦察敌情的“探子”。他们负责先了解四周的环境和敌我界线、我方资源、寻找安全的驻扎地点、确认最好的前进路线，并且随时注意任何敌方行动的迹象。长久以来，当一群人在一个危险的旅程中，通常都会先派出侦察员去了解一下前方的路况。倘若软件项目不是大批精英的危险道路，就没什么好谈的了，所以当然得派个先知先觉者去为项目事先“侦探”一番。

倘若没有人事先研究出最适合的路线，各版本的开发过程就像在沙漠中胡乱游荡。

“侦察员”必须敏锐地嗅出任何软件产业的蛛丝马迹：软件开发团队与大环境之间的依存关系、正在开发中的操作系统新版本、与本项目有关的技术发展现况（也许还得另外找人来研究这项新技术）等等。“侦察员”能建立或修订多版本的技术规划（请参阅法则 3），他们访问顾客、

学习有竞争力的新技术（至少要有基本的了解），并且研究决定项目该以何种方式或路线进行。

“侦察员”必须能够提出硬件最低需求的建议书，分析所有的使用者需求及各项软件开发规范，准备原型产品，草拟未来的产品计划，以及建议下一版产品的重要特色。

“侦察员”的任务和贡献可以说是无限的。倘若没有人决定好最适合的路线，你的开发过程就会像在沙漠中迷失了方向，版本与版本之间毫无组织和明确的方向，注定了悲惨的命运。虽然有好的侦察员不能保证情报够充分，至少“侦察员”会让你不致盲目乱走。

侦察员的重要性

最近我与一家大公司的MIS部门有接触的机会。他们在过去三年内成功地将原本在大型主机上的终端机全部（有上千部之多）换成PC及Windows的作业系统与网络环境。可想而知这是非常艰巨的任务，需要大量的人力与金钱，和一段很长时间，整个公司在这个转换的过程中，受尽各种煎熬。

完成了Windows 和 PC的转换之后，MIS部门开始发展一套分布式的应用软件，以期充分发挥新科技

的效用，这是他们第一个重大的投资回收项目，预计降低的成本和增加的利润可达每年数百万美元。

这个工作团队大约由100个人所组成，主要是软件工程师与品保工程师，大部分都是公司的新人，或对技术还不是很熟悉，再加上对公司而言也是第一次开发这种关键性任务（mission-critical）软件，可想而知他们一定得克服无数的困难。在我与他们接触之初，这个项目的进度已经严重落后，并且大大超出预算，更糟的是眼看期限就要到了。

这项 PC 软件的项目自开始至今已经有三年了，当初他们开发软件时规划的硬件、网络协议等环境早就已经过时，目前所使用的软件开发工具和操作系统竟然落后了两个版本！这三年来，硬件价格大幅滑落，PC效能大幅提升，所要求的硬件运算能力也提高了几倍，他们使用的PC已经在市场上绝迹，但运行在上面的应用软件却无法淘汰。

我实在无法帮助他们解决这个难题，只能说这个实例证明了“侦察员”的重要性。当年他们几乎是凭着直觉和谨慎保守的作风来决定软硬件环境和工具策略：“我们尽量不要改变任何东西，反正它能用就好。”

倘若他们曾经好好调查一下计算机的发展趋势，倘若他们曾经考虑过兼容性的问题，他们就会知道必须使用最新的工具和最新版的操作系统，并且尽可能将未来的发展也估算在内，多花一点钱也是值得的，也不会因为当年决策错误而造成今日难以挽回的局面。

不过还好，这支勇敢的 MIS 团队毕竟最后还是圆满达成了任务，成功地完成了新的软件。这次他们学乖了，事先派了两位最优秀的组员担任“侦察员”，做了一次彻底的技术调查和完善的规划，终于在危机爆发之前将之化解。

老实说，对于类似上例的公司，我只能感到同情。计算机科技进步太快太剧烈，冲击着组织不够健全的公司，旧的系统难以割舍，新的技术又逼着你不得不进步。是的，使用第0版的软件也许会让你惶恐，但从第 7 版升级到第 8 版可能意味着重大的进步。如果你做决策之前，好好做过“侦察员”和完善的规划，而且知道这些对系统的重要性，你就会乐于接受科技带来的改变。

本书强调的基本观念是，我们并不企图减缓科技进步的速度，也不是要建立僵化的系统，而是要在改变中获得益处，将不断变化的科技管理得宜。行动总比停滞好，行

动力高的组织会有较高的适应环境能力。“侦察员”就是为科技的改变而准备的，如果你决定永远停着不动，那你不需要“侦察员”。

当然“侦察员”也可能带来问题。如果他对多版本计划没有非常清楚的认识，他可能不知道要侦察什么；他们也可能不懂这项任务的重要性而随便做做。“侦察员”必须要非常了解自己肩上所担负的责任：一旦选错了一个操作系统或开发工具，可能会害死整个组织。

