第一章
公司组织与管理

寻求既懂技术又善经营的精明人士

在公司组织和管理方面，微软始终遵循着这样一项策略，即坚持挑选那些既懂专业技术又懂经营之道的精明人士担任领导职位。我们可以把这项策略归结为四个原则：

- 聘请一位对技术和经营管理都有极深造诣的总裁。
- 围绕产品市场，超越经营职能，灵活地组织和管理。
- 尽可能任用最具头脑的经理人员——既懂技术又善经营。
- 聘用对专业技术和经营管理都有较深了解的一流职员。

从理论上来说，这些原则并非微软独家所创；但从实践上看，它们对整个计算机行业有着深远的影响。很少有公司能够拥有像比尔·盖茨这样既精通专业技术又知道如何把它们转化为数十亿美元资产的总裁。事实上，许多公司在采用与微软类似的管理方法，即围绕产品市场，超越经营职能进行组织和管理。但是，他们往往不能同时维持一个强大的产品市场和一系列领先的核心技术。作为一个处于迅速发展的主导行业中的公司，微软在加强组织管理以适应不断变化的市场方面做得颇为出色，有时甚至可以领导市场潮流。它逐步建立起来的技术力量已经能够满足巨大且日益增加的产品系列的需要。

许多公司雇用或提升职员的条件仅仅基于对管理能力的考虑，而并不看重他们的技术知识和经营管理相结合的程度。微软在选拔经理人员时，把他们的技术知识和运用技术知识去赚钱的能力放在首位。虽然这种方法可能导致公司缺乏素质良好的中层的职员管理人员，但这对微软在竞争激烈的高科技产业中进行软件开发是大有裨益的。此外，通过新一轮的雇用和汲取，微软可以不断地扩大已经具备的技术基础，比如为用户的软件增加新的程序以及开发信息高速公路产品和服务等等。

微软对筛选职员，尤其是软件开发员条件特别苛刻。在挑选软件开发员时，往往在所有的应征者中只雇用 2%—3%。在考虑聘任经理人员时，微软则重视选拔那些既拥有较高的技术能力，又睿智明理，知道如何运用自己的技术来为公司推出新产品的出色职员。

在很多方面，微软的生产单位经理们就像两千年前古罗马军团中的百夫长那样运作。他们都精明能干，不需要过多的指示。对于突如其来的机会和威胁反应迅速。他们在组织机构中拥有足够的资源，独立运作；百夫长们平时各尽其责，只是偶尔进行汇报，但他们只处于一定的限度范围内而不能越雷池一步。自然，首领也深信这些百夫长——还有他们的部属——是在为整个团体的利益而奋斗的。微软的业绩证明：虽然无法做到使每位用户都满意，然而毋庸置疑，微软拥有一位杰出的领导者，一支卓越的高层管理队伍，一群勤奋努力的职员；他们不仅对专业技术和微机软件行业都有很深的了解，也知道如何去赢得成功。

原则一：聘请一位对技术和经营管理都有极深造诣的总裁

许多成功的企业都归因于总裁的技术知识和商业敏锐感及其领导和管理才能。在今日美国产业界，比尔·盖茨也许是最精明的企业家，同时也是最被低估的经理。无论是在对软件和计算机技术的独到见解上，还是在创建并维持一个庞大的盈利企业上，他都表现出了惊人的天赋。在过去，他曾被认为是一个脾气暴躁喜好吵闹的人，经常对其职员横加指责（甚至大声咆哮），但他终于随着公司的发展逐渐成熟起来。他持续不断地对新产品和业务的选择进行引导，尤其注重关键产品的特性。不过现在他更多地依靠着几十名高级管理人员和技术领导，他所创立的各种正式的和非正式的机制也正帮助他指挥着微软这架庞大的机器。

盖茨其人

威廉·享利·盖茨 1955 年出生于华盛顿州首府西雅图的一个富有家庭。他的双亲都不是技术人员：父亲是一位律师，母亲是一位教师。从各方面来看，少年盖茨和成年盖茨十分相像。他的传记作者把他描绘成“一个精力充沛的年青人”，喜欢在椅子上晃来晃去，如同我们采访他时所见。教过他的一位老师说他“彻头彻尾是一个淘气包”。他童年时兴趣广泛，喜欢玩各种各样的游戏，其中有一个称为“冒险”，参加者们为控制地球竞相争斗。[1]

盖茨首次接触计算机是在 1968—1969 年，当时他在私立湖滨中学读二年级。学校有一个简单的电传打字电报机，可以限时接通一台计算机。盖茨在学会了 BASIC 编程语言之后，又与一个十年级的电子专家保罗·艾伦合作学会了更多的编程知识。当盖茨 14 岁时，就已和艾伦一起通过编写和测试计算机程序来赚钱了。不久，在 1972 年，两人创建了他们第一家公司“Traf-O-Data”，并且销售一种用以记录和分析机动车辆交通数据的小型计算机。

1973 年，盖茨被哈佛大学录取，而第二年，艾伦就中止了在华盛顿大学攻读计算机科学学士的努力，离开了校园，在波士顿地区的豪尼威尔（Honey Well）找到了一份工作。以下的故事是人们所津津乐道的：包括艾伦如何在 1974 年注意到了《大众电子学》杂志为 MITS 计算机公司的新型牛郎星微机所做的广告，以及艾伦和盖茨又是如何利用哈佛大学的计算机设备编写 BASIC 的一个版本的。

盖茨在 1975 年离开哈佛，全身心地投入到为牛郎星微机（后来为其他的个人机）开发程序语言的工作中去。他和艾伦一起搬到了新墨西哥州的阿尔伯克基市，和 MITS 计算机公司相邻。同年，以盖茨为首，总共大约 40 至 60 名成员一起创办了微软公司，当时他在公司的第一个产品——微软 BASIC 的开发中占有举足轻重的地位（参考附录 1，微软大事年表）。

青年盖茨有许多杰出的优点。他聪明，极富进取心，与他的朋友们一样。他能高度集中注意力，把握住自己感兴趣的事物，尤其是计算机和为实际应用编写程序。盖茨为人们所展现的计算机世界，不仅仅是追踪记录交通数据，更为重要的是人们可以坐在桌旁运行他的软件。在个人微机时代即将来临之际，这无疑是技能和雄心的伟大结合。

局外人士和微软的职员们对盖茨的描绘惊人相似。他被形容为一个幻想家，不断地蓄积力量，疯狂地追求成功，凭着他对技术知识和产业动态的理解大把地赚钱。正是盖茨的独特个性和高超技能造就了微软公司今天的业

绩。

盖茨是个不折不扣的幻想家。还是在个人计算机的早期发展中，他就清晰地阐明了这个行业的未来发展方向。尽管经历了许多策略上的失败，他从未放弃过实现自己梦寐以求的幻想的努力。[2]

这个家伙聪明得令人畏惧。但从一定意义上来说，他是独一无二的、地地道道的利润导向型的人才。你怎样去赚钱呢？当然不应不择手段，唯利是图，赚钱就像生活中许多美好的事物一样，是吧？如果你能赚钱，显然是一件好事。他就像是一台知道如何去赚钱的巨型计算机（吉姆·康纳尔，Office产品单位的程序经理）。

他很疯狂，对产品的了解甚于任何人。每次开会时，你都会大汗淋漓，因为他会马上注意到你的任何纰漏，并穷追不舍，非要打破沙锅问到底。他是那么令人不可思议（戴夫·马里茨，前任Windows/MS-DOS的测试主管）。

IBM曾经认为他们能把盖茨置于股掌之上任意摆布。他毕竟只是一个“黑客”（hacker），不关痛痒，无足轻重。他们这次却看走了眼。这个怪杰的降生，便是为了在人间攫取巨大的权力和最大利润；而作为一位律师的孩子，他对合同语言知之甚详，驾轻就熟，甚至把IBM那些自以为是的家伙弄得狼狈不堪，不敌而走。[3]

管理者盖茨

在一次采访中，我们曾要求盖茨描述一下微软在管理产品开发方面获得如此惊人成就的背后有什么秘诀。他开列的清单虽然包含了一些我们已经注意到的因素，但还是给我们留下了十分深刻的印象。下面是一些从采访录中剪辑出来的简短摘要，阐述了盖茨在管理产品开发方面遵循的主要原则，我们在后面还会进一步涉及到。

- 精明人士和小型团组 “我们一开始就采用最先进的管理办法，那就是聘请一批了不起的人物并成立小型团组……我们最突出的优势是：优秀的开发员总是喜欢与优秀的开发员在一起工作。”
- 使大型团组的工作方式小型化的开发过程 “于是我们不得不拥有规模稍大的团组……我们被迫在许多方面正规化。经过每一个里程碑式的重要阶段时，我们都力争做到没有任何瑕疵，就像做项目评估工作那样。以上所有的这些东西都与项目组的规模大小密切相关。”
- 使团组之间相互依赖性减少的产品结构 “在微软，其良好的组织结构能使相互依赖性降到最低限度，即使在开发组内部也是如此。”
- 集中力量开发新产品 “公司所有的成员都在一块儿工作，只有少数例外。这样，当你急需某人帮助时，你便可以与他碰头，当面求教……这也是一个很重要的优势所在。”
- 公司构造的软件产品，直接用于其工作机器 “我们的开发系统和目标系统完全一致。就我所知，许多公司不这么做。”
- 单一的主要开发语言 “员工们可以争论什么是最优的开发语言。但是，一经确定，即是唯一。”

- 大量投资以支持人们的工作 “我们乐意花费巨资为员工购买所需工具。员工们都有自己的办公室。”
- 公司内部使用自己设计的软件工具 “我们使用自己的软件工具，所以我们能毫不费力地掌握它们……总的来说，我们从使用自己的软件工具中获得了莫大的好处。”
- 多人了解有关产品细节 “我们极少出现这样的情况：仅仅一个人熟悉一个庞杂的代码，而其他人都嚷嚷：‘噢，上帝！咱们头儿是唯一能修改这个代码的人’。”
- 经理人员既创建产品又进行技术决策 “我们没有不懂技术的管理人员，因此，去寻求技术和管理之间的平衡毫不费力。”
- 迅速在技术和业务之间作出取舍 “[我们有]能力迅速进行这类决定。如果其中出现纰漏，业务[产品]单位的经理会迅速作出反应，或者，通过电子邮件由我来作出选择。这些都是在瞬息之间完成的。”
- 一个范围广泛的消费者信息反馈圈 “大部分公司没有利用广大的消费者对其软件产品的意见信息反馈……而微软仅在美国就拥有 2000 人的队伍，就我们的产品与消费者进行经常性的电话联系并记载发生的所有事情。可以说，我们拥有包括市场在内的更为优质的信息反馈圈。”
- 从过去项目中汲取经验教训 “我们对项目进行了大量的事后分析并查看‘臭虫（bug，即错误）’产生的根源，进而考虑如何设计以减少‘臭虫’以及怎样使用软件工具来‘捉虫’等。”

随着微软的发展壮大，盖茨遇到了许多前所未有的问题。他只有不懈地努力，以使自己岿然屹立于这个迅速发展的公司和行业的最前沿；并且保持对微软及其竞争对手不断膨胀的产品组合有一个细致的了解。他每天都得决定什么该“管”，什么该“放”，十分忙碌。可以确信的是，盖茨有一些能干的好帮手，虽然他和其他微软经理们都极力反对动不动就扩充人员。盖茨本人只有一位私人行政助理和一位技术助理，都是在最近几年内雇用的。技术助理常由年轻的程序经理或软件开发员兼任，一般任期一至两年。助理需要复审产品构造思路和说明书，作会议笔录，追踪竞争对手的行动和展览会最新的动态，并帮助盖茨跟上不同项目的进度。

微软领导层采用相当传统的管理方法，不过盖茨事必躬亲，始终保持高度参与。他主持每年 4 月份和 10 月份的程序复查和计划会议，制定大批量生产新产品的时间表并拟定预算的日程安排。10 月份的程序复查重点在于制定 3 年生产计划；每个部门都详细阐明计划上市的产品及其与其他产品之间的相关关系。10 月份的复查结束之后，微软的营销人员（称为产品经理）在各部门的生产计划基础之上进行销售预测。具体的预算计划就此展开了。经理们对预测的销售额和费用支出预算进行分析，看其是否能达到公司的利润目标。盖茨与其他领导人员根据这个分析，才决定在 7 月份开始的下一个财政年度所要雇用的员工数目。盖茨不仅在所有重要的程序复查与计划会议中起主导作用，而且对核心产品单位直接进行指导。

一般地，盖茨主要通过各项目小组成员与经理发出的电子邮件和产品进度报表来定义战略性新产品（或者新的产品版本），并监督开发时间表的执行情况。他每月从各项目组收到许多简短的进度报表——过去每两周一次——并且认真阅读。他参加许多项目的季度程序复查，有时也写写战略备忘录，

大约一年四到五次吧。（有一次我们去采访他时，他正奋笔疾书。他的助手告诉我们，其备忘录内容大致为“我们正面临几个艰巨的技术挑战，谁能提出解决疑难的方法，什么样的方法从战略角度来看是正确的”。）一年之中有一到两次，他要过一个“思考周”。在这段时间内，他把自己与世隔绝，思索一些问题，比如：怎样获得更多的用户支持？怎样使员工们更为合作团结？五年之后产品世界会发生什么样的变化等等。盖茨试图从其时间花费上获得最大效用，即赚得最多的美元：“我对构成公司 80%收入的产品有着最深刻的了解。”他把大部分时间用在一些关键性的应用程序上，如 Office 及其 Word、Excel、Windows 等组成部分。他也十分关注快速发展的新领域，如多媒体技术和联机信息产品与服务等。

项目进度报表

微软的每一种软件产品都相应地有一个项目进度报表。盖茨以及其他高级行政管理人员（包括相关项目组的领导）每月都会接到从不同的项目组递交的此类报告。这些报告在使高级管理人员全面掌握各项目组的项目实施状况方面起到了关键性的作用。盖茨常常直接通过电子邮件与相关的经理人员或开发员进行交流，并把所收集的信息用于正式的程序复查之中。项目组的成员同时也通过这种报告来设定自己下一步的目标。盖茨是这么说的：

> 我拿到了所有的进度报表。现在恐怕有上百个项目正在铺开……[进度报表]包括时间表、重要阶段日期、说明书中的任何变动，还有一些评论，比如“嗨，我们不能再雇人了！”或者“去你的，如果这个 Mac 的 OLE(对象链接与嵌入)2 的发布版还完不成的话,我们都死定了。”……他们知道他们的报告会递交给相关项目组的领导，所以如果他们想表达自己的意愿，这是一个很好的办法。如果他们不在进度报表中提出，那么两个月后他们会以其他方式来说明，不过，这就会造成信息的中断……内部各小组统统照此拷贝，这在一定程度上是小组成员意思一致的表现。

进度报表言简意赅，并且有一套标准格式。盖茨能够迅速将其浏览一遍，他敏锐的目光往往能轻易地找出一些破绽，如项目潜在的迟延或改变。有一些项目和事件倍受其重视：“我把所有的东西全部扫描了一遍……看上百个进度报表对我来说易如反掌……如果报告内容无关紧要，如关于邮件网关产品的，并且报告中没有漏掉成串成串的重要东西，我会敲击‘删除’键把它们删去。报表十分简练……大约有两屏……每个日期各占一行，如里程碑式重要阶段日期、说明书编制日期、代码完成日期……有一栏专门写起始日期、最后日期以及本次报表日期。”盖茨对时间表有无疏漏、是否砍去太多的产品特性或有无必要改写说明书等事项尤为关心：

> 每个报告阶段,仅有 10 到 15 个报表吸引我的视线……[因为有的评论]说：“我们砍去了一些产品的特性。”那么，请告诉我，剩下了一些什么东西呢？又有的评论写道“你正落后于时间表的进度。”这类评论都是隔靴搔痒，无济于事。员工们真正需要指明的是规模大小及速度

要求。或者当一个竞争对手在市场上发行了一个新的版本时，员工们就必须在进度报表中注明：他们是继续保留原说明书呢，抑或加以修改使其与新的时间表一致。这才是至关重要的……有时候马上就会有一些事情跳出来，比如说，这次他们修改日期吗？……当你在阅读开发评论、程序管理评论、测试评论、营销评论时，你会发现，虽然通常它们只有三四句话，有时候，也会出现滔滔不绝的长篇大论。你对这些报告都有了个大致印象后，便会禁不住信口开河，发出一个邮件说：“得了，我让你谈一下有关拖放操作的事儿，你怎么一句也没有写上去。”或者“难道我们不需要这玩意儿吗？我们没有答应把这东西给惠普（HP）吗？RISC 版本怎么样了？你怎么什么都没有提！”……值得庆幸的是，不止我一个人在这些问题上明察秋毫，判断无误，另外有一批员工也深谙此道。

程序复查

大约每过三个月，微软就召开各项目的程序复查会议。这些会议一般延续两个小时，盖茨与其他高级管理人员通常都会参加。项目组一般从各职能领域派遣一至两个关键人员为代表列席会议，有程序管理组（专门负责编制产品说明书），软件开发组（编写计算机代码），软件测试组（测试代码），产品管理组（产品策划及营销），用户培训组（准备产品文档）。盖茨告诉我们，他所关注的问题类型与进展报表如出一辙：

> 你会关心项目进展是否顺利，因为这是一个基本问题。你会想知道员工们是否团结互助……这也很重要。如果他们添加了一些东西，将会使生产速度放慢，你就得干预：“你得给我保证速度不会下降10倍”。……如果有些产品完工比
>
> 他们想象的时间要长，你就得考虑，是因为他们不了解有关的产品设计吗？
>
> ……你必须仔细询问很多问题使自己清楚我们对正在做的事情是否真正成竹在胸。你必须留心倾听项目实行过程中冒出来的好点子，兴许它会将整个说明书的面貌彻底改造一番
>
> ……他们能否将突然闪现的灵感捕捉并使之升华？他们与其他小组的关系处理得如何？
>
> ……我认为小组之间应该共享代码。我常有意向他们灌输这一意识。不仅在相关小组之间应该代码共享，毫无关联的其他小组也应当如此。如果他们提出，相互依赖会制造出大量的“臭虫”或使速度大为降低；或者他们从未能够及时从另一小组拿到代码，那就有必要摆到桌面上进行讨论……
>
> 如果市场环境改变了，我会让他们修改说明书，那么他们及时从我这儿得到指示就显得十分重要。你一旦拥有充足的信息资料，你就可以直截了当地对员工说，他们干得很出色，或者成绩平平，这样，你便营造出了一个良好的工作氛围。有时候，在开会之前你就已经心里有数了，因为给我的有关项目的电子邮件实在是很多很多。

史蒂文·斯诺夫斯基，是盖茨 1993 年与 1994 年的技术助理（现在是 Office 的组程序经理）。他极力贬低程序复查的戏剧性效果，声称高级管理人员通常都事先知道问题所在并且与项目成员们一起先开一个预备会议“程序复查唯一的作用，是让头儿们确信，大家众志成城，行动一致。他们以前经常为麦克·梅普尔斯或史蒂夫·鲍尔莫或其他高级副总裁做此工作，然后再为比尔精心准备，这决非突如其来。”但是盖茨极力使他的问题搅和在一块儿。他有时会思考一些极为明显的事情，不过想得更为深入一些：“我提出的大约有一半问题经常使人莫名其妙；但是，另一半往往相当清楚易懂。我会提醒员工什么样的标准检查程序是真正重要的，什么样的系统配置是真正有价值的。”在一些情况下，他也会播洒及时雨，给予相应的帮助：

> 在我的职位上，我实际能够控制些什么了呢？我有一批核心开发员，在我转移到某项目视察时，帮助我检查算法[一种数学上或程序上的方法，通过计算机代码完成特定任务]，核实代码或进行实际操作。一个项目如果看起来岌岌可危，你就会考虑独立地复查一下代码。在过去，我会对他们说：“嗨，给我源代码，我拿回家看去。”现在我不这么做了。我会命令手下的 D14 或 D15（公司高级技术职衔）：“给我回去好好钻研一下，回头再告诉我。”或者，“选派更多人员，务必解决这个问题。”他们经常就关于砍掉哪一块特性提出一些建议……因为他们了解所有的相互依赖关系以及各部分是如何互相配合成为一体的。

盖茨也坦率地承认，他发现取消一些陷入困境的项目并不容易：

> 一般来说，我们不会轻易取消项目。但是，软件市场时时风起云涌，变幻莫测，我们不得不作出相应的调整。这属于技术——业务复合型决策。最著名的例子是我们的数据库产品。第一版本的 Windows 数据库并不完善……你可以说我们是另起炉灶。有人说我们仅仅重写了其中的 80%，这种说法未免太过极端。而 Windows 的 Word 倒不愧是一个典型。一波三折，屡遭挫折，迟迟无法上市，成为微软发展史上的一段趣事。

在复查说明书时，盖茨把重点放在一些关键问题上。他想知道一个新产品能引起多大轰动效应，是否能与其他微软产品兼容，他也关注质量问题（主要以“臭虫”数目来定义）。盖茨通过对这三大问题的分析来决定是否推出一个新产品。另外一些正越来越引起盖茨关心的问题便是小组之间的相互合作与构件共享以及生产是否落后于时间表的进度等。对于独立性很强的产品组们来说，这种互相依赖实非易事，而且越来越令人挠头，因为许多微软产品可能每年都要推向市场（例如，Windows 95 和 Office 95）；任何项目的失控延迟都可能导致尴尬的结局——产品重新命名。

盖茨喜欢运筹帷幄，确定总体方向，让各产品组自己解决微观问题。令人们拍案称奇的是，他能迅速发现各小组正费尽心机设法解决的产品技术细节问题，而不论此项目课题是属于其所擅长的专业领域（如 Word、BASIC、Excel）之内还是超出他个人经验之外（如 WindowsNT）。麦克·康特，以前是 Excel 的产品和程序经理，现在在 Office 工作，向我们回忆起大忙人盖茨是如何在几分钟内就发现新特性的缺陷的。而 Excel 组，虽然知道有缺陷存

在但未曾告诉盖茨，已花费了整整一个月的时间来寻找问题症结所在：

> 在 Excel3.0 中我们确立了一个重要概念，即“次最小重算”。所谓最小重算，是指你只需要重算相关的数值；当用户改动某数值时，软件只计算变动的单元格，而不必把全部工作表重新计算一遍。这是重算速度上的一个重大突破。而 Excel3.0 的次最小重算功能，在中间变量计算结果未改变时，甚至允许不重算相关的数值……我们把这些情况汇报给比尔，而……[他]说：“这种情况是怎么回事？还有这个，嗯，那个呢？”克里斯·彼得斯，当时是我们的开发经理，说：“哎呀，真是有意思。一个月的潜心思考，我们才发现这三个问题。我们真是好生惭愧。”可以说，盖茨非常善于发现技术问题。

即使是老练的职员和经理也会因准备与盖茨的正面交锋而忐忑不安。克里斯·彼得斯，现任 Office 产品单位的副总裁，在他名噪一时的 1990 年的录像“软件推出”中给予他的同事们如下建议：

> 你在做什么，为什么这么做，都必须向比尔交待清楚；你从不应该隐瞒什么，因为他很精明，能洞察一切。但是，你必须从容不迫，信心坚定，你必须顶撞他，毫不示弱。我所能给你们的唯一建议便是在开会的时候，带上你最最拔尖的开发员，他们能不加思索源源不断地旁征博引，把盖茨埋葬在不容置辩的铮铮事实之中……永远不要仓促上阵，毫无准备。但是必须要学会说“不”。这样，盖茨便会很尊重你的意见。他很清楚得及时推出产品。我认为我们最后要说的是……对微软来说，了解延迟推出一个产品将付出什么代价十分有用，所以，我们最好不要做这个特性了。我想最终我们还是会制作那个特性的。但，这就是他成其为比尔·盖茨的原因。

控制新产品开发

至于什么样的事情该放手不管，什么样的事情应牢牢控制，盖茨是这么说的：

> 嗯，最初，我不让任何人编写代码。我拿走所有的其他人写的 BASIC 语句，自己再重写一遍，因为我不喜欢他们编程的方式。这其中是颇需要技巧的，我不情愿让任何人介入其中。但是我们马上就有了新产品像 FORTRAN 和 COBOL。在这里，我只是确定一下软件说明书是否设计正确，我们是否遵循基本算法等……我从来不会让别人来确立待开发产品的总体思路。我不会说：“我不知道这儿有一个新产品组，没有人告诉我这些。”对于一个软件公司的总裁来说，这是一个很好的控制公司的手段，也是我现在唯一能真正把握的东西。

盖茨的一个重要角色是根据他对未来发展方向的展望，包括可能的竞争对手的行动，来定义公司全部产品系列，并在技术和业务之间作出取舍决策。一个例子就是盖茨在 5 年时间内，推进 Excel 与其他组采用 VisualBasic 为通用的“宏语言”（宏是一组特殊的命令，由用户自行处理，使许多复杂的

操作仅需一两个快捷键就能激活。标准宏语言的使用，能使用户用同一种方式来定制各种各样的微软产品）。盖茨也促使各小组采用对象链接与嵌入（OLE）标准使构件共享更为便利。

盖茨也积极参加关键项目的开发。他确定现有产品，并制定其未来发展方向。Word 的开发经理爱德·弗莱斯回忆道：

> 我们与比尔之间互相影响，互相启发，尤其是在产品开发的早期阶段，我们一起仔细审查说明书并决定产品特性……比尔几个星期前就到我们小组来了，我们围着他向他展示 Word 的各种资源。上周是他的“ 思考周 ”，其间，他给克里斯·彼得斯发了大约 10 个电子邮件，邮件中说：“ 噢，这个 Word 版本中的某些东西我不太赞赏。你得给我好好考虑，怎么样使我们所有产品的拼写检查功能更为成熟。还有，Word 和 Help 等产品怎么样才能合成为一种类型。 ”比尔花了很多时间定义我们产品的未来方向，他也对我们现在的进程提出批评……他经常勾画出 5 年后产品的概貌。我们只得在遵循其指示与满足用户需求两者之间努力寻求平衡，寻找一个折衷方案……这便经常造成矛盾冲突。

虽然微软的经理与开发员崇尚一种高度独立的企业文化，盖茨却习惯于事必躬亲，大包大揽，极少委托他人。据斯诺夫斯基说：“ 每样事情都极为琐碎，但都各有其独到之处。 ”

在推进小组们采用 Visual Basic 与 OLE 的同时，盖茨还要求在各产品中增加一些关键的功能。包括 Excel 的分级特性，Word 的表格和拖放操作以及 Visual Basic 中的自定义控制项（称为 VBXs）。盖茨也曾经坚持要求程序经理与开发员采用 Visual Basic 来编写原型和特殊的软件程序；一些小组发现 Visual Basic 语言由于技术上的原因不太适用，便予以抵制，他们成功了。公司副总裁史蒂夫·鲍尔莫在盖茨的默许下，也曾授权微软的小组们采用 Windows NT 的测试版而不是 Windows 或 OS/2 作为其操作系统。另外，盖茨还通知微软的商业工具小组（一项重要工作是构造语言编译器），不仅仅要面向消费大众，还得增加特性以支持内部的软件开发员。偶尔，盖茨会中止因为延误或出故障而难于为继的项目,比如 Windows 的 Omega 数据库产品(后来演进成为 Access）。他还把版本有异但从事同种类型功能的项目合而为一，如 Word、Excel 与 Mail 中的文本处理代码。

人们屡次三番与盖茨进行争论；就克里斯·彼得斯的观察，只有这样才能获得盖茨的尊重。但是人们必须有显明的技术根据和数据事实作为后盾。那些基于私人的或感情上的原因，或者是内部政治而建立起来的观点，对盖茨及其他关键人物丝毫不起作用。特别地，对于盖茨来说，需要用新型的智力模式（其观点必须颇有见地，其世界观则需与众不同）来改变他冥顾不化的头脑，引起其共鸣。史蒂文·斯诺夫斯基解释说：“ 如果他一意孤行，执意不肯接受你的观点，你一遍又一遍地重复只会白费力气。你必须另辟蹊径来击倒他。 ”

原则二：围绕产品市场，超越经营职能，灵活地组织和管理

比尔·盖茨向我们强调，微软以产品为中心来组织管理公司。我们发现，

在很大程度上，这句话是正确的，不过，微软也非常重视其技术与专项功能（虽然与前者有重叠交叉）。员工们在多功能小组（根据产品组织而成）内工作，另有一些机制将这些产品小组联为一体。除了部门和产品组这两大结构外，微软另有两类组织机构。一类是较为正式的，组成了微软的管理体系。另一类属于非正式组织，由一些被尊称为“智囊”的管理人员和一群执行特殊任务或项目的技术人员与经理组成，直接听命于盖茨或其他高级领导。

组织和过程演变

微软通过不断地摸索与试错，经过漫长的阶段，终于形成了今天的组织结构和产品开发过程。在 20 世纪 80 年代早期，微软成立了一个名为“最终用户组”的机构来开发应用程序，这使系统与应用组截然分开，各自循着不同的轨迹独立发展。公司在发展过程中，始终为改善其专项功能，尤其是软件测试，而奋斗不息着。公司也不得不克服许多难题，例如，产品小组缺乏中心领导，质量控制与项目管理方面破绽百出，无一可取。与其他许多公司一样，一些具有历史意义的转折点与关键人物构成了公司的发展历史（见表 1.1）。

第一个转折点是 1984 年成立测试组的决定。测试组与开发部门的分离，使得对开发员的工作进行独立的检验成为可能。第二个转折点大约在同一时间发生。程序管理开始区别于产品管理与软件开发而成为一个独立的职能。第三个转折，则是由 80 年代中期一系列迟延产品和“臭虫”引起的。这导致公司上下意见趋于一致，认为应当严格进行质量控制与项目管理。微软小组们现在已形成一种制度，即在事后分析书面报告中记录项目完成的心得体会。这反映了一个越来越为人们普遍接受的观点，那就是通过“向错误学习”，可以把工作干得更为出色。第四个转折点，是 1988 年 IBM 的麦克·梅普尔斯来到微软，在他的提议下，创建了小型业务单位，这使操作组增加了核心领导，并且更易于管理。

第五个关键事件是 1989 年的“休假会”。高级经理与开发员致力于减少故障，并提出多种方案，以帮助微软小组们在软件开发与

表 1.1 微软组织演变进程中的关键事件

1984	在各个部门内部建立独立的测试组。 Macintosh 电脑用的 Multiplan1.06 电子表格软件产品回收。 建立除测试组之外的专项职能如程序管理和产品管理机构。
1986	建立事后分析文档，鉴别产品质量与项目管理方面的问题，并提出解决方案。
1987	Macintosh 电脑用的 Word 3.0 文字处理器产品回收。
1988	IBM 的麦克·梅普尔斯就职于微软，在系统部门和应用部门为每个产品组建立独立的业务单位。 在 Publisher1.0 项目中采用里程碑方案。
1989	5 月份的“休假会”与 11 月份的“零缺陷代码”备忘录突出了“每日构造”的重要性。
1990	Excel3.0 项目（1989 — 1990）顺利完成，仅推迟 11 天。 采用同步一稳定过程中的关键要素（里程碑和每日构造）。
1992	聚合性总裁办公室成立。 把系统与应用归于全球产品集团之下，由执行副总裁麦克·梅普尔斯领导
1993	集中营销小组（产品经理）成立独立的部门，产品策划者除外，将业务单位改名为产品单位。 建立 Office 产品单位，由副总裁克里斯·彼得斯领导。
1995	成立平台集团，由集团副总裁保罗·马里茨领导，成立应用与内容集团，由集团副总裁内森·梅尔沃德和皮特·赫金斯领导。

质量控制方面更为系统化。一种思路是把一个项目分解为很多个子项目或里程碑，1988 年 Publisher1.0 就采用了这种方法从而顺利地完成了。另一种思路便是产品的每日构造，许多组都实行了这种方案但未执行“零缺陷”目标。这些关键性思路都已成为同步一稳定过程中的精华部分。Excel3.0（1989 和 1990 年开发）是第一个采用这些先进技术的规模宏伟、营业额颇丰的微软产品，它的问世仅仅推迟了 11 天。

第六个关键事件便是 1992 年建立四人总裁办公室，并由麦克·梅普尔斯全权负责所有的产品开发。这使计划表一直难以预测的操作系统组更为规范化。第七，则是 1993 年的重新改组。大多数营销组的产品经理们被集中起来组成一个独立的部门，而一些产品策划者仍继续留在产品开发组里。微软还把业务单位改名为“产品单位”来反映这种变动。此外，微软成立了 Office 产品单位。这些变化使营销和产品策划及开发脱离，有利于在微软的核心应用软件产品之间实现构件共享。

乔恩·薛利是麻省理工大学的中途辍学生，曾任坦迪（Tandy）公司（销售“无线电小屋”手提电脑）的行政主管。1983 年，盖茨聘任他为微软总裁。他把软件开发分为两部门：操作系统部门以及应用软件部门。盖茨曾经规定开发员必须向高一级的开发员而不是总经理负责，每一个开发员都必须向他汇报工作，因为他是公司的最高开发员。盖茨也一手控制了其他的重要领域，如营销等。现在，微软急剧扩大，产品大大增加，这种规定已是过时了。薛利认为盖茨管得太多，公司整顿势在必然。举个例子，盖茨喜欢把开发员东抽西调，今天来这个小组，明天去那个小组，疲于奔命。不仅如此，盖茨有

时心血来潮，随意改动说明书，虽属“神来之笔”，却因无半分预兆，令手下人员无所适从。薛利认为盖茨应在高抽象层面上确定产品，引导开发组织的战略性发展方向，而不应陷在项目的细枝末节中无法脱身。[4]盖茨同意对公司进行大整顿，主动减轻自己的工作量，集中精力进行应用软件的监督。他任命史蒂夫·鲍尔莫为系统部门的主管。盖茨对这一次公司组织的大整顿作了如下描述：

> 在麦克·梅普尔斯来公司以前……所有的开发小组以我为中心展开工作。当时盛行这样一种理论，即开发员不应被迫给那种一天写不了上千行程序的人工作；否则，对于开发员来说，这是一种侮辱。所以所有的开发管理人员全是清一色的软件开发者，每一个管理环节都安排了能够编制程序的开发员。现在这个传统被打破了。我把软件开发分为系统部门与应用部门，并且让史蒂夫负责[系统部门]。史蒂夫不是技术人员，但他为人十分精细，品格高尚。人们都很信服他出众的才华。当进行一项技术决策时，他总能知人善任，调度有方。他所选择的即使不是经理人员，也是广为人们所爱戴的人士。所以我们每一回都能圆满地完成任务。

1984 年，微软决定把测试组从开发部门中分离出来。因为它的许多软件产品都出现了“臭虫”，由此激起许多采用微软操作系统的 PC 机制造商的极大不满（微软把这些公司称为原始设备制造商，简称为 OEM）。比如，1981 年与 IBM PC 机一起推出的 BASIC 版本就比较粗糙，用户在用“.1”或者其他数字除以 10 时，就会出错。这次事故后，IBM 坚持要求微软改善其软件开发和质量控制的过程。在 20 世纪 80 年代早期微软还有其他的未向世人公开的问题。比如，FORTRAN（一种技术性的程序设计语言）上就有一种腐蚀数据的“臭虫”。[5]个人用户也开始纷纷进行投诉，说他们从零售商店买回家的微软应用软件有许多质量问题。