另一方面，“侦察员”本身也可能因为这项任务的重要性而过度膨胀自我，看不起其他的人，自以为是“定义未来的人”。团队中的其余成员会因此而觉得沮丧，对未来的方向完全没办法掌握，觉得自己不受重视；甚至以为事情已经多得做不完，管理者竟然把人调出去研究无关的技术：“我每周工作 70 小时，而那个家伙悠闲地在看书!”这种抱怨在派最顶尖的人去当“侦察员”时尤其会发生。

团队成员对“侦察员”的工作不免嫉妒。当“侦察员”比当开发者好——比较光鲜亮眼、比较先进、比较酷。当几个月的“侦察员”似乎真是不错，你可以观摩别人的公司、跟厂商谈谈、玩玩软件原型、趁机多充实自己的技术

能力、有权力影响未来的方向。但是“侦察员”不参与软件开发，只负责做初步的规划、培养团队对于目标的共识。如果“侦察员”能够让团队信任，并凝聚出真正的共识，开发人员就比较能够自在工作，相信这些担任“侦察员”的优秀同事是在为大家的未来披荆斩棘地开路。

对开发团队来说，“侦察员”能够运用得愈成功，代表他们愈相信项目经理的领导，团队的共识愈强；不论是否处于技术转换的过程，对团体和个人都是一种很好的现象。

法则 6

Watch the ratio
注意人员的组成比例

项目经理经常犯的错误之一，是以为只要雇用软件工程师就好，其他的人都不必要，或是让软件工程师占整个团队很高的比例。也许是认为开发人员愈多，写出来的程序就愈多，这是错误的观念，项目的目的是完成软件，不是完成很多程序代码。在开发团队中，事实上有一些工作是不适宜交给软件工程师的。

在我的小组中，比例通常是6位开发人员，2～3位品保人员，一位项目经理，以及两位技术文件撰写人。在微软的各个部门中，这个比例会稍有不同，也许和您的人员比例也不同；但基本原则是开发人员和品保人员的比例不超过2:1。其实真正负责软件如期完成的是品保人员。当进度落后时，我们第一个要看的是品保人员：人数够不够？有没有充分授权？有没有确实参与设计？进度上能不能跟开发人员配合良好？能不能一有问题出现就立刻提出警告？品保人员和开发人员的理念一致吗？是不是跟开发人员过度亲密而放水？

对于人员的组成比例应该着重于有效的人数比，而不是实际人数比。一个健全的软件开发团队一定要符合上述的人数比例原则，平均每一位品保人员所支援的开发人员不超过两位：前者是思考并监督软件的状况是否达到预期

水准，后者专心写程序和抓错虫。注意人员的组成比例，可以帮助（但不保证）团队的运作取得平衡，记住，平衡才是你真正的目的。

法则 7

Use feature teams

运用特色监督小组

在微软的C++开发小组中，我们就用过特色监督小组（feature team）的横向组织。我觉得这是非常好的做法，对整个团队的工作品质产生了绝佳的影响。

如果你问一个品保人员他的工作是什么，他会回答：“监督软件开发的进度，确保如期完成，并且确保品质达到预定的目标。”他绝不会说：“测试程序。”那是肤浅的答案。品保人员的工作是如期完成软件，是开发“产品”，是去了解顾客，而且还知道技术规划中每一件大大小小的事情进行得如何；可以说，品保人员必须要管我们的产品、我们的市场和我们的整个事业。

我们的团队组织像一个二维矩阵，传统的组织模式为经，特色监督小组为纬。以经理、品保、开发、文件四种角色，虽然是一个传统的阶层式组织，但还算相当扁平化的；除此之外，这四种角色必须各派代表参加特色监督小组，每一项产品特色都有专属的特色监督小组，以确保每项特色都能照日程做出来，这个小组必要时可自行开会。

(特色(feature)一词，也有人为了听起来较大众化而译为功能，这是因为中文里的功能一词，意义非常广的缘故。事实上特色是指完成某项功能的独特方法，特别是与竞争者不同的地方，它适用的范围很广，您可

以说，采用Control-C.Control-X.Control-V来操作“剪贴簿”是微软产品一贯的特色，您也可以说，某软件具有随插即用的特色。——译者注）

特色监督小组运作模式有几个重要因素：分别是充分授权、赋予责任、融入任务、建立共识和地位平等。特色监督小组是我所经历过最神效的组织方法，我将它的五项重要因素讨论如下：

充分授权（Empowerment） 像特色监督小组这样的编制似乎比较非正式，无法像阶层式的组织那样容易管理，但它需要被充分授权才能发挥作用。譬如像开发人员，他们是“专才”，几乎专职负责某一特定的技术领域，若是让他们完全主导一项产品的设计显然太不智，如果加进一些别的角色，形成“通才”的局面，就平衡多了。单靠经理人的决定，恐怕不够全面，我看过太多好的（或坏的）决策，一下子就被另一位经理完全推翻，我倒是没见过特色监督小组把事情搞砸过。也就是说，只要特色监督小组能被充分授权，并且发挥它的作用，就可以确保决策的品质。（当然特色监督小组的必须挑选适合的人参加，本书附录中将说明这一点。）

赋予责任（Accountability） 我认为软件开发问题中

最有趣的就是赋予责任这个主题。我们讨论过的题目是：“你负责什么？”我的理论是，大多数人的思考都没有好好利用到，因为人们并不把提出新想法当成自己的责任之一（请参考法则4：别做笨蛋）。

首先，人们可能认为：“这不是我的责任范围”，因此少管闲事，不提任何有建设性的提议。如此一来会有很多不良的后果，尤其是当事情没有用语言沟通，会引起许多误解、猜忌、行动力的相互抵销，而最后只有用行动来抗议了。

只要单纯地请对方表示看法，这样的姿态很容易解除对方的心理防线。

其次，如果建议的来源真的说出了他的想法（可能要冒很大的风险），被建议的对象很可能产生防卫心理（又不是你的事情，管什么管嘛！），导致争端。有时候，防卫心理是非常隐讳的，特别是当这个组织的文化将防卫心理视为缺点时。讽刺的是，愈是成熟的团队或个人，愈会执着于现有的优良管理制度，愈会提出各种反驳的理由来

证明现有的做法才是最好的，不需要什么建议。

第三，除非建议者克服自己的攻击心理，否则这个建议可能永远到不了对方的心中。就像防卫心理是普遍存在一样，攻击心理亦然。通常提出建议的人都有某种攻击性或包含价值判断的态度，这也是绝大多数的人都有的弱点，而受到批评的人会感觉到威胁而更强烈反弹，于是提出建议的人立刻下结论：这人是个笨蛋，沟通的大门立刻关上。

有很多建设性的想法就此消失。我所想到惟一的解决办法，就是所谓的研讨会（workshop）。在研讨会中人们主动上前邀请同事提出批评指教，这样单纯的提出、接受、反馈的循环中，每个人都会受到批评、提出建议，这是比较容易让人卸除心理武装的气氛，比较容易对事不对人，如此单纯的信息传达让每一个人都轻松接受建议而愿意改善。令我惊讶的是，由于研讨会的练习使得组员熟悉沟通技巧，最后反而不需要经常开研讨会了。

我有过多次运用研讨会的方式解决软件开发问题的经验。方法像是个案研究（case study），由一位志愿的组员主持；这个案例当然是现在发生的真实情况，通常是关于组内的人际关系，主持人先描述案例的大致情况，有时故

意不说明当事人的名字，比方说：

——我实在无法接受这种想法。

——没有人看我的电子邮件（指建议）。

——某人根本没有把心放在项目上。

然后其他的人开始讨论，询问主持人关于此案更详细的信息，并理清一些盲点，或是询问主持人尝试过那些方式，之类的问题。最后事情会在讨论中澄清，问题也就差不多解决了。

前述的两种简单方法会产生两项很重要的结果（从来没有让我失望过），其一是这样的讨论会造成非常多的新点子伴随而生：你会看见主持人不断地说：“这个想法太棒了！”然后当场记在笔记本上；其二是这种案例研讨的结论通常可以应用在其他类似的情况，人们会发现，这就是我上个月讨论过的某案嘛！当类似的过去案例累积到相当的数量，人们就会发现本案也不是什么特例，然后就会发现处理这类事情的通则，把它记在笔记本上作为日后的参考。

在横向的特色监督小组中，一个人的成功或失败是完全透明的。

“赋予责任”有很大的好处，很少有应用上的限制。如果一个组织健全的团队成员，对于整个任务的过程——设计、开发、除错、品质、期限——彼此都负有责任，就会自然地互相给予建议和建设性的批评，而且用正面的态度接受。因为每个人都有相同的责任。一旦建立团队的责任感，这种对责任的认知会分享、传递到团队中的每一个份子。

融入任务（Identity） 充分的授权和责任感，使人具有控制权和影响力，愈能使自己与任务融为一体。控制权愈大，融为一体的一致性愈高。这种融为一体的一致性是开发任何好软件的先决条件，它会将个人的心理状态健全与否，表现在软件作品中。譬如说，有一个人心中有挫败感或是自我毁灭的倾向，就会在他所负责的软件功能中表现出来。

在横向的特色监督小组中，每个人会将产品当作他的

任务，而非狭隘的技术而已。没有什么人可以推诿塞责，一个人的成功或失败是完全赤裸而透明的。你不能责怪管理不良，因为你自己就是管理者，你不能责怪其他的功能支持人员没有好好配合你的产品特色，因为你们是相互负责。因此，你有责任自己找到答案，你有责任克服自己的障碍。

建立共识（Consensus） 共识是特色监督小组的气氛。因为大家聚在一起的原因是“产品特色”而非功能，由于责任是互相的，敞开心胸是安全也是必要的。我见过有些特色监督小组自动地重新组织他们的关系，建立共同目标、重新规划资源、适度修改日程，而没有发生严重冲突。即使有冲突，由于不是对人的或是出于私心，通常都很容易解决，不需要主管以职权介入。

地位平等（Balance） 由于特色监督小组的每一位成员都是不同的背景、专长，不同的工作角色和不同的观念，没有谁比谁优越的情形，所以每个人的地位都是平等的。通常人们对于不属于自己专业的领域，反而会有一些全新的看法，刺激彼此打破过去的观念限制或盲点，激发更好的创意。由于各人的意见来自不同的着眼点，就可以使产品的开发顾全更广的层面，而更周全、完美。不同角色的