山雨欲来风满楼。微软的高级经理人员终于醒悟，发觉很有必要引进更好的内部测试与质量控制的方法。但是，许多人坚持反对，包括大多数程序设计师甚至一些高级经理（如史蒂夫·鲍尔莫）。他们认为在高校学生、秘书或者外界订约人的协助下，开发员可以自己测试产品。在 1984 年 1 月推出 Macintosh 多元计划（Multiplan）电子表格软件的最新版本之前，微软就曾特地请 Arthur Andersen 咨询公司进行测试。但外面的公司一般没有能力也不太可能全面地精确地测试一些较为复杂的软件产品。结果，一种相当厉害的破坏数据的“臭虫”迫使微软为它的 2 万名 Mac 多元计划电子表格软件的买主免费提供其更新的版本，代价为每个版本 10 美元，一共花了 20 万美元，[6]可谓损失惨重。

痛定思痛，微软的经理们得出结论说，如果再不成立独立的测试组，他们就不可能达到更高的质量标准。IBM 与其他有着更为长远更为成功的软件开发历史的公司便是效法的榜样。戴夫·穆尔，微软的开发部门主管，回忆道：“我们清楚我们不能再让开发部门自己来测试了。我们需要有一个单独的小组来设计测试，运行测试，并把测试结果信息反馈给开发部门。这是一个伟大的转折点。”微软未曾照搬 IBM 的方法，即组织一大群人检查所有的软件项——从说明文档到代码及测试计划——或者要求行政管理人员在各开

发阶段“停止活动”。这种官僚制度在大型电脑和容错性应用软件的生产中较为普遍，但在 PC 机世界中却很少采用。微软也不像 IBM 与其他的一些公司（包括日本的软件制造厂）那样对关于开发员与测试员如何度过他们的时间这类细节问题进行监控。微软有选择地采用一些看起来比较先进的技术，如独立的测试小组、自动测试以及为新手与关键性构件进行代码复查等。他们并不照搬 IBM 的经验，而是引进，使之成为微软特色。（穆尔指出：“IBM 经验并不适用于微软。”）

但是，穆尔接着说：“我们做错了。”自从微软在 1984 年与 1986 年之间扩大测试组以后，开发员“开始变懒了”，认为他们能够“把代码扔在一边等着测试”，他们忘了唯有开发员自己才能发现更多的错误，只有他们自己才能防患于未然，首先阻止错误的发生。在此同时，微软历史上第二次大灾难降临了。原定于 1986 年 7 月与公众见面的 Macintosh 版 Word3.0，千呼万唤方于 1987 年 2 月问世，而且先天不足，里面竟然有大约 700 多处错误——有的甚至破坏数据，摧毁程序。一下子微软便名声扫地了。公司不得不在初始发布后的两个月内免费为消费者提供升级版本，其费用超过了 100 万美元。[7]

至今，一个连公司内部的怀疑论者都深信不疑的事实是，微软在产品开发管理方面困难重重，举步维艰。为重获消费者欢心，各小组必须更为规范化。盖茨亲自掌管应用软件部门，但一些主要项目却是一团乱麻。除 Excel 外，Windows 的其他新应用软件开发毫无进展。而一个数据库程序（别称 Omega 项目，以后演化为 Access）和一个 Widnows 的项目管理应用软件严重受挫。

Opus 项目，后来更名为 Word for Windows，激发起工作人员的灵感，杜撰了一个现在相当流行的新名词“无穷错误”。这个词描绘的是这样一种情形：测试员寻找错误的速度远远快于开发员修正错误的速度，而每次错误修正以后，又会产生新的错误。如此循环往复，错误无穷，使时间表和最终产品上市日期难以确定。这种情况的出现，有时是因为暑假实习生在编写完代码后未完成测试便返回大学，有时候，则由于开发员被从一个特性或项目抽调到另一个，所以来不及完成测试工作。在微软曾经经历过的“迟缓臃肿”集成期和测试阶段，开发员只能求助于旧代码。但遗憾的是，代码的具体内容大多已被淡忘，或者其作者已不知去向。在重写代码以修改数不胜数的错误中，开发员常常会重蹈旧辙，新一代的“臭虫”又大肆猖狂起来。[8]微软的测试主管罗格·舍曼，向我们回忆微软历史上这一段黯然无光的日子：

> 人们得到消息说他们实际上把事情弄得一团糟……他们像驾着一辆飞驰的赛车，不时地撞到墙上，不过现在他们总算知道墙在哪儿了……人们认识到不管程序是多么地充满想象力或创造力，如果不实用，也只能忍痛割爱。他们学会从各个角度全面观察问题并着重分析外部因素。他们知道了他们所编写的代码必须是可测试的，并且其测试资源一定是有限度的……他们得编写稳定性很强的代码。可以说，从 Omega 和 Opus 的反面教训中，人们获益非浅，并逐渐走向里程碑式过程。我认为他们从 Access 而不是 Word 的教训中学到了更多的东西。“臭虫”数据库是如此的硕大无比，以至于几乎不可能存放在一个服务器中。而“活性臭虫”多如牛毛，使测试小组几乎无事可干：“急什么？开发部门必须处理完 2 年的积压待办事项才能赶上咱们现在的进度。”测试员几乎把全

部时间都放在开发自动化工具上。终于，有人得出结论，说所谓的出品时间统统都是不现实的，项目也凌乱不堪，我们永远不可能及时推出产品。这大概意味着所有产品的定义……也是虚假的。开发员不得不专注于一小部分产品使之性能稳定……他们削减了许多开发计划，转而务实，力求产品具有稳定的代码基础并且可能继续开发。

盖茨招贤纳士，决定到公司外务色更多的管理型人才，以使项目不再失控。求贤若渴则英才群集。1988 年，在 IBM 服务 23 年之久，领导其软件战略和业务评估工作的麦克·梅普尔斯投入微软门下。他还是开发 OS/2 系统和 Presentation 管理器（IBM PC 机的早期图形界面产品）的中心人物。[9] 具有讽刺意味的是，微软员工们经常对 IBM 的一些现象冷嘲热讽。比如，IBM 聘用太多的非熟练开发员充任程序员（微软员工戏称之为“蠢驴编程”），开发过程过于连贯且封闭等。[10] 微软经理们十分自信地认为微软集中几百个高级开发员所取得的成就 IBM 得汇合成千上万个员工才可能完成。不过，梅普尔斯看起来与众不同，很有天份，而且盖茨希望开发过程安排更为合理，使微软能有效地构造、推出产品并控制其质量。（所以梅普尔斯理所当然地被重用了。） 1989 年的一份重要备忘录总结了五月份“休假会”以来的讨论（由戴夫·穆尔组织），使各产品小组勇于采取正确的行动。这份名为“零缺陷代码”的备忘录，由 Word 组的一位开发经理克里斯·梅森撰写，主要记录了公司现状及即将采用的新开发过程。

在操作系统领域，微软的 Windows 同样步履艰难。就如同盖茨的传记作者在 1990 年 Windows 的第三个版本问世以后所观察到的那样,产品依然显得粗制滥造：“又一次，微软的测试者们未曾根除问题，留下了无穷后患。比如说，无法将程序安装于某一类型的机器上，网络老是出错，鼠标无法使用，一种第三方磁盘管理软件的数据被破坏，还有普通的假信号收集和文档错误……软件行业中流传着这么一则笑话，说微软产品直至第三版本才开始进行 β 测试。”[11]

盖茨及其他微软员工还有另外的隐忧。股票期权已成为公司资金来源的一个不可或缺的组成部分，但是经常性的产品迟延上市，使微软股票价格狂跌不止，回升乏力。产品的迟延与反复也使用户，包括 OEM 与零售商在内，都疑惧不安，大失所望。有一位股东甚至因微软未能及时推出 Word（该产品占微软总销售额的 20%）而对微软起诉，欲与之对簿公堂。最后，公司在 1990 年耗费 150 万美元才了结此案。（该案件控告微软经理人员故意隐匿迟延交货的消息。）至于数据库项目，当梅普尔斯 1988 年到任时，原假定 3 个月即告完成。而在一年半以后，他与盖茨取消了这个项目。[12]

微软备忘录

送达：应用软件开发员和测试员

作者：克里斯·梅森

日期：89/6/20

主题：零缺陷代码

主管：麦克·梅普尔斯，史蒂夫·鲍尔莫，应用业务单位经理和部门领导在 5 月 12 日和 13 日，应用软件开发部的经理们与他们的项目领导、麦克·梅普尔斯以及其他的应用和语言的代表们一起开展了“休假会”活动。我的讨论小组对零缺陷

编写代码的技巧进行了深入调查与研究。这个备忘录就记载了我们大家所达成的共识……导致我们产品错误越来越多的原因是多方面的，不知诸位注意到了没有，事实上，我们的产品正越来越趋于复杂化，但我们并未曾相应地改变我们的经营管理方法……列举出一大堆问题只是为了让大家更清楚地看到，是我们现行的管理制度，而不是我们的员工，导致这一系列问题的发生……我们的时间表设计和长期以来形成的企业文化鼓励我们花最少的时间来完成一项特性，从不要求精益求精。只要它能被很好地演示，我们就觉得可以了，所有的人也都这么认为。于是这项特性就算是按照计划圆满完成。几个月以后“ 臭虫 ”不可避免地出现了，而我们却认定它与原来的工作毫无关联……当不能按时完成时间表规定的任务时，我们便想走终南捷径……产品的日趋复杂怎么会成为培育“ 臭虫 ”的温床呢？许多人也许会大惑不解。很简单，因为我们不懂得怎么样协调各部分来进行新产品的生产，或者修改旧产品……我真正的意思是：你们的目标应该是每天生产那种能产业化的、易于推出的产品……人无完人，人类自身的缺点都无法克服，又怎么能强求代码中没有故障呢？当出现 “ 臭虫 ”以后，你必须仔细分析并立即着手解决……我们每天主要的工作就是编写代码，而出现 “ 臭虫 ”就意味我们努力的失败。[下面是另加的斜体字]成百成千的公司和个人用户都依赖于我们的产品：“ 臭虫 ”将会使他们蒙受时间与金钱上的巨大损失，可以想象，电脑病毒悄悄地在电子表格、数据库或者文字处理器中蔓延时，将会有多少家公司被迫停业！所以，我们必须开始更加慎重地处理故障问题。

应用和系统部门不断出现的问题，为梅普尔斯改革方案的推出和贯彻创造了一个极为有利的环境。特别地，他要求项目小组跟踪领先市场的竞争对手及产品，如 WordPerfect、Lotus1-2-3、哈佛 Graphics。他还要求每个小组对软件开发、测试和项目管理的过程进行定义并且精心改进。在与盖茨讨论之后，梅普尔斯把应用部门分成五个业务单位：Office，分析（Excel），图形（PowerPoint），数据访问（Data Access）和 Entry（Works）。这些业务单位现在都已成为自负盈亏的盈利中心——能独立支配各种资源来为程序管理、开发、测试、营销（产品管理和产品策划）和用户培训服务——与 IBM 在 1981 年推出最初 PC 机时采用的业务单位形式十分相似。当时，IBM 独立业务单位的实行，获得了巨大的成功。梅普尔斯在为 IBM 业务单位服务时便与盖茨等一见如故。后来他回忆起他对微软重组的影响时说：

这实在是很有趣。当我初来乍到时，我参加所谓的资源计划会议。开发、营销、程序管理还有测试部门的头头们都来了，他们对项目进行了详细的审查。每个星期每个项目都会有变动；有的项目已经迟延了 18 个月，但还想将其计划完成日期往后推迟，这时他们会要求说：“ 我们需要更多的测试员。 ”于是我们就从别的项目中抽调一些测试员过来。第二周，我又召开资源计划会议。上周人员被抽走的小组陷入了窘境，不得不要求增援，我们只好再次进行人员调动。公司内部员工经常被莫名其妙地东抽西调；人们从来不隶属于一个固定项目，这便造成了一种极为混乱的局面。于是，我们决定构造业务单位。从表面上看，它使人员自由流动所产生的好处——高效率——丧失殆尽。比如，Word 的测试小组成员固定不变，即使身边没有需要测试的软件，也不予外借。事实上，业务单位的建立，使得资源库能够保持其连续性和稳定性，人们可

以从容安排各项计划。它也给员工更加广泛的权利与责任，使得一大批管理人才脱颖而出。它使公司在制定其发展战略时更加注意面向用户并跟紧其竞争对手。总之，微软现在的组织机构就是以小型业务单位为基础构建起来的。

微软一直沿用这种基本的组织结构，但也有一定改进。1992 年，盖茨把史蒂夫·鲍尔莫调离操作系统部门，转而担任销售和支持集团负责人。盖茨还创建了一个不断扩大的全球产品集团，由麦克·梅普尔斯对应用和系统两大领域的产品开发进行监控。这个任命使梅普尔斯有机会接触并驾驭操作系统组，直至 1995 年他辞职为止。而自梅普尔斯卸任以后，应用与操作系统又再一次分道扬镳。1993 年，微软对营销工作渐渐重视起来，建立了 Office 产品单位。此外，微软又任命了关键职能领域如开发、测试，程序管理以及用户培训的负责人；自 1994 年以来，这些职能组都必须向产品开发的总负责人汇报工作。各负责人并不直接监督项目的实施，也没有明确的责任分工。他们的主要工作是帮助各小组总结并推广最佳经验。盖茨对公司组织的增减变动作了如下说明：

这只是一种取舍关系，所有的组织建设都属于取舍关系。它优化了产品制造的团体精神，我们会问：如何处理我们的产品？如何取舍来制造这个产品？我们推出产品了吗？产品。顺利通过复检了吗？所以，渐渐地，专门化的观念被破除了。

产品组织上存在两大缺陷：其一，公司虽然在面试求才、新工具开发、新方法运用等许多方面都有着很好的经验，但这些经验明显没有在各领域之间得到传播和推广。如果你的小组仅有一位测试员，那么就不可能求全责备，更不可能在考察过后诘问他：“喂，你怎么能连最新的有关测试的书籍都没有看过呢？”所以我们给每个小组都配备了一些为数不多但熟知测试方面最新动向的人员。其二，代码共享极为不易。虽然存在以上缺陷，但不应以一眚掩其大德，要看到公司现行组织给自身带来的利益要远远超过其造成的弊端。如果在过去的 8 年中，我们不使用业务单位来规划公司资源，微软早就分崩离析了。

微软公司组织体系的一个明显优势是它给各小组提供了充分的自由，每个小组就是一个相对独立的开发中心，致力于把各类产品推向特定的市场。克里斯·彼得斯解释道：

虽然我们自以为是地把业务单位当作微型公司看待，可实际上它们是不折不扣的产品开发中心……那里没有负责销售和财务的部门——其相关职员早已从业务单位中调离。在业务单位内部工作的每一个人的职责都是相同的，就是不断地为公司推出新的产品。他们不需要编制代码，也不必进行测试或撰写说明书，他们唯一的工作就是为公司推出产品……在这里，你最好能努力避免编制代码，如果你能够不通过编码就赚足美元，那何乐而不为呢？

特别是由于 1989 年公司“休假会”活动的刺激，微软的经理们如今特别强调开发员与测试员要留在自己所属的小组内，至少要超过一个产品周期。他们要求开发员努力工作，争取产品“一次到位”，并且“以质量为本”。

为了达到上述标准，微软采用了“每日构造”与多里程碑技术——这就是我们在第四、五章内将要描述的同步—稳定过程的精华。

现行的管理结构和组织体系

微软目前位于董事会之下的最高管理层，是由 1992 年首次创立的“公社制”领导班子——总裁办公室演化而来的。在 1995 年 7 月麦克·梅普尔斯归隐之后，又进行了一次改组。现在包括身兼主席和总裁二个主要领导职位的比尔·盖茨以及五个分别主管微软四大业务集团的高级经理人员：集团副总裁内森·梅尔沃德（前任高级技术部门领导）和皮特·赫金斯（前任桌面应用部门领导），二人共同负责新成立的应用和内容集团；集团副总裁保罗·马里茨（以前负责产品和技术战略）领导新成立的平台集团。这两个集团构成了微软的生产、研究以及开发的基本框架。执行副总裁史蒂夫·鲍尔莫掌管销售和支持集团，而罗伯特·赫伯德则负责营业集团并同时担任营运主管。部门副总裁与总经理们对这些集团领导负责。他们下面是产品单位经理，产品单位经理直接领导职能小组经理，再下面便是微软最基层的管理干部——产品小组组长。

如图 1.1 所示，联机应用和内容集团下设四个部门：桌面应用部门、用户服务部门、联机系统部门以及研究部门。平台集团也分为四个部门，分别负责个人操作系统、业务系统、开发员与数据库系统、高级用户系统。大多数部门自身就拥有市场营销机构，其职员由产品策划者直接担任。它们之间共享一个中央可用性测试实验室（由 30—35 个职员组成），用以测试产品特性及原型。销售和支持集团下设有许多独立的部门，有的负责面向全球 OEM 的销售（这些 OEM 主要是指 AST、DEC、Dell、康柏、富士、Gateway、IBM、NEC、好利获得、Packard Bell、东芝、优利、Zenith 等大公司），有的负责产品支持服务（简称 PSS），还有国际营业部门（主要面向亚洲）、高级技术销售部门、战略企划部门（为一些大公司提供咨询及特定产品）、北美销售部门以及欧洲销售部门。营业集团则负责财务、磁盘生产、说明手册及相关书籍的出版（微软出版社）、信息系统和人力资源管理。

在平台集团内部，个人操作系统部门生产开发 Windows 和 MS-DOS，业务系统部门则负责 WindowsNT 与 OLE（对象链接及嵌入）技术的开发，它下辖一个独立的产品单位（负责电子邮件及个人机服务器系统）。开发员及数据库系统部门编写程序设计语言（如 Visual Basic），并为支持工具以及数据库产品（如 Access 和 FoxPro）编写程序。高级用户系统部门负责交互电视系统、宽带通讯及多媒体技术的开发。在应用和内容集团，桌面应用部下设有 Office 产品单位,它协调 Word 与 Excel 两个产品单位并与 Graphics 产品单位（PowerPoint）紧密合作，以使这三个产品的功能在 Office 应用套装软件中能更好地协调。这个部门同时也生产 Project，一种流行的项目管理工具。用户服务部门包括微软家用软件产品组，负责生产家庭理财工具软件以及为家庭娱乐和教育设计的多媒体应用软件；同时也生产 Works 套装软件，它集电子表

图1.1 微软公司組织体系（1995）

格、文字处理、数据库功能于一身，主要提供给初学电脑的人使用。（联机）系统部门开发和管理微软网络产品。研究部则负责新产品研制和程序设计技术创新，并与各产品组紧密配合，尽量把研究成果早日产业化（参照附录 2、3 中关于微软主要应用产品与操作系统的描述）。

图 1.2 是桌面应用部门的结构示意图，清晰地展示了微软是如何将产品单位与专项功能结合在一起的。该部门包括多个产品单位，每个产品单位一般都设有五个职能小组，各由独立的经理人员管理，分别负责测试、程序管理、开发、产品策划（由委派到产品组的产品经理担任）及用户培训。微软将这些小组按照其生产的产品不同分为几个区域，例如，在 Office 产品单位成立之前，Excel 单位的开发员被分成五个小组：重算/功能，图表，打印/格式化，加载项（一种特殊的软件程序，如统计分析或拼字检查等）和宏/转换。一些产品单位把他们的小组集合起来共同进行用户培训，准备培训手册和联机文档（例如 Help 菜单的撰写）。在桌面应用部门，由 Office 产品单位的用户培训小组来准备 Word、Excel 与 PowerPoint 的培训手册。在开发组内也成立了一些小组，与微软的海外分支机构一起准备产品的非英语版本，而产品管理组一般都有国际经理来主管外国市场营销。微软的一个日本分部微软 K.K.，则制作了桌面应用软件的日文、中文及朝鲜版本，该分部直接向部门经理负责。

表 1.2 将微软的 17000 个雇员按其职能进行了分类。大约有 30%的员工（5100 人）在海外 36 个外国分支机构从事销售、产品支持以及制作当地语言版本软件的工作。（海外销售额占微软零售销售额的 70%左右，具体数据见前言中的表 2）。在美国本土工作的 13300 名雇员中，大约有 1850 位软件

开发员，1850 位软件测试工程师，400 位程序经理与产品策划人员。大约有 2100 位员工从事消费者支持服务；4000 人从事销售、市场和咨询工作；600 人从事用户

图1.2桌面应用部门(1995)

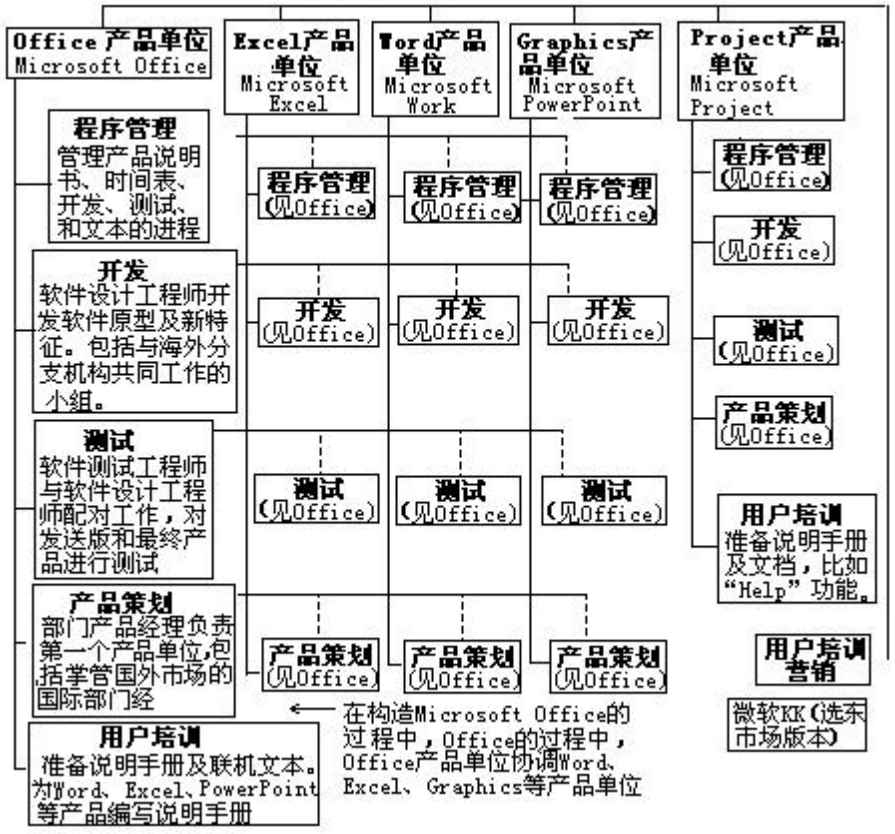


表 1.2 微软职员的大致分类（1995）

数量	职能领域
400	程序经理和产品策划人员
1850	软件设计工程师
1850	软件测试工程师
2100	消费者支持工程师
4000	市场、销售与咨询服务
600	用户培训
2200	营业与行政管理
300	研究
4500	海外人员（各个职能部门，其中包括 400 名开发员）
17800	总计

数据来源：开发部门主管戴夫·穆尔于 1995 年 7 月 18 日所发的电子邮件。

培训工作；2200 人从事营业与行政管理工作；300 人参与研究工作。

规模稍大的产品单位，如 Office，Windows NT、和 Windows 都各自有 300—400 名员工，其他的一些产品单位也都拥有 200 名以上的员工。一般来说，

操作系统开发组拥有比应用软件开发组更多的开发员与测试员；而后者则更需要产品策划与程序管理方面的人才，因为应用软件的各特性很直观，其市场也多定位于非技术型顾客。总而言之，表 1.2 的这些数字反映了近十年来微软翻天覆地的大变化。想想看，在 20 世纪 80 年代早期，Excel、Word、MS－DOS/Windows 等部门竟只有 10 位或更少的开发员！产品支持

服务部的人员在 20 世纪 80 年代早期也就几十个人，现在已发展到 5000 人，为产品开发组提供了宝贵的信息反馈（见第 6 章）。微软素以高效率著称的销售与营销队伍，包括了成百名咨询人员（他们帮助公司安装数据库和网络系统）。这支队伍也是由 10 年前的寥寥数人发展到如今的 3000 员工。

销售与营销开支（包括广告费用）构成了微软最大的花销，大约相当于 1994 年营业收入的 30%。营业成本（包括产品支持）大约相当于 1994 年营业收入的 15%，研究与开发成本（看作当期发生的费用）为 13%，一般管理费用为 3%。扣除以上各项成本支出，微软的税前净利为其营业额的 37%（见表 1，导论）。RD（研究与开发）、销售与市场营销以及产品支持等各项开支的急剧增加，使盖茨、梅普尔斯与公司其他高级经理寝食难安，千方百计想办法来削减这些费用——比如，在公司内实施更为规范的管理制度，促进产品组之间的协调合作，提高产品质量，改进产品功能使用户使用更为方便（用以削减用户支持费用）。但在另一方面，产品组之间相互依赖性在加强；产品的生产规模在扩大，其复杂性也在增加；产品不断升级换代，使新版本与以前版本之间兼容性的保持也越来越困难。这一切，都使公司管理任务更为艰巨，有时，它们甚至使微软为在预定日期内及时推出产品上市而进行的各种努力付之东流。Office4.0 与 Windows95 就是因为上述各种原因而迟迟未能问世，对比后面我们还会详细讨论。

系统部门和应用部门的古老博弈

1992 年，微软高级管理层的变动中，最引人注目的便是系统和应用部门正式合而为一，划归麦克·梅普尔斯统一管理。这一措施很容易令人想起创业之初，各项目组直接向盖茨负责的情形。把这两个部门划在一块儿很有意义，因为它们之间有着长远的相互排斥的历史，但是在战略上它们之间又变得越来越相互依赖，越来越不可分离。我们可以毫不夸张地说，只有深入了解 Windows 如何运行才能写好应用软件，而对 Windows 与一些技术（如对象链接与嵌入）的演进施加影响力也同样十分关键；另一方面，如果以微软为首的应用软件开发公司不再编写应用软件，一些新的操作系统如 Windows NT 就绝对不可能在市场上占据一席之地。问题在于，在历史上，微软的系统与应用部门相处并不融洽。系统人员（尤其是 Windows 3.1，Windows 95 的开发者），与应用软件人员相比，更偏向于采取松散懈怠的工作方式，虽然系统软件往往更难开发与测试，更宜采用严格的组织管理制度。

梅普尔斯成功地使应用部门做到以质量为本，管理更为规范化。他对系统部门也进行了改革，赢得了一部分人（如原先在 DEC 工作的戴夫·卡特勒）的支持。遗憾的是，直至 1995 年他还未曾完全成功。他总结教训时说，自己习惯于把重点放在用户问题与产品进程上，这种作法更适用于应用软件部门而非系统软件部门，其原因有二。其一，操作系统得在各种不同类型硬件上运行，而对各种不同硬件的测试需要“冗长及大规模的β测试”（指从用户的角度对初级产品版本进行测试），使用户能够使用各种不同的机器与应用

配置的组合。这种测试的进程很难预测。其二，梅普尔斯发现操作系统项目一般来说需要更多的员工，其构件也与许多不同产品相关联，这使得项目管理繁琐不堪。他解释说：“应用软件部门效率比较高的一个原因是他们能够以小组形式工作，交流便利，依赖性少。而系统软件部门的队伍庞大臃肿，相互依赖性强，往往人浮于事。这两个部门的工作过程截然不同，应用软件部门的小伙子比系统软件部门的小伙子更易受工作过程的驱使。我之所以这样说，不是基于主观臆断，而是基于客观事实。”

结果，在系统软件部门内部经常出现一些人所谓的混乱局面，甚至涉及到形象极佳的 Windows95 项目，但 Windows NT 却是一个例外。正如梅普尔斯所说的：“在系统软件部门，各产品开发过程之间的差异很大。Windows NT 组主管戴夫·卡特勒组织严密，专备了一本工作簿极有条理地记载了所有事项……这是一种非常严格的开发程序。而相反的，Windows 组与 DOS 组却更像一群持枪抢劫的乌合之众，毫无计划可言。他们常满不在乎地说：‘让我们开始编程吧，看看会出什么结果’。”梅普尔斯认为所有的系统软件组必须集中精力减少产品缺陷并且提高预测完成日期的准确率。

虽然两个部门界限分明，但是两边围墙内所有富有经验的经理人员都认为系统软件部门比应用软件部门更成问题，即使两者间的界限正在逐渐变化。克里斯·彼得斯在成为 Office 副总裁之前，是 MS-DOS 2.0 与 Windows1.0（还包括 Excel 和 Word 的各种版本）的开发员。他指出说明书和项目管理严重缺乏规划：“你会发现系统软件部门比应用软件部门要远为缺乏组织性……我从不认为他们有合适的说明书……他们会说，我们在这项目中采用的是里程碑式的技术，而当你问他们在这一项目中一共有多少个里程碑式子项目时，他们便会面面相觑，无言以对。”Windows NT 的高级产品经理理查德·巴斯同意克里斯的上述看法，他说：“当我 1990 年刚进微软时，发现应用软件部门和系统软件部门在管理方法上委实有巨大的差异。事实上，系统软件部门时刻关注着应用软件部门，认为他们管理有方，在那里开发过程得到更为有效的控制。”

系统软件部门与众不同的文化氛围，与史蒂夫·鲍尔莫的管理风格密不可分。多年来，他偏爱于营造相对宽松的环境，从不设独立的测试组，给予程序经理和开发员创新的自由，即使到了开发环节末期也可以进行产品的变动。前任 MS-DOS 与 Windows 的测试主管戴夫·马里茨（现在已离开微软），对他以前的上司史蒂夫·鲍尔莫作了如下评价：“他十分精明，只是在他下面做事很困难，因为他总是随随便便……他是那种散漫不羁的人，所以他无法控制系统软件部门。”Office 组的高级程序经理麦克·康特，娓娓道出了系统软件部门拥有独特文化的原由：

> 我认为他们正在进行着许多观念的转变：其中之一即从面向 OEM 转向面向最终用户……在一定程度上，这是微软历史和文化的一部分。史蒂夫·鲍尔莫是系统软件部门的最高权威。他不是那种架子很大、官僚气息浓重、有板有眼、一丝不苟的人。所以，系统软件部门从来也没有在规范化方面有所进展。另一个不可忽视的原因就是他们倾向于技术主导一切。开发员拥有极大的发言权，但他们很不理性，没有严密的组织结构，如同一盘散沙。如果依着他们的性子去做，那决不会遵循获得用户或增大市场占有率的原则来办事。所以系统软件部门一直没有起色。

虽然应用软件部门以前也如同未开发的西部一样没有秩序，但麦克·梅普尔斯却将它治理得井井有条。以上的观点来自于应用软件部门的一个职员，所以你可以对此持怀疑态度。史蒂夫·鲍尔莫在公司内具有很高的威望，尽管已经离开系统软件部门，但仍然保持着强大的影响力。盖茨对他也是言听计从，倚为股肱……但麦克却是一位深谋远虑的人物，他不会下车伊始就发号施令：“好吧，让我们打破一切旧规，按我的方式从头开始。”恰恰相反，他做事很有策略，极有分寸，我认为他是在潜移默化地改变公司。

康特的这个观点在微软各阶层的员工之间引起了共鸣，他们认为除此之外，还有其他许多理由，其中之一便是地域上的差异：应用软件部门座落在整个微软园区内更具现代风格的北部地区，一条大路把“宏伟豪华的棕色大厦与南部郊区”分隔开来（戴夫·马里茨如此描述）。（尽管应用软件部门条件更为优越，但盖茨仍然将他的办公室与系统软件部门设在一处。）产品的不同技术特性也使这两大部门之间差别很大，不过，由于操作系统中用户界面和特性越来越重要，这种差别已在逐渐变化。产品开发主管克里斯·威廉姆斯认为系统软件部门的员工在取得更多的实践经验后会逐步改变，他说：

> 系统软件部门的员工仍然觉得一个产品在成熟之前，不应进入市场，所以他们开发软件的方法全然不同……你将不得不熬过三个或四个项目周期，不得不数次忍受糟糕的管理过程所带来的痛苦……而系统软件部门员工的周期……比应用软件部门的要长两倍。刚从 Windows NT 组离开的家伙跑到我这儿来抱怨：“我们实在不想再干了！”至于在 Chicago（Windows 95 项目）组的那一帮人，我估计等产品推出以后，也会有一部分跑到我这儿来说同样的话……这就是两个部门的本质差别。我认为另一个重大差异就是在系统软件部门总是由那些过去获得成功的人来具体负责项目。Windows 3.1 毫无疑问是个相当成功的项目，人们用老一套的方法成功地开发了这个软件。既然它并没有失败，你怎么能指望他们去改变做法呢？

新市场和技术

微软并不仅仅注重操作系统与应用软件的开发和生产，虽然这两大领域给微软带来了丰厚的利润，构成了公司营业收入的绝大部分。新的用户产品和联机系统的市场增长最为迅猛，这两个部门有着截然不同的另一种文化，它们更关注于前沿科学技术和用户日常生活的结合。集团副总裁内森·麦尔沃德（曾任微软的高级技术主管）在最近的一次采访中对他的主要工作任务进行了描述：（1）帮助盖茨制定公司的技术策略，比如对一种技术作出购买还是自己开发的决定等；（2）创造未来的产品；（3）开展研究活动，重点集中在计算机科学领域，以支持微软各产品组，并努力使盖茨对未来个人机的想象变为现实。

从一个更为广泛的意义上来说，麦尔沃德所扮演的角色就是为微软进行“拳头产品的转换”，正如他所描述的：

大部分公司都面临着这样一个传统问题，就是它们都有自己的拳头产品，但却难以再进一步地发展。微软的特别之处就在于它从来都不惧怕改变自己的拳头产品，其中主要的原因就是我们的高级经理都是技术型人才，我们都懂得技术，而技术的力量是无穷的，一天之内就可使整个产业发生巨变。也许你在商业上很精明，但是无论你有多么能干，如果不懂技术，就无法保证自己在技术革命的浪潮中安然无恙，它会使你在倾刻之间粉身碎骨。[13]

高级用户系统部门是平台集团的一个组成部分，在战略上显得极为重要。它在研究、宽带通讯和联机系统等部门的配合下，开发出家用和信息高速公路多媒体产品并打入市场；它还涉及到交互电视系统和微软网络等许多领域；它也生产未来产品。这个部门与其他产品单位十分相似，都有程序管理、开发、测试与产品策划等职能机构。不过，在促进发明与探索新思路方面，各职能小组之间的职责分工不很明确。[14]

瑞克·罗歇德，一位前卡耐基——梅隆大学计算机科学系的教授，掌管着微软的研究部门，其工作也包括交互电视系统的研究。这个部门大约由 250 位科学家组成，每年的预算支出超过 1.5 亿美元。它的主要工作目标是使新技术在 5—10 年之内实现产业化，同时也包括一些能迅速带来效益的短期项目。研究人员们正在研究：能提高编程效率的工具；操作系统和用户界面的人工智能应用系统；计算机决策系统；语言处理和识别系统；超级计算机的设计；快速响应的视频服务器技术；运用录像、声像、文本等媒体信息创造交互电视和多媒体产品以及袖珍信息系统（例如袖珍 PC）。罗歇德解释说盖茨认为微软的规模已经足以维持一支需投入大量资金的研究队伍了，盖茨本人以及其他一些微软领导也认为必须拥有更多的微软独创技术：

从本质上来说，比尔希望微软的研究部门能推进特定领域的研究工作……我们希望自己的研究部门是世界一流的，与研究领域内任何一位竞争对手相比都毫不逊色甚至更胜一筹。比尔把这当作一项长期的奋斗目标，微软应该有能力达到这一目标，因为它已具备了一定的条件，达到了一定的规模。这一目标的完成已变得越来越重要，因为公司要想立于不败之地，必须开发具有自己特色的技术和思想。

研究部门为许多产品的开发作出了卓越的贡献，比如新的联机服务，新的操作系统（Windows 95，Windows NT）以及提供“智能”帮助并使用户能又快又准地掌握 Windows 操作的新用户界面等等。梅尔沃德和罗歇德都对微软能最终生产出新一代的技术与产品充满信心，他们希望自己的研究部门能超越许多领先公司的中心实验室。比如，施乐公司（Xerox）的帕罗阿图研究中心（PARC）一直在个人计算机和图形用户界面的研究方面处于先锋地位，但是始终未能将之商业化。IBM 在将它的研究人员发明的计算机技术成果产业化方面也存在着重重障碍（如 RISC）。

罗歇德向我们解释了他们如此自信的原因：“施乐公司的 PARC 的问题并不是它自身的问题。它没有做错任何事；它的研究人员都干得很出色，其工作无可指责。只可惜明珠暗投，施乐公司墨守陈规，鲜有创新……我认为这是一个教训，公司的经营运作必须与研究人员的工作紧密配合，这样才能使

研究成果迅速转化为生产力。”梅尔沃德接着说：“微软与其他公司的不同之处在于它的研究机构直接与公司的最高决策层——总裁相联系，而其他公司的老式实验室则由于它们一贯的‘象牙塔综合症’，极易与其他部门隔离起来。我可以直接向盖茨汇报工作，盖茨也经常参与我们部门的决策，同时我们也对公司内其他部门的决策发表意见。”[15]（据报道，盖茨花费其 1/3 的时间与梅尔沃德一起构想未来的产品。[16]）

原则三：尽可能任用最具头脑的经理人员——既懂技术又善经营

我们前面谈论到的微软高级经理们（比尔·盖茨、麦克·梅普尔斯、史蒂夫·鲍尔莫、内森·梅尔沃德，保罗·马里茨、皮特·赫金斯、克里斯·彼得斯、瑞克·罗歇德等）也许有其弱点，但他们都很有能力，对微软的发展起到了不可替代的作用。事实上，盖茨与其他经理经常鼓吹他们只聘请聪明人充当经理、开发员与测试员等等。但是到底‘聪明’是什么意思呢？比尔·盖茨认为，“聪明”就是能够迅速地有创见地理解并深入研究复杂的问题，如他在 1994 年接受采访时所解释的那样：“聪明人一定反应敏捷，善于接受新事物。他能迅速进入一个新领域，给你做出头头是道的解释。他提出的问题往往一针见血，正中要害。他能及时掌握所学知识，并且博闻强记。他能把原来认为互不相干的领域联系在一起使问题得到解决。他富有创新精神和合作精神。”[17] 盖茨自身便显示出这些可贵的素质，并积极在微软的经理、职员和应聘者当中挖掘此类人才。

随着微软的发展壮大，盖茨已经私下访查了上百名程序员、经理以及技术专家，他们丰富了盖茨对技术知识的了解。他从中挑选了一小群更为出色的员工组成了一个非正式的智囊团，帮助他进行决策并监督关键项目和首创性举措的实施。盖茨习惯于提拔年轻的程序设计专家进入高级管理层。但随着公司的快速发展，盖茨的这种习惯使得微软内部既懂技术与业务又擅长经营管理的中层经理十分短缺。

智囊团

微软智囊团的核心大约由 10 来个人组成，他们管理关键产品领域和公司新的举措，组织非正式的监督组来评估每个人的工作。也有许多在各项目工作的高级技术人员，组成了智囊团的外围；一些人还是公司的元老，从微软建立之初便一直在这儿工作，然而越来越多的成员来自对手公司或者是个人计算机领域之外新技术方面的专家。戴夫·穆尔从不认为自己是智囊团核心圈子的成员（“我比那些人要平凡一些”）。他说，智囊团的成员不属于任何部门，但是他们以其非凡的知识与经验而得到广泛的承认。他对智囊团作了进一步的解释：“它很不正式。你成为其中一员的一个主要原因便是你的事业阶梯。每个开发员都有一个事业阶梯评级……每一个职能领域——开发、测试、程序管理、用户培训，都有其自己的事业阶梯。”

微软智囊团成员包括公司的最高层领导、高级开发员和程序经理。[18] 下面对其所包括的公司领导人员作一大致的描绘。执行副总裁史蒂夫·鲍尔莫（39 岁）在公司内德高望重，深受爱戴。他是盖茨在哈佛的同班同学，在 1980 年加入微软，以前曾在斯坦福学习一年的管理。人们可能会觉得他不是管理系统软件部门的最佳人选，但他不惧困难，勇于接受挑战——在 20 世纪 80

年代中期 Windows 项目迟迟无法完成，成为一堆无法收拾的烂摊子时，他挺身而出，承担了开发责任并成功地将其推向市场。他现在主管销售和支持集团，因为产品销售与用户支持领域变得越来越重要。集团副总裁内森·梅尔沃德（36 岁）是普林斯顿大学的物理学博士，师从于诺贝尔奖获得者斯蒂芬·霍金。在 1986 年盖茨兼并梅尔沃德创立的软件公司后加入微软。他负责公司网络、多媒体技术、无线电通讯及联机服务等。

集团副总裁皮特·赫金斯（37 岁），在取得斯坦福大学的工商管理硕士（MBA）学位后，于 1986 年加入微软。他以前主管桌面应用部门，现在与梅尔沃德一起负责新成立的应用和内容集团。集团副总裁保罗·马里茨（40 岁），是微软的元老重臣，1986 年就在微软服务了，其主要工作是制定公司长期产品战略并监督微软的操作系统软件和其他关键的用户技术等的开发。在这里，我们有必要介绍一下新加入微软的罗伯特·赫伯德（53 岁），他在凯斯西部保留地大学获得了计算机科学的博士学位，然后在宝洁公司干了 26 年，最后升为宝洁公司的高级副总裁，主管管理系统部门、市场调研与广告策划部门以及技术获得部门。他于 1994 年加入微软后，因其在市场营销方面具有十分丰富的经验而深得盖茨敬重，因为用户产品日新月异，已成为微软发展最快的领域。

高级副总裁布兰德·斯尔文伯格（41 岁）是个人操作系统部门的经理，1990 年从 Borland 公司来到微软就职负责 Windows 3.1 与 Windows 95 项目的开发。罗格·海纳（43 岁）是负责开发员与数据库系统部门的高级副总载，曾在 DEC 工作，之后，跳槽到苹果公司，负责 Macintosh 软件的开发工作。1993 年加入微软。詹姆斯·艾尔金（43 岁），是领导业务系统部门的公司副总裁。他于 1991 年从 Banyan Systems 公司来到微软。现在他主管 Cairo（Windows NT4.0）项目。高级副总裁克雷格·蒙代（46 岁），负责高级用户系统部门。他以前经营过一家超级计算机公司：Alliant Computer Systems，于 1992 年加入微软。副总裁乔纳森·莱泽洛斯（44 岁），是公司战略关系部门的负责人。他于 1986 年加入微软，是一位市场营销专家。副总裁克里斯·彼得斯（36 岁），主管 Office 产品单位，该产品单位每年创造的销售额和利润大约占整个公司的一半。

副总裁瑞克·罗歇德（43 岁），研究部门负责人。在 1991 年加入微软以前，他是卡耐基—梅隆大学的计算机科学教授，颇有名望。罗歇德是 Mach 的主要设计师（Mach 是一种用 UNIX 语言开发的操作系统，用作 Windows NT 的一个组成模块）。戴瑞尔·鲁宾（41 岁）是软件战略部门的副总裁，网络技术的专家。他在 1986 年就已经是微软的一员了。因为优秀人才的不断加入，他的影响力在慢慢减弱。查尔斯·西蒙尼是这精心挑选出来的智囊团成员中一位大师级的程序设计师。他是斯坦福的计算机科学博士，曾经为施乐公司的 PARC 设计图形应用程序，包括著名的“华丽”（Bravo）文字处理软件。他在 1981 年离开施乐公司，在接下来的 10 年中一直致力于微软的应用软件开发，Multiplan、Word、Excel 等都是在他的指导下开发成功的。

智囊团中的有些人物尤其出类拔萃，显得比别人要技高一筹，斯尔文伯格曾经对其年轻的同事克里斯·彼得斯作过这样的评价：“克里斯·彼得斯无疑是深知开发过程要髓的人物。他可以说是最为成功的。他的开发过程正被公司其他员工竭力仿效，堪称完美的榜样。我们大家都认为可以从他身上学到很多很多的东西。”与微软其他高级经理人员一样，彼得斯具有很过硬

的技术背景，而且他比许多人都要见多识广，经验丰富，因为他在系统软件、应用软件与一般管理部门都干过一段时间，是位难得的通才。他在华盛顿大学取得电子工程的学士与硕士学位，对软件工程颇有心得。1981 年加入微软以后,他先后为 MS－DOS 2.0、Microsoft Mousel.0、PCWordl.0 和 Windowsl.0 进行程序设计。之后大约有 5 年时间，他担任 Excel 的开发经理。由于运用了新的开发过程技术，他成功地把 Exce13.0 推向市场，仅仅比预定时间晚 11 天。之后他成为 Word 产品单位的总经理，为微软创造了好几亿美元的收入。1993 年，彼得斯荣获副总裁的头衔，主管新成立的 Office 产品单位，是微软制定开发桌面个人机战略的中心人物。

智囊团另一位重要人物是麦克·梅普尔斯（53 岁），在 1988—1995 年间担任公司的执行副总裁。现在他已激流勇退，成为公司顾问，专门对有关兼并与招聘新人等事项提出建议。梅普尔斯不是开发员，但他却是俄克拉荷马大学的电子工程学士与工商管理硕士。他看起来知道如何来管理手下这一批桀骜不驯的技术员。梅普尔斯徐徐地却又坚定有力地推动了公司沿着产品开发不断规范化的大道前进。他当初毅然离开 IBM，是因为他看到了微软的巨大发展潜力的同时，对 IBM 十分地失望。进入微软后，为帮助盖茨摆脱繁重的官僚式的组织管理包袱，他把 IBM 的四项基本准则引入微软。第一条准则便是包括雇用、培训、工资报酬及晋升道路在内的一整套人事管理方面的作法：

> 在人事管理方面，IBM 干得很出色。他们有一整套的评估、监督和奖励员工的方法并能保持其连续性。微软在这方面将采用 IBM 的经验并加以改进。长期以来，微软没有人喜欢被检查系统、管理系统或工资提升系统给“固定公式化”……由此造成的结果便是人们做自己喜欢做的事，并把所有的东西都只记在脑子里面而不愿提笔写下来。不过由于公司小，没有造成什么更大的恶果。但现在公司规模扩大了，由于没有一套连贯的人事管理制度，内部开始分化，而且这种分化比我们现在所认识到的要严重得多。所以，我们开始让员工在各组间自由流动并树立员工工资提升和职位晋升的一套制度。戴夫·穆尔便花了很多时间来定义一个 D10 开发员与 D11 开发员之间的区别以及他们如何提升等等……我们企图使每个人的头脑里都存在一种根深蒂固的观念，即在项目组之间调动人员时，在工资待遇、职业升迁等方面不会有多大的差异。

第二个准则，就是培养中层经理。微软向来缺乏这方面的管理人才。梅普尔斯说道：“我们经常采取休假会、经理会议等方式进行管理开发。这可能比较浪费时间，但是大家聚在一起聊天可以集思广益，互相切磋管理经验。”第三个准则，则是继续设立专项职能，如开发、测试，但是必须保证让员工自由流动以获得更丰富的工作经验。梅普尔斯希望公司内能避免思想偏狭、部门敌对以及争权夺利，这些陋习已经使 IBM 与其他一些大公司深受其害：

> 员工们现在思想开放多了。开发员不再对测试员横眉冷对。一般来说，各职能组之间总是会有隔阂与敌意。事实上有时怨结深了无法解开，一些人不得不另换一个工作或干脆离开微软，不过我认为现在已经

很少听到这样的一些话了，比如：“ 测试员水平太低了，开发员才是真有水平。 ”或者说：“ 开发员不行，营销人员的素质要高得多。 ”每个小组也越来越具有凝聚力了。不过，公司对开发员还是另眼相看，无论工资、声名还是职位晋升都向他们倾斜。在微软你若是个开发员，你会觉得趾高气扬，比别人高出一头。这显然是历史与文化的原因造成的。我们公司是技术推进型的，高级员工当然大多数都是技术人员出身。

如今的 IBM 公司内部冲突不断，各小组都隐恶扬善，尽量隐藏‘ 坏消息 ”，使公司管理层与质量保证人员难以知晓。梅普尔斯竭尽全力来避免在微软发生类似的情形。比尔·盖茨正属于那种不愿听到坏消息的人，一些微软经理们也极不情愿承认错误或执行时间表上的疏忽。不过，盖茨一旦察觉真相，会因人们不及时告知以致贻误战机而暴跳如雷。梅普尔斯认为员工之间应该进行经常性的坦诚的对话，旨在开展建设性的相互批评使任务完成更为顺利圆满。盖茨持相同的看法，他希望人们能敞开胸怀无所保留，尤其是在产品开发出现严重问题的时候。他希望人们为公司的总体利益着想，而不仅仅基于单个项目或个人职位的考虑。盖茨与梅普尔斯也都不希望看到人们犯同样的错误的情形一而再、再而三地出现。下面梅普尔斯把微软与 IBM 的气氛作了一下比较：

[在 IBM]有一些质量保证人员，不过没有微软这儿的多，并且它起着更大的审计监督作用。每每开发部门进行测试之后，质量保证人员总是说他们弄得很糟糕，全无可取之处。微软不像 IBM 这样互相之间对抗性如此之强。IBM 似乎故意采用冲突性管理制度：如果你让员工显露其真实偏好，他们各自所代表的利益便也能一览无余，这样你就可以获得比较全面的信息进行最佳决策。在一定程度上，这可以说是一种很有效的管理方法。我们微软不这么做。我们试图使员工意思一致，达成共识；并做到胸襟宽广，顾全大局。当我在 IBM 开发部门时，我发现在这儿你别无长进只能学会如何向上级隐瞒坏消息的伎俩……实在走投无路、被逼无奈时你才会把坏消息给捅出去……最后你将自食恶果，在劫难逃。事情与你所料的正好相反，老是与你作对，你也没有规避风险的经验，这样，在每个人都觉得项目进展顺利时，突然它就整个都崩溃了。微软的风格是把事情全部摊开，员工间开诚布公地进行交流。例如我可以收到一份由 Word 开发员发来的电子邮件，说事情进展不顺利或者其他什么。总而言之，这种交流是有利于微软发展的。

梅尔普斯的第四个准则便是：一个软件公司必须为产品开发确定一个过程。他说：“ 如果你在 IBM 工作，你会觉得过程很有价值但也有不少问题。确定过程很重要，每一个特定的组织或单位都必须有自己的特定过程。 ”梅普尔斯回忆说，自从 1989 年的休假会活动以后，他和公司其他领导决定“ 每个小组只要找到自己的过程并能坚持之，便可以确定他们自己的过程了。 ”他希望能发现一个可重复出现的开发过程，但从不企图确定一个适合微软所有的小组的标准开发过程。每个业务单位的核心人物都努力把他们开发与测试软件的最优过程记录下来。这样过了大约一年半，在公司内部出现了一致意见，正如梅普尔斯所说的：“ 虽然人们还是觉得在开发项目时应该保持一

定程度的自由。不过我认为有一些基本的做法已经渗透到整个组织。正是这些基本做法推动着整个组织前进与发展。”下面是梅普尔斯列出的一些一般原则；至于盖茨所开列的清单我们以后将会讨论到：

- 人们自己制定时间表。
- 为常常发生的预料之外的拖延留出一些缓冲时间。
- 考虑到有发生变化的可能，不要过早地完成说明书以免浪费时间。
- 采用里程碑式管理，从最难的问题入手。
- 将用户问题置于技术或过程的考虑之上。
- 将能力不同的员工进行合理配置。

技术过硬的经理人员

盖茨与公司其他的早期领导很注意提升技术过硬的员工担任经理，这不仅仅局限于开发机构之中。瑞克·罗歇德在卡耐基—梅隆大学任教时，常与出色的计算机科学系的学生打交道。在他作为一个不带任何偏见的旁观者接触微软时，却也禁不住为微软高级员工的高超技术倾倒。他说：“我对第一次访问微软的情形记忆犹新，整个管理层几乎都由技术上十分出众的人员所组成。在整个计算机行业这种情况很少见。”戴夫·汤普森获得康奈尔大学的学士和硕士学位以后，曾在 DEC 与 Concord Communications 工作了 13 年，之后于 1990 年加入微软并主持 Windows NT 的网络设计。他说：“微软的一个强项便是它的经理人员都是技术型人才，即使在最高管理层也是如此……我认为经理确实应该由技术人员来担任。小组长，即位于第一线的经理，绝对都得会编写代码，能够对小组开发的软件作出适当的修改。”

在微软，各职能部门的经理在升任高级领导之后，并不会放弃其原先的职能性工作。即使像 Word 与 Windows NT 这样大组的开发经理也会利用其 1/3 至 1/2 的时间编写代码。经理们为了及时解决各自组内发生的问题并作出正确的决策，有必要对技术领域的种种问题有一个清晰的了解。Windows NT3.0 项目的软件设计经理娄·帕雷罗里让他手下的经理们像他一样每天花一半的时间编写代码：

> 我在组内制定了许多规则，其中最重要的一条是每个工作人员都得编程，谁也别想坐在那儿发号施令、管理别人……我发现管理者很容易失去目标，他们总是无法认识到问题的本质并且反应迟缓。如果你始终不放弃编写代码的工作，你就能对项目的进展情况了如指掌，及时地发现并解决问题……我大概每天花一半的时间编写代码并寻找项目的缺陷。

作为应用软件领域的经理，克里斯·彼得斯也持同样的看法。在他还是 Word 单位的总经理时就认为：

> 在微软内部，职能部门的经理还是会花很多时间去干自己的老本行的，这就意味着一个拥有 60 位开发员的开发小组经理会花费相当多的时间来进行程序设计……在一些大公司内部……他们把具体操作的层次向下移。你一旦当上开发部门经理，很快就会以自己身居高位、日理万机为由放弃编程；同样地，特性小组组长会以自己重任在肩而不愿编

程；至于程序设计员，也会觉得自己十分繁忙、分身无术而不再编程……虽然，我是 270 名员工的领导，似乎不再需要做什么具体的工作了，但是我还是为 Word 新版本编写了其中一个特性。

中层管理的不足

微软提拔技术骨干担任管理人员的初衷之一是鼓励他们继续在技术领域发挥作用，但这样一来他们花在管理方面的时间就大打折扣。事实上，正是对正规培训、晋升机制以及成文规定等管理程序的忽视，才导致了微软在管理上的不尽人意。戴夫·马里茨认为微软的症结在于过分重视管理人员的技术能力而从不过问他们的管理才能。他说：

> 这家公司在管理上已近枯竭……在获得提升时，我竟毫无察觉，直到有一天发现工资涨了才明白是怎么回事。由于经理对我很是敬畏，所以我的工作常常不用接受检查，这就是微软在中层管理上的问题……在我看来，微软缺乏行之有效的工资晋升和奖金制度。选拔管理人员的标准就是他们的技术水平，这种政策导致一旦你被雇用为开发员后，就会逐级晋升，最好的开发员将最终成为最高级别的管理人员。

马里茨和公司内其他许多人一样都认为技术管理人员应当具备超群的技术才能，否则在公司内将无法得到其他员工的尊敬。但同时他也同意管理人员不应当仅仅是优秀的程序设计师，他们应当具备领袖的气质和魅力，可惜在微软的中层管理人员中，这样的人实在是凤毛麟角。他进一步阐述说：

> 要想成为一位经理或者领导——虽然在本质上经理就是领导——你必须具备两种基本的素质，如果你两者兼而有之，那就是天生的领导者：其一，你的技术才能应当超过你的同事或将要成为你手下的那批人。其二便是领袖气质。一般来说微软的经理人员都不具备这种素质。我认为自己属于后一种气质类型而非技术型的管理人员，就这一点来说，我与微软其他人不太一样。你会发现，在第一线管理层的经理人员大多不具备领导才能……而在第二线，具备这两种素质的经理人数要多一些……职位越高的经理素质越高。在第三、四层，经理们能把管理工作干得很出色，同时他们的技术才能也十分卓越。

梅普尔斯承认他加入微软以后不久就发现公司中层经理人员的缺乏，他与戴夫·穆尔以及其他一些人都在日复一日、孜孜不倦地努力以求改变这种状况。但是微软发展实在太快了，盖茨与梅普尔斯不得不放手让年轻人承担更大的权利和责任，让公司的高层领导们也继续提升那些具有技术背景并且在技术方面取得巨大成就的员工。理查德·巴斯承认说，寻找合适的中层经理是微软现在所面临的“最大挑战”：

> 毫无疑问我们没有一支优秀的中层管理人员队伍，像克里斯·彼得斯那样出色的经理实在不多，所以现在麦克又多了一项日程安排——培养更多的管理人员……我们公司现在面临的最大挑战便是发掘一支中层管理队伍……但显然这项任务比较艰巨……通过几年来的经验积累

> 我们终于发现应当重视并且培养管理才能……微软的文化传统是如果你能大叫大喊并且干劲十足，你就是经理。不过就在去年，情形有所改变……在过去，只要你能真正地做出一番事业，你便是经理。但现在这已经远远不够了。

原则四：聘用对专业技术和经营管理都有较深了解的一流职员

我们在下一章将会讨论到，微软有一套严格的招聘制度，以寻求具有技术才能并且渴望能利用这种才能来开发与销售软件谋取商业利润的有为之士。那些能融入各专项职能组并与同事愉快合作的候选人倍受考官青睐。经理人员一般都愿意挑选那些能够适应微软工作方式，具有独立思考、学习和活动能力，决策迅速，不需要正规培训和规章制度加以约束的年轻人。但众多人才济济一堂，既给微软带来巨大的好处，也相应地会产生一些不良影响。

一流职员的正面效应

在产品和职能经理以下，微软大约有 5000 名软件开发员和测试工程师。在软件领域，最好的软件编程员在同一时间内能比同一组内效率最低的程序员多编制 10 倍到 20 倍的代码，从这一点来看，吸引众多的计算机天才将是微软的一大优势，因为经理们所拥有的人力资源显然不能以员工人数简单相加来衡量，高比例的优秀人才，将使资源内涵大为丰富，其所创造的生产力更是难以估算。（我们可以拿微软与其他的软件公司作一比较。无论在欧洲、美国还是日本，这些公司在选拔软件开发员方面远不及微软苛刻，许多公司甚至雇用那些对计算机一窍不通的人，让他们在屋子里接受编制软件程序的训练。对这些经验不足、缺乏培训的员工来说，很难干出骄人的成绩，在复杂的技术面前也难以保持“清醒的头脑”。[19]）

事实上每一个与我们谈及这个话题的微软经理都大力鼓吹一个经过仔细挑选的精英群体的价值。麦克·梅普尔斯说：“员工素质是无法估量的财富。不是每个公司都像微软这样拥有最好的麻省理工大学或斯坦福大学的毕业生。”瑞克·罗歇德也认为“这里的优秀软件开发员比我所见到的世界上任何地方的都要多”。戴夫·汤普森持同样的观点：“我们公司与其他公司最本质的区别就在于所雇佣的员工素质不同。公司整个系统的基础就是员工们敏锐的思维方式和惊人的工作效率。如果你的员工在几个小时内就完成别人需要花许多天的工作，你就会拥有更大的灵活性，就会觉得有更丰富的资源可以利用。”比尔·盖茨也颇为自得地吹起了法螺：

> 这些最优秀的员工参与着程序设计、思想创新以及具体的编码过程，他们所熟知的代码范围宽广，这使公司受益匪浅。公司能拥有精通所有程序设计变化的人才真是非常幸运，尤其在工作进程受阻、你不得不小心翼翼的时候。他们能检查出任何编码过程中的缺陷，他们对各种程序设计变化所产生的副作用了如指掌……员工们都不情愿与差劲的开发员合作，他们会想方设法使那些人明白，如果你不是真正拔尖的开发员，你呆在这里只会痛苦失意。

优秀员工不仅能够利用其技术才能给公司创造丰厚的利润，还能使一个

大公司摈弃其官僚作风，变得相对灵活——更像一个小型公司。克里斯·彼得斯便持这种观点，他十分强调员工独立工作与解决问题的能力。他说：“我们公司有一大群优秀人才，个个雄心勃勃想干一番大事业。而有的公司却雇用一堆没脑子的笨蛋，由经理立下许多规矩来控制他们……我见过这种笨蛋公司，他们喜欢雇用平庸之辈并企图通过制定很多规矩来弥补其不足。这显然不太能行得通，因为问题的根本原因不在于缺少规矩，而是由于雇用了一堆需要规矩制约才能干活的员工。”

正因为微软聘请的是一群聪明人，公司从不强求员工墨守规章制度，尊敬正式职衔，员工也不会费尽心机拉帮结派以争权夺利。公司强调的是技术以及产品推出能力。戴夫·穆尔评论说：“在我们公司官衔并不受重视，把产品推向市场才是至关重要的。”布兰德·斯尔文伯格认为，权威和责任都会与那些最有才能的员工相伴：“微软开发部门的一个重要特点便是各个开发组内部的权力分布状况更多的是每个人的力量和能力的反映，这决不是千篇一律的俗套……我们公司的管理制度是很有伸缩性的，如果我雇用了一个对特性构造颇为在行的极为出众的开发经理，我估计权力会自然而然地向他转移。我对此并不会介意，而只会调整自己去适应这种情况。”

如果个人或者组之间发生诸如何种构件应当共享之类的冲突，盖茨会作出最终裁决。约翰·法恩以前是程序管理部门主管，现在担任 Excel 组程序经理，他对微软文化以及比尔·盖茨的作用作了恰如其分的评价：

> 在这儿成功行事的办法形形色色，从最不正式的耳边轻声嘱咐到最为正式的行政命令下达……规则丝毫不起作用……行政机制更不占统治地位。所以，当一位软件开发业务单位经理对另一单位经理提议说‘也许我们开发的软件应当包含在你们所开发的软件之中’或类似的其他建议时，你不要感到不可思议。这与行政机制毫无关系，只是为使公司利益最大化或者说为了赚取最大利润而采取的一种策略。不过因为这种决策的机会成本很高，一旦失败会造成巨大的潜在的负面影响，所以最后裁决权集中于比尔。在这种意义上，传统的管理方法起了作用。

一流职员的负面效应

雇用一些在工作中不愿受纪律约束的员工所带来的后果便是这些人不得不经常经历痛苦的试错过程。我印象很深的一点就是微软的开发员和测试员（也有少数例外）不喜欢翻阅软件工程方面的科学文献，一些其他公司在好几年以前就觉得很重要的软件开发方面的作法（包括进行更为详细的程序设计和代码的检查，对结构设计、构件共享更为关注，在项目管理过程中采用更好的定量度量制与历史数据等），微软却很迟才发现以至于痛失许多良机。

高级经理们——包括比尔·盖茨在内——很难指挥这一群优秀员工。他们不喜欢与人合作也不愿意妥协或与别人共享自己的发明成果。在第 6 章中我们讨论到微软在这方面做了很多工作并有了一些进步，但是公司做得的确还很不够。麦克·梅普尔斯给我们描述了他刚到微软时所见到的一些问题：

> 很少有人拥有项目管理的经验，整个公司的管理都很松弛。员工都很优秀，但他们不愿受人支使，不过他们很乐意被引导着去发现该如何去做。于是你可以想出一些主意让员工自己寻找更好的办事方法，而绝

不应该命令说“你必须选择这样的过程，你必须这么做”，这肯定行不通。只有推进员工坐下来进行探讨并找出解决问题的方法……才是合理的现代化管理模式，你绝对不应该采取独裁式管理……我以为我们的管理过程很有特色，只适用于微软而不适用于其他大多数[公司]……因为我们公司拥有非常非常出色的员工，他们的主观能动性很高。

随着微软的发展壮大，领导们引进了许多政策来避免各小组的重复作业，但这也仅仅只是工作指南或原则而不是严厉的规定。布兰德·斯尔文伯格对公司的这种转变作了如下评价：“回顾创业之初，我们确实不需要规章制度的约束。现在情形改变了，有必要引进一些新作法来适应新的变化。不过我认为传统文化势力强大，根深蒂固——比如权力分散，管理灵活但又不走极端等。虽然现在有一些硬性规定，但每个组遵守规定的程度大不相同……戴夫·穆尔或麦克·梅普尔斯不会给我下达命令说：‘你必须这样去管理你的开发组。’这不起作用。”在下一章，我们将要讨论微软如何定义其工作范畴以及它如何去组织与培养有创见的技术专家等等。

第二章
管理创造型人才和技术

组建职能交叉的专家小组

为管理创造型人才和有关技术，微软遵循了一条我们称之为“组建职能交叉的专家小组”的策略。我们将这一策略归纳为如下四条准则：

·以小组形式工作的职能交叉型专业部门。

·让各部门专家自行确定其技术专长并负责人员招聘。

·通过边干边学和言传身教培养新雇员。

·创立晋职途径和“升迁级别”以留住并奖励技术人才。

建立职能型专业部门或者为技术人员创立晋职途径和升迁级别并非什么独到良策，尽管这些措施对于那些技术居于主导地位的公司非常有用。微软的独到之处在于它授权这些专业部门人员自己来定义他们的工作，招收并培训新雇员。我们还感受到对于拥有如此众多员工和产品的微软公司来说，其既定方针和正规教育相对较少。这既是优点（工作种类灵活机动，人们有独立的思想性），也是缺点（人们不得不通过“试错”来学习知识，在许多时候不得不改弦易辙，从头做起）。

这些准则的产生、发展与微软公司特殊的历史、文化有很深的渊源关系。微软早期主要由软件开发员组成，这些人从发明、测试新产品，作出全部关于人员雇佣、市场营销和订立合同的决策，直至回答客户的电话咨询，工作事无巨细。随着微软产品和经营规模在数量以及质量方面不断的扩大和提高，公司增加了成千上万的员工。由产品设计失误或者交货延迟、产品带有严重偏差所带来的风险和成本迫使微软的经理们鉴别出关键的职能并使之制度化。但为避免职能过于分散，技术专家们实行责任交叉并采用多职能小组形式开展工作。

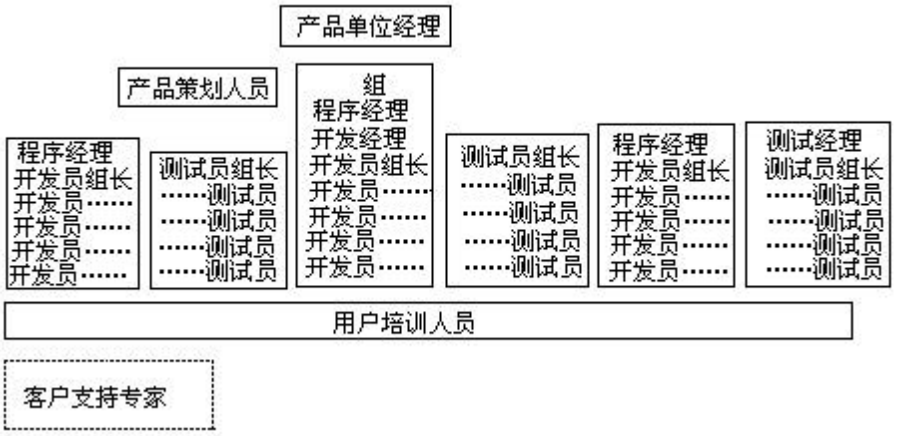
程序管理和软件开发，以及测试，是产品单位里主要的技术工作。微软特别注重寻找初露锋芒并能兼顾 PC 软件商业价值的“超级程序员”。他们可能并非本行业中最富革新意识的程序员，或者并非对其他公司软件工程方面的实践活动了如指掌，但他们在生产人们竞相购买的新特性、新产品方面具有很高的创造能力，并使之转化为公司巨大的收入和利润。从而，这些微软程序员得以跻身于本行业最具生产力和最有效率人员的行列。[1]

在产品单位里，程序经理、开发员和测试员以“特性小组”形式并肩“作战”。典型的小组由一位程序经理（他通常从事不止一个“特性”的设计开发工作）和 3 至 8 位开发员（其中一人为组长）组成，同时还配备平行的测试小组，其成员与开发员组成“搭档”（见表 2.1）。用户培训专家则作为产品小组的成员进行工作。产品经理负责为产品单位内的营销部门和产品规划组选定人员。客户支持人员虽是另一个独立部门的成员，他们中的产品专家却与单个产品单位密切合作。而某些职能领域（如程序管理）依然难以准确定义，其他一些职能领域（如程序管理与开发、程序管理与产品管理、开发与测试）难以截然分开，微软也从未试图把它们分开。

产品单位由高级经理负责每个职能领域存在困难（比如开发领域和测试领域）。职能经理们在特性小组里工作并向产品单位的

图 2.1 微软项目和特性小组组成情况

注：粗体字表示项目领导



主管汇报情况。这些主管非常类似其他行业（比如汽车行业）的项目经理。譬如在Office产品单位,此人负责协调几个相关项目——主要有Word、Excel和 PowerPoint。[2] 微软经理们将许多决策权授予特性小组。不过，产品单位经理和高级功能经理作出主要技术决策，在单位预算约束下配置资源，并解决不同组别之间的冲突。他们还与高级产品经理和主管组成小组来确定项目时间表，产品阶段性目标和产品上市的决策。

：建立以小组形式工作的职能交叉型专业部门

微软为每个基本的职能领域创立了工作思路，它使人们广泛地学习和分担责任，它把人员组合成多职能小组来作为更大的项目小组的一部分进行工作。每个小组的职能如下所述。

程序管理

也许微软最难以定义和胜任的职位是程序经理的工作。程序经理必须能管理产品项目的全过程，而且他还是连接软件开发与市场营销（产品管理）的关键纽带。一份由程序经理布鲁斯·雷恩起草的公司定向报告把他本人及其同事们描述为“开快车，坐办公室的技术本科生”。他还告诫说：“程序经理是领导者，是帮手，是协调人，但决不是老板”。雷恩将程序经理的主要职责范围开列如下：