人会关心不同的问题：

项目经理：团队目前的状况如何？开发过程顺利吗？领导是否有效而得宜？我们走到开发周期的哪个阶段？进度是否控制得当？需要别部门协助的地方是否已经取得支持？我们的目标明确吗？还有什么尚未完成的工作？本周的工作主题是什么？

品保人员：我能不能依照预定的时间自开发人员手中取得阶段性程序代码？程序有什么样的错虫出现？这些错虫对付得如何？有什么功能尚未开发出来？程序的执行效能如何？本周的产品状况是否合格，需要对谁提出警告吗？项目经理知不知道程序的稳定程度？我们团队的沟通如何？是不是每一个人对产品状况都有相同的认知？本周合理的短期目标应该是什么？

开发人员：本周我能不能完成预定的工作进度，将阶段性程序代码交给品保人员和文件人员？这一段程序代码写得够好吗？使用者需不需要这个功能？这个程序容易使用吗？程序是否执行够快体积够轻巧？我清除了所有的错虫吗？有没有人因为在等待我的工作进度而耽误进度？我是否完全了解本周的短期目标？我的工作是否符合策略上的方向？而且工作内涵对技术规划有无贡献，或只是个早

晚会被丢掉的垃圾？

产品经理／行销：产品特色是否吸引顾客（包括潜在顾客）？我该如何生动地表达这个产品的特色？它能否牵动顾客心中的心理或情感反应？我们应该如何修改产品，才能加强顾客的购买欲？当顾客听到我们的产品时心里会有什么样的反应？顾客在什么时候会使用这个产品？产品特色是否能加强我们与顾客之间的关系？媒体是如何报导我们的产品？开发这项产品特色背后的动机是什么？我是否完全了解产品特色及其重要性？准时完成的可能性有多大？我是否应该向公司说明产品的不确定性有多高（或是非常确定）？产品成功推出的机会点在哪里？

技术文件／教育训练：这项功能是否以更容易的方式使用？我对它的说明够清楚吗？使用者接口能不能设计到不必学就能操作？如果我现在还没有完成文件，会不会太迟？这项产品特色是否已经通过品保人员的核准？我对这项产品特色有什么感觉？我能不能用更简洁的方式来说明？其他的组员认同我写出来的文件吗？

虽然特色监督小组是非常的重要，但要成功地组织起来可不是件容易的事。在传统的阶层式组织中加入横向的特色监督小组是非常困难的转变，这种转变上的困难可能

来自团队的内部或外部。

在特色监督小组的内部，本身就会面对相当大的不确定性，大家不知道自主权的范畴是什么，只知道我们是一组人，应该在一起做事，但不确定的该用什么样管理方式。特色监督小组的本意是突破传统角色的藩篱，但是组员却会很自然地以原来的传统角色来为自己的定位，如此特色监督小组的作用就会明显受限。开发人员常常会主导小组的运作，导致使其他的功能不容易发挥。如此一来，特色监督小组的横向组织功能大概只能改善沟通，这似乎与它的高成本（组织小组、召集开会、物色人选、行政作业等等）完全不成比例。

在很多人的心中,所谓的团队只不过是
虚伪矫情的共识和假装一团和气罢了。

而且，人们总是要等到问题已经很严重时，才会成立特色监督小组。如果这个时候管理阶层放手让他们自行决定命运，组员反而会有被抛弃的沮丧感。向来不习惯自治的组员突然要组织起高难度的特色监督小组，一定会觉得

非常不安。特色监督小组的功能在这样的情况之下很难发挥，问题反而很可能雪上加霜。

特色监督小组本身也充满了矛盾和冲突，因为在很多人的心中，所谓的团队只不过是虚伪矫情的共识和假装一团和气罢了。

特色监督小组惟一不可磨灭的贡献是创意，代价是无数不眠的夜、对拒绝的害怕、对个人勇气的磨炼。于是，“冲突”便成了特色监督小组的标志，虽然它也是成长的动力。

所以，项目的初期往往看不出来特色监督小组的重要性，因为没有什么挑战性，特色监督小组看起来像是管理阶层在搞的流行玩意。

最后特色监督小组还是会发挥很大的效用，只要它能被适当的组织和赋予足够的权力。它有资源、有创意、有管理者支持，能够发挥惊人的力量，在关键时刻，他们会发现自己竟然能够搞定这么多、这么重要的事情。

当然在特色监督小组内也有它的问题。很多头脑优秀而充满创意的人不敢提出他们的构想，有一种内在的负面力量阻止他们将自己的天份表现出来。大多数人都会害怕在团体中被排斥，这种心理源自于做个乖小孩的童年经验：

由于父母很少会限制小孩的发展，因此小孩对于“不要这样”的信息特别敏感，为了保护自己，尽量表现得与大家一样，服从于大多数平庸者的价值体系，而不太敢表现自己的才能。然而，与众不同才是软件开发的成功要素。