- 产品策划。
- 产品规格说明书。
- 产品时间表。
- 产品开发过程。
- 实施折衷方案。
- 协调产品开发组。[3]

微软程序经理职能的起源可以追溯到80年代中期。那时，情况很明显，高级经理必须从“超级程序员”手中接管某些产品开发的控制权。同样明显

的是，管理设计过程不同于软件开发。杰夫·雷克斯，现任市场营销高级副总裁，1981 年由苹果公司转而加盟微软。他回忆起在 1990 年的一个案例研究中由超级程序员所制造的麻烦时说：“依赖单个超级巨星会产生许多问题：（l）他们的供给很少；（2）有些人不得不维持或更新他们已编写的软件（这通常提不起他们的兴致），并且他们的代码常令人费解；（3）有时他们不了解市场的需求；最后如果你试图把他们中的几位调配到同一个项目，那么你会在设计决策中遇到真正的麻烦——所谓“厨子太多汤难做”。[4]

吉布·布鲁门萨，1982 年加盟微软并在雷克斯手下工作。他曾帮助程序员定义了 Multiplan 电子表格的规格并将其推向市场。他的所作所为后来演变为程序管理一职，他说：“在 1984 年初，我们开始开发 Mac 机的电子表格。我参与其间并为开发组提供了一系列服务：帮助制作规格说明书，做人工检验并且决定哪些错误应马上修正，哪些错误可以留到下一个版本。我并不做设计决定，但我必须确保设计决定已经做好。整个过程进展顺利，于是他们决定给我的工作命名并将其制度化。”[5]

比尔·盖茨支持规范程序管理的决策，尽管这是对他所喜好的工作方式的一次大变动。在公司早期，他经常几乎重写微软所推出产品的每行代码，同时还管理产品的市场营销。但正如在第 1 章所讨论过的，到 80 年代中后期，盖茨管理所有事务已经不可能了。他同意建立正规的工作种类，比如产品经理处理市场营销事务，而程序经理的工作介于市场营销和产品开发二者之间。但盖茨并不想让程序经理与开发或营销工作截然分开，他想让程序经理如同吉布·布鲁门萨那样与开发员在小组里并肩工作，使二者相得益彰。盖茨深信，如果没有这种职责交叉与密切合作，各部门职能截然分开会导致失败：

> 我们过去没有程序管理一职，而开发员不喜欢做繁杂的文牍工作，并且不愿意参加讨论市场营销的冗长会议。于是最终出现了一种新的技术组合，一种非常有影响力的技术组合，它能处理所有繁杂事物……其他公司现在也有了程序管理一职，虽与我们的略有不同，但我认为全行业都已认识到这是一个重要的办法。……程序经理与开发员每天一起工作……这是所有事务中最密切的关系——比开发与测试之间的关系，市场营销与销售对象之间的关系更为密切。你可以将程序管理与开发置于同一经理的管理之下。事实上，只要找对了人选，就能配成小组。并且我们就是这么做的。但如果程序管理与开发在一起合作得不好，别的就甭提了。

现在，在新项目初期，程序经理与产品策划者密切合作以确保每个新产品都有与众不同的“特色”。他们一起撰写产品规划书，这份规划书集中了从开发员、客户支持人员直到包括盖茨等高级经理在内的所有人的智慧。接着，程序经理开始撰写产品规格说明书，它以提纲形式描述了产品的不同特性。这份说明书随着程序经理与开发员讨论产品特性的细节和建立产品原型的过程而不断扩展充实。每位程序经理通常管理几个特性的开发，因此，常在几个特性小组里工作。

在许多公司，通常有软件设计员（有时称为系统分析员或需求工程师）组成的独立部门，他们负责为其他从事产品制作的人员（像程序员、执行员

和代码员）做好产品功能设计。通常设计员负责设计一项新产品并把他们的工作局限在项目的这一初始阶段。而微软人则多次向我们声明，在微软，程序经理并不是这种意义上的设计者，相反，微软程序经理的工作从始至终贯穿一个项目的全过程。因为他们设计的规格随着产品制作的过程在扩展在变化。程序经理在项目起动时，可能花费许多时间来定义许多种产品所共有的特性，解决产品结构方面的问题，模拟用户界面（用户在计算机上看到的屏幕或菜单），并且他们经常提出新的特性构思。但程序经理的所有工作是在开发员的帮助之下进行的，而且开发员也提出许多特性构思（特别是在系统软件部门）。

故而程序经理所做的是记录产品的规格，而非设计，他们与特性小组一道工作并帮助管理该项目。克里斯·彼得斯，Office 软件的副主管，总结说：“程序经理负责写规格说明书，他不负责所有的构思。重要的是产品规格被记录下来了，并有专人负责。”盖茨同意这种看法，他强调开发员真正控制着最终设计：

> 我们可以争论是谁提出了好的特性，因为在有的组是开发员，在有的组是程序经理。但是……从代码如何运作的意义上讲，程序管理从未作出过任何“设计”。开发员仍然……处于强有力的位置，如果他们有一个想法，他们就照此编写代码；他们可能说这个很容易，别的特别难。于是从这个角度上讲，开发员愿意做设计。我们也愿意让他们设计。有一些组，像 Windows 或 NT，或 C 编译程序——大多数系统软件组基本上是开发员设计，程序经理予以记录……当谈到应用软件时，就很难说谁在进行设计。最完善的应用软件是 Excel，因为它总为其他许多事情提供范例。这其中也许 60%的构思来自程序经理，而 40%来自开发员。以乔恩·德·沃恩或一些特性小组的领导为例，这些人真正以他们开发的特性为荣。在任何一个特性中都不好说是由程序经理提供了意见，开发员认可并照此编码，然后把结果给程序经理看。实际情况是高度的相互作用，相互影响。请相信，开发员一想到他们拥有这些特性，就会用难以置信的热情来关心这些特性。

管理设计和开发产品的过程意味着程序经理管理整个项目中的关键活动。但是他们对那些他们赖以完成工作的小组却不享有正式的权威。例如，除了准备产品规划书和规格说明书外，程序经理负责提出折衷方案，比如哪个特性要保留，哪个特性要排除在外等等。他们经常通过一种称为“活动基础计划”（见第 4 章）的微软技术来了解顾客真正需要的产品用途，进而确定某一产品特性是否有效地满足了这种需求。而且，作为设计和决定待性过程的组成部分，程序经理制作产品原型并且管理特性和界面初始版本的测试。这种测试是通过新用户在微软的的可用性实验室中完成的（见第 5 章和第 6 章）。

程序经理在综合开发、测试、产品管理和客户培训等部门意见的基础之上，负责拟定初步的产品开发时间表。他们始终关注着时间表的进度，并确保开发员在可用性实验室里测试了特性；同时为方便内部维修和测试以及方便外部用户，已记录了实验结果。他们与小组中的其他成员商定项目的阶段性目标，并且与测试、开发经理共同确定何时阶段性目标内的特性编码工作

算是完成了，以及何时一项产品可以推出了。程序经理还负责通过正式会议，每天的电子信件和非正式会议（走廊会议和午餐会议）与其他组协调工作。这种协调近年来变得非常重要，它包括界面和代码的标准化（这些界面和代码属于不同的产品，但必须一块儿工作）和相关组之间的进度同步（这些组可能共享特性或代码，或者将一起推出它们的产品）。这些情形发生在 Excel、 Word PowerPoint、 Access 和 Mail 中，也发生在 OLE 和 Visual Basic 等应用软件中——在 Office 的程序系列之内。

作为领导者和帮手（而不是老板），程序经理力图确保在不同功能领域间进行着信息交流并且这种交流贯穿开发周期的始终。开发员、产品经理、测试员、用户支持工程师和高级行政人员——从比尔·盖茨算起——都对有关新产品的决策发表意见：何种特性予以保留，什么时候进行下一步工作或什么时候推出产品。一旦人们在目标上达成一致意见，那么接下来，引用微软文件的话“程序经理的责任就是负责传达”。[6]约翰·法恩，前程序管理主管，现任 Excel 组程序经理，谈了程序经理所起的关键作用：

> 如果微软没有程序经理，情况会不大相同。程序经理的关键作用表现在两方面。首先，程序经理确保产品的特性能吸引顾客，并使顾客愿意为此掏腰包……要创造出除了特性不能吸引消费者之外，在各方面都很出色的软件并不难……也可以根据“设计”的概念，把这种情况产生的原因称“设计管理”。
>
> 其次，他们确保产品在最佳时刻以最优内容冲击、打开
>
> 市场。通常人们深知产品各部分很容易脱节，而且推出时稳定性不够或稳定性太高……产品信息在制作软件、规格记录、测试软件的不同小组成员之间得不到很好的交流。项目越大……产品规模越大，成功率越低。如果这个问题不解决，将导致产品各部分脱节，产品说明书与产品实际功能不符，产品功能在两个不同的特性中不能连贯。而解决这些问题的关键在于程序经理，他以某种方式使信息得以交流。

程序经理与开发员、程序经理与市场的密切联系也会产生问题，他们总是把过多的市场准则强加于开发过程，但有时又难以作出取舍。在微软一个很典型的讽刺例子是，如果开发员对实现同一特性的两种不同方法难以取舍时，程序经理会建议他们两种方法都试试。程序经理还倾向于列出长长的特性清单，当产品经理、开发员和行政人员想要缩小选择范围时，这种清单会成为激烈争论的焦点。正如盖茨强调的，开发员在选择过程中仍有特殊的影响力，因为他们必须评估实现这种特性的技术可行性。并非所有的程序经理都有软件设计专长，然而他们影响着微软产品，特别是应用产品高水平层次（或缺乏层次）的决策。而在有的组里，程序经理变得相当技术化，并比一般用户更关注开发员的观点想法，这种情况经常发生在操作系统和语言产品开发中。正如 Office 软件的高级程序经理麦克·康特所观察到的：“在许多组里……人们过多地钻入与市场营销大不相同的、过分技术化的象牙塔之内，这对于程序经理和产品本身都不是一件好事。”

开发

麦克·梅普尔斯在第 1 章中说过，开发员——正式场合称“软件设计工

程师”——在微软的地位相对高一点。在调查过程中，我们发现情况正是如此。程序经理通常把注意力集中在整个产品的策划，以及什么类型的特性能达到目的方面。与此相反，开发员确认这种产品策划并创造单个产品特性的细节。Office 组的程序经理史蒂文·斯诺夫斯基简要概述了开发员所从事的工作：“他们编写代码，他们完成产品的特性，他们知道编码基数。他们的工作就是通过编写代码来制造产品。”

我们此前已提到微软开发员通常不会从程序经理那里得到产品的详细技术要求，并仅仅照此编码。提供详细的技术要求说明书是一种产品开发类型，在大型软件厂商，如 IBM 中使用相当普遍。查尔斯·西蒙尼在 80 年代初也曾试图将这种方法引进微软[7]，但没有成功。开发员通常和程序经理一道扩展产品说明书，并创造出他们自己的特性细节。然而，在每一个项目中，少数高级开发员和程序经理确实负责产品的结构，并且开发经理（一线经理）为他们的小组提供指导。开发员与测试员共同负责，以确保产品特性能行之有效。微软的条例将开发员广泛和交叉的责任归纳如下：[8]

- 确定新特性的轮廓。
- 设计特性。
- 配置项目资源。
- 制作特性。
- 测试特性。
- 准备出品。

开发员、程序经理和其他人员提供有关产品特性的种种建议。但是，开发员必须很清楚每个特性完成了什么并帮助程序经理决定哪个特性应被包括在新产品之内，哪个特性应予删除。因此，开发员必须评估每个特性并了解人们会怎么使用它。特性设计主要是找出合适的算法（计算机指令）来完成预想的任务。这还涉及一个特性与另一个特性或另外的产品版本之间的兼容性，以及是否可能从其他微软产品中借用代码等问题（这是一件日益重要的事情）。

配置项目资源要求开发员估计他要花多长时间编写每个特性的代码。单个开发员依照过去项目的记录和自己的判断，通过征求组长或开发经理的意见来作计划，另一个步骤是与程序经理一起工作，根据给定的时间、人力资源和市场需求来确定新产品实际的特性系列。随后开发经理与开发员和程序经理协商如何把产品特性分派给开发员或特性小组。

特性制作由编码和检查代码等环节组成。微软文件敬告开发员们当心出现采用了太多的存储器，或者编写的代码依赖于特定的硬件处理器这样的问题。开发员还会从公司收到“无缺陷编程”指令——即依赖于每天的产品构造，阶段性目标的稳定性，每天调试和测试的同步稳定过程（见第 4 章和第 5 章）。

检验代码是开发员和测试员的责任，他们配对进行工作。开发员检验自己开发的特性，经常（通常是每天）启动自动装置来检验代码。在他们的工作成为待检与“正式制品”之前，开发员还把所谓特性的“私人版本”提供给他们的测试“伙伴”。“私人版本”使测试员有机会找出错误，开发员有机会修正错误，开发员可以在把代码传递给本项目的其他成员或其他内部使用者之前修正错误。再次，开发员将他们的代码置于可用性实验室中以便从普通用户中得到更多的信息反馈，这样做有利于进一步的开发工作。

开发员最后的任务是准备推出产品。这要求有规则禁止对代码再做任何重大的改变（比如增加特性）。开发员必须将错误的数量降到几近于零，并且维持相当一段时间。[9]

经常与测试员一道或在可用性试验室里测试代码，并站在用户立场上用β版本测试代码，是开发员工作的重要组成部分。这是因为开发员有偏离程序的长度和速度以及用户关心的特性用途的倾向（微软产品在这些方面并不总是做得很好，特别是第一个版本的产品）。高级副总裁布兰德·斯尔文伯格毕业于布朗大学计算机系并在多伦多大学获硕士学位，他在珀兰工作时曾任 Windows3.0 版本的β测试员。他于 1990 年加入微软并领导 Window/MS-DOS 组。斯尔文伯格提到他本人作为微软经理人员的一项重要任务就是在负责 Windows 95 项目中，提醒开发员多为用户着想：

> 我总是积极致力于建立最终用户与开发员之间的联系，以便让开发员了解用户的需求……Windows 的下一个版本中……主要的阶段性目标是尺寸大小和运行状况。我们将用三个月时间使产品变得更小而速度更快。我认为尺寸大小和运行速度不是你可以仅在最后时刻加上去的东西，并且说："好了，我们已经加上了所有的功能，让我们来调试它。"我认为在任何时刻都不能离你的最终目标太远，正像你不可能让一头大象节食之后变成一只老鼠一样。

测试

源于计算机主机和微机行业的软件公司依靠开发员测试自己的代码，同时还有单独的测试部门，其人员负责进行软件产品的最后测试。但在许多 PC 软件公司，测试依然是一种特别的活动——主要还是靠开发员来测试他们的产品，有时候会得到外部合同工的帮助。微软过去也是这样对待测试的。但从 80 年代后期开始，特别是 1991 年后，戴夫·穆尔和其他经理积极致力于使测试成为公司内部被接受的专职部门，它独立于开发，但又配合开发。

他们尚未完全成功，开发员还是高高在上。但是微软的培训文件指明了为公司上下所接受的，测试独立存在的三条理由：

（1）开发员不可能编写出完美无缺的代码，程序经理不可能制作出完美无缺的说明书；

（2）必须让某些人的工作独立于制作说明书和编写代码，以便对它们的质量能有一个公正的评价；

（3）在开发过程中，当代码群尚未交织在一起时，及早发现并修正错误对于开发员来说更节约成本和更容易——对提高产品的稳定性和顾客的满意度更有益。

为了更有效率，微软的条例建议测试员从六个方面来检验产品：[10]

（1）用户方面　模拟普通客户对某一特性的使用，并判断这个特性是否能让他们理解；

（2）国际方面　确保对不同的语种、国家或地区，产品的格式及语言使用是正确的；

（3）硬件　评估产品与不同的硬件平台和设备是否兼容——比如与 Compaq、IBM、Gateway、Apple 机——或者和特殊的打印机及其他附件是否兼容；

（4）软件　评估产品是否与其他软件兼容。比如，Word 可以被读成或生成 Word Perfect 文件，同时其特性能向 Word Perfect 用户提供特别帮助。系统产品像 Windows 和 NT 必须能与非微软应用软件一道工作；

（5）规格检查　检查产品是否依照原先的设想制造，并且规格是否按照程序管理的预想来完成；

(6)产品稳定性　应在两个层次上检查产品的稳定性。一是定量分析(如错误与错误严重性分析，以及其他我们以后将讨论到的计量手段）；另一方面是定性分析——被定义为判断是否已作好产品上市准备的“主观感受”。

微软在 1984 年开始聘用测试员——正式场合称“软件测试工程师”。当时公司首次建立独立的测试组。在此之前，为数不多的几位测试员向开发经理汇报工作，并且没有他们自己的职能机构或职务级别。直到 1986 年，微软才设立了应用软件测试主管一职并聘人担任此职。但是戴夫·穆尔此后在担任开发主管的同时兼此职务。直到 1993 年，罗格·舍曼出任该职，微软才有了第一位专职测试主管。（正如我们第 1 章所指出的，部门主管实际上并不监督开发、测试或其他功能经理们的工作。他们主要是为本部门筛选出好的实践方法，并向全公司推广，他们不时地审计产品单位的账目。我们将在第 6 章中进一步讨论他们的作用。）

大多数微软的测试员以及几乎所有的开发员都驻在华盛顿位于内蒙得的公司总部。公司还有两个国外测试基地：爱尔兰分部处理使用欧洲语言的微软产品，而日本分部（微软条例称之为“远东”）处理韩国语、日文和中文版本的软件。总体上，微软近乎是一个测试员对一个开发员，即在公司全体 4100 名员工中（包括程序经理和产品策划人员），有 1850 名测试员（见表 1．2）（相反，日本和美国大型软件厂商一般用于测试的人员不会超过项目人员的 10%至 15%，甚至更少。[11] 但 Lotus 与微软人数相仿。[12] 微软或许可以减少测试员数目，但需要程序经理和开发员在开始编写特性代码之前花更多的时间来做结构规划和细节设计，或者使开发员更多地检查自己的设计和代码。然而，这样做会减少微软项目的特性以及构件必须日益增强的灵活性。大量的测试员提供了一种保障：与因错误而使产品被取消或被替代的成本相比，这样做要经济得多。因此，微软分派测试员和开发员并肩工作，并且在项目初始阶段，测试员就组成与开发相平行的小组。同时，测试员向测试组长报告工作。测试组长管理着一组测试产品某个组成部分的测试员，并向产品单位测试经理汇报工作。测试经理向产品商业单位经理而不是向开发经理汇报工作。

准备一项新的测试需要做几项工作。一是查阅以前项目的事后报告以及其他测试组的报告。这些报告提供了需要寻找什么类型的问题以及如何发现这类问题的信息。微软经理还鼓励测试员与产品支持人员及用户进行交流，并浏览媒体的评论以判断工业批评家们的好恶。这些早期工作完成之后，测试员开始设计特殊的工具或代码例行程序来帮助他们进行测试。另一项活动是调研，即研究微软竞争对手的产品和新特性来决定微软应该采取的行动。接着是通过找出可能延误出品的高风险区（包括对其他组的依赖性）来发展一套测试方法。微软的培训材料用 Excel 作为一个例子来说明怎样进行充分的测试准备。它包括用于特性测试的计划和“脚本”——即用于测试过程的命令或功能清单，其他测试员和经理可以参考它来完善自己的工作。测试员还为自己的程序员开发了自动“测试程序”，这些测试程序比人工测试快捷

且更完备，并且它们消除了反复运行同一套命令或功能的单调沉闷。最后，测试员花时间来做微软所谓的“非结构化测试”、“方案测试”、“特别测试”、“大猩猩测试”和“自动式星期五”等种种测试。此时，他们已超越了计划测试书，采取各种不同的方法来使用产品，并尽可能使其暴露问题。

测试员的一个主要责任是测试开发员提供的软件私人版本。他们将开发过程中的错误迅速信息反馈回去。这种信息反馈既反应从用户角度来评估的设计问题，也反应编码出现的疏漏和错误。（按照 Word 测试经理詹妮·希尔顿的说法，大约 80%微软软件开发过程中的错误仅仅是由于“漏掉代码行”造成的）。测试员通过电子邮件向开发员通报在私人版本中找到的错误。只有当私人版本的测试过程结束之后，测试员才向其他人发布一个公共测试版本。测试员还跟踪测试过程中发现的错误，根据他们的特性区域和严重性加以归类。同时，开发员创造了一种测试发布文献（TRD），这种文献不仅说明了哪些特性已经完成，等待测试，而且还指出了应予关注的特殊领域或代码细节问题，以帮助测试员集中注意力。

其他部门

微软还有三个定义更明确并与其他部门交叉的职能部门。产品经理是市场营销专家。虽然从 1993 年后期以来微软将他们中的大部分划归市场营销部，但还有一些人在产品单位中做产品策划工作。用户支持工程师为用户提供技术服务并且分析客户反馈的信息，他们的工作归属产品支持服务部（PSS）。该部向史蒂夫·鲍尔莫报告工作。用户培训人员准备用户手册或帮助文件。他们作为产品小组的成员在产品单位里工作。

生产单位里的产品策划人员与程序经理密切合作，确定产品特性的新构思，并且撰写市场营销说明书（即产品规划书）。除了归纳特性之外，这个说明书涵盖了诸如产品包装、定价、相对于其他产品的市场定位之类的众多问题，并且它要求分析竞争对手的情况和市场趋势（市场营销部的产品经理也做一些类似的分析）。在这些活动之外，产品经理确定新产品领域，提供关于新产品和新特性的建议，并且通过独立的营销部门来准备市场营销和新产品的销售。微软的培训条例，把产品经理描述为“一群工商管理硕士”，“拥有房产的时髦着装者”。条例列举了产品经理的五个关键责任领域：[13]

- 管理“商业活动”。
- 识别和寻求市场机会。
- 在产品开发过程中充分代表客户意愿。
- 负责在功能和出品日期二者之间作出权衡。
- 负责市场营销和产品销售过程。

客户支持工程师主要通过电话，有时也通过电信方式来回答顾客的问题。他们对有关微软客户和竞争对手的数据进行深入分析。他们还通过指明需要发现的问题和帮助产品组确定优先权等方式为产品开发测试提供有益的意见。通过回答客户的问题，PSS 部为客户提供服务和帮助。这种服务被大多数客户视为他们所购买产品的一个重要组成部分。微软每天至少收到来自世界各地的近 2000 个电话，以及成千上万的电讯查询（比如一种自动的快速查询服务）。为了更好地与客户打交道，微软条例阐明了客户支持工程师的五个关键职责：[14]

· 处理客户的电话，及时弄懂客户的问题并找出解决方案。
· 研究问题，并向客户提供所有可能的解决方案。
· 把电话编码，组成支持数据库。
· 对于非客户失误导致的问题提交有关软件错误的报告或文件错误的报告。
· 如系客户失误，就把问题整理成一篇可编入知识库工具的文件。

PSS 制作了一份名叫“Off-line Plus”阅读者甚广的周报，它把问题报告详细化，能帮助确定测试日程、错误修订以及下一个版本的开发日程。此外，该部门的人员还与程序经理、开发员、测试员以及可用性实验室、用户培训人员密切合作，当新产品、新特性尚在开发之中时，他们从用户支持的角度提出意见和建议（关于 PSS 作用的详细讨论见第 6 章。）

除了编写用户手册和其他文件外，用户培训部与市场营销部一道提供有关产品的包装、标签以及国际化的建议。他们遵从其他部门的领导，并使其工作过程规范化。他们甚至借用软件开发的术语来表明他们工作过程的特点。在他们发现和修正材料的错误时，他们借用术语表示这是他们的“产品开发周期”和“制造”，以及包括“测试”文件在内的质量保证过程。用户培训部门还雇佣了各类专家，比如职业作家、图像设计师、技术编辑、文字编辑、资料员、索引员和负责打印的生产小组。[15]

准则二：让各部门专家自行定义其技术专长并负责人员招聘

在 80 年代，微软创立了职能型专业部门（比如程序管理和测试），这些部门本身在公司里并没有约定俗成的传统，也无法与大学课程相对应。因此，微软的经理让这些部门和其他部门（比如软件开发、客户支持和用户培训）的创始成员自己定义他们工作的性质，并负责招收成千上万的新雇员。这样做是合乎逻辑的。从此以后，微软一直以这样的方式开展工作。

人员招收和筛选过程

微软公司员工的平均年龄约 30 岁，大多数员工相当年轻，特别是应用程序组里的开发员。全部雇员中有一半直接来自大学，并且许多微软经理更喜欢这条途径，他们愿意招收年轻人，因为年轻人更容易融入“微软模式”之中。[16]

在公司的早期，微软采用亲自面试人员的招聘方法。那时比尔·盖茨、保罗·艾伦、查尔斯·西蒙尼以及其他高级技术人员对每一位候选人进行面试。微软用同样的办法，即按早期雇佣开发员的办法来招收程序经理、软件开发员、测试工程师、产品经理、客户支持工程师和用户培训人员。微软每年派招聘人员去大约 50 所美国大学。招聘人员既去名牌大学，同时也留心地方院校（特别是为了招收客户支持工程师和测试员）以及国外学校。招聘人员通过人力资源部组织参观，处理日常文书工作，并参与由技术部门和产品部门资深人员进行的面试。有希望的候选人还要再回微软总部进行复试。可见招聘人员并不直接雇佣人员，而是管理招聘的全过程。在 80 年代末 90 年代初，这种方法占很大比重（见表 1）。WindowsNT 组的开发经理戴夫·汤普森曾谈到招聘人员的作用及微软招聘开发员的方法：

在校园里进行一种特别的面试，有人去那儿做预选工作。此后，有经验和没有经验人员的招聘过程基本相同，变化不大。被面试者在一天之内将与 4 至 6 位面试者交谈。最后他们将与作决策的人交谈，合适的人将被聘用。面试过程非常灵活机动。招聘人员是这个过程中的关键因素，他们帮助管理此过程。他们使得这个过程对于开发经理来说既不痛苦又不费力。过去在小公司，我们在雇人时花去许多时间。

而现在我仅在只有我才能答复的问题上花时间——比如评估反馈信息，面试候选人，作出决策等等。

当你想发展得快一些，就要有高效的人员面试过程。好的招聘人员对于某些重要的品格具有不可思议的洞察力……他们知道什么样的人更可能成为一名优秀的微软雇员。任何只靠人事部门来招收人员的公司其结果是注定要失败的。

微软总部的面试工作全部由产品组职能部门的人们承担，开发员承担招收开发员的全部面试，测试员承担招收测试员的全部面试工作，以此类推。面试交谈的目的在于抽象地判定一个人的智力水平，而不仅仅看候选人知道多少编码或测试的知识，或者有没有市场营销的特殊专长（在判定新雇员四种重要的素质，即雄心、智商、专业技术知识和商业判断能力时，盖茨常被作为典型引用，而这四种素质中智商最重要。[17]）微软面试中有名的一般性问题包括：估计密西西比河河水的流量或美国加油站的数目。被面试者的答案通常并不重要，而他们分析问题的方法却很被看重。

能通过筛选的人员相对较少。在大学里招收开发员时，微软通常仅挑选其中的 10%—15%去总部进行复试，而最后仅雇佣复试人员的 10%—15%。总体上说，微软仅雇佣参加面试人员的 2%—3%。瑞克·罗歇德，微软研究部副总裁，在称赞筛选过程时说：“ 这很像是进行口试。面试过程相当严格，我不敢保证筛选出了所有优秀人才，但被筛选出来的肯定都是优秀人才，他们具备一定的才能和天资，以及独立思考问题的能力。 ”

一旦被雇佣，新雇员将面临一系列的挑战和考验。这些考验可能来自每年一度迎新会上盖茨本人的考查，或者甚至可能来自微软某一条洞穴状的走廊（公司的每一幢大楼都有 X 型的双翼和各种各样的棱角，使每个办公室的窗户增多，能很好地欣赏附近的山林和如画的风景。只有聪明的人才能在过道里成功地找到自己通过的通道。）而且只有愿意长时间工作的人才能坚持下来。微软员工颇似日本人即使休假也是休短假。戴夫·穆尔描述了微软典型的一天，他说：“ 在微软情形是这样的，早上醒来，去上班，干活，觉得饿了，下去吃点早餐，接着干，干到觉得饿了，吃点午餐，一直工作，直到累得不行了，然后开车回家睡觉。 ”

为了更深入地考验雇员的决心，微软付给他们相对较低的工资。盖茨通常不付给雇员高薪，并且在一开始时甚至拒绝向秘书和其他人员支付加班费。他在事实上设立了不给加班费的政策。但在 1982 年，他开始发放年度奖金，并给雇员配股。在 90 年代，此类补偿金数目相当可观，因为微软的股票价格持续上涨。给雇员的补偿金现在包括高达 15%的一年两度的奖金、股票认购权以及用工资购买股票时享受的折扣。（一个雇员在微软工作 18 个月后，就可以获得认股权中 25%的股票。此后每 6 个月可以获得其中的 12.5%，

10 年内的任何时间兑现全部认购权。每 2 年还配发新的认购权。雇员还可以用不超过 10%的工资以 85 折优惠价格购买公司股票。[18])

员工离退和调整

诚然，微软人员管理的办法也存在消极的一面。严苛的工作时间表会使员工精疲力竭。人们持续长时间的工作，并且总是在某一特定产品上长时间工作，他们容易变得厌倦，并最终离职而去。理查德·巴斯将其部分归于微软“有意识的只雇佣所需人数一半的政策，这有助于权责分明，减少官僚主义。另一方面，我们倾向选择那些甘愿献身的人们……我们喜欢雇佣这类人，因为他们愿意苦干…… ”戴夫·马里茨，前 MS-DOS/Windows 的测试经理，抱怨经理们缺乏足够的办法来阻止人们过度工作和过早达到高峰期。精疲力竭尤其困扰着开发员，因为他们总是严重低估编码所需时间。精疲力尽也同样影响到测试员，他们与开发员配对工作，经常通宵达旦地测试开发员编写的代码。马里茨自己就是拼命工作的受害者，在 1993 年完成 MS-DOS 6.0 版本的出品之后不久就退休了：

> 比如在测试 DOS 6.0 版本的压缩数据时所有工作都是在晚上进行的。我们告诉开发员五点下班，随后我们进入办公室检查他们的磁盘，整理文件，并运行压缩器。当 Windows 组、WinWord[Word for Windows]组和 Excel 组的开发员早上返回时，他们的磁盘已被压缩，或者已被调试过了，即使是我们不能做的，影像也被转存起来了——全部工作都在晚上进行。这是家常便饭，并且每个周末都如此。如果经常这么做，人们会不愿在这里工作。
>
> 我说的是在微软，你永远得不到休息。你总是处在一种紧张状态。我所做的够不够？有没有把所有事情都办好？这是很痛苦的。我知道我不想在微软累死，一两年后，我离开了那里。我这一辈子再也不想看见软件了。在那个地方，我已经看够了。我愿驾一叶扁舟，烟波垂钓，捕鱼为生。

戴夫·穆尔估计每年高达 10%的新雇员离职而去，并且在新雇员进入公司的头 5 年里，这个比例一直保持着。但是有了 5 年的经验之后，几乎便没有人会永久性离开公司了（有人可能请一两年假去休养或做别的事情，像道格·克隆德就曾去加利福尼亚休假）。穆尔进一步断言，从 80 年代末起，人员调整比例保持稳定，而且开发部门的人员调整比例每年仅为 3%，他还谈到大部分公司范围内的调整来自于产品支持部和制造部，这两个部门的工作都是机械的重复作业，他们雇佣了许多高中生来“每天 24 小时装箱 ”。微软鼓励那些工作达不到标准的人员离开。许多资料表明，在 1993 年和 1994 年，根据经理们对员工生产能力的评估，公司解雇了 5%工作业绩最差的员工（但穆尔坚持说，这种宽泛的人员裁减不适用于开发员。）

从严格招收、筛选和离退中脱颖而出的那些人必定才华横溢，雄心勃勃，并且愿为长远的经济利益长时间超负荷劳动。一言以蔽之，他们非常像比尔·盖茨和其他创立微软公司的高级人员。正如 Office 的程序经理吉姆·康纳尔所说的：

这是我钦佩微软的地方之一。他们在雇佣工作狂方面无与伦比……在面试过程中做了大量工作。他们真是非常善于激励人。我自己的感觉，在某种程度上是在说我自己，我认为我们真的善于寻找不达目的绝不罢休的人。在这里，他们给你许多机会和考验。他们非常高兴让你满负荷地工作。人们也接受这种挑战。一个人也许不够，但工作却还是能完成。你会看到许多例子，人们准时地到达他们的办公室。当我还是个测试经理时我常遇到此类问题，我不得不命令大家回家休息，因为他们已精疲力尽了。他们可以接连三四天一直工作……他们仅仅是有这样的冲动。并且我认为这正是微软的成功之处。

程序经理

程序经理面临最广泛和最难承担的一系列责任。他们必须与其他职能部门，特别是开发和产品管理部门的人员密切合作。没有哪一所大学的学位正好能使人胜任程序经理一职。这样，寻找新雇员总是有点困难。正如比尔·盖茨所说：“程序管理有些不可思议。我们不清楚我们该去哪里招收程序经理，程序经理所需的背景是什么。”

但经验丰富的程序经理知道不招收什么样的雇员。他们不招收能编码或有市场营销背景的专业人员。相反，他们更偏爱那些对公司总体上怎样设计产品，具体地怎样设计软件感兴趣的人员。这与对产品经理的要求有某些相似之处。所以，微软积极鼓励有才华而不当程序经理的人试着干产品管理的工作。但是产品管理专业的人员招收相对容易一些，正如麦克·康特所说：“要找到程序经理的适合人选要难得多……可用于设计今天的商务应用软件的正式背景极少，而在学校里学到的关于应用软件市场营销的东西却多得出奇。”约翰·法恩谈及他要寻求的候选人的品质时说：

用人得当极其重要。适合于当程序经理的那些人的魔力体现在两方面。一方面他们完全热衷于制造软件产品。他们必须是那种对所有权极度关心的人。一个每时每刻为产品所有细节操心的人总会比那些不负责任的人制造出更好的产品。另一方面，他们能专心致志地从始至终关注产品制造的全过程。他们总是善于从所想到的每个方面来考虑存在的问题，并且帮助别人从他们没想到的角度来考虑问题，而不是仅集中在其感兴趣的某些方面，不去考虑其他东西。

微软努力寻找程序经理，这种努力主要集中在大学生上，大学提供大约80%的新人选。依照系统产品市场营销的产品经理克里斯蒂的看法，另一个渠道是从微软其他部门转行当程序经理，比如从产品管理、测试和开发等部门寻找程序经理。许多程序经理具有工科本科学历，一些人有硕士学位。一部分人在大学里学文科，偶有一两个 MBA，而 MBA 在产品经理中更为普遍。专门受过与工作有关的正规训练的程序经理都从事各组的用户界面标准化工作。比如 Office 组，研究组以及消费者小组里的程序经理。他们通常有心理学或工业设计文凭——这是开发新型的，使用更简便的界面所需要的重要的背景。

无论所受正规教育是什么，所有的程序经理候选人都必须表现出理解基本技术问题的能力，尽管他们本身不是熟练的程序员。康特估计在应用领域

“大约半数的程序经理可以编写像样的Visual Basic的应用程序……技术能力通常不是判断一个人是开发员还是程序经理的标准。许多程序经理具备成为一个开发员的所有能力，但他们喜欢解决什么样的问题却是一种偏好。”在系统软件领域，程序经理显然更技术化一些，因为他们帮助设计某些特性，这些特性由开发员编写代码时使用的应用编程界面构成。

因此，程序经理一般具有设计方面强烈的兴趣以及计算机编程的专业知识或熟悉计算机编程。比如，约翰·法恩，曾就读于俄勒冈州瑞德大学珀特兰分校人类学专业。他在大学里修了许多计算机课程，并且在一个为 C 语言制作开发工具的小公司谋得一份程序员的工作。他于 1986 年加入微软并成为编程工具方面的程序经理。后来，他做了 Quick Basic 的程序经理，并最终成为相当成功的 Visual Basic 产品的程序经理。他还做过一个时期的程序管理主管。（这项工作涉及筹办会议、培训新的程序经理，帮助需要帮助的部门，并考虑有关人员下一步要加入哪一个部门。）

麦克·康特集对技术的理解能力与商业兴趣于一身，他的晋职经历与吉布·布鲁门萨相似。他有纽约大学斯坦商学院信息系统与管理的学士学位，刚开始他在一个小型应用软件公司工作。在 1989 年加入微软之前还做过自由咨询顾问。在微软，康特开始在 Excel 市场营销部作公司财会工作。作为这项工作的延伸，他创造了在市场营销部门内规划新产品，为 Excel 新版本作高级设计的工种。两年之后，他的头衔转为程序经理。他还成为 Excel4.0 版本的小组领导人，接着成为 Excel5.0 版本的高级程序经理。1993 年加入新 Office 组。另一位高级程序经理，Office 组的吉姆·康纳尔具有更高的学术背景，他在加州大学圣迭戈分校学习，研究行为和神经生物学，并获心理学哲学博士，他作过伯克利和密歇根大学的博士后，后来又辞去华盛顿大学的研究工作并于 1983 年创立自己的软件咨询和开发公司。康纳尔在 1989 年加盟微软，开始是做合同测试员，接着成为网络系统产品的测试经理。后来他成为互用性设计组的程序经理，从事有关界面标准化似及 Word、Excel、PowerPoint 及其他应用软件的共有特性的定义工作。在 1993 年，该组成为 Office 产品单位的一部分（见第 3 章）。

开发员

对微软人来说，他们清楚地知道一个开发员需具备怎样的技能，并且知道如何鉴别这些技能。开发员寻找那些熟练的 C 语言程序员。C 语言是一种高级可移植性语言，可以与低级的汇编程序语言互换，而汇编语言有助于程序更快运行，使用更少的存储器。开发员还希望候选人具有一般逻辑能力并在压力之下仍能精确工作。在微软，开发员们相信，在面试中表现良好的人，在许多人急需产品时的最后期限里，更可能编写出过硬的代码（如果这时犯错误，必然会在计算机出版界曝光，并使公司国内或国际范围内的声誉扫地。）他们所需要的人员不仅局限于为编程而编程，而且还追求个人挑战并乐于将产品投入市场——既为自己又为公司赢利。微软经理还喜欢直接从大学里雇佣计算机专业的毕业生，并使他们尽快投入到微软项目中去。

因为开发员在公司的作用很重要，他们编写的代码就是微软的产品，故而招收开发员的过程尤其严格。这在招收人员的年度里占用了许多时间。根据麦克·梅普尔斯的分析，在 80 年代末 90 年代初，微软开发员们花费了他们 15%的时间来招收新开发员。管理层现在试图限制单个开发员花在面试上

的时间——规定每周不多于两次，每次为一个小时。大体上，一位普通候选人在学校面试时会遇到一个开发员，在微软公司会有三个开发员对他进行面试，接着午餐时会遇到另外一两个开发员，所以至少有四到五个开发员面试了每一位最终候选人。如果某部门还不能作出决定，它可以邀请候选人回来再进行面试，如果不止一个部门对某个候选人感兴趣，该候选人可以返回并在其他部门再进行一整天的面试。

微软还要求每一位面试者准备一份对候选人的书面评估报告。由于许多人（包括高级经理们）会阅读这些报告，所以面试者常感到来自同事间很强的压力。他们必须对每一个候选人做一次彻底的面试，并写出一份详细优质的评估报告。戴夫·汤普森在解释 NT 组招收开发员的办法时论及此事：

> 开发员是唯一胜任面试新开发员的人……我们组织面试。面试至少有一个主要部分是编码问题……面试过程组织得很好。每个人为别人提供信息反馈，所以有大量有效的信息交流。你的信息反馈会被所有人看到，这一事实会提高反馈信息的质量，而这无疑又会提高面试的质量。如果面试结束后你写了一份什么也不是的信息反馈报告，里面没有任何数据，你会很尴尬。尴尬会催人进步，不是吗？于是你下次肯定会认真做好面试工作。

测试员

微软在招聘软件测试工程师上遇到了困难。很清楚，应用软件测试员必须懂得普通人怎么使用软件产品，而系统产品测试员则需要懂得开发员如何使用特性，以及应用程序和打印机等附件如何与操作系统配套使用。软件的这两类测试员必须热衷于“破坏事物”。他们必须想办法让产品失败，使之不能正常工作——这恰恰是与程序经理和开发员相对立的工作。这并不需要有共同的技能和偏好。

戴夫·穆尔简要地说道：“寻找愿意成为职业测试员的人很费力。”这种困境起因于微软不愿意招聘其实想当开发员的人来当测试员，因为有些人认为他们可以通过当测试员来达到目的。又由于编程方面的知识对于有些测试，如对操作系统和语言的测试是有用的，甚至相当重要，情况就变得更复杂。

Excel 的测试经理马克·奥尔森具有微软经理们在测试员中寻找的那种类型的背景和智力。奥尔森，1965 年生于华盛顿塔科马，可算是与计算机一起长大。在大学读物理学时，他修了几门编程课和其他技术课程。毕业后，他在 IBM 呆了几年，从事半导体材料开发工作。由于他没有研究生学历，他被限制在初级研究职位上，他本人对此感到厌倦，于是回到华盛顿寻找新工作。奥尔森谈到 1989 年加入微软的经历时说：“我在我的母校进行软件测试职位的面试。他们面试各类事情。我是物理学专业的理科学士，那时这样的学位并非求职时理想的敲门砖……于是我抓住这个机会，研究微软公司的所有情况，以及有关测试员的所有情况……我知道微软需要有人来分解、破坏软件，理解它们怎样运作并确保它们能正常运作。”

奥尔森迅速成为公司几位顶尖的应用测试员之一。但他还是觉得准确鉴别称职的测试员所必备的素质很难。在面试过程中，他寻找与开发员想法不同的人——这些人从不想当然地认为一个特性会正常运作。他们研究可能导

致特性失败的隐性内容，比如“多特性测试”（测试一个特性与另一个特性的关联）。奥尔森说：

> 在面试过程，很难确定具备什么样的素质会使人成为一位优秀的测试员。要确定这种素质比仅凭借编码知识就能推断工作能力的情况要难得多。要做测试员，不仅需要有能力在没有任何记录的情况下，学习新软件并抓住主旨，而且需要具备检查它的能力，于是我们寻找永恒的怀疑论者，他们从不认为任何事情是想当然的。他们不会因为开发员说软件能正常工作就随声附和，他们调试它，把它推向极限……对于特性应怎样工作，测试员必须在另一方面比开发员更精明……必须能够懂得电子表格的用途以及客户怎么使用它。然后由一两个人通过在 6 个月内模拟 100 万人对电子表格的使用来对产品进行测试。这是真正的挑战——要抓住所有的情形。

像奥尔森那样有一个理科本科学历背景的人，在测试员中相当普遍。同时，他们也普遍缺乏为其他公司测试软件的经验。微软经理们喜欢直接从大学里招收测试员，或至少是招收在测试方面没有任何经验的人员。缺乏经验但聪明的人能够很快掌握微软那套测试方法，同时对微软的低工资和高要求也会感到满意。

不排除有些测试员在加入微软之前已有些经验。比如 Word 组的测试经理詹妮·希尔顿，她曾在塞乔州立大学主攻历史和自然科学，并在威斯康辛大学获科学史硕士学位。她曾在一家计算机系统公司干销售工作，接着在微软的竞争对手——软件出版有限公司（出品哈佛图像）干了 5 年的测试和质量保障工作。希尔顿在 1990 年加入微软，并且职位升迁得很快。

有时，测试员还来自微软公司内部的其他职能部门。奥尔森估计在 1993 年 Excel 组里共有 50 位测试员，其中 15%来自客户支持部门。他们对于一般客户面临的问题有深刻的洞察力，莫希·邓尼曾做过 Windows NT 3.0 版本的测试经理和程序主管，早先他曾是一名程序经理。但是，经理们不鼓励开发员们转行干测试工作——特别是在应用软件部门。因为程序员们经常假定他们的代码能很好地进行工作，而好的测试员可能不会接受那些代码。奥尔森强调指出：“我们愿意要那些想当测试员的人，我们不要那些想当开发员的人。开发员用非常结构化的方式编写代码，他们作出假定，并根据这样的假定来编码。我们不想要任何预先作出假定并满足于此的人。我们需要对这些假定提出质疑的人，并且他们能够提供更多的东西。”

与应用软件部门不同，系统软件部门的许多测试员与开发员具有相同的背景。而且，他们偶尔也会由测试转向开发，也有人从开发转向测试（但在真正有才华的开发员中这种情况很少）。这种人员的混合是有意义的，因为用户并不直接地与操作系统打交道。操作系统特性的关键用户是为这些系统编写应用程序的开发员。对于计算机语言和软件开发工具，情况也是一样，比如戴夫·马里茨，有生态学（从南非大学获得）和计算机（从以色列特克里昂大学获得）的学士学位，在 1987 年加盟微软之前，他还做过以色列武装力量的坦克司令员和空降控制系统的汇编程序员。马里茨在微软先做图像工程小组的领导工作，后来成为 MS-DOS/Windows 的测试经理。

莫希·邓尼曾有程序开发经验，但不是在微软。他曾在以色列特克里昂

大学学习计算机工程并从事过军事应用软件的开发工作。他在 1988 年加入微软并成为 OS/2 软件的程序经理，后来又成为 OS/2 的测试经理。在做 Windows NT 的测试经理之后，邓尼雇佣过 50 个测试员，其后，他又雇佣了更多的测试员。他通常试图招收开发员来做测试员。邓尼承认有过很艰难的时刻，他说：“我们不得不推销我们自己，因为在许多人的心目中，软件测试……与干练的软件工程师认为的富有挑战性的工作不符……（但是）他们要通过同样的面试过程。”这种面试过程的筛选标准是任何被招进 OS/2 项目的开发员也必须通过的。戴夫·马里茨在雇人当测试员并说服他们乐于从事这项职业时感到非常费劲：

> 大家变得厌倦测试……干测试员这行我们晋职机会不够，并且我认为我们雇人的办法也很成问题。戴夫·穆尔和其他人可能会有不同的看法，但是我们从大学里雇佣所谓的 STES（即软件测试工程师），而这些年轻人他们全力以赴学习计算机并决心取得 CS（计算机科学）学位，他们向往编写代码。但我们却把他们领到这里，让他们从事测试。这些年轻人会觉得疲乏之极……但是这是唯一能够达到招聘标准的候选人。我们的这些标准是你首先要进行面试，必须问被面试者编码方面的问题。如果你偶尔遇到一个非常棒的人——你也许能够让他在实验室里工作三个月，测试应用软件。学习连通性和 Novell 工作原理——但他却不懂多少编码知识，他不会把一个整数插入到一串 C 语言整数链表中去，他就永远不会被这里雇佣，因为他无法通过面试。

为弥补测试员人手不足，同时应付产品测试的高峰期（比如当一个产品β版本刚完成时），微软经常雇佣兼职或合同测试员。这些人通常没有编程方面太多的知识，他们一般工作时间长，而工资较低。其中有些人，像吉姆·康纳尔，知道许多编程知识，于是成为了正式雇员和经理。

其他部门

微软不要求产品经理有编码的专业知识，尽管他们中多数人有工程学背景，并且一些人还学习过计算机。他们一般能在各种设定值下熟练地使用计算机软件，他们中越来越多的人有商务管理方面的高级学位（如 MBA），以及市场营销的正规知识。比如麦克·康特，在加入微软并成为 Excel 组的产品经理之前有管理与信息系统专业的学位，并有销售和信息咨询方面的经验。理查德·巴斯有美国海军学院电子工程专业的本科学位，在海军部研究与开发程序管理部门工作了 10 年，接着又在 MIT 获管理学硕士学位，他于 1990 年加入 NT 组，成为产品经理。桑贾伊·帕塔萨拉蒂作为高档消费者技术部商业发展组的产品经理，具有计算机本科学位和 MIT 管理学硕士学位。而克里斯蒂·威奇斯，曾在布格桑大学攻读政治学和心理学，他从人事部转到 Excel 组从事产品管理，后来又转而从事系统产品的市场营销工作。

客户支持工程师同样也有广泛多样的背景。在招收该专业人员时，PSS 经理寻找具有如下品格的人选，如候选人看上去是否感情投入，是否乐于解决别人的疑难问题并教给他们正确的使用方法，是否具有良好的分析解决问题的能力和交流技巧，而不是仅有技术文凭。[19] 前 PSS 市场营销主管特瑞希·梅补充说微软还进行一些基本的心理评估和测试，特别是对于那些即将

走上管理岗位的新雇员。一些客户支持工程师有编程的背景，但他们主要的工作是帮助客户，其范围从 MS-DOS 和 Windows 的初学者直到高科技产品，如语言编译程序或网络交流系统的熟练用户。比如马克·辛登沃格有数字电子专业学位。在 1984 年加入微软之前，从事计算机硬件修理工作。他领导过一个支持 Mac 机的 Excel 软件小组，后来转到 Windows 支持组。现在他是 PSS 部 Excel 组的产品开发顾问。

准则三：通过边干边学和言传身教培训新雇员

怎样教育和引导加入微软的新雇员？这个问题随着公司产品的多样性和复杂程序的增加变得越来越棘手。特别是在程序管理之类难以定义的功能领域，这一问题尤为突出。早期，人们通过交谈或边看代码边使用产品来交流产品设计知识。新雇员注意观察有经验人员的工作，每个人通过“试错法”来学习。在许多领域，微软现在还是这样运作的。微软试图雇佣能自学业务的人员，而不愿在培训项目、正规条例和流程，或者详细的产品记录上大量投资。它依靠熟练员工来教育新雇员，这些熟练员工有组长，某些领域的专家以及正式指定的指导教师，他们除了本职工作外还要担负起教导新雇员的责任。

这种办法有它的优点和缺点。优点在于大家觉得有权学习并自己决定学什么和不学什么，他们在公司里的作用灵活机动，只要有能力应付，这种作用可以变得相当广泛。但是这也有消极的一面，新雇员的提问常常会暂停熟悉员工的工作。而且由于微软的快速发展，有经验的中层经理和组长很缺乏，新雇员常常不得不通过试错来学习或者跟着经验不超过两年的指导教师学习。这时的错误成本可能很高——会在世界市场上有损公司的收益和声誉。

程序经理

微软发现雇佣，培训程序经理都不容易。麦克·康特谈到他的经验时认为，程序经理的工作是“部分在学习，部分在创造。”在公司早期，这一概括尤为正确。甚至在吉布·布鲁门萨成为第一位程序经理的十年之后，微软还没有为这个职位制定详细的工作指南。康特承认这一点，他说：“总体上说是微软，具体地说是微软的程序管理一职，都没有正式的取向。这里没有正式的培训项目，没有任何事情必须要由手册规定，比如所有程序经理必须这样编写说明书，或者必须这样制作产品模型，甚至这样来安排时间表等等。”

程序经理现在可以受到一些正规的培训，包括一个供选修的为时三周的培训项目。另外，微软不定期举行“蓝碟”午餐会，届时会有程序经理介绍他们自己的经验。比如，约翰·法恩曾经介绍过怎样制作一份好说明书，他的这次发言被制成录像带在公司内部传播。但是，总体上微软经理们深信，边干边学，言传身教以及开始时选择合适人选是培训程序经理的良好途径，这些比试图在教室里教育他们要强得多。法恩在他还是程序管理主管时谈到并强调了这一点，他说：“假如你雇到了合适人选，那么有效的培训带来的好处 90%来自于言传身教，即新程序经理与更有经验的成功的程序经理一道工作，并向他学习。这些年来在培训方面通过其他机制的努力达到剩下的 10%。与开发员和测试员相似，法恩强调程序经理也通过从相对简单到更复杂

的项目来学习经验：

> 刚开始时，你的任务可能是一句话就能说清楚的单特性领域，对初学者来说或许是一个很难透彻理解的领域：负责一个单独的特性。这意味着你应确保产品推出时特性领域各个方面也完成了——它不会导致产品脱节，它具有一定的稳定性和功能……这种工作要干一段时间——6 个月，1 年，无论多长时间，都会有人对你进行密切的指导……随后，当这种工作你已做得相当熟练之后，你会在更大的特性组中从事类似的工作，但指导会少得多。一段时期之后，你会拥有一个小项目或一个大项目的一部分。如果你从事的是一个小项目，你会是它的唯一的程序经理。这是对程序经理的第五档要求。这是一个我称之为“ Ninja 程序管理 ”的职位，也就是说，无论一个项目对于你有多么难，从刚开始至任务完成，你要想方设法……当然有许多发展方向。像在其他公司一样，你可以从事管理工作。这样，你要想到越来越多的跨产品的战略问题。当你达到这个深度时，就像在其他公司一样，你的功能领域已变得没有意义了，你可以做公司需要你做的事情。

开发员

新开发员必备的经验水平以及特殊的培训定位要求，随不同的组，不同的产品和特性而变化。在诸如 Excel 和 Word，甚至 Windows 和 MS-DOS 组里，大多数开发员只有两三年的经验。但在 Windows NT 组里，核心设计者在大型组织，如 DEC 的操作开发上有 12 年甚至更多的经验，大多数其他 NT 开发员有 4 年的经验。这其中一些人来自公司外部，一些人来自公司内部（如从 OS/2 或语言软件转来的）。在所有的组里，让新开发员加入进来变得很难，因为软件主要产品构件的数目非常巨大，而潜在的互相作用也越来越复杂。Word 的开发经理爱德·弗莱斯曾在新墨西哥州技矿学院主修计算机专业。1992 年转到 Word 前在 Excel 组里工作，他解释他对待新加入者的办法时说：

> 新雇员是一种挑战……非常难。他们在大型复杂程序开发的中期加入进来。所幸的是，他们不必知道有关程序的所有事情，还能做一些工作。但是，他们会得到一些痛苦的教训。他们学习……他们不得不为尚不明了的事情担忧……比如，他们认为他们将在某处装入一些特性，但他们必须理解其中的宏语言，并且那种宏语言……影响到他们怎样做他们的特性。如果他们做得不对，他们会破坏这种宏语言，或产品中任何其他模式……他们的特性可能在一个模式里能工作，但在另一个模式里却不能，或者它可能工作，但不能在打印前预览，或者根本不能打印。有许多互相影响的事情。

或许因为开发员编写的代码不留余地，代码不是工作得很好就是工作得很糟——微软对这个领域人员有更长的正规定向和培训的历史。微软为新开发员提供了几个为时两天的实习班，培训他们处理开发过程、产品、工具和其他专题。微软人仍在继续争论这些课程的用处，经理们也没有使这些课程变成强制性的。戴夫·穆尔特别强调他喜欢让新雇员马上投入工作，甚至不必“ 看上半个小时录像带以了解微软如何起家。他们或许已经知道了这些事

情，这是对士气的抹煞。从校园来到这里的最初时刻是令人兴奋的，这样的兴奋将贯穿他们今后的工作。项目急需这些人员加入，立刻加入，在第一天就加入。”

在头几天里，新雇员与经理们及来自其他专业部门的高级人员见面。他们会听到有关开发周期的一个方向性的简介，详细的开发过程他们将在以后的项目工作中学到。开发经理会立即派给新雇员一个单独的任务或者让他与特性小组一起工作。他们还可能把新雇员介绍给愿意当指导教师的高级开发员。在许多组里当指导教师是一个正式任务，一个人当 9 个月。“指导教师是真正做培训工作的人”，穆尔说。新雇员开始从事相对容易的特性编码工作，这种工作需要一周左右时间并且与其他特性关联甚少。高级人员（特性组长、领域专家、指导教师）随即非常仔细地检查新雇员编写的代码。

在有些组，新开发员能看到描述产品内部结构的有关文献。比如 Word、Excel、Windows NT 软件都有入门书，分别为“Word Internals、“Excel Internals”和“Newcomer Doc”。但这些入门书很快就变得过时了，并且不像新雇员所希望的那样详尽。这些入门书也不可能通过书面语言解释清楚在大型系统中计算机代码的复杂性。所以新雇员还必须依靠指导教师，并且通过亲自阅读代码，做实际编码和调试工作来学习知识。乔恩·德·沃恩以前在 Excel 组工作，现在是 Office 的开发经理。他曾在俄勒冈州立大学学习数学和计算机，1984 年加入微软。他谈到了“Excel Internals”这类文件和微软的“口授传统”：

> Excel Internals……解释了 Excel 一些基本事情，像单元表格式，存储器的配置，层次（操作系统的特殊界面，使在 Windows 和 Macintosh 平台上使用相同的 Excel 的核心成为可能。见第 5 章）的一些知识，介绍非常零散。我们必然依靠这点东西来学习知识。我想说我们有很强的口授传统，这就是指导教师来教导新雇员或新雇员通过读代码自学……在一个项目完成过程中，这些文件的真实性在减少，于是我们必须修正它。在我们做项目时我们不修正它，但我们会在项目之间给予足够的关注。

德·沃恩认为这些文件尽管不完美，但仍有用。因为 Excel 现在由上百万行难度很大的代码组成，这使新雇员学习起来相当吃力。通过自学来掌握有关系统的知识变得特别重要（和困难）。因为大约半数 Excel 开发员在微软工作的年限不超过二三年，而且整个项目仅有一位技术领导。不断更新这些文献是另一项费时的工作，是德·沃恩和其他微软员工所不喜欢并倾向于放弃的工作。结果，他们使文献尽量短小精悍，并且仅在产品的两个版本之间才更新它：

> 在组里，我现在正把重点放在这上头。Excel 很难学。你初次面对几百万行代码——你从何开始？……我们总觉得要使它变得容易的途径是记录更多的东西——但是…现在，你不仅得保留你的源代码，而且保留你已写下的所有东西。我已说过，在过去，指导教师的作用发挥得很好，但 Excel5.0 版本的开发是个例外。也许我们真需要写下更多的东西。但是，草率的一面是所有新雇员已投入工作，尽管他们的认识还

相当模糊。

Excel 组在微软公司发挥了特别的作用，它不仅是软件开发过程和特性标准化的技术领导，并在事实上成为培训有才华的开发员和经理的基地。正如程序管理部门的约翰·法恩一样，德·沃恩认为新雇员通过实践会学得更好，他说：“我们所做的仅仅是根据大家的级别层次，把任务分派给他们，或他们接受任务，然后靠他们的聪明劲儿把任务完成……许多人为此感到不安，他们说：‘我们不能让他们做这些工作，他们没有任何经验，’但是这里，我想我们应该说：‘是的，他们没有经验，但他们可以边干边学。’”德·沃恩回忆起他在微软干的第一份工作，那是让他为 Mac Excel1.0 版本写拷贝保护软件。这项工作迫使他去学习，在高深的技术层面上，Macintosh 机磁盘驱动器在软件和硬件方面怎样工作。德·沃恩还认识到让人们自己去学习的结果是不时会出现错误，而这是一个好经理应该预见到的：

> 首先，人们可以犯错误。犯错误之后，然后观察怎样修正它，这样可以学到很多东西。这也是我学习知识的关键途径之一……当某人从事一项新工作时，其他人或许是这个领域的专家，当他们的代码写成之后，会有最好的专家来查阅他们的代码并告诉他们，这样编是对的，或这么做会更好……这样每个人都边干边学……指导教师会解答疑难问题……但当他们觉得自己的代码编写完之后，会有人对代码进行检查并找出错误。

当然，并非所有新来者都有乐观的初次经历。在小产品组里的新开发员，受到来自高层管理人员的注意较少，很容易在忙乱的工作中被忽略掉。大卫·惠特尼，MIT1990 年计算机系的毕业生，从事微软 Mail 的工作，他联系自己的经验评论说：“他们分派给我零散的任务。在搞一个新项目时，他们把新雇员使来唤去，一会儿让干这个，一会儿又让干那个。我干的都是一些低层次应用结构方面的杂事，处理存储器管理程序并进行调试，就干这样一些事情。”

微软过去为开发员提供了更多的定向培训。在 80 年代，微软最富才华的程序员之一道格·克隆德（他最近离开了公司）为新开发员办了一个叫“训练中心”的培训项目。这个培训中心的培训计划根据受训开发员的能力持续几周或几个月。这个培训中心被赞成者称为“克隆德大学”，被反对者称为“克隆德花园”。新来者还通过其他事情学习。比如，查尔斯·西蒙尼发明了一种独特的编码规则，在微软使用时被称为“匈牙利标记法”（见第 5 章的讨论）。在经理们允许他们加入项目开发时，开发员还必须通过其他的考验。微软人经常争论花很长时间对开发员进行定向培训的用处。事实上，PC 机软件公司一般不搞广泛的新雇员培训，它们大多数都雇佣熟练的程序员。（这与主机和微机的软件生产者相反，他们通常有二三个月到一年的培训项目）。几年前，“克隆德大学”结束了，因为克隆德本人对此感到厌倦。此后，微软转而直接把开发员安排进项目，只提供指导教师和几门供选修的短期课程，以提高他们的技能。

指导教师制并非尽善尽美。它依靠熟练员工来指导无经验人员，并主要是通过示范和口头说明。但当微软熟练员工变得相对短缺，而同时产品越来

越多样化和复杂化时，情况就不太妙了。这正是 Excel5.0 版本所遇到的情形。由于种种原因，该产品直到 1994 年初才面世，比计划时间晚了几个月。（以前，Excel 项目最多只延误几天或一两周时间）。德·沃恩谈到这个问题并分析了原因——微软没有及时安排有经验的人做指导教师。他说：“ 上个夏季，我们没有那么多有足够经验的指导教师来做工作。而且，我们野心勃勃，急于求成，为了该项目的某部分投入了大量人力。但当别的工作开始时，会产生很大的冲突。也许下次我们应该提出建议，每周给出一些时间来安排指导教师。 ”当微软减少新雇人员并且规定任何一组当中，新雇员不得超过开发员的 50%时，这类问题减少了。微软也增加了更多的正规培训。正如盖茨所说，微软人还是喜欢主要靠言传身教来培训新雇员，并且他们认为现行方法不会导致严重问题。

> 指导教师制？实际上并没有什么不好……我们过去有非常正规的培训，但人们对此并不特别热心。不是有些人不学，光浪费时间，就是有一些超级巨星对此根本不屑一顾。存在固定的事情，固定的计算技巧，但是给这些人一本“ 编程入门书 ”（怎样编写计算机代码的初级课程）足矣，只要确保他通读了全书并理解了书中的东西，给他看代码，并问他一些问题。现在我们非常正规，我们提供了一系列课程，大家可以以适当比率来选修。但是，进入一个项目，从更高级人员处学习，这对我们来说似乎更为成功。

测试

某些定性策略、技术工具（如自动测试程序和错误数据库）可以使测试变得更有效。在加入产品组之前，微软为所有的测试员提供基本的培训。测试员可以在实习室和讨论会上学到上述那些东西。而其他的知识，如有关特殊产品的知识，或者怎样最有效地使用特殊工具和技术，则需要导师的言传身教和在职经验。

尽管在不同产品领域内存在着许多共性，但在接受最初的基本训练之后，各个产品单位都有各自的方法向新测试员传授有关产品和测试支持工具的知识。较大的组里还有文件概括测试概念，样本测试计划和清单，以帮助测试员确定什么时候算是完成了测试任务（见第 5 章）。Excel 组甚至有“ 电子帮助文件 ”来收集对 Excel 测试员有用的信息，但是目前尚没有建立结构化系统方式来进行测试。马克·奥尔森在谈及检查使用软件的所有不同方法的困难时，强调了这一点，他说：“ 使你的测试覆盖产品的方方面面，并保证不会把事儿弄得一团糟的方法是建立测试结构。你有系统的方法来浏览和修正一个特性，你有需要测试的事项的清单，并且延长这张清单，你必须确保没有遗漏任何一个特性中细微的环节，你必须使你的开发组尽可能明白这些。当他们编写代码时，必须将这些问题考虑在内。而不是当你找到一个错误时才轮到你来修改，以使它正常工作。 ”

像其他的专业一样，测试员作为特性测试组的成员来学习专业知识。小组由三至八九个熟练测试员和新测试员混合组成。测试员还和单个的开发员配对进行工作。测试员仍向组长（测试小组的“ 主管 ”）和部门测试经理汇报工作。测试员也有指导教师，但微软并不像依赖开发员指导教师那样依赖测试导师来进行培训。

不知如何编码的测试员进入微软后通常要学习这个技能。他们学习一门 Basic 语言或更高级的语言课程。编程对所有组里的测试员都变得日益重要。凡是带有宏语言的产品，包括 Excel 和 Word，需要进行编写代码的测试。事实上，越来越多的应用产品包括了宏语言，以使用户能按规格改装产品，（一个宏指令是指用户可以把一系列命令变成一个简单的命令或一次击键，在一步中完成一系列任务。）在系统领域，许多测试是通过编码运用不同的应用编程界面（API_S）构成的，所以在这里编码很重要。但甚至在应用程序组里，测试员越来越多地使用自动测试生成器和宏语言来编写测试案例。这些需要基本的 Basic 知识或者 Basic 语言的图形版本 Visual Basic 的知识。

其他专业部门

产品管理和用户培训专业的人员培训是非正式的，新雇员通过边干边学和从指导教师那里学习知识。新雇员很大程度上还依靠过去项目的范例来学习，这些范例包括以前完成的成功的市场营销报告和计划，或较好的记录文件。大学市场营销课程中的产品管理以及相关科目（比如英文写作和编辑，图像设计和图书出版）对于用户培训人员也是有用的。

微软为客户支持工程师提供了正式的培训。这种培训与公司在过去几年里努力改善公司形象和提高为顾客服务的能力是密不可分的。顾客不仅仅只是购买微软的产品，他们还会享受到许多售后服务。客户支持工程师不必像开发员那样有必备的职业教育，但他们必须有关于微软产品如何工作的广博知识，并且实际上必须在某种产品上具有专业知识。

新的客户支持工程师在分专业之前，接受三至四周培训。这种培训从基本的系统产品 MS-DOS 和 Windows 开始。 他们还接受交际技巧，包括如何与顾客打交道等方面的一般性训练。为便于边干边学，新 PSS 雇员通常从 Windows 支持分部开始，接着进入应用程序支持分部。作为定向培训的一部分，他们还接电话，与导师一道工作（每位技术员有一位导师）。在他们被分配处理客户的电话之前负责答复客户来信。[20] 工作确定之后，每个雇员每年还要接受大约 20 小时的再培训。[21]

培训电话服务员的高级 PSS 人员，与开发组织密切合作以理解他们开发的新特性。同样，数据库工具之一的知识库（支持人员用它来帮助回答问题）的管理员也与开发员和用户培训组织密切合作以理解他们开发的新特性。马克·辛登沃格，负责 PSS 产品开发分部与 Excel 软件单位的联络，他解释培训和对产品开发的投入在本部门所起的作用时说：

> 世界范围的培训计划将从事对新雇员的培训工作。在我们组，特别是 Excel 组，我们的培训是整个开发交流周期的组成部分。他们是负责文件浏览和说明书浏览人员的组成部分，他们从产品设想开始紧紧跟踪它的开发，所以他们后来实际负责进行有关该产品的人员培训。知识库的编写也是同样，他们为工程师提供许多关键信息。所以，他们要经历整个产品开发周期，他们了解产品工作的目的，工作方法以及决策的制定。接着他们就进行培训。

准则四：设立晋职途径和“升迁级别”以留住并奖励人才

微软在产品组里创立了技术专业并雇人充实这些专业部门之后，一个新的问题产生了，这是许多公司遇到的典型问题：怎样把人才留在技术岗位上和产品组里，以便利用他们积累的专业知识和公司已付出投资的工具、技术和培训。正如前面提到的，微软倾向于把有技能的开发员推到管理者的岗位（比如产品单位的主管），而且中层经理人员还继续编码或测试程序。但是当开发员，测试员和程序经理只想呆在他们的专业部门里，并且只想升到本专业的最高位置而又不必担负管理责任时，职业管理的问题就产生了。

微软做了一些在鄙视官僚主义的公司里很少见而在高科技公司里相对普遍的事情，他们在技术部门（比如在测试和开发领域）和一般管理部门（比如在产品组和公司一级）建立了正规的升迁途径。

建立技术升迁途径的办法对于留住熟练技术人员，承认他们以及给予他们相当于一般管理者可以得到的报酬是很重要的。梅普尔斯说：“我们非常清楚地意识到双重职业的概念。当一个家伙不想做经理时，他也能像一个愿意做经理当头头的人那样发展并升迁。”

在职能专业部门里典型的晋职途径是从新雇员变成指导教师、组长，再成为整个产品单位里某个功能领域的经理（比如 Excel 的程序经理、开发经理、或测试经理）。在这些经理之上就是跨产品单位的高级职位。这包括职能领域的主管或者在 Office 产品单位中的某些职位,他们负责 Excel 和 Word 产品组并且构造用于 Office 应用软件的共同特性。

微软既想让人们在部门内部升迁以产生激励作用，还想在不同的职能部门之间建立起某种可比性。它通过在每个专业里设立“技术级别”来达到这个目的。这种级别用数字表示（按照不同职能部门，起始点是大学毕业生的 9 或 10 级，一直到 13、14、15 级）。这些级别既反映了人们在公司的表现和基本技能，也反映了经验阅历。升迁要经过高级管理层的审批，并与报酬直接挂钩。戴夫·穆尔回忆起 1983—1984 年微软建立的晋级制度。这种制度能帮助经理们招收开发员并“建立与之相匹配的工资方案。”当微软建立起其他专业部门时，每一个领域都建立了相似的晋级制度。