这种自我设限的负面行为，虽然在一般的生活环境中是正常现象，但在正常的软件开发环境中却是病态。一般人都会为了在阶级系统中求生存而学会“善伺上意、察颜观色”，自然而然地会去迎合管理者的态度而改变自己的行为。然而，管理者也会不知不觉地发出“停止！不要这样”的信号（也许是出自盲目的策略或单纯的固执，就像父母对孩子的行为教育一般），导致属下害怕功高震主、得不偿失。管理者应该率先鼓励人们尽量发挥，对于优异的组员表示肯定和支持，至少对于不同的意见采取开放的、正面的态度。

另一方面，每个人对于这种意见自由也会有不同的反应。每个人感受到自己被充分授权，一肩挑起一件大事的成败责任，会自然地把这种情绪传达给团队中的其他成员。这种个人自由（personal liberation）在不同的时机会以不同的样貌呈现，也会以不同的方式强化，每个人都不一样，但这并不容易被观察出来。每个人都需要被挑战、被鼓励、

被栽培；弹性和耐性使个人形成团体，而以自己的脚步共同成长。这种成长过程虽然艰辛，却是成为优秀智能工作团队的先决条件。

在特色监督小组中容易发生奇怪的对话，像是谁能决定什么事，除了控制之外到底该有什么样的管理角色是管理者应该担当的。

我也同时注意到，每个人对横向组织（特色监督小组）的参与程度，恰好与他原功能组织（例如系统文件等角色）的成功程度成反比。如果整体组织的功能表现不彰时，其中某个功能部门的表现还算不错，致使这个部门的表现显得突出，那么在整个组织效能改善之后，原来的疏离感反而使得这个部门的人较不容易融入横向组织中。

我们再来谈谈特色监督小组可能面临的外部问题。参与者本身所隶属的功能部门管理者（如开发经理、品保经理等等之类的各式经理）有他原来的部门目标，现在要抽调人力参加特色监督小组，当然不免有些挣扎。也许其原

属的功能部门不是那么有创意，但在这“疯狂的特色监督小组”成立之前，他们对组织也有一定的贡献，或多或少都对特色监督小组抱持着对立的态度，毕竟，他们已在原来的专门领域建立起一定的责任感和权威性，现在要他们加入一个陌生的组织，做一些自己不擅长事情，而且在原本未被授权的环境中，他们凭着能力取得一席之地，但现在不需要了，却在新环境中被授权却更多，使得他们对原有的成功价值产生怀疑，因而使他们更难适应新的环境。

管理者知道如何以传统的方式推出软件产品，当他看到特色监督小组刚成立时所作的尝试，会觉得太幼稚了，会感到非常地担心：“他们这样做是错的，应该由我来做，或是由我来教他们如何做。”这真的是善意的关心，管理者无法忘记自己对产品的责任，会怀疑特色监督小组是不是疯了（我自己就曾经这么想），竟然让这些没有经验的笨蛋决定技术与产品的发展方向，并且管理者从前犯过的错误他们现在一样照犯！

管理者学习在“鼓励属下”与“控制属下”之间取得平衡，于是发生奇怪的对话，像是谁能决定什么事？除了控制之外到底该有什么样的管理角色？如果不是管理者该做的决定，那么他要如何负责？如何自处？如何定位？

当年我也曾经怀疑过，和其他多位资深经理怀疑我们为什么要大费周章地成立特色监督小组，我曾经在深夜醒来，自问我们究竟为什么要这样自找麻烦，想出这种疯狂的主意使每个人都全心投入。渐渐地，我们了解到管理者的角色应该是教导、挑战、鼓励，并肯定这个创意思考的过程。如果我们的观点是对的，事实自然能够证明它。在授权给特色监督小组的同时，经理挑战组员假设的前提，让他们能够自我检视自己的动机和行为，协助他们达成共识和化解冲突，并让他们明白管理者了解他们，会支持他们的决定。我们训练组员自己去追求效能，这是非常基本的要求，因为我们要让组员自行负责，让组员有能力决定，并且以自认最理想的方式去实现。