级别对微软雇员最直接的影响是他们的报酬。通常，盖茨的政策是低工资，包括行政人员在内，但以奖金和股权形式给予较高的激励性收入补偿。行政官员和高级雇员的基本工资比公司的平均工资高不了多少（像许多日本公司一样）。盖茨 1994 年只拿了 27.5 万美元的工资和 18 万美元奖金。但他还拥有公司 25%的股票。其他高级行政人员拿的也差不多或更少。像史蒂夫·鲍尔莫（他拥有公司 5%的股票，是公司第三大亿万富翁，仅排在盖茨和保罗·艾伦之后——保罗·艾伦作为董事会成员拥有公司 10%的股票），1994 年他工资收入为 24 万美元，奖金为 19 万美元，麦克·梅普尔斯工资额是 24 万美元，奖金为 25 万美元。[22] 刚从大学毕业的新雇员（10 级）工资为 3.5 万美元左右，拥有硕士学位的新雇员工资约为 4.5 万美元。对于资深或非常出众的开发员或研究员，盖茨将给予两倍于这个数目或更多的工资，这还不包括奖金。程序经理和产品经理与开发员的工资几乎一样多。测试员的工资要少一些；刚开始时为 3 万美元左右，但对于高级人员，其工资亦可达 8 万美元左右。由于拥有股票，微软的 17800 雇员中大约有 3000 人是百万富翁——大概算是相似规模公司中百万富翁比例最高的。

即便是技术级别或管理职务上升得很快，有才华的人还是易对于特定的工作感到厌倦。同时，产品组和专业部门也得益于有不同背景和视野的人员

的加入。相应地，微软经理鼓励在产品组之间保持某些人员的流动性，并且他们不阻止合格人员转到不同的专业部门。但是，人们只有在某一特定领域积累了几年经验之后才能换工作。微软的大型产品，像 Office、Word、Excel、Windows 和 NT，需要花几年时间来积累经验，频繁地变换工作是不足取的。对有兴趣更换工作的人，在产品组里为其安排一个顺序。虽然这种顺序随时间而变化，人们通常还是认为某一个组与一些产品比另外的产品更值得去做。在应用软件中，这种看法似乎是根据该组对微软的销售和贡献大小来决定的。而在系统和语言部门，这种标准却围绕着技术挑战性和声誉的高低。比如，Windows NT 和 OLE 是比 MS-DOS 更吸引人去做的产品，因为，它们一直是巨大的收益发生器。Visual C++和 Visual Basic 比其他不很普遍的语言更具挑战性。公司的部分其他产品——比如多媒体、交互式电视机或者联机 Video 和网络服务之所以吸引人是因为它们属于高科技产品，拥有快速发展的新兴市场。在应用软件部门，Excel 一直是最著名的，但要求也最高。又由于 Office 现在是主要的应用型产品,它对于应用程序开发员来说是新的乐土。

程序经理

程序经理具有与其他专业人员一样的晋职途径。他们的职务范围从指导教师、组长到部门的程序经理，还有一个负责人员调动的职位：程序管理主管。（这是一个临时的和业余的职位，没有多少人愿意干。程序管理仍是一种复杂的难以定义的职能领域，供主管开发和利用的工具、技术和思想都很少。）这个部门的新雇员一般有大学毕业文凭并从 10 级开始干起。在 1994 年，程序经理的最高级别是 14 级。由于产品的重要程度不同，工作难度的变化也相当大。

程序经理有广泛的职责。程序经理负责与其他组的大部分协调工作，开发员也负责协调特性和共有构件的级别。在微软初期，这种协调主要采用标准化直观界面（比如，菜单和屏幕怎样在用户面前出现）和功能命令（比如，确保 Word 和 Excel 中存储文件的方式相同）的形式。直到最近，标准化的努力才有了收获。人们通常共同使用的产品当中的命令、工具条和界面不尽相同。最近，协调的努力主要集中在开发相同的特性和共享构件上。一些有开发经验背景的高级程序经理负责此类型的结构标准化工作。这种工作包括应用型产品（比如 Office）和系统产品（Windows 95 和 WindowsNT）。

开发员

确定开发员的级别（指 SDE，即软件开发工程师的级别）是一个比其他专业更正规的过程。这也许是因为确定开发员的级别为其他专业提供了晋级准则和相应的报酬标准。开发经理每年对全体人员进行一次考查并确定其级别。开发主管戴夫·穆尔也进行考查以确保全公司升迁的标准统一。尽管经理们关注平均晋级年限，但微软没有标准的固定晋级时间表。一个从大学里招来的新雇员一般是 10 级，新开发员通常需要 6 至 18 个月才升一级。有硕士学位的员工要升得快一些，或一进公司就是 11 级。从 10 级升到 11 级很容易——用戴夫·穆尔的说法就是“不需动脑子”，“ 11 级的人是那些可以自行工作、自己编写产品代码而不需要太多监督的人。”开发员平均在 11 级上要呆上两年半时间，但有些人要在这个级别上干许多年，因为升迁考查

变得深入了许多。摩尔补充说：“ 典型情况下，从 11 级升到 12 级，我问几个问题；从 12 级升到 13 级，我会问许多问题；从 13 级升到 14 级，要求经理向我和麦克·梅普尔斯汇报此人对所在部门的贡献；14 和 15 级必须得到比尔的同意 ”。穆尔，这位 14 级开发员，详细阐述了升迁标准和要求：

> 当你显示出你是一位有实力的开发员，编写代码准确无误，而且在某个项目上，你基本可以应付一切事情时，你会升到 12 级，12 级人员通常对项目产生重大影响。当你开始从事的工作有跨商业单位的影响时，你就可以升到 13 级。当你的影响跨越部门时，你可以升到 14 级。当你的影响是公司范围的时候，你可以升到 15 级。通常那些“ 智囊团 ”成员一般都是 14 或 15 级的软件设计工程师。公司里 15 级的人员很少。内森·梅尔沃德是 15 级——他还有副总裁的头衔。查尔斯·西蒙尼是 15 级。总共只有五六个人是 15 级。比尔不在这个范围之内，他是总裁。

穆尔估计所有开发员中 50%至 60%是 10 级和 11 级人员，20%属于 12 级，大约 15%属于 13 级，而剩下的 5%至 8%属于 14 级和 15 级。这些数据显示了微软的快速发展。在过去五、六年里，一半开发员仅有一两年的工作经验。显然，经理们一般不用技术级别来估计一个小组的能力，但他们通常让高级人员来开发特殊的关键特性。经理们会提出他们“ 需要一个 12 级 ”或“ 需要一个 13 级 ”员工来对付项目中某一特别困难的部分。特性小组组长通常是 12 级，或者非常有经验的 11 级人员，开发经理一般是 13 级或 14 级。14 级和 15 级人员经常从事一个或几个产品中特别困难的体系结构设计。

开发员可以在一到两年内升为指导教师，接着再过几年升为组长。一个特别出众的开发员如有志于管理可以升为部门的开发经理，或者是整个商业单位的首脑，就像克里斯·彼得斯那样。但是，随着开发员人数增长变慢，升到最高开发级别或成为更高级别的管理人员的机会要少得多。马克·沃克，曾在杨伯翰大学受过计算机教育，现在是 Word 组的开发员，反映了这些变化：

> 从我开始工作的两三年时间里，公司的变化挺大，机会也变得不同以前。我们仍然干许多有趣的工作，或许比以前还要多。但是现在从人员雇佣的角度来看，我们不再像过去那样增加员工。我所在的组从以前的7人发展到40人,不久又上升到60人。但是这种增长现在结束了。……现在整个结构在变。以前，当公司人数增长时，你可以升得很快，并且有许多机会。但现在公司看上去正朝着传统组织的方向发展，当某人离开时，你才能顶上去，并且只能等着被消耗，如此而已。我只好留在组长一职上，我的升迁实际已经停止了。

高级开发员的工作特别富有挑战性，因为微软希望他们既进行人员管理又编写代码。产品组里的开发经理也许是最需要吃苦耐劳精神的职位。此人几乎总是在产品如何工作以及特性如何相互作用方面的专家，同时人们还希望他（她）既尽可能多花时间编码，又能帮助组里的其他成员。由于特性小组组长也是产品特定领域的专家，他们必须全天候地既帮助整个项目又管理他们的小组，以正常开展工作。正如爱德·弗莱斯总结他在 Excel 和 Word

的工作经验时所指出的：“我们举荐某人当技术组长。此人一般应对产品通常如何工作有最广泛的理解……组长是个要求很高的职位。一方面，他们要用所有时间来编程，另一方面，他们要进行管理和分派任务，回答问题。这也就是说他们必须精通编程，同时还要能处理其他琐事。”组长满负荷的工作是基于这样的假定：因为他们有更多的知识和经验，他们可以用更少的时间来做与其他成员一样多的工作。弗莱斯解释了其中的逻辑：

> 我们相信，组长能做至少与组员一样多的工作，并且还能办其他事情。换句话说，在我的组里，我已经有一些很好的程序员。如果我真能胜任组长一职，我在一周内应该能够干和他们同样多的编程工作，并且我只要三天时间来做这些工作……剩下的两天用来完成我的其他任务……有时你在计算机界能听到此类说法，即一个好的程序员比一个差的程序员强上 10 倍。我不知道差别是否真有这么大，但事实确实如此，他们精通于他们的工作。除此之外，他们还有时间做其他事情。而且一般来说，这就是他们能得到这个职务的原因。他们不仅从事自己的工作，而且还照看其他领域……并且学习有关产品的新知识……我们总是在寻找发现这类人才。

除了在技术级别和管理职务上可以升迁之外，开发员的另一选择是在微软的其他专业部门寻找新的挑战。这类事情经常发生，特别是在部门内部。正如克里斯·彼得斯所观察到的：“如果你是非常优秀的开发员，你可以在公司的任何一个项目里工作。如果你是一个非常糟的开发员，就很难换组……但事实上许多人并不换组，因为他们开始在那些几千行代码组成的程序里学习并提高着专业知识。”在微软公司内部存在着某些分歧，这些分歧所围绕的中心问题是人员在不同产品组之间多大程度的流动于个人和公司均为有益。长时期留在一个组里的好处是开发员能深刻理解复杂产品的大部分。不足之处在于从事同样的工作如果时间过长，人们容易疲倦，甚至精疲力竭，而且人们的工作范围会变得狭窄，不利于部门之间的交流。于是经过梅普尔斯的推动，人员流动在增加，特别是从 Excel 组流向诸如 Word 和 Project 这样的领域。比较统一的意见似乎是开发员在考虑换工作之前应该在一个项目里至少工作两个开发周期或者大约三年。乔恩·德沃恩在谈到人们应有多大程度的流动时说：

> 我们愿意做的另一件事是在项目的两个版本之间给相当数量的人员一次换工作的机会。这种想法在于人们可以学到有关产品的更多知识，并且我们有一流的程序员，他们掌握着更多的一手资料，即其他人应如何处理问题……在公司范围内，我们还有一定比例的人员在项目之间流动……我们不鼓励所有人不停地流动，因为这样可能导致某些人在同样的事情上重复工作。我们没必要设计工作指南，因为人们基本上做他们愿意做的工作。我们在过去已作出决策，我们不会让 20 个人去做同一件事，如果这件事只是一个人的工作。因此，没有必要将其规范化……我要说如果有人在同样一件事上工作了三年，这也许足够长了。

区分个人和不同专业的另一途径是报酬。开发员在微软属于报酬最高的

一类人。一批有才华并已在其他公司积累了丰富经验的开发员（比如最早加入 Windows NT 组的开发员）可以不时协商他们的工资额并使工资额超过本级别的平均工资水平。当然这样做的人很少。这种不平等把开发员放在“ 主角 ”位置，这有可能加剧员工的紧张关系。但是其他部门的人员似乎意识到在软件公司，开发员必须要有一个特殊地位。理查德 · 巴斯，Windows NT 组内的一位高级产品经理就持这样的看法：

> 我们中有些人对开发员怀有极度不满的情绪，简直就是忌妒。达瑞尔 · 希文斯是 Windows NT 的主要开发员之一。达瑞尔有 9 辆波尔舍（Porsche——名牌跑车），我希望我也能有 9 辆波尔舍。但我怨恨达瑞尔吗？当然不。他绝对当之无愧，他棒极了。如果是用我的支票来付给他报酬，我也会乐意的。所幸的是从长期来看很清楚，过一两年，你就会得到你应得的报酬。如果由于某些原因，我们请来了达瑞尔并付给他能买 9 辆波尔舍的报酬，而他干得并不成功，达瑞尔就不会在这里呆很久……但是，这些开发员都是精心挑选
> 出来的人才……唯一的影响是，某些人感觉到开发员是“ 主角 ”，自己只是“ 配角 ”，但这是我们这行的特点。

测试

当微软经理们逐渐认识到制造性能可靠、使用方便的产品很重要时，他们更注意在测试领域内留住优秀人才。这有很强的经济含义。因为测试需要在工具、培训上作大量投资，并且个人在特定产品和特性上的经验对于有效的测试是无价值的。马克 · 奥尔森谈到这一点时说：

> 我认为，微软的成功，产品的优秀质量，开发活动的卓有成效主要在于测试是有效而独立的领域。它并非不称职的开发员的收容站，也不是想做程序员的人们的出发点。我们真正致力于使它独立……所以我们只要那些愿意干测试这一行的人……我们希望吸引有兴趣干这种工作的人。我们想要留住他们，并在他们身上投资，培训他们成为适合于各种项目的测试员。我认为这方面我们已经做得很成功，并且还成功地扩大了我们的组织，并沿着这条道路改进了我们的工作程序。

测试员的级别从 9 级到 13 级。还没有测试员能得到像许多高级开发员或程序经理一样的地位和报酬。戴夫 · 穆尔解释了其中的缘由，他说：“ 如果有测试员向我表明他是 14 级的材料，我会给他 14 级……如有必要还会有 15 级。但我还没有见到这类人。 ” 奥尔森虽不是最高级别，但他已经提升得很快了，他说：“ 实际上我是 12 级。但当你计算多数级别的年限时，你会发现每个级别要呆上两、三年时间，而我从 9 级升到 12 级只花了大约一半时间。尽管晋级途径有限，但我感到高兴的是我有机会学习并且我们将工作干得更好的能力和改进工作程序的理解力都在提高。 ”

熟悉某些特定的产品很重要。因此，微软经理希望测试员像开发员一样，在同一个产品组干上几年。至少，他们希望测试员留在同一领域内，像在桌面应用软件，系统软件领域或消费者分部内。而且，由于测试员的经验在不断积累，经理们不鼓励他们转到其他部门，比如转到开发或程序管理部门。穆尔强烈反对测试员更换工作，他说：“ 他们应该一辈子搞测试。如果当初

他们是被当作测试员雇进来，就应该作为测试员从事测试工作。”但穆尔承认微软仅到 1984 年才开始有测试员，到 1991 年，人数才多起来。公司尚没有为测试员规定清楚的晋升途径和流动范围。像乔恩·德·沃恩他愿意让人在同一个工作至少干上几年。

当然，测试员可以向上升迁。通常升迁的途径是当测试组长，而后是主要产品的测试经理。下一步是测试主管的位置。但大多数测试员更愿意做关键产品单位里的测试经理。一些测试员相信他们至少可以转入其他组和部门，尽管这种流动的可能性相当小。奥尔森描述了这种感觉：

> 我认为你在这种职业中能干多久并保持头脑清醒有一个极限……人们转向程序管理。人们偶尔从测试行当转向编程……在你想要的技术级别和管理职务上会有更多的限制。
>
> 在我的部门里共有 6 组人员。每一组有一位测试组长领导，他对本组人员负有管理责任。在我的组织里只做测试的人在增加，有人只做测试并且乐此不疲，他们不愿意管理别人，他们只想搞技术……紧接着他们变换产品，因为他们想体验新东西，这当然是一种选择……但出现得不多。
>
> 对我来说下一步是呆在技术管理岗位上还是转向一般性管理岗位。在我上面的职务不太多。在测试一行，我现在的职务是最高的，我可以在报酬上得到提升，但我的头衔还是一样——Excel 的测试经理。

正如在管理和开发部门一样，测试部门的某些工作比其他工作所需的技巧要少一些。从这个意义上讲，测试系统产品可能是测试技术等级中要求最高的，通常它需要大量的编程知识。测试低容量消费者产品的简单特性也许对技术的要求最低。在每一个产品组里，有些特性比另一些特性更富有挑战性，但所有的测试都很重要。因为用户可能会就最简单的问题抱怨公司并打电话质询（一个电话大约要花微软公司 12 美元）。正如奥尔森所说：

> 还有一定比例的特性不是那么有吸引力，或者工作起来更枯燥……并且不需要技术理解力。比如像测试“帮助文件”……你看着文字并按按钮，……如果主题出现了，那么一般说来做这种工作不需要多少脑力，但需要有人承担任务，了解帮助文件如何工作以及理解使它正常工作的重要性。我们必须做这种工作。某人可能不适合测试 Excel 中的程序环境，但他可以弥补我们必须做的工作的缺口。……这些人可能没有其他产品测试员那样的技术能力，但很难忽略其工作的重要性，很难通过二者的比较来评估他们的工作是否有同等重要性。你必须检查他们正在做的工作是否与部门职责相吻合，工作是否努力，是否把小事情办好了，所以对工作表现的评估是以个人为单位而不是组范围内的，正像我们已经做的那样。

与每年大约 10%的新雇员离开公司相比较，解雇测试员相对要少得多。奥尔森回忆在过去几年里只有 4 个测试员（他的组里共有约 50 位测试员）因为工作表现差而离开。因为在这些人身上已经投资，经理们愿意为不放弃努力的测试员提供更多的帮助和培训，而不是解雇他们。然而奥尔森认识到测试员升迁的上限导致了人们的离去，对奥尔森来说一个更棘手的问题是如何激励选择留下来从事测试职业员工的积极性，他说：“对于在组里已干了两

三年的人来说，这类问题更大。我怎样继续激励他们？怎样才能使他们不断取得新进展？”

激励测试员的一个日益普遍的方法是送他们参加职业软件工程会议。这种会议集中讨论软件质量或软件测试。微软还发起主办室内研讨会和研习班，并请开发员介绍他们的专业以及代码的功能。这些方法以及 1993 年罗格·舍曼被任命为专职测试主管，都增强了测试作为一个独立的职业技术部门的形象和地位。他们还让微软人更多了解被该行业其他地方和其他公司运用的测试观念、工具及其实践活动。戴夫·马里茨欢迎这些新进展，他说：“每年预算可供 10 个人去全国各地参加研讨会。我想这很好，因为……我不是测试方面的世界级专家。”

其他专业部门

微软的其他主要职能型部门——产品管理、客户支持和用户培训——也有与开发员及其他部门相类似的晋职途径、正规级别和报酬支付体系。正如我们以前提到的，这些部门员工可以更换工作，特别是从产品管理部门转向程序管理部门，从客户支持部门转向测试部门。

总之，微软的人员管理是比较成功的，特别是对于这样一个快速发展的公司，在发展进程中没有“职业化”的管理人员，这种成功经验尤为可贵。1991 年在应用部门进行的一次调查（见附录 4 数据摘要）表明：大多数雇员认为微软是该行业的最佳工作场所之一。当然还有一些问题：比如对新雇员培训不够、部门之间缺乏合作等等。这些问题，盖茨和其他经理们正在想办法解决。员工对于低工资（不包括津贴在内）以及工作中“质量”与“数量”需求存在冲突这两个问题也有反映。但整个调查结果显示，微软提供了一个非常好的，尽管不是尽善尽美的工作氛围，这里有精干和平易近人的经理，而且几乎没有人耍政治手腕，搞官僚主义。微软为员工提供了有趣的和不断变化的工作，并有强烈的团队文化以及大量学习和作决策的机会，同时它允许人犯错误。第 3 章将讨论微软开发的各种产品以及公司如何在不断扩张的 PC 机软件市场上开展竞争。

第三章
产品和标准的竞争

开拓并适应不断演变的大规模市场

为了在软件产业日益多样化的各个相关领域中参与竞争，微软奉行一种我们称之为“开拓并适应不断演变的大规模市场”的战略。我们将这一战略分成五个原则来讨论：

·尽早进入不断演变的大规模市场，或以能够成为行业标准的“好”产品促进新市场的形成。

·不断改进新产品，定期淘汰旧产品。

·推动大批量销售，签订专有供货合同，以保证公司产品成为或继续成为行业标准。

·充分发挥作为新产品和关联产品的标准供应商的优势。

·整合、拓宽并简化产品以进入新的大规模市场。

这些原则都不是微软所特有的，然而很少有公司能够在一个产业的几代产品中将这些原则结合起来运用。我们认为微软作为一个公司在其始终如一地将这些原则结合运用的能力方面是超群的，这一能力便是其形成目前市场实力的关键原因。

不是作为发明者，而是作为开拓者之一创造或进入一个潜在的大规模市场，然后不断改进一种成为市场标准的好产品，这是日本 VCR 产业中众多公司的做法。索尼、JVC、松下和东芝沿用了 50 年代安佩克斯（Ampex）发明的录像机，推广了一系列家用录像机；这些公司还开创了一个新的录像带市场。微软公司供应 MS-DOS 和 Windows 操作系统时采用了大批量排他性合同，这与 JVC 公司及其母公司松下公司为使 VHS 进入 VCR 标准的天下而签订一系列大批量供货合同的方式是类似的。英特尔公司采用了类似的策略来支持其微处理器。微软公司还利用其标准提供者的地位向市场推出新产品，这一点与 VHS 制造商及英特尔公司的做法也极为相似。[1]

并非所有行业都涉及平台标准（像操作系统、微处理器或录像机等）和必须具有兼容性的辅助产品（如计算机应用软件、计算机硬件设备或录像带）。同时，尽管平台技术和辅助产品使其提供者能够将消费者和供应商锁定在结构标准、构件和特定产品上，这种优势总是不能永远维持下去。一些公司如日本 JVC 公司和松下公司（及其 VHS 录像机）以及英特尔公司（及其微处理器）现在就承受着来自其他标准的竞争压力，比如数字系统和精简指令集计算机（RISC）微处理器带来的压力。

还有其他一些依靠产品标准和辅助产品暂时取得市场支配地位的著名例子。施乐公司曾经垄断了最初的普通纸复印机和各种复印机的供应和服务。IBM 公司曾控制了计算机以及相关的硬件外部设备、操作系统、语言和开发工具。RCA 公司曾以可替代构件和许可经营费支配了彩色电视机市场。索尼公司发明了 8 毫米摄像放像机并在销售这种摄像放像机和 8 毫米盒式磁带方面处于领先地位。任天堂公司和世嘉公司用家庭录像游戏机、录像游戏机盒带及许可经营合同支配了这一产业。索尼公司和飞利浦公司控制了激光唱盘的标准，并进而控制了计算机 CD-ROM 的标准。

但成为标准提供者并非是一件很容易的事情，还有两个更不易应付的挑

战。一个是在足够长的时间内维持这一地位，以便在几代产品中利用这一优势。另一个是将这种支配力扩展（比如通过产品或营销联系）到新的市场上去。尽管创建已有 20 年之久，但微软公司相对来说还很年轻；它成为一个销售额名列前茅的 PC 软件公司只是 1988 年以后的事。它控制了不止一代的关键工具和平台技术——语言和操作系统（字符形的和图形计算机），但它控制的仅是一些应用软件的领域。不过，微软是少数维持并扩张了其市场实力的公司之一。我们认为本章提及的两个原则可以解释微软公司是如何做到这一点的。

首先，微软公司不断进行渐进的产品革新，并不时有重大的突破。尽管这通常不过是将许多渐进的革新进行包装，但这种重大的变化使老产品已显陈旧。渐进的、有时是重大的革新形成一种不断的新陈代谢机制，使竞争对手们很少有机会能对市场的主导者构成威胁。微软公司积聚了巨大的财力和技术资源，使其能够维持这一 R&D 水平。它还遵循着某种不同寻常的战略原则：主导性企业一般不愿因引进新产品而使其现有产品的销售量减少。[2]

例如，微软公司用 Windows 3.1 操作系统取代了 MS-DOS 曾拥有的支配地位。由于 Windows 3.1 既是一个图形用户界面又具有一系列新的操作系统功能，因而使 MS-DOS 的销售量锐减，而 DOS 应用软件也大都过时。同样，Windows 95 和 Windows NT 以及为这些系统编写的一些新应用软件又使 Windows 3.1 和老的 Windows 应用软件不再时兴。Visual Basic，一种大大简化了 Windows 应用软件屏幕和菜单的 BASIC 图形版，取代了市场上该语言的传统版本，因为几乎所有的新应用软件开发都与 Windows 及其他图形操作系统相一致。Office 套装软件由于其技术和定价两方面的原因，事实上已挤掉了个人应用程序的销售和使用。Microsoft Network 甚至有可能使目前一些专为 CD-ROM 销售的微软多媒体产品过时。

其次，微软公司在不断地整合、嫁接、重组并简化其产品。这样做的主要目标是以同时拥有多种过去一度是相互分离的功能的产品进入新的市场，并使产品更广泛地走向广大用户。这种竞争和革新模式使微软公司的影响力扩展到包括计算机初学者及家庭消费者的规模巨大的市场上。

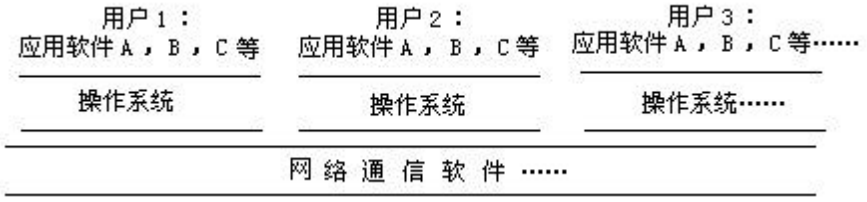
同样，这一战略也并非微软公司所独创。VCR 生产商已经使其产品结合了电视机的功能，并使 VCR 和摄像放像机特别易于使用。微软公可现在的做法也类似苹果公司 10 年前开始的做法 :苹果公司在其 Macintosh 计算机中集成了不同的应用软件，使其特别易于使用，并且它还不断提供新的软件产品甚至是联机服务。任天堂、世嘉、索尼和其他家用电子企业也正在扩展其录像游戏机的功能。电话、有线电视和无线电通讯公司都在为其网络开发新的用途，从而使其影响力扩展到新的市场上。微软公司正在做同样的事情：维持现有的技术和产品，同时又在新产品开发方面显示出非凡的能力。它正在通过创造产品间的相互联系和利用广大消费者网络来进入新的而又相互关联的大规模市场。

原则一：尽早进入不断演变的大规模市场，或以能够成为行业标准的“好”产品促进新市场的形成

在 70 年代中期，个人计算机领域的第一种大规模市场产品是计算机语言，人们用它来编写能使新 PC 硬件运转的软件。第二种产品是操作系统，它

是一些控制计算机基本操作的特殊程序。之后是一些悄悄出现的台式计算机应用软件，如文字处理软件和电子表格。更近期出现的大规模市场包括（那些通常是供公司或其他大型组织使用的）分布式计算机的网络操作系统以及多用户通讯和应用软件产品。对家庭消费者来说还有联机网络系统和应用软件。一个典型的 PC 用户要有几种应用软件、一个操作系统和一种网络通信程序，其层次如图 3.1 所示。

图 3.1PC 软件的主要类型



然而，有两种趋势使得 PC 软件的这一层次图更加复杂化了。一种趋势是计算机公司越来越多地将一个软件层次上的功能加到另一层次上，这种做法使得生产简单产品的厂商难以竞争。比如像我们后边要讲到的，微软公司就把包括一种原始文字处理器和计算器在内的许多小的应用软件功能加进 Windows 3.1 操作系统里边。Windows 中还包含了一些基本的网络通信功能。另外，微软在 Windows 95 中加上了更复杂的网络通信特性。类似地，Novell 公司也因其在 Netware 操作系统中所增添的广泛的网络能力而独树一帜。Lotus（现由 IBM 拥有）正在销售一种称为 Notes 的产品，其中就包含了最新的数据库管理应用程序、操作系统和网络通信程序等多种功能。

第二个趋势是，PC 软件公司为了使其产品出类拔萃并更多地向个人用户和公司用户兜售这些产品，引进了更多类型的系统、应用软件和网络软件。表 3.1 列出了 9 种不同的分层次，后面都附有微软公司的代表性产品。从列举的这些产品中可以看出，微软参与了每一主要软件层次的竞争，尽管竞争的有效性不尽相同。我们提到过，微软是从生产 BASIC 编程语言开始的，然后进入台式计算机

表 3.1PC 软件和微软产品的层次

————————————————————————————————————

系统软件

1.编程语言及其他开发工具（如： BASIC， C， Visual Basic， C+ +，OLE，用来编写其他软件产品）

2.桌面操作系统（如 MS-DOS， Windows 3.1，Windows 95，用来运行 PC 应用软件及其他软件产品）

应用软件

3.桌面应用程序（如 Word， Excel， Office， Access，用来完成各种工作）

4.公司应用软件开发和服务器工具（如 Microsoft SQL Server，Microsoft Back Office，用来为 PC 网络建立数据库系统）

网络软件

5.网络操作系统（如 Windows for Workgroups， Windows NT，用来连接不同的 PC，并使接合起来的 PC 能够通过用户服务器功能存取信息或访问应用软件）

6.网络通信程序（如 Microsoft Mail，Microsoft Exchange，Microsoft At Work，用来使 PC

能够与其他 PC 及可编程设备交流和分享信息）

7.联机网络系统（如 Microsoft Network，用来使 PC 能够通过电话线或有线电视系统取得电子信息和电子化产品）

8.联机网络工具和服务器（如以 Windows NT 为基础的 MicrosoftTiger Video Server，用来通过网络产生和输送信息及其他网络产品和应用程序）

9.联机网络应用软件（如依赖于 Microsoft Network 的一些现有软件，以及为家庭提供金融服务的 Microsoft Money 等产品，用来为联网的 PC 用户提供大量产品和服务）

操作系统的大规模市场，进而又涉足公司用网络操作系统。操作系统现在占微软收入总额的大约三分之一（见导论中表 2）。它还很早就开始进行多样化经营，其所生产的桌面应用软件和其他类型的应用软件产品，现在已占到收入的 60%。90 年代，第二和第三分层次的软件构成了整个行业中最大的大规模市场，尽管从长远来看，第七和第九分层次将有可能成为新的最大市场。在后者中竞争的厂商可以对进入其联机网络者收费，也可以通过使用户购买产品和服务或进入其他数据库和网络而获取利益。由于对 PC 网络和建立办公系统所需的通信程序（如 Lotus Notes）的使用正与日俱增，第五、第六分层次也是重要的市场。

我们将在本章的后面进一步说明这些不同类型的产品以及微软的战略。这里需要着重理解的是关于公司如何进行竞争的四个关键。第一，微软咄咄逼人地从一种软件和大规模市场转向另一种软件和大规模市场，从一代产品转向另一代产品。这种多样化和更新换代有时是通过技术上有联系的产品和某种启动性技术完成的。我们在本章中谈到过的例子有 MS-DOS 与 Windows，DOS 应用软件与 MS-DOS，Office 套装软件与 Windows，对象的链接与嵌入（OLE）与 Windows 和 Windows NT 及应用软件，以及 Windows 95 与 Microsoft Network 等等。微软还利用同硬件与软件零售商的关系及类似的营销办法来推销其新产品和换代产品。

第二，微软创立了一些标准，它们本身能促进新的大规模市场的形成。例如，用于 IBM 兼容计算机的 MS-DOS 操作系统为 PC 硬件制造商开创了 IBM 兼容产品的市场的形成。这些价格低廉的 PC 机的出现又扩大了操作系统市场以及应用软件与其他软件产品的市场。MS-DOS 和 Windows 开创了自己的应用软件市场。Visual Basic 为定制程序开辟了市场。OLE 为 OLE 启动的 Windows 应用软件开创了市场。我们相信，Microsoft Network 将为联机网络应用软件开创一个新的大规模市场，至少它应该能够大大扩展现有的相对先进的联机网络用户群。微软标准开创或拓展的市场为微软增加了发展机会，只要它还继续在这些新的战场上竞争。

第三，积极的扩张一直是很重要的，因为微软早期的大规模市场已经饱和。除了一些例外情况（如 Visual Basic）外，PC 编程语言方面的创新已不如应用软件和网络系统等其他软件领域的创新多了。不仅如此，计算机用户多年以来已经能够购买操作系统然后把“现成”的应用程序包装起来使用。这使编程语言对大多数人来说已不必要，结果它只成为整个软件市场上很小的一部分。操作系统，然后是文字处理软件及电子表格等独立的桌面应用软件，都已成为大规模的商品化产品。这一事实促使微软及其他公司去开发套装软件，并涉足各种类型的办公室与网络软件、家用产品、多媒体出版物以及联机（“信息高速公路”）产品和服务。

第四，微软现在在模糊不同层次的软件之间以及这些层次内不同类型的软件之间的界限方面一路领先。这种将某些特性合并或“集成”到一小部分产品上的趋势使微软更易于推销产品，吸引大量新的消费者。增加更多的特性还有助于维持已有的用户群：当消费者的需要随时间发生变化的时候，他们更愿意从微软购买升级产品和新产品，而永远不需要到其他公司去购买不同类型的软件（我们将在第 7 章中把这一竞争方式作为微软“向未来进军”战略的一部分加以详细讨论）。例如，微软在一段时期内给 MS-DOS 和 Windows 3.1 附加了很多特性；创造了集成套装软件 Office；开发了含有网络特性的更简单的家用套装应用软件 Bob；将新的 Microsoft Network 产品与 Windows 95 操作系统联系起来。

市场资料

各种各样的市场资料显示了微软在这一非常广阔的 PC 软件产业中所处地位的影响程度（与局限）。在 1993 至 1995 年间，微软占台式 PC 新增操作系统销售量的至少 80%，每年的销售量为 4 000 万套（IBM 的 OS/2 操作系统占这一市场的大约 10%）。Windows 和 MS-DOS 分别是两种最畅销的系统产品。MS-DOS 在已安装的基础上也是领先的，全世界 1.7 亿 PC 用户中有 1.4 亿在使用 MS-DOS；这些 MS-DOS 用户中有大约 7000 万同时使用 Windows。尽管微软已将 MS-DOS 并入 Windows 95（出品日为 1995 年 8 月），作为一种独立的操作系统 MS-DOS 不会再卖出很多套，但从 1981 年以来 MS-DOS 一直是名副其实的摇钱树。多年以来，它带来了几十亿美元的销售收入和高达 25% 的公司利润，而其支持与开发成本却微乎其微。1994 年，Windows 和 MS-DOS 一共为微软带来约 10 亿美元的销售额，其中约 80%是来自康柏和德尔计算机公司这样的稳定的 OEM 渠道。Office 及其主要部分（Word 和 Excel）共有约 3 000 万用户，共带来 17 亿美元的销售额，为微软创造了大约一半的利润。[3]

微软在应用软件和网络产品方面一直是市场份额较小，起步较慢。在 1990 年引进 Windows 3.0 和 Microsoft Office 套装软件以前它仅占有 10% 的电子表格市场和 15%的文字处理软件市场。但是到了 1995 年，微软在这两大类 PC 应用软件的新增销售额中已占约 60%，其中 Word 和 Excel 在 Windows 和 Macintosh 计算机上的使用都是领先的。单单 Office 一项就占了迅速发展的套装软件市场大约 70%的份额。[4]微软在较复杂的数据库管理工具及应用软件的市场上与公司用网络操作系统的市场上仅占有百分之几的份额，不过最近它已进入这些领域，各种面向公司用户的产品（如 Windows NT 3.5）目前销量不错。

总之，微软在 PC 软件产业中拥有最为广大的产品组合。其产品从编程语言到联机网络系统，既包含复杂的办公系统，也有专供儿童和电脑初学者使用的产品。 1995 年 6 月结束的财务年度中大约 60 亿美元的销售额使只经营或主要经营 PC 软件的其他竞争者们相形见绌(Novell 公司及其 WordPerfect 分部在 1994 年的收入约为 20 亿美元，Lotus 则不足 10 亿美元）。微软在主要的大规模应用软件市场之外并无支配地位，然而，该公司正对涉及 PC 的几乎所有重要领域内的竞争者构成严重的挑战。微软的目标，正如麦克·梅普尔斯所说，一直集中在那些潜在销售额和利润都特别丰厚的新兴市场上：“我们正着手创建一些我们认为可以凭其赚大钱的东西，这样即使你需要一些什

么别的东西，你也可能无法从我们这里得到……我们正处于我们开始权衡利弊、进行明智投资的阶段。我们在这一领域中所占的比重太大了，我们必须开展新的业务。我们在我们所处的产业中占有如此显著的一部分，以至于很快我们的增长就只能和整个产业的增长率一样慢了，这并不是我们愿意看到的。”

从语言到操作系统

1975 年，比尔·盖茨和保罗·艾伦通过引入第一种计算机语言 BASIC 促成了 PC 软件产业的发端。BASIC 是英文“初学者通用符号指令码”的首字母缩略词，发明于 1964 年的达特茅斯学院，是一种用于主计算机和微型计算机的相对简单的语言。盖茨和艾伦采用了这种语言的一种公开使用版本，将其运用到第一种价格低廉的个人计算机——MITS 计算机公司的 Altair PC 上。紧接着，他们又推出其他语言产品，使它们能够运用于 Altair 之后出现的所有廉价 PC 上，这些计算机有 Commodore64、Apple II 及 IBM PCjr 等等。[5]

盖茨和艾伦在 1981 年之前一直在销售编程语言，这时 IBM 要求他们为其新型 PC 提供一种操作系统（微软的两位创始人以前从未制造过操作系统，不过他们总算抓住了这笔跟 IBM 之间的交易）。盖茨和艾伦开发这种系统并不是从零做起，而是以 75 000 美元的价格从当地一家小公司西雅图计算机公司那里购买了一种原型产品 Q-DOS——英文“快速而有隐患的操作系统”的缩写。他们以此作为 DOS 第一个版本的基础。Q-DOS 虽然不是一种商业产品，但它是第一个为新型 16 位英特尔微处理器设计的操作系统，从而是一种开创性技术（最新的 PC 和录像游戏机上的微处理器是以 32 位甚至 64 位的字节片处理信息的）。

从某种程度上说，IBM 从微软购买操作系统的决定对盖茨和艾伦只是一次很好的发财机会而已；事实上，IBM 曾试图同数字研究公司达成协议，因为该公司的 8 位 CP/M（Control Program/Monitor 的缩写）为 Q-DOS 提供了模型，不过没有成功。但 IBM 转向微软至少部分上是因为盖茨和艾伦已因推出 BASIC 这种现在已成为 PC 机标准编程语言的产品而享有很高的声誉。不仅如此，通过制造 BASIC，微软表明了其拥有制造可靠的可推出产品的模拟工具和编程技术。相比之下，IBM 的经理们却拿不准自己的程序员能否快速地编写出一种适用于有严重内存限制的 PC 机的小型操作系统（这一工作迥异于编写主计算机软件），而其他的 PC 软件公司在质量和交货方面的声誉又很差。[6]

桌面应用软件

正如并不是盖茨和艾伦发明了 BASIC 编程语言或 DOS 产品的核心一样，微软也不是其第一种 PC 应用软件电子表格和文字处理软件的发明者。这些软件在几年前为主计算机和微型计算机以及为 Xerox Alto PC 原型产品和 Xerox Star 工作站编写的应用程序中就已经有了原型（施乐公司在 70 年代设计了这些应用程序，但从未使它们取得商业上的成功）。其他公司在 70 年代后期和 80 年代早期也诱导着微软为个人计算机引入产品。然而，一旦 IBM PC 与 PC 兼容市场开始扩张，微软便开始推出具有竞争力的优质产品，并积极地推销这些产品。随着比尔·盖茨及后来其他经理们对新市场的不断追求，微软也不断在产品上加入有用的和富有远见性的特性（要了解微软主

要桌面与商业应用软件产品的大致情况，请参阅附录 2）。微软在应用软件开发方面的勃勃雄心可以从麦克·梅普尔斯的另一段话中看出，尽管它明显是一种夸张，或者也许只是一种意愿：“有人认为如果我们既不追随 Lotus，也不追随 WordPerfec 或 Borland，则很难让人理解。我的工作是争取应用软件市场的合理份额，对我来说就是 100%。”[7]

电子表格 当微软于 1980 年开始搞电子表格(项目的名字叫‘Electronic Paper”）时，盖茨雇了一位顾问来研究当时已打开市场的两种产品：一种是 VisiCalc，是由软件艺术公司（后为 Lotus 兼并）于 1979 年为 AppleII 引入的；另一种是 SuperCalc，是由一个名叫 Sorcem 的公司（后被计算机同仁公司兼并）为 CP/M 计算机开发的。曾在施乐 PARC 为电子表格构想做过一些基础性工作的查尔斯·西蒙尼于 1981 年加入微软时接手了这个项目（后来经营微软程序员训练中心的麻省理工学院计算机科学专业的毕业生道格·克隆德也于 1981 年加入了公司，并成为五人开发小组的一位主要成员）。西蒙尼的小组创造了一些新奇的特性，比如在屏幕下端出现的菜单和分菜单，这使得使用者能够选择命令（如“打印”或“存盘”）并运用简化公式。微软于 1982 年 8 月为 AppleII 发布了这种名为 Multiplan 的新型电子表格的第一个版本，之后又为 CP/M 计算机和 IBM PC 提供了几个版本。Multiplan 在行业出版物上和 Apple 的公共认可系统中都得到了很高的评价。[8]

Multiplan 在电子表格销售中马上处于领先位置。不过微软在推出能够充分利用有更大内存的新上市 PC 的升级版方面行动迟缓，这一延误使得 Lotus 以其 1-2-3 电子表格在 1983 年超过了 Multiplan。Lotus 一直占有美国电子表格市场的80%,每年的销售量在PC软件公司中都保持领先,直到1988年，微软才重新坐上第一把交椅。[9]

Lotus 是促使 Multiplan 的下一代产品 Excel 出现的主要原因。克隆德这样评论该小组的开发战略：“我们的任务是，1-2-3 做到的我们都要做到，并且要做得更好。”[10]微软这一项目始于一次旨在确定如何同 Lotus 展开竞争的三天休假会（盖茨和西蒙尼均出席）。这次会议的结果是一份有关新的电子表格应当具有的理想特性的大体说明，这便是现在微软称之为对新产品的“想象性描述”文件的前身。克隆德把这些想法汇集到一份 20 页的备忘录中，作为一份概要说明书。负责完成这项任务并在后来成为程序经理的是曾在推销 Multiplan 期间研究过 VisiCalc、SuperCalc 和 Lotus1-2-3 的特性的吉布·布鲁门萨，他承担了制定产品说明书的任务，并同开发者密切合作，确定出每一种特性的细节。最终的产品结合了 Multiplan 的优点和 Lotus1-2-3 及另一种竞争产品 Framework 的长处，之后微软将新程序翻译到 Macintosh 平台上。[11] 正如我们下面要讲到的，微软软件销售量的回升，是以其在开发 Word 和 Excel 的 Macintosh 版过程中精益求精的图形编程技术为基础的。[12]

文字处理 一家名叫微处理机公司的企业于1979年开辟了PC文字处理的大规模市场，其产品是为 8 位 CP/M 计算机编写的，称为 WordStar。[13] 由于 WordStar 用起来比较困难，因此这一市场仍为后来的进入者如 WordPerfect 公司（1979 年建立）和微软公司等敞开着。西蒙尼在施乐 PARC 工作时曾设计了一种名叫 Bravo 的图形文字处理软件， 1982 年他接管了微软文字处理

软件开发工作，编写了一种专门用于 MS-DOS 和 16 位 IBM PC 的新产品 Word。微软于 1983 年将此产品投放市场,并同时举行了一次成绩非凡的大规模市场促销活动，在一次《 PC 世界》杂志的特别发行中发布了 450000 张演示盘。[14]

Word 吸引买主是因为它有许多革新性特性，在这一点上它比 Multiplan 或 Excel 的早期版本出色。这些特性包括增加了一个图形用户界面、“ 所见即所得 ”（WYSIWYG）的直接可视功能以及盖茨坚持要加载的一个激光打印机驱动器。[15]微软还在其 1984 年引入的 Word 的 Macintosh 版使用了很多相同的编码。Mac 版没有 PC 版那么多的特性，但当时是除苹果公司的 MacWrite 以外唯一的 Macintosh 文字处理软件,而 MacWrite 尚不能处理 8 页以上长度的文件。微软的 Word 由此填补了 Macintosh 软件的一个主要空白,并与 Excel 1.0（1985 年引入）一起促使 Mac 成功地成为一种商业产品。[16]Word 与 Excel 的 Macintosh 版也为微软奠定了与新型 WindowsPC 平台下的图形应用软件产品竞争的基矗不过，微软 Word 的 PC 版并未像更高级的图形应用软件那样引起微软的关注，在对 DOS 机的销售方面很快就大大地落后于 WordPerfect。在对 Windows 计算机的销售中并最终在对全部文字处理软件的销售中，Word 将来的版本都将直接与 WordPerfect 竞争并将最终超过 WordPerfect。

应用软件集成　微软在电子表格和文字处理程序之后又推出了许多其他应用软件产品（比如显示、数据库管理、项目管理、电子邮件、个人进度安排和桌面印刷等方面），通常用于 Macintosh PC 和 Windows PC。公司起先一心一意地开发独立产品，如 Word 和 Excel，但不久发现将不同产品捆在一起并降低价格能吸引更多的消费者。结果，微软开始逐步尝试以 Office 之类的“ 套装 ”软件来取代单个的桌面应用软件。

Office 第一版于 1990 年推出，其主要的引人注目之处在于它是一套以 Word 和 Excel 为主体，以 PowerPoint 为图形程序包的优惠应用软件。一种电子邮件产品 Microsoft Mail 现也包含在这一套装软件内。Access，一种数据库管理产品，现在套装于售价较高的 Professional 版内。微软逐步将这些不同产品的特性集成起来，使 Office 成为一种价格低廉的应用软件产品。盖茨在 1993 年 10 月推出 Office4.0 时对这一战略解释说：“ 我们第一次将我们的主要应用软件定为 Office 而不是各种单个的软件……这并不是说单个的软件不重要。它们很重要。但是现在 Excel 和 Word 作为 Office 软件包组成部分的销售已经超过其总销售的一半。这样，我们就使 Office 远不再是一种优惠推销一组应用软件的方法了，而毋宁说是一种单个产品。 ”[17]

图形用户界面

随着微软从一个大规模市场走向另一个大规模市场，它也从字符界面转向最具大规模市场性质的界面：图形用户界面（GUI）。字符界面比如 MS-DOS 以及为此操作系统编写的老软件主要是在计算机屏幕上显示字母和数字，用户与计算机程序的交流主要通过英文式命令或通过一系列击键动作，其中大部分都难于记忆。相反，图形用户界面依靠一种象征性的“ 图标 ”，使人们使用计算机时仅需移动一个由“ 鼠标器 ”或“ 跟踪球 ”这样的装置控制的指针即可。图形界面使得计算机操作起来容易多了，因为人们启动程序和开始做其他工作时只要指住并“ 咔嗒 ”一声点一下某一特定图标就行了（一些 DOS

应用软件如 WordPerfect 5.1 和 6.0 现在也有原始的图形界面，就像 MS-DOS 的最新版本一样。但这些产品没有图标，而且仍基本上是字符性质的）。

GUI 程序比 MS-DOS 或 DOS 应用软件需要多得多的硬件处理能力和内存。但这一事实使得 Windows 标准促成了两个新的大规模市场：一个是有大量内存和快速处理能力的 PC 市场，另一个是与 Windows 兼容的 GUI 应用软件市场。这两个市场的成长反过来又刺激了对更多 Windows 操作系统与新 Windows 应用软件产品的需求，从而对微软和其他应用软件产品供应商形成了一种利润循环圈。

微软的开发者们也从与苹果公司合作研制用于 1984 年引入的 Macintosh 的应用软件产品中学习如何创造 GUI 产品。1982 年 1 月，微软与苹果公司签订了一项有关设计电子表格、数据库和图形应用软件产品的合同，当时 Macintosh 尚处于开发阶段。这使得微软开发者们能够熟知 Mac 的用户界面和内部运作。合同禁止微软为任何其他计算机引入 GUI 或鼠标程序，但仅限于 Macintosh 出品后一年内或最晚至 1983 年 1 月 1 日。[18] 此后，微软可以自由地为 PC 兼容设备开发类似的应用软件。微软这样做了，尽管由于 DOS 的兼容性和 PC 硬件方面的限制，微软的行动有点慢。

微软在创造 Windows 操作系统的过程中使其图形编程技术迅速完善。第 1 版（1985 年）和第 2 版（1987 年）有很多问题，包括 MS-DOS 的内存限制问题，所以它们都未能成为商业化产品。但微软锲而不舍，3.0 版（1990 年）突破了 MS-DOS 的内存限制，销量达到几百万套。1992 年 3 月的 3.1 加强版很快销售了几千万份，成为在与原来的 IBMPC 和英特尔微处理器标准兼容的 PC 上进行图形计算的新的市场标准。盖茨与施乐 PARC 公司和苹果公司的先行者们一样,在 80 年代早期就认识到 GUI 操作系统及其应用软件是大规模计算机市场的未来：

> 微软把公司的赌注下在图形界面上……图形界面成为主流的时间比我原先预计的要长，但今天我们可以说这是人们使用个人计算机的主导方式。我们只要看看 DOS 应用软件和 Windows 应用软件的销售对比情况就会发现，在过去两年半的时间里，字符式应用软件从占市场份额的约 80%降到不足 20%……而当最后一种主要的字符方式应用软件更新版出品时，即使这 20%也将大幅下降。行业的所有资源最终都转为制造图形应用软件。[19]

正当微软努力钻研其 GUI 技术时，苹果公司的人们意识到如果微软开发 Windows 的努力得以成功，那么他们自己明显的优势就会丧失。苹果公司还特地提出抗议，声称 Windows 第 2 版过分地模仿了 Macintosh 的“look and feel”。但是，微软争辩说实际上是施乐公司发明了该项技术而未能使之商业化。在付出了 900 万美元的律师费后，微软使苹果公司的上诉于 1992 年 4 月被驳回。[20] 在拒绝了前几次申请之后，美国专利与商标局终于不负微软的长期努力，于 1994 年 8 月批准微软使用标签为“Windows”的注册商标。[21] 这一决定没有承认微软是图形计算机的发明者，但却承认了它是 Intel 兼容 PC 图形计算机标准的开发者。

联合大规模市场的网络系统

随着 PC 硬件处理能力的提高而兴起的另一 PC 软件市场是联网计算机市场。本来，这些用户使用的是以公司或其他大型组织（如大学或政府公共部门）为基地的强功能计算机。从 60 年代开始，这些机构开始能够用与大型机或微机相联的终端——80 年代后是个人计算机——把愿意分享同一数据库或通信系统的人们联系起来。最近一段时期，各种机构都创造了强功能台式 PC 的网络或工作站，一些计算机充当信息或通信控制的“服务员”，它们连接了大量充当“顾客”或信息接收者的计算机。这些网络一般都使用以 UNIX 为基础的操作系统，如 Novell 公司的 NetWare。

为进入这一潜在的规模巨大的市场，微软紧追 AT&T（UNIX 的发明者）、Novell 及其他公司如 DEC 和 Sun Microsystem 公司等。微软开发了 WindowsNT 并于 1993 年 8 月 31 日引入第一版（编号为 3.0，以便与 Windows3.0 相符），于 1994 年推出升级版 NT3.5（参见附录 3 关于微软操作系统的说明）WindowsNT 要求具有大量内存和计算能力的强功能计算机，但它也提供了一些吸引公司用户的独到特性。

与基于 UNIX 的系统或专有性操作系统（比如 DEC 开发的系统）不同，NT 可以运行 Windows 和 DOS 应用程序，有与 Windows3.1 相同的用户界面。NT 还可以在英特尔兼容机以外的计算机上使用；在将 Macintosh、PC 和工作站联网时有一种 NT 特别版可以充当服务器。像 UNIX 和其他高级系统一样，它设有防止一个用户未经允许进入另一用户文件的安全功能。

另外，像 UNIX（及 NetWare 与 OS/2）一样，NT 是“多线程的”：它可以自行分解成几个处理线路（称为“线程”或“虚拟计算机”），然后既可独立、又可同时执行不同工作或应用程序。每一线程都保护一个软件不因另一软件运行失败而受到影响。相比之下，Windows 3.1 虽能执行多种工作或应用程序，但只能在一条线程上进行；如果一个应用程序运行失败，整个系统就可能要关闭。单线程应用程序还必须占用计算机的时间，一个高强度应用程序（如一个多媒体程序）会导致同时分享处理器的其他程序运行特别缓慢。这些不足之处（主要是单线程 MS-DOS 的人为缺陷）使 Windows 3.1 的用户们在操作时经常遭到失败，这使微软很难将 Windows 3.1 销售给公司用户。

NT 的起源可追溯至 1988 年，当时内森·梅尔沃德开始了一个编写一种优于 OS/2 的操作系统的小型项目（OS/2 是一种本来由 IBM 和微软于 80 年代中期共同设计以取代 DOS 的操作系统，但在两个公司于 1989 年停止合作后仅由 IBM 独家开发与销售）。米尔沃德希望新的系统能在 RISC 工作站上使用，也能在英特尔芯片和多处理器计算机上使用，他决定申请 Mach 的使用许可证并对其作进一步研究，这是一种于 80 年代在瑞克·罗歇德领导下在卡耐基-梅隆大学开发出的 UNIX 型系统（瑞克·罗歇德现已是微软负责研究的副总裁）。与此同时，DEC 公司畅销的 VAX 微机的 VMS 操作系统的主要设计者之一戴夫·卡特勒也正在华盛顿特区贝尔维尤附近的 DEC 办公室里从事开发一种 RISC 操作系统。在 DEC 管理层取消了他的项目以后，他辞职并加入微软指导 NT 的开发。[22] 微软同时还在另一独立小组里继续进行 Windows 大规模市场版本的开发，尽管这两种产品有一些可共享的构件和设计概念，包括用户界面等。

Windows NT 的响应者并没有招之即来：盖茨宣布第一年可望销售 100 万套，可实际只售出 30 万套左右。原因主要包括运行 NT 所需要的硬件升级价格昂贵，利用 NT 32 位处理能力的应用程序严重短缺等，还有其他一些新产

品中常见的不足之处。不过，微软在其第二代操作系统和升级版中展示了一种全新的结构以及许多新特性，这些产品包括 Windows 95 和于 1994 年 9 月发布的受人欢迎的 WindowsNT3.5。NT3.5 在 4 个月之内就卖出 700000 套，至 1994 年 12 月，NT 总销售量达 100 万套左右[23]。微软还把 NT 作为用于诸如交互式电视和点播式电视等有线电视线路和电话线上的信息高速公路产品和服务的服务器技术。[24]

原则二：不断改进新产品，定期淘汰旧产品

微软在竞争中先发制人，并大规模地扩张其市场覆盖面，增强市场渗透，这主要是通过两种方式。第一，通过每年为 R&D 开辟几亿美元的财源和稳步地改善其产品，微软使得其竞争者要想赶上来是难乎其难的。我们可以从微软不同版本、不同口味的操作系统、电子表格、文字处理产品和 Office 套装软件中略见一斑。微软甚至还花费数年时间建立了图形 Windows 界面层并逐步使之成为一种功能全面的操作系统（将 MS-DOS 嵌入其中），从而使 MS-DOS 免于遭到淘汰。

第二，微软定期推出新的换代产品，而不是沾沾自喜地看着现有产品的销售相对于竞争者来说逐步降低。这些新的换代产品（以 Office、Visal Basic、 Windows NT 以及 Windows95 为代表）仍严重依赖于原先的产品，并且在某种程度上不过是一些渐增改进的拼凑。但是，一旦几种新技术放在一起，许多这种小的改进就会导致一种新产品的出现，使老产品几近过时。正如 Office 产品部副总经理克里斯·彼得斯所说，微软的战略是每 3 至 5 年就将几种渐增改进组合起来，从而使产品发生较大变化。这种想法是要配合甚至促进计算机硬件处理能力的“指数增长”：

> 我们所做的一切在 3 年以后将不再有意义。对产品来说，未来 5 年人们拥有的计算能力将等于过去创造的一切计算能力之和。这一事实意味着你必须时时冲在别人前边。如果你把指数增长内在化，别人都会觉得事情与去年相似，只有小小的不同，这在很短时间内当然是对的。当你在两年的小区间内看一种产业发展，它看起来确实像是直线式的，这时你必须再往回看，你就会发现它通常是按指数发展的。这时，你会下更大的赌注，下与以前稍有不同的赌注……
>
> 我想这就是微软的一个内在原则。我想这就是比尔一直了解和关注的事情，你[必须]……大刀阔斧地改变事物，制定长远的计划……一个典型的例子是，DOS 曾经辉煌一时。[我们当时可以说，]“我们只需出产 DOS 的新版本，何必研制 Windows？”结果却是，Windows 促进了多得多的硬件和计算能力的创造……从某种意义上说，Office[这种革新]使单一产品过时了。正当我们的竞争者们竭力适应 Windows[应用软件]时，[现在]他们……又不得不制造一种好的文字处理软件加好的电子表格加好的数据库。你现在必须成为那样的软件公司。

渐进的革新

批评家们称，微软的产品很少一开始就“命中目标”，而通常是经过两三个版本才创造出能与其头号竞争者相匹敌的产品。[25] 例如，Windows 的头

两个版本的功能特性非常有限，即使 3.1 版在易用性和其他技术特性方面也比 Macintosh 操作系统落后。与 Novell 公司的 NetWare 相比，WindowsNT 在公司用户市场上的销售一开始并不好，尽管 NT 3.5 正在好转。在微软推出的电子表格（Multiplan、Excel）和文字处理软件（Word）、集成的套装软件（Office）以及家用产品系列（多媒体产品以及个人财务和其他产品）中，我们都可以看到这种类似的随时间变化而不断改进的模式。微软于 1991 年推出的 Money 经过了两个版本才达到了 Intuit 公司的 Quicken 的性能，而销售量却从未接近过（Quicken 于 1985 年左右推出，用户有大约 700 万，而 Money 只有 100 万）。即使像 OLE 和 VisualBasic 这样具有重大革新特性的产品和技术在其第一版与最近的版本之间也经过了相当长的演进过程；而且它们仍有待于进一步的提高。

随时间的变化而不断变化并加载新的特性，这导致了富有竞争性的产品的不断出现。比如，看一下行业评论我们就会发现微软产品在标准特性、易用与易学性、文档与技术支持等方面现在已趋于占优。有些产品在高级特性方面有些滞后，不过多数情况下微软仅瞄准大规模市场的用户或仍处于向多版本中加入新特性的过程中。[26] 多数微软产品目前不仅销售状况良好，而且技术评价也不错，它们越来越多地受到行业设计方面的奖励。例如，《PC 杂志》将 1994 年应用软件技术优胜奖授与微软 Windows 下的 Office，并将 Office 的三个组成部分——Word6.0、Excel5.0 以及 Access2.0——评为该年度各自领域的最佳产品。另外，编委们还授与 Windows NT 3.5 系统软件类技术优胜奖，将其命名为 1994 年推出的最佳操作系统。[27] 这些成绩是非常显著的，因为微软的业务并不专一；它在几乎所有的 PC 软件大规模市场上都参与竞争。

MS-DOS

微软的第一个操作系统从 1981 年以来已历经 6 个版本，但其基本程序却很少有重大改变。1983 年推出的 DOS 2.0 主要是增加了对 IBMPC-XT 及其兼容机的硬盘驱动器的支持功能，还增加了原始的用户打印时的多任务功能。1984 年推出的 MS-DOS 3.0 支持 IBM PC-AT 及其兼容机上更先进的硬件，这些计算机都装有速度更快的 Intel286 微处理器。（在微软开始向 IBM 以外的公司出售操作系统以及 IBM 与数字研究公司推出他们自己的 DOS 系统以后，DOS 逐渐被人们称为 MS-DOS。）

1985 年的 3.1 版增加了网络功能。1988 年推出的 MS-DOS4.0 提供了一种为使用鼠标设计的简单图形界面或者说是“外壳”，但这没能吸引住多少买主。1991 年推出的 MS-DOS5.0 能够为应用软件存取更多内存，并提高了本来有限的多任务能力，使装入一个以上的软件（但不能同时运行）成为可能。这一升级版被证明是一个推出 Windows 的优秀平台，其销售量达几千万套。[28]

微软在 1993 年又推出一个版本，不过经理们也许会觉得还是不推出的好。MS-DOS6.0 主要是增加了一种称为 DoubleSpace 的数据压缩特性以及一些新的内存管理特性和诊断工具。这一产品一推出便遇到很多麻烦，包括一次迫使微软作出让步、更换其原先的数据压缩技术的专利侵权诉讼案件（StacElectronics 公司胜诉）。[29] 据《PC 杂志》对几千个用户的民意测验，MS-DOS 在产品支持和一些其他技术特性方面与 DR-DOS、IBM 的 OS/2 以及

DESQview 相比都“大大低于平均水平”。[30] 更严重的是，被调查者对 DoubleSpace 数据压缩特性的问题颇有微辞（见第 6 章中更详尽的讨论）。但是，微软通过几次小小的改进克服了这些困难。最新的 MS-DOS6.22 现在已成为一种稳定产品，销售量为几百万套，并且通常与 Windows 3.1 配套出售。

尽管批评家们喜欢指责 MS-DOS 技术上的不足之处,但它毕竟还是一件出色的产品。它在为整个产业充当平台技术的同时为微软带来了几十亿美元的收入。MS-DOS 在很低的价格上提供了较强的功能，而仅占用很少的内存和处理能力。这些优点使得用廉价的 PC 运行成千上万种应用软件成为可能。微软还通过不断扩充 MS-DOS 的功能而使其作为推出 Windows 的平台一直生存到 GUI 时代。

但 MS-DOS 作为单独的操作系统和 Windows 平台的时代很快就要结束了。微软从 80 年代中期开始便不再对其发展投入大量资源，而是着重发展 Windows 和 NT 系统。结果在某些领域，竞争对手在特性方面已经超过了 MS-DOS，尽管在市场份额方面他们还进展不大。因为它包含或模仿了运行老的 DOS 应用程序所需要的编码，所以 Windows95 没有和 MS-DOS 配套出售。假设微软人认为 MS-DOS 还有很大的市场，那么可能会有新版本的 MS-DOS 出现。不过，除了没有足够的计算能力来运行图形程序的少数用户以外，对大多数用户来说，Windows 95 从根本上已使作为 MS- DOS 一种独立产品的面临淘汰。

桌面应用软件

直到 1988 年 Excel2.0 面市，行业评论界才认为微软的电子表格设计技术堪与 Lotus 开发公司相媲美。但是 Excel2.0 如此完美地利用了 Windows 接口，从而保护了羽毛未丰的 Windows 产品并帮助它成为 PC 的新标准，这与 Lotus1-2-3 早年为 IBMPC 和 DOS 所做的极为相似。新 Excel 还使工业观察家们大加赞赏，甚至说出了“这一时期里程碑式的产品之一……一件杰出的艺术品”这样的溢美之词。[31] 与 Windows 的竞争者（Lotus1-2-3、QuattroPro 和 SuperCalc）相比，Excel 后来的版本有时只处于一般甚至低于平均水平，但在 Macintosh 上它们一直得分最高。[32]Excel 目前是最畅销的 Windows 和 Mac 电子表格。

Word 作为一种产品起步较晚。PCWord1.0（1983 年推出）和 Mac Word1.0（1984 年推出）尽管有很多革新特性，但仍销售平平，而且由于它们有太多错误，行业批评家们对它们也褒贬不一。Word 的早期版本还显然有点儿难以学习，尽管要比 WordStar 容易。[33] 不过，抑制销售量增长的最主要原因还是 WordPerfect 的迅速普及。WordPerfect 有许多优良特性，而且通过免费电话提供非常到家的消费者支持。至 1986 年 WordPerfect 已成为头号 PC 文字处理软件，占有大约 30%的市场份额——约为 Word 市场份额的 3 倍。

像电子表格一样，微软的第三次努力向成功迈进了一大步。1986 年为 PC 引入的 Word 3.0 更为精致，并配有联机指导，从而大大简化了程序学习过程。[34] 不过微软还是碰到了麻烦。如第 1 章所述，1987 年的一个 Macintosh 版本由于有几百处错误而迫使微软免费为消费者发布了一个改进版，这使得 1987 年版本成为一次商业和公共关系的灾难。微软还比预想时间多花了 4 年才完成了在 Windows 上使用的 Word 版本。然而，当 Windows 上的 Word 在 1990 年面世的时候——同时也有一个着意设计的用于 Macintosh 的新版本出台——微软超过了 WordPerfect，成为文字处理软件销售的市场主导。在对消费

者支持的满意程度和技术特性进行的民意测验中，这一版本有时落在 DOS 上的 WordPerfect 和 Windows 上的 Lotus 的 Ami Pro 的后面。但 Word 6.0（1993 年末推出）是继 Excel 之后第二大畅销的 Windows 应用软件，同时也是最畅销的 Mac 应用软件。不过，目前 Word（还有 Excel）销售量的一半以上是通过 Office 套装软件实现的。[35]

我们应当附带说明一下，尽管 Word 拥有约 65%的 Macintosh 文字处理软件市场（WordPerfect 约为 30%），MacWord6.0 却很难为用户接受。[36] 这是第一个几乎包含所有 Windows 版本“核心代码”的 Mac Word 版，只为适应 Macintosh 而作了少许改动。微软花了几年时间才将这一版本与 Excel 研制成功。但是，如我们在第 4 章中所讨论的，Excel 包含了一个特殊的功能层使其与 Macintosh 和 Windows 操作系统分离开来。由于 Word 没有这样一个功能层，微软只好搞一个庞大而运行缓慢的 Macintosh 版本。为了在新 Windows 版本出现以后尽快推出 Macintosh 版，微软还决定不要花专门时间使 Word 6.0 编码适应于 Mac 硬件。开发员最终不得不修改编码，为 Mac 消费者推出了一个特别的升级版。从长远来看，将 Word 的 Macintosh 版和 Windows 版合并——如同 Excel 所做的一样——对微软是一件好事情。这意味着微软只需建立并支持该产品的一个版本即可。不过，Word 6.0 事件反映了微软缺乏对 Macintosh 用户的关心而更偏爱 Windows 用户。如果不纠正的话，这一态度显然将对微软的 Macintosh 应用软件尤其是文字处理软件构成伤害。不过，尽管 Mac 用户传统上主导更大的市场，购买大量的软件，但这一市场比 Windows 用户市场要小得多。

Excel 和 Word 的新版本以及其他微软新应用软件（如 PowerPoint）包含了一些创新特性，这些特性被证明是对消费者具有吸引力的，帮助这些产品从某些竞争者那里赢回了市场份额。特别是微软已能够克服那些通常会导致软件消费者不能切换到其他产品上的技术和心理因素。例如，Excel 中含有“Lotus1—2—3 用户帮助”，Word 有“WordPerfect 用户帮助”。这些特性提供了一些指令，以对启动 Lotus 1—2—3 或 WordPerfect 功能的击键作出反应。Excel 和 Word 还能够从大量竞争产品中阅读和转换文件。另外，两种产品都加载了能简化并自动执行普通任务而且从某种程度上能预期用户要干什么的工具。Word 的 AutoCorrect 可以自动改正拼写错误和排印错误，将每一句话的首字母大写，当 I 单独出现时将其大写，或者在适当地方加入斜体字的引言和“警句”（令人讨厌的是，它还常常把 PC 变成 Pc）。用户还可以通过启动 AutoCorrect 来显示一个缩略语所代表的整个单词或短语，例如显示 NYT 所代表的 NewYorkTimes。AutoFormat 特性可以将文件按某一式样格式化，而“小能人”——即循序渐进帮助屏幕——能帮助用户使用详尽的指令完成复杂的工作。[37]

而现在 Office 正使单个的桌面应用软件遭到淘汰。这一产品与 Windows 操作系统和界面结合在一起，运用了诸如 OLE、动态链接库（DLL）等技术和其他设计方法。结果， Office（以及由 Novell-WordPerfect 和 Lotus 引入的类似套装软件）使得其各部分应用软件以一种仅仅几年前在 PC 上还不可思议的方式分享特性和数据。MS-DOS 计算机及应用软件现在看起来确实过时了。套装软件还为消费者升级到 Windows 水平以取代旧应用软件提供了一种廉价的方式。

Windows

我们说过，Windows 的早期版本销售很糟糕，使用起来也不顺手，尤其是在当时 PC 硬件有很多限制的条件下更是如此。为了使画面和运行速度达到消费者可接受的程度，Windows 需要增加一个彩色监视器，还需要增加比 80 年代末以前通常状况下更为强劲的处理器和更多内存。另外，在 80 年代中期几乎没有什么图形应用程序，编写 Windows 上的应用软件比在 DOS 标准下编写软件更为复杂，微软花了几年的时间才引入了 Visual Basic 这样的编程工具。其他公司的应用软件开发者们在学习 Windows 和为 Windows 编写软件方面也踌躇不前，因为他们拿不准这东西能不能以及什么时候能变为市场标准。[38]

像开发 Excel 和 Word 一样，微软也在这一产品上不断地下功夫。1987 年 10 月它推出了 Windows 第 2 版，是专为新英特尔 80386 微处理器所设计，并且是在康柏（其 PC 市场份额因其积极引入 386 计算机而剧增）的合作下开发完成的。新 Windows 对于许多评论家（及用户）来说运行仍嫌缓慢，使用起来也不灵活，但其显然是一种更为精致的程序。最重要的是，Windows 为易用图形应用软件开辟的新市场，终于出现了。微软通过编写专门配合新 Windows 程序的 Excel 而一路领先。Excel 促进了 Windows 的销售，并使微软与 Lotus 之间争夺电子表格市场主导者的竞争更加白热化。1990 年 5 月份推出的 Windows 3.0 在 5 周之内就卖出近 40 万套，当微软于 1992 年 4 月用更为精致的并排除了臭虫的 Windows 3.1 代替 Windows 3.0 时，Windows 已成为一种相当成功的产品。不出一年，每 10 台售出的个人计算机中就有 9 台是与这一版本配套售出的，与此同时，微软持续每月推出超过 200 万套。[39]