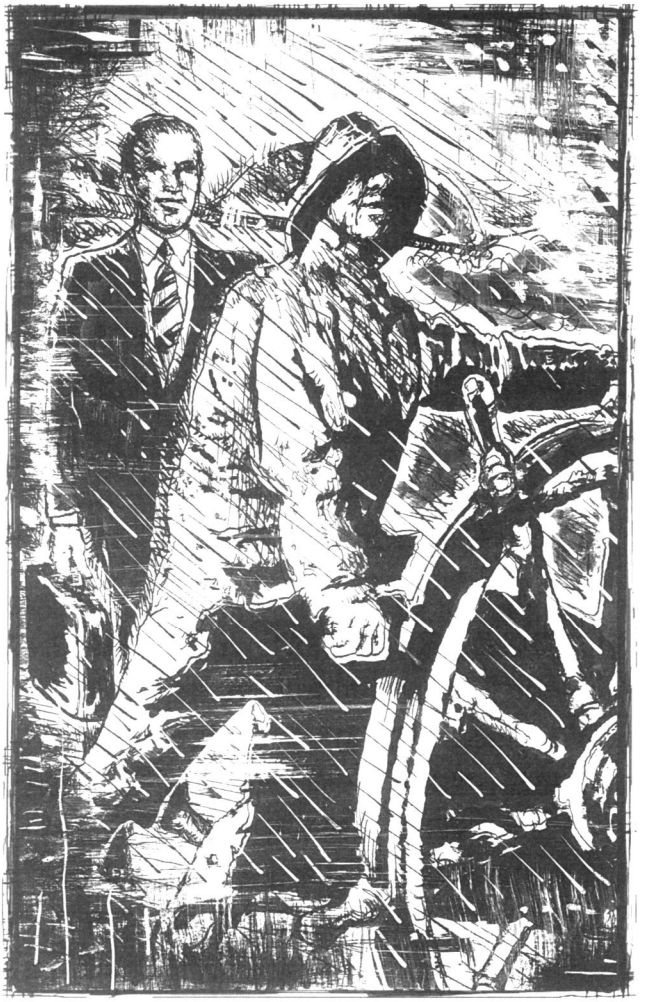
权威是来自学识，而非职位。

当然，这整个过程是循序渐进的，就某种意义来说，它仍然不断地在发展。有很多时候，小组希望由经理直接替他们做决定和设立目标；也有些时候是经理不当地命令小组做事。传统性的功能组织和横向特色监督小组之间，

没有全然的界线。但是在这一切都在转变中，我清楚地感觉到大家都朝着正确的方向在进步。

在一个理想的项目中，基本上有两种角色存在：创造者（creator）和推动者（facilitator）。创造者是一些专业人员，例如开发程序、行销、软件品保和文件撰写；而推动者则负责凝聚团队共识和维持最佳的开发环境。在这里可以无忧无虑地尽情发挥创意，有充足的资源解决问题和实现理想，组织的运作非常有效率。推动者就是项目经理或一般的管理者，他们的能力不直接显示在产品中，但却是推动产品成功的保姆。

推动者必须像创造者一样对最后的成果负责，而创造者也必须像推动者一样对团队共识负责。让两种角色互相负责是很好的做法，可以互助互补。传统阶层式组织的重要性逐渐降低，权力来自于知识，而非地位，这是一项革命性的突破，值得所有的软件开发组织深思。



管理者的角色是舵手

法则 8

Use program managers

项目经理的职责

项目经理是软件开发团队的一份子，他的职责是：

- 领导大家定义出一个成功的产品。
- 引导大家对产品注入深切的期望和信念。
- 带领团队将理想实现，变成可预见的产品诞生。

要定义“项目经理是什么”，倒不如从定义“项目经理不是什么”，还比较简单些。一位项目经理实际所扮演的角色并不是一般人直觉上想到的工作性质而已。

项目经理并没有（或只有很少的）正式的权力或地位，至少刚开始时是如此，所以项目经理通常会为此感到非常地焦虑。笨蛋项目经理可能以为他的工作是写出软件规格，让其它人去写程序、测试程序、撰写文件等等，最后完成项目经理心目中理想的成品。在最好的情况下，别人会认为他天真；在最坏的情况下，别人会认为他愚蠢、成事不足败事有余。在项目经理可以对团队有任何的价值之前，不应该有任何直接的控制权；幸好，一个健全的开发团队不会让这种事情发生，组员会适时阻止项目经理不当地直接控制。

当然这类事件会导致在项目经理心中有些气愤和挫折，因而要求上级授予更多的权力，如果上级愚蠢到真的这么做，就会导致更严重的愚蠢事件（请参考法则 9：要

权威，不要霸权）。

> 通常是等到情况已经坏到无可救药时，项目经理才会明白“政治”只是合法的借口。

对于这类项目经理的挫折感，我见过很多不同类型的反应。最常见的一种是项目经理开始对他真正的角色——领导——做出负面的、反叛性的行为，这种行为通常是通过对“政治垃圾”（political bullshit）的厌恶来表现。除了对科技要有一定的知识之外，软件开发团队的领导人还得具备对人性高度敏锐的观察力，必须能够看出在团队外在行为背后那些隐藏着的情绪因素。可想而知，有些专案经理为了避免那些剪不断理还乱的人性问题，开始寻求比较“清爽”的自我形象，而在软件公司中免不了的科技挂帅文化，很容易视“政治”为畏途。当然，驾驭技术要比掌握人性要简单多了。然而，当人们抱怨政治的奸诈诡谲时，其实是对不良领导的反感所致。不幸的是这种不满的情况比比皆是，健全的组织却少得多。于是失败的项目经理把

过错推到“政治”这个字眼。然而，大概要等到情况已经坏到无可救药时，专案经理才会明白“政治”只是合法的借口，他应该用领导代替控制，才懂得为自己的偏执负责。有谁愿意身处于乌烟瘴气的组织环境？但是当项目经理或任何人伤害组织的健全性时，那就是走向失败。

虽然刚开始时项目经理对自己的角色非常不确定，不能肯定自己对项目的贡献度，还对政治嗤之以鼻，不过当他了解到应该以领导代替控制时，就会觉得一切都得心应手。项目经理应该是以栽培和教育的全方位领导，来引领软件的发展。

项目经理应该要有技术的背景，而且必须在两种层面非常专精：一是对开发产品所使用的技术很熟悉，二是拥有建构软件的技术领导能力，而后者是本书主要的讨论范围。项目经理必须精于哄骗、驱策、鼓励、要求他的团队做出最好的软件和表现出最好的工作效能，他清楚知道软件制作过程中每一项的投入和产出细节，他必须懂得用最好的方式定义产品和维持健全的技术。最后，项目经理还必须是团队的发言人，面对媒体、客户、以及整个组织。

项目经理是维系团队灵魂的关键人物，他应该是擅长

沟通和倾听的人，具有设身处地为他人着想的本领。总之，项目经理是软件开发的核心人物。

团队的精神

在此我强调一个观念，在后面也会再重复提及：软件代表着创造它的团队。你想了解这个团队？看看他们的软件就知道了，反之亦然。我特别强调这个观念是因为这是软件开发管理的基础。在某一个时间点或是从某一段描述中，或是从这个团队的行为，常常无法准确判断这是个什么样的团队，但软件不会说谎。软件会忠实地展现创造它的团队：一切优点和缺点，天赋和天谴，从潜伏期的小病到最极致的团队合作。若对这个团队有任何疑问？他们的软件作品就是答案。如果你的询问对象是人，你还得注意他的表情和肢体等等隐性语言，但是如果你仔细研究他们的软件作品，你不必问任何人就知道他们团队有没有问题。软件会自我表达，它绝不说谎，也不隐瞒。

这项理论是软件开发管理的基础，可以用一个恒等式表示：

团队＝软件

(team = software)

基本的原则是：如果你对团队有任何疑惑，去看他们的软件；如果二者一致，你可以相信自己看到的团队现状。相反地，如果软件未能表现出应有的水准，表示团队有问题，不论团队看起来多么健康，你都应该去分析、去发掘，找出问题解决掉。

团队的精神

团队等于软件，不错，但团队是什么呢？你是以什么方式或风格领导这个团队？我相信软件开发的关键是与“团队精神”（group psyche）保持密切的连系（Psyche是小爱神丘比特所爱的美少女的名字，原意是灵魂或精神，这里的灵魂不是宗教上的含义，那是soul，精神也不是强调团队的合作协调，那是teamwork，group psyche是指一群人所共有的中心思想和心理状态）。这样说很抽象，下面是“团队精神”的具体描述：

1. 一群人同心协力，集合大家的脑力，共同创造一项智能财产。

2. 个人的创造力是一种神奇的东西，源自于潜在的人类心智潜能，它被情感丰富，而被技术束缚。

3. 一群人全心全意地贡献自己的创造力，结合成巨大的力量。结合的创造力由于这一群人的互动关系、彼此激荡，而更加复杂。

4. 这种复杂的情况之下，领导变成像是人际互动的交响乐指挥，辅助并疏导各种微妙的人际沟通。

5. 在团体中的沟通和互动是正确而健康时，能够使这一群人的力量完全结合，会产生相加相乘的效果，抵销互斥。沟通顺畅能使思想在团队中充分交流传达。

6. 团队工作的品质比时程更重要，而作品的伟大是需要对“团队精神”特别加强，才能达成。“团队精神”可视为个别成员精神的平均值，而个人的精神（psyche）则是使他能感觉、能思考、能推论的内在力量。

7. 倘若忽视了“团队精神”，则只会有平庸的成果。

那么，如何照顾“团队精神”呢？我的答案就在本书中。

法则 9

Be an authority, not an authority figure

要权威，不要霸权

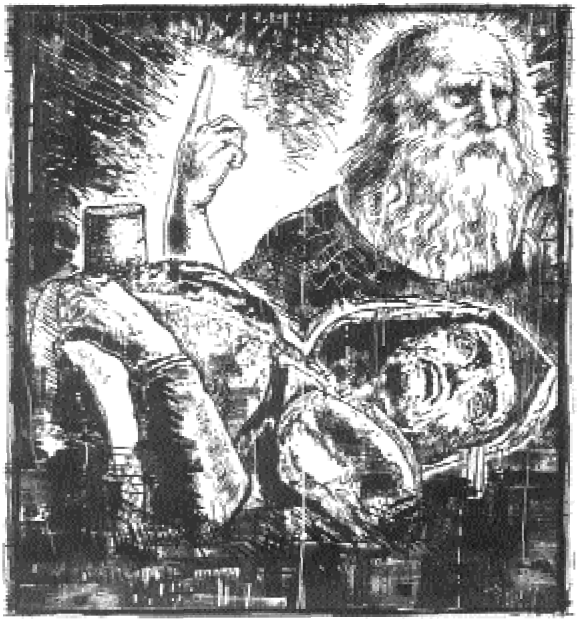
在组织中，大多数人都会寻求一种权威，而不在乎这种管理风格好或不好。这种心理现象可能是因为人们希望重建来自家庭的权力结构：父母亲控制并保护孩子，而孩子依靠父母。无论这种需求来自何处，人们迫切地需要权力和依靠的对象，以致将领导者投射成权威的化身——也许另一方面管理者也有迫切的掌权欲望吧。

权威的目的是让每一位团队成员都有自己的专业权威，和团队的专业自信，这才是管理者真正的权威。

在任何社会中，人们总是会将群体投射到自己所熟悉自在的心理结构，这种倾向在工作团体中特别明显。人们会很自然地将工作中所遇到的他或她，比拟成过去相处过的人物，特别容易将管理者当成父母，更有甚者，团队中最优秀的人可能会将组织中的高层人物比拟成某位权威人士。这种奇怪的不理性现象必然造成管理者有一种“ 霸权 ”的形象产生。

就像在家里一样，一个健康的组织一定会经历三个阶段：孩童期、青春期和成熟期。在团队成立之初的孩童期，组员通常会不知不觉地将管理者当成无所不能的权威人物：决定事情的人、保障资源的人、主宰奖赏和惩罚的人。而组员则早已预设了自己的角色：服膺阶级。由于他们对

权威的信赖，在组员心里会投射出一大堆“应该如此”的模式到管理者的形像上，他们会说：“事情应该是这样。”并把团队与管理者之间的关系看得单纯而天真。这种管理者“应该如此”的模式包括了领袖特质、权威和成就等等，其实没什么实质上的意义，睿智的管理者应该避免浪费时间去探究在每个人心目中的权威形象。



让每个人都有权威，而
不是让管理者独霸权力

人们倾向将管理者当成像父母般的权威。

如果管理者接受了这种幼稚的权威形象，和不健康的“领导者－团队成员”关系，整个团队就永远停留在孩童期。千万记住我们的目标是让团队中的每一个份子都有权威，而不是把权威集中给管理者。管理者也许要花一点时间才能让大家接受这个观念，也许在过程中会有人抱怨，但团队一定要有对权威的正确观念才能成长。对于一个不成熟的团队，为它描绘目标比凝聚它的共识容易，你可以召集所有的人在一起，告诉他们你为他们拟定的目标是什么，这是短期的做法，长期来看，一定会有技术或产品的专家脱颖而出，主导大家对产品的目标，而管理者必须相信并支持这个目标，并且将它传递给整个团队。

充分授权

我很想找另一个字来表达“充分授权”(empowerment)的涵义，因为这个字现在已经被用滥了。不论它

用什么样的名词表达，这个观念永远是创造智能财产团队的中心价值。人们经常分不清楚准许（permissiveness）和授权（empowerment）的差异，前者是“让组员去做任何他认为最好的事”，后者是“让组员能够思考，然后尽全力去做”，完全是不同的两回事。

充分授权是让组员能够全力发挥，无所阻碍，是为他们清除各种不同的障碍，让他们以自己的力量达到某种成就。自由是充分授权的基石，让他自由判断和实行判断，自由地思考和表达他认为应该思考和表达的事情，自由地冒险尝试，而不必怕受到额外的惩罚。充分授权是充分学习的结果，而不是忽视他，让他胡搞瞎搞。当管理者对属下说：“这是你的决定”时，他必须已经提供完善的支持——训练、信息、资源——让属下能够做出正确的决定，这样才算是充分授权。否则的话，让属下做决定等于是弃他于不顾。

如果每个人都能获得充分的授权，那么在冲突发生时该怎么做决定？这是一个理论上的问题，实际上不会发生。在充分授权的环境中，不是无政府的混乱而是事实和才能领导一切，每个人都有安全感，因

此而会放弃因为骄傲所造成的愚蠢、去除自我的狭隘心态，于是软件的设计开发以及决策问题就变成是单纯的资源分配运用。一个充分授权的团队有能力分析各种方法的优点和缺点，并且能够选择对目标最有利的方法，决策没有对或错，而是在产品功能、资源与时间三者之间的权衡（trade-off）。

你知道如何组织团队，辅导团队成长，由于你提供好的技术和程序等等环境，充分授权团队如期完成软件产品，不断成长再成长，这一切造就了你个人的权威。你的见识是“团队精神”、“团队作品”以及二者之间的关系。你真正的权威是来自这些事情，而不是公司赋予你的阶级或是地位，也不是属下的权威依靠需求所投射出来的虚象。但威权是人们对权力形象的需要，那是向人们保证有人关心他，有人保护他的利益，有人让他尊敬崇拜，而不是真正的权威。

那么，真正的权威会有什么影响力？那是你知道自己在做什么，让组员了解你在做什么，并且让组员和你一起努力，大家一起为目标创造价值。

一个孩童期的团队自然不太懂得接受充分授权，有人会抱怨没有目标，缺乏明确的目标，并且不愿去做困难的

决策等等。这种时候，要激发团队成长最好的方法是集中心力在软件如期完成的目标上，管理者只要对软件产品进度和品质负成败责任，让每一个人都卷起袖子干活儿，不要去理会什么阶级和分工的界线，专心使软件如期完成，将目标单纯化，排除一切干扰。管理者以这种姿态融入团队的工作，很自然会打破霸权迷信，引领团队自我成长。这是“权威”，是行动，是一起创造软件的工作，绝不是谁决定什么、谁决定奖惩的“霸权”。