在早期进行的用户调查中，Windows 3.1 在消费者支持方面得分并不高，不过微软后来提供了更多的电话线、支持人员、训练项目、支持工具及其他措施来解决这一问题（参见第 6 章关于微软产品支持服务的讨论）。Windows 3.1 在用户对技术的满意程度和易用性方面的得分比 OS/2 高得多，它运行应用软件的速度通常也比 OS/2 快得多，尽管在其他一些项目上它比 OS/2 得分低。许多行业评论家们长期以来一直盼望着它还能有一些技术上的改进，包括更好的内存管理和保护、实现真正的多任务、改进程序和文件管理器等。[40] 不过，至少对于新 32 位应用程序来说，微软已经使这些问题在 Windows NT 和 Windows 95 中得到解决。这两种产品使 Windows 3.1 显得多少有点过时。

Windows NT

NT 的第一版使用户怨声载道。NT 要求有大量磁盘空间和计算机内存来安装和使用该程序。它在运行许多应用程序时特别缓慢，而且难于安装，在设备驱动程序和与其他应用程序的兼容性方面也有问题。对许多消费者来说，OS/2、NetWare、NeXTStep（UNIX 的一种为 Mach 设计的版本）和 UNIX 的其他版本更为可取，它们都经历了许多版本，多数错误已被排除。相比之下，NT3.0 是一种全新的产品——一种前途未卜的公司用操作系统——在这一领域中，微软还从未以严格的承诺与对手竞争过。[41]

NT 的一个改进版于 1994 年秋推出，序号为 3.5，它解决了许多老问题，对硬件能力要求降低，而增加了更多的特性。值得一提的是，3.5 版包含了一些设计上的变化，使其在运行 Windows 应用软件时更为可靠。它还改进了工作站图形，提高了网络能力。另外，它更充分地利用了 OLE，使各独立的

应用软件和对象能够通过一个标准接口而相互作用。[42]

在行业出版物上和微软内部又引起广泛争论的是下一种主要版本，在内部以代号“Cairo”相称，公司外则称之为 Windows NT4.0。[43]Cairo 不是一个严格管理下的项目，它更像是一项随意性的研究。然而这一项目却正在开发非常重要的技术。特别是 Cairo 利用了 OLE 的网络版并加进其他新特性，这将为 NT 把多个计算机联系成网络提供更广泛的能力。例如，Cairo 将使用户能够通过一系列联网计算机分享应用程序、存放或寻找数据及其他对象(这是在一个用户的计算机上除支持应用程序及数据或信息对象以外的附加功能）。Cairo 还包含一种新的文件管理技术，它使得文件不光可以按文件名存盘，而且还可以保存文件内容索引，使用户能够按不同标准寻找或整理这些文件（用户可以要求操作系统列出所有关于某一特定主题的文件，或者列出某一日期产生的所有文件。这对于那些创立了成千上万份单个文件的消费者以及那些人们需要分享大量信息并顺利存取信息的团体来说是极为有用的）接口构件也是 OLE 式的，这意味着用户可以随心所欲地增减——换句话说，可以随心所欲地改变计算机屏幕。行业观察家们预计 Cairo 可以在 1996 年或 1997 年成为商业产品，不过微软尚未正式宣布该产品或其出品日期。[44]

Windows 95

Windows 95 原来代号 Chicago，它使微软的大规模市场操作系统在性能和易用性方面可与已有 10 年历史的 Macintosh 势均力敌。这种新系统如同 Windows 3.1 一样，运行时对硬件没有特别的要求：4 到 8 兆字节的随机存储器（RAM）和一个快速的 386 处理器即可。与此相比，NT 或 IBM 的 OS/2 却至少要求有 12 至 16 兆字节的 RAM 和一个快速的 486 处理器（这对 Windows 95 当然更好）。Windows 95 吸收并加入了 Windows 3.1（及 3.0）上业已存在的先进的结构特性，从这个意义上说，它是一种技术上渐进的革新。但它还增加了新的设计概念和特性，其中有些来自于 Windows NT 和其他微软项目。

例如，像 NT 和 Windows 3.1 的加强版一样，微软在 Windows95 里也建有一个 32 位应用程序接口，称为 Win 32。（Win 32 是微软系统软件战略的核心。只要应用软件开发者们坚持这一标准，他们开发的任何应用软件就都可以在支持这一接口的 Windows 家族的任一种系统上运行）。但是，Windows 95 围绕一种“小核心”有一种分功能层的结构，这一点更像 NT 而不是 3.1。这一内核进行内存管理和多线程任务转换，用相互独立的分系统来完成显示器管理和文件输入输出等任务（参见第 4 章中对 Windows95 结构的更详尽说明）。

从性能方面来看，也许最具重要意义的是 Windows 95 像 NT 一样超过了 MS-DOS，至少是在新 32 位应用程序方面。这是很有用的，因为在直接运行 MS-DOS 时，操作系统将其控制功能让与单个 DOS 应用程序，每次只能运行一个应用程序。Windows 3.1 通过“合作执行多任务”避开了这一局限，操作系统可以交替执行各种任务和应用程序。在用户看来，这就似乎是计算机在同时执行多种任务和应用程序。但是，如果其中一个任务要耗费大量处理时间，那么共用时间就变得相当漫长，因为这种轮流时间对所打开的各种应用程序和任务是平均分配的。

Windows 95 像 NT 家族一样，对新 32 位 Windows 应用程序实行“多任务优先制”。这使操作系统能够决定哪些任务或程序消耗的处理能力最多，它

还可以停止一种应用程序来运行另外一种程序或任务。MS-DOS 和 32 位应用程序在 Windows 内如同在 NT 内一样拥有其自己的“被庇护”的空间。当计算机在运行多种应用程序时，如果一个程序遇到问题而停止执行，则其他程序通常可以不受影响。这是对 MS-DOS 和 Windows3.1 的一个重要改进，MS-DOS 和 Windows 3.1 只包含有一个处理道，更容易产生应用程序“崩溃”问题。

Windows 95 的另一长处是界面。尽管 Windows 3.1 使用起来比 MS-DOS 的任何版本都简便得多，但行业批评家们还是将屏幕说成是“一个满是图标、程序组、目录和菜单的莫名其妙的大杂烩”[45]。用户们还必须用两个独立的称为程序管理器和文件管理器的分程序来操作文件和应用程序。Windows 95 界面更为简练，在“集成”（微软所喜欢的一个词）方面也优于以往，而与 IBM 的 OS/2 Workplace Shell 及苹果公司的 Macintosh 相似。在 Windows95“桌面”（主机屏幕）上是一些带有许多图标的折页，这些图标分别代表一个应用程序或文件。屏幕下端是一个新的任务条（TaskBar），告诉用户哪些程序正在运行；它还可以使用户无需进入、退出、再进入单个的程序（像 Windows 3.1 所要求的那样）就可以转换任务或应用程序。Windows 95 还借用 Cairo 项目的技术使用户可以随意改变他们所面对的屏幕，并可通过结合不同程序的特性来自动完成应用程序。

Windows95 的有些特性含有全新的编码，需要大量结构性工作。特别是即插即用功能使得操作系统能够自行判断用户将何种硬件设备接进系统，从而自动启动适应该设备的软件驱动器（Macintosh 在几年前就有了类似功能，其操作系统中包含了苹果硬件设备驱动器）。另外——这一点又与 Macintosh 相似——Windows 95 能够容许长文件名，而不是像在 MS-DOS 和 Windows3.1 中那样最多只能有 8 个字符外加一个 3 字符的扩展名。它还有电传和电子邮件功能，以及微软早期在 Workgroups3.11 的 Windows 上以及 MicrosoftAt Work 上就存在的基本网络特性。它甚至还有一种按钮功能，可以简便地连通新上网的 Microsoft Network 而无需转换到一个单独的通信程序上去。

Windows95 从 1994 年 5 月以来已将其 β 测试版发布到 400000 个消费者测试点上。1995 年出品的目标已经比原计划后推了一年半；微软需要额外花时间以使 Windows 95 与现有应用软件更加兼容，还要完成和测试一些新特性（特别是即插即用和多任务功能）。[46] 这一延误对 Windows 开发组关于他们需要多长时间才能研制成一种复杂的新型操作系统的预测能力并未造成不良影响，即使他们原先的进度表有点过于雄心勃勃了。同时，微软也并不急于推出 Windows95。比尔·盖茨和其他经理们显然想避免 80 年代他们常常经历的以及目前英特尔奔腾机所面临的那种质量问题。苹果公司也不愿仓促推出新产品，它已将其一再推迟面世的 Macintosh System 7.5 操作系统升级版再次推迟整整一年，于 1996 年中推出。[47]

原则三：推动大批量销售，签订专有供货合同，以保证公司产品成为或继续成为行业标准

早在 1981 年，在一次公开会议的谈话中，比尔·盖茨就论述了大批量销售和标准之间的战略关系：

> 我们为什么需要标准？……只有通过大批量销售，你才能以合理的低价提供软件。标准提高了你能销售到[市场上]的计算机的水平……我也许不该这样说，但这确实以某种方式导致了一种个人产品达到一种自然的垄断：一些人合理地编制、改进、宣传某一特定产品，并通过努力、用户信赖、信誉、销售力量和价格为这一产品确立了非常牢固的地位。[48]

十几年之后，这段话成了对微软 1975 年以来所作所为的精确概括。微软定价、促销、授权和支持产品的方式都有力地促进了大批量销售。他们为产品提供大量优惠，前提是他们认为如果低价格能导致市场的指数扩展，那么对每件产品只取薄利要好得多。长期以来，这些策略与其他一些策略一起使微软在编程语言（BASIC 和 Visual Basic）、字符操作系统（MS-DOS）、PC 图形操作系统（Windows）和桌面套装软件（Office）等主要市场上取得了压倒性的优势。Windows 和 Office 几百万套的销售量也促使 OLE 成为应用软件开发者事实上的标准。

大批量销售对于维持这一循环过程和公司发展一直是至关重要的。首先来自 BASIC 和 MS-DOS，继而来自 Windows 和 Office 的大规模收入和利润，使得微软能够为其产品的不断改善和持续销售以及近期在 R&D 和其他新事业上的巨额投资提供资金。没有哪一个专门从事 PC 软件产业的公司在财力资源方面能与微软相匹敌（微软仅在 1995 财务年度就向 R&D 投入 8.3 亿美元资金，它还拥有 40 至 50 亿美元的现金[49]）。盖茨这样评论微软的战略优势：“这都是规模经济市场份额问题。当你每月推出 100 万套 Windows 软件的时候，你就可以每年拿出 3 亿美元改进产品而仍能使其以低价格出售。”[50] 副总裁史蒂夫·鲍尔莫同意这种说法：“软件业是固定成本产业。因此量绝对代表着一切……因为你可以将固定成本在广泛的范围内分摊。”[51]

BASIC 与 MS-DOS

微软最初于 1975 年通过低价向硬件公司出让许可权来促进大规模销售。这些许可权合同规定硬件公司有权将 BASIC 与其所销售的计算机集成出售。例如，根据最初的 Altair BASIC 定价条款，MITS 计算机公司每销售一套要付给微软一小笔专利权使用费，但总额不超过 18 万美元。MITS 还有权将 BASIC 许可权再转让给其他公司,这种情况下由 MITS 和微软分享专利权使用费。起初，MITS 出售 BASIC 给那些购买 AltairPC 的用户时只额外收费 75 美元，而对只想购买语言的用户收费 500 美元（Altair 计算机未配有 BASIC 或操作系统时售价约为 400 美元）。当时主机和微机语言的租金大约为每月 400 美元或更多，相比之下，MITS 的价格是相当便宜了。[52] 在其他有潜在的大批量销售的 OEM 情况下，盖茨同意以很低的固定收费标准达成交易，例如同苹果公司（用于 AppleII）和 RadioShack 公司（用于 TRS-80）的交易就是这样做成的。这样他就不必担心侵犯专利和收取专利权使用费带来的麻烦了。[53]

微软因将 Q-DOS 运用于 IBMPC 而从 IBM 获得了 200000 美元，又因提供 DOS、BASIC 和一些语言自动编译程序而获得不属专利权使用费的 500000 美元。与同意 MITS 再转让 BASIC 许可权的合同不同的是，这一新合同禁止 IBM 转让 DOS——但对微软却无限制。于是微软又将 MS-DOS 以低价卖给其他一些硬件商，从而使之成为 PC 产业（不光是 IBM）的标准。它最初对其他 OEM 消

费者收费 95000 美元，但之后按这一价格的一半将 MS-DOS 卖出，以使尽可能多的公司汇聚到这一标准下。微软对其专利权使用收费很低（通常 MS-DOS 和 BASIC 搭配），一般每套在 1 美元至 15 美元之间，依套数而定。[54] 零售价相当之高，以留出巨大的利润厚度给硬件制造商和零售店（在全国软件连锁店里 MS-DOS 6.22 版的最新售价为约 50 美元）。正如盖茨所说的：

> 在将 MS-DOS 转让给其他计算机制造商方面我们要限制 IBM 同我们竞争的能力，这是谈判的关键。我们希望保证只有我们有权对其发放许可权。我们以相当低的价格与他们达成这一交易，是希望这有助于它的普及。这样我们就可以采取我们自己的行动。因为我们坚持要让所有其他的厂商同我们站在一起。我们知道好的 IBM 产品通常是可兼容的，因此自然而然就可知道最终我们将能够把 DOS 卖给其他公司。我们知道，如果我们能靠 DOS 发大财，这将是从其他兼容机制造商那里而不是从 IBM 那里赚钱。他们为使用 DOS 而付给我们一笔固定费用。我们没有要专利使用费，尽管我们靠这一交易赚了一些钱。其他人支付了专利权使用费。因此这对我们总是有利的——市场在成长，其他硬件商也可以卖出一些……最终出现了 DOS 的兼容竞争者，也有人搞出具有竞争力的新操作系统。但是我们已经占据了主要地盘，所以我们可以继续以低价销售。[55] [着重号为著者所加]

各种非定价策略进一步降低了竞争产品对 MS-DOS 及后来的 Windows 构成威胁的可能性。例如，数字研究公司的 CP/M-86（80 年代早期开发）本来可以成为主导的 PC 操作系统；据报道，它有更好的内存管理特性，还有其他一些超乎 MS-DOS 之上的优势。但是微软是主要的语言生产者，它并不急于推出与 CP/M-86 兼容的语言。而当它推出这种兼容语言时，微软对其定价超出 DOS 兼容版 50%，因此销售量很小。微软还只销售一种低版本的 BASIC 而去掉了图解法。结果，应用程序开发者们发现很难为 MS-DOS 以外的软件编写程序，CP/M-86 作为一种竞争产品也就遭到了失败。[56]

微软通过运用其最有效也是最具争议的革新促销方式——按处理器转让许可权而使得来自 CP/M-86 后继产品 DR-DOS、IBM 的 OS/2 及 DOS 内部版 IBM PCDOS 的挑战一一被击退。1988 年，微软开始以特别低的专利使用费向硬件公司提供 MS-DOS，条件是它们要为所推出的每一台计算机支付一定价格，不管该硬件公司是否将 MS-DOS 与每一台计算机搭配出售。微软的开发主管戴夫·穆尔声称微软这样做的一个原因是简化计算和预测。微软只向硬件公司提供一个软件的一张母盘，后者可以简单地复制，而无需保存记录，也无需支付专利使用费。但微软可以随时了解它们推出了多少计算机（处理器），并且只要将英特尔和兼容微处理器公司计划出品的估计数加起来就可以预测销售量。不论这一做法背后的意图是什么，反正微软产品立刻变得如此畅销，以致硬件公司都不愿集成其他软件，因为它们一般都希望取得销售折扣。

在 DR-DOS 5.0 版和 6.0 版分别于 1990 年和 1991 年出现以后，微软在促成按处理器转让许可权的合同方面显得尤为积极。由 Novell 公司（它在 1991 年收购了数字研究公司）积极推销的 DR-DOS 实际上在 1992 年初就在零售方面超过了 MS-DOS。尽管 MS-DOS 的销售主要通过 OEM 硬件商，零售只占其销

售额的很小一部分，但微软还是迅速对这一挑战作出了反应。

首先，在 1990 年 4 月 DR-DOS 刚刚面市之时，微软就宣布要将 MS-DOS 升级到 5.0 版。尽管它直到 1991 年 6 月才推出这一产品，但它先前的声明可能已经降低了硬件商和其他消费者购买 DR-DOS 的积极性(行业批评家通常把这种为先发制人阻止别人竞争而老早就宣布将要推出的产品称为“朦胧件”，尽管这也是这一产业中让消费者预先知道下一代产品是什么的惯常做法）。其次，微软似乎向硬件商施加了更大的压力，以使它们接受按处理器转让许可权合同。第三，微软于 1992 年公开宣布 DR-DOS 可能与 Windows（当时 90%的新 PC 上都安装 Windows）不完全兼容，因为微软没有对它进行过测试（微软没有把 Novell 作为 Windows 3.1 的β测试点）。微软甚至附带了一份说明，警告那些准备在非微软的 DOS 版本之上使用 Windows 的用户，他们可能会遇到兼容性问题，尽管这种问题出现的机会实际上微乎其微。[57]

Windows 和 Windows95

微软同样使用价格优惠、积极促销和转让许可权的策略来推销 Windows。当 1985 年 Windows 的第一个版本问世时，微软把它的批发价定为每套 90 美元，但对其最大的消费者“每台处理器”只收费 8 美元，或每套 24 美元。至 1987 年春，微软声称已销售了 50 万套以上的 Windows1.0，但这是通过将 Windows 与 MS-DOS 搭配销售的；可能只有不到 20%的用户在其计算机上安装了这些操作系统。[58] 最近的 Windows3.1 版是按批发 35 美元、零售 95 美元的价格出售，而微软对 Windows 95 版的定价也与之接近。如果硬件商同意在其出品的 50%的 PC 上安装 Windows95 或者采用 Windows95 标志，并在一特定日期内签订合同的话，它们还可以将 OEM 价格降至 30 美元。[59]

与硬件商的长期合同中即使没有按处理器转让许可权的条件，也使竞争者很难引入竞争产品。例如，当 IBM 试图引进 Presentation Manager 以及后来的 OS/2 等 Windows 的后继产品时，微软与主要 PC 生产商签订的合同实际上就孤立了 IBM。[60] 微软还早早从硬件制造商那里争取到了使用 Windows 95 许可权的承诺。一位行业杂志记者在预测这种新产品（此处指 Windows4.0）在市场上遭到失败的可能性时说：“有钱能使鬼推磨。微软的秘密武器就是与世界上几乎每一个 PC 制造商签订合同。正是由于达成了这些交易，你可以肯定 Windows4.0 将预装在 1995 年所销售的 70%以上的 PC 上……你可以找微软去做任何事情，但它绝不必挨门挨户地去推销新 Windows 以使你的计算机软件升级。”[61] 尽管面市有些迟，但多数行业分析家仍预计 Windows95 头一年的销售量将至少达到 3000 万套。

同对待 MS-DOS 及 Windows 早期版本一样,微软也不会让这种主要的新产品的成功仅仅出于侥幸。它已宣布将为 Windows95 的促销活动投入 1 亿美元的资金。[62] 除此之外，它还另拿出 1 亿美元来为微软的名字及“Microsoft Home”等产品系列向一般消费者作宣传广告。不仅如此，如下所述，微软还制订了许可权使用条件,大大降低了应用软件开发者们为 Windows3.1 研制更多产品的积极性。这些措施将保证 Windows 95 及其后继产品会最终取代 Windows 3.1 而成为桌面操作系统的标准。

应用软件的促销和成套包装

MS-DOS 和 Windows 的普及为微软推销应用软件至少提供了一个间接

优势。公司与许多硬件商和软件零售店保持有密切的联系，成千上万的消费者一启动他们的计算机就能看到微软的名字。这些营销关系看起来有助于公司销售本来不受欢迎的或新研制的应用软件，如 Works、 PowerPoint、Mail、 Access 以及微软的多媒体产品。值得一提的是，微软将本来不为人所知的 Works 卖给 OEM 商，他们像以往集成 BASIC、 MS-DOS 和 Windows 一样把 Works 预装在他们的 PC 上，使之变成了一种最畅销的产品。

成千上万的用户还转而使用 Word 和 Excel——并尝试名声小得多的 PowerPoint 和 Acces 程序——因为微软已经把它们套装进（技术上也已集成进）Office 里了。消费者单买一套 Office 标准版本仅需要约 250 美元，而相比之下，分别单独购买这三种软件的话，每种的价格都是 300 美元或更多，因此购买 Office 可以享受到巨额优惠。几千万的 MS-DOS 用户只需做出一个简单（而又便宜）的购买决策，从一个他们都熟知的制造商那里购买一种 Office 最新版本，就可以使其计算机软件升级到 Windows 水平。将这些应用软件套装成 Office 还使得微软售出了更多的图形软件 PowerPoint，扩大了这种先前销量很低的产品的市场接受程度和使用范围。不过，对应用软件的销售来说，仅仅将应用软件进行套装、降低售价、宣传微软的名字以及积极向零售商推销还不够。产品必须接近市场主导者，或和市场主导者一样好，并且必须尽早进入市场。微软 Money 相对于 Quicken 销量很低，就证明了这一个事实。

海外市场

微软将其在美国这个世界上最早的,也是最大的 PC 市场上取得的大部分成就推广到海外。查尔斯·西蒙尼说出了这一经济逻辑：“我们一开始，或者至少从一开始研制应用软件，就意识到国际市场是要害。它要求产品的复杂程度提高 3—5%，但市场却扩大一倍，至少扩大一倍。”遵循这一理念，微软设计了 MS-DOS2.0，并于 1983 年发布，开始了最初的国际化。[63] 从此以后，它为其主要产品都设计了用于向国际市场销售的最新版本，并在全球建立了 36 个子公司。毫不意外的是，目前微软 40%以上的收入来自海外，主要是日本和欧洲。在这些市场上的竞争战略与在美国一样：建立联盟并运用多种定价和非价格策略订立长期合同，由此使这些公司将微软的软件与当地软件套装出售。

在日本，微软通过与 NEC 和其他日本 PC 制造商建立联系而在 PC 操作系统市场上确立了其早期的地位。微软先是从 1983 年引入的 8 位操作系统 MSX 开始。之后，它推出了 BASIC 的日语版，用于日本 PC 销售市场的领头羊 NEC 所制造的计算机以及 IBM 兼容机上。微软在 1983 年开始发布一种 MS-DOS 的日语版，不过这种版本只有在 1987 年以后随着 16 位机的销售上升才开始受到欢迎。[64] 微软现在已经研制出了 Windows、WindowsNT 及所有其他主要应用软件的日语版。微软日本分公司（桌面应用软件部的分部）还负责将主要的应用软件翻译成汉语、韩国语以及日语。

在欧洲，微软于 1982 年在英国建立了一个销售公司，随即进入法国、德国和其他国家。苹果公司当时已占有欧洲市场的 50%，但微软说服一些欧洲 PC 制造商将 MS-DOS 和 BASIC 作为其 PC 的预装软件产品推出。微软还迅速将 BASIC 使用手册翻译成欧洲语言，并生产了 Multiplan 的欧洲语言版，这促进了其在欧洲的事业。1983—1984 年以后 IBM PC 在欧洲的销售也大大促进

了欧洲 PC 公司采用 MS-DOS 作为其操作系统标准。[65]

反垄断问题

在微软成功的上面笼罩着一个巨大的阴影，那就是对潜在的反垄断打击的担心。1990 年 6 月美国联邦贸易委员会（FTC）开始对微软的做法进行质询；在 Novell 公司的坚持下，欧洲联盟也实施了一个类似的调查。尽管 FTC 于 1993 年 8 月停止了质询，但美国司法部继续进行调查，最终与微软于 1994 年 7 月达成了一个双方同意的协定。与此同时，微软也同欧洲联盟达成了一致。

然而在 1995 年 2 月，一位美国联邦法官史无前例地拒绝批准司法部的处理办法。他做出这一决定的理由是该协定范围过窄，只适用于将来的情况，而对于纠正微软目前对操作系统行业的控制却未采取任何措施。尤其是协定没有处理微软提前宣布新产品（“ 朦胧件 ”）的产业惯常做法，该法官担心这会阻碍竞争产品的销售。协定也没有对微软 “ 将其在操作系统市场上近乎垄断的地位的影响带入更具竞争性的应用软件领域 ” 的能力作出判断与规定。[66] 司法部只在相对有限的条件上达成协议，是因为在法庭上很难对广泛的反竞争行为作出确认。微软和司法部对此法庭判决提出上诉并最终胜诉。即使在这一裁决作出以前，微软就一直在美国和欧洲遵守着原先的协定（欧洲协定不需要法庭认可）。[67]

1994 年 7 月达成的协定包含三个主要方面。第一，微软同意，如果许可权使用者按推出的处理器或计算机付费而不是按推出的软件数付费，则微软停止给予大量折扣优惠。这类合同占微软操作系统销售量的大约 60%。第二，微软同意放弃让计算机厂商提前几年就承诺购买一定量软件的长期合同。第三，微软同意停止其与独立应用软件开发者之间订立的严格的非公开协议（微软一直要求其 β 测试者和其他公司在其操作系统投放市场后三年内不许公开该系统的详细情况。这限制了其他公司应用软件开发员的活动和动力，而微软却可以自由地随处招集开发者，来开发微软销售的应用软件）。在这些协议条件中，微软还再次保证不对竞争对手或者消费者隐瞒新产品信息，不过它们仍保留不转让或不公开其源代码的权利。

即使这些相对温和的限制也促进了操作系统市场的竞争，而对微软操作系统的销售造成了一些损害，因为他们为硬件商购买竞争产品敞开了大门。比如在德国这种事情已经发生了。欧洲最大的 PC 卖主 Vobis 微型计算机公司从 1994 年秋天开始将 OS/2 和 Windows 而不是 MS-DOS 与 PC 一起销售。结果，大约 20%的欧洲 PC 都装有 OS/2（这一数字是美国的两倍）；在德国市场上这一数字是大约 40%。由于这些计算机装有 OS/2 同时也装有 Windows，因此尚不清楚人们实际使用哪种操作系统。其他 PC 卖主都没有仿效 Vobis，而且 Vobis 也计划在 Windows 95 出品后在 PC 上安装 Windows 95，但微软目前在将新产品确立为事实上的标准方面面临的任务难度至少比以前增加了。[68]（Vobis 还与微软谈判，对其按处理器购买 Windows 3.1 的许可权方面作了一点变化。例如在 1995 年 3 月，它只按在其 PC 上安装 Windows 3.1 的套数付费。[69]

不过，我们仍然认为协议的条件对微软的全球市场份额和财务收入可能不会产生太大的影响。几乎所有的硬件商仍然将 MS-DOS 和 Windows 装在他们的机器上，尽管他们也安装 OS/2。微软产品有一个巨大的既有用户基础，

因此已经成为事实上的行业标准。不仅如此，微软仍可以使用销售折扣以及排他性的许可权转让条件来推进 Windows95 及其未来版本与其他产品的销售。使用竞争产品的用户将不得不为失去与成千上万现有 DOS 和 Windows 应用软件接近的潜在机会而担心（即使 Windows NT 和 Windows95 在运行所有者应用软件方面也不是完美无缺）。出于这些原因，不管有什么样的反垄断协定、有什么样的条件，总归只有少数硬件商愿意放弃在其 PC 上安装 Windows 95 或其未来版本而冒风险。[70]

但与此同时，由于政府反垄断措施的原因，微软也正在改变它的行动。特别是在美国司法部上诉要求阻止微软兼并 Intuit 公司之后，公司老总们于 1995 年 5 月决定放弃兼并 Intuit 的努力（微软最先于 1994 年 10 月宣布要实行兼并）。[71]

原则四：充分发挥作为新产品和关联产品标准供应商的优势

定价、促销以及促成大批量销售的销售协议，使微软确立了其公司的地位，也使它的一些产品成为工业标准。但为维持和扩张其市场地位或者为向新市场销售新产品，一个工业标准提供者还要争取利用其对产品“结构”的影响力。做为一个工业标准提供者还具有超乎协作者网络之上的优势，这一协作者网络遵循某一特定结构，比如决定产品如何配合 MS-DOS 和 Windows 运行的技术标准。对微软来说，这一网络包括应用软件的制造者，以及软件开发工具、硬件外围设备和驱动器（如打印机和监视器）以及为微软操作系统设计的其他产品的生产者。革新专栏作家迈克尔·施拉格在 1994 年的一段话中精辟地论述了比尔·盖茨和微软作为工业标准提供者的目标和成绩：

> 谁比其他人更理解标准的意义？可能是比尔·盖茨以及微软。微软实际上不是在做软件生意，而是在做标准的生意。微软成功不是因为它编写了最好的程序，而是因为它确立了最好的标准。微软 Windows——使盖茨成为亿万富翁的这种个人计算机软件——被培育和发展成一种标准，而不仅仅是另一种操作系统。微软的目标绝非收入甚至市场份额的最大化；它是在与消费者、软件开发商和英特尔这样的微处理器生产商建立关系，以给予微软操作系统最充分的支持——战略上、财务上和技术上的支持。这些关系网络正是使标准成为标准而不是一件产品的东西。标准不是一个公司的产品，而是这些网络的副产品。支配标准意味着支配这些网络。[72]

提供标准

对一台个人计算机来说其硬件和软件结构标准有四个主要因素：微处理器、操作系统、数据控制器（称为“总线”）和视频系统。[73] 这些要素决定了在操作系统和应用程序或网络通信软件之间、操作系统和硬件逻辑和内存芯片之间以及外围设备驱动器（用来运行打印机、监视器、键盘、鼠标指示设备等的软件）和操作系统和应用程序之间发出和接收数据与命令的接口。因为用户需要向后的兼容性，而目前没有哪一个公司能够像 PC 产业早期时那样主观地改变这些要素。例如，微软就不能改变 Windows 的与现有应用软件保持兼容性的要素，这些应用软件包括 DOS 应用软件和那些为 Windows 3.0 和 3.1

编写的应用软件。高级副总裁布兰德·斯尔文伯格（原为 Borland 公司和苹果公司的开发员）曾指出这是他在微软的小组没有及早改进 Windows 的主要原因之一：

> [Windows]3.0 非常大、非常慢；3.1 作了大量改进……[但是]在某些地方你也不能破坏其兼容性，这就是接口。它们当中有一些通过应用产品定义了 API。在某种情况下，如果我们能够重新研制它们，我们会知道怎么做，我们会编写出更快的系统。可是一旦你已经有了那些接口，你就基本上被锁定了。你就不能改变、拆毁这些应用软件。像我们现在的系统，我们并不拥有它； ISV[独立软件商] 拥有它。我们[Windows/MS-DOS 小组]只为一个目的而存在，就是运行这些应用软件。而[如果]你破坏一个软件，你就没有理由继续存在了。

不过，硬件和软件关键技术的提供者——英特尔和微软——都对其 PC 微处理器和 PC 操作系统的既成标准的演进施加了巨大的影响。微软尤其善于利用它的地位：它不仅生产 OLE 这样的关键性启动技术，还生产辅助产品（应用产品）。因此它能够而且正在创造技术与营销联系，以支持其每种主要软件产品的销售（见表 3.1）。

辅助产品

Windows 是复杂的，但它是几乎所有软件应用程序编写人员（以及打印机及其他外围设备制造商）都支持的环境。微软发布了大量有关其新产品的详细信息，甚至在产品进行商业销售之前，在行业会议上以及借助于向消费者发布β版、向应用程序编程人员发布软件开发工具的机会向外公布。微软人还回答来自任何公司的应用软件开发者的问题。

但是微软设计了 MS-DOS 以及 Windows，它的开发者们比其他任何人都更清楚这些系统的复杂性和特异性。从 1988 年至 1995 年，系统软件组和应用软件组也同时向同一位总裁麦克·梅普尔斯报告工作。他和比尔·盖茨都公开地鼓励人们分享技术知识的做法（盖茨还在 1995 年 3 月份《华尔街时报》刊登的一次专访中指出在应用软件组和系统组之间“没有一道不可逾越的鸿沟”。他说：“我们对这两个方向的投入都不会停止。”[74]）。让行业标准的设计人员们一同走过工作厅或在一张午餐桌上吃饭，或在隔壁办公，或安排在同一开发组，这些做法都有不可替代的好处。

有些行业观察家称，微软的应用软件编程人员因采用 Windows 以及 MS-DOS 中“未公开的系统调用”或低水平应用程序接口（API）而具有一种特别不公平的优势。API 是一些激活操作系统功能的例行程序；它们对于编写应用程序来说是必要的。问题是微软的开发员可能有办法进行这种调用，并可能在编写应用软件是使用这些调用。[75] 在某些情况下可能真是如此，尽管这些调用对应用软件开发的重要性有多大尚不清楚。戴夫·穆尔对我们说，许多未公开调用的存在只是因为开发员已经决定不使用它们了。他还解释说，它们存在的目的是操作系统开发员可以借以清除内部臭虫或检测各种功能，而非借以进行软件编程。

但是，即使不考虑 API 的问题，微软还是充分地利用了其作为 Windows 提供者的地位。除了采用特别的标准和政策，它还做出了特别的投资和承诺，使其应用软件和系统产品相互加强了地位。例如，微软很早就许诺把 Windows

作为它们新的标准环境，并迅速审定了微软主要应用产品的图形版。这些产品在电子表格和文字处理软件市场上对 Lotus 和 WordPerfect 提出了严峻的挑战。在 Windows 最初出现的时候这些竞争者还一直是不容置疑的市场领导者，但它们又都不愿采用 Windows，因而没能及时引进其产品的图形版。它们的 DOS 版与 Windows 兼容，但不像后者那样易于使用，功能也没有那么强劲。盖茨在回忆起这一点给微软带来的优势时说：“当 Windows 出来的时候，我们的应用软件竞争者迟疑了。它们花了很长时间才推出它们的第一个 Windows 版，许多人可能便因此满足了。只是最近它们才推出或准备推出一个不错的 Windows 版。而我们在这段时间里却继续前进，继续加强我们的领导地位。”[76]

微软还设计了 Office 套装软件和 Microsoft Network 以有效地配合 Windows 95 和 WindowsNT，它还有充分的意图研制更多的应用软件，以充分利用这些操作系统上的特性。盖茨在 1993 年秋公开地表达了这种意图：“我们在通过一个代号为 Chicago[Windows95]的项目加强桌面 Windows 时，充分利用了 Windows 上的所有用户界面的能力。我们要首先利用它……我们发现有大量的机会将我们的应用产品集成到一个外壳中，不光是 Mail，而且包括所有群体功能、文件搜索功能。Chicago 将带来用户界面的进步，从而使应用软件全面升级。Windows 的未来版本也是如此。我们保证我们的应用软件要充分地利用这一点。”[77]

关键的启动性技术

对象的链接与嵌入（OLE）技术极好地说明了微软如何利用其作为标准提供者的地位。它创造了一种关键的启动性技术并把它与一种操作系统集成在一起。微软在行业中利用 OLE 从事应用软件和操作系统开发方面也是领先的。OLE 也是一种复杂的、仍在演进的技术；不过，微软已经使它成为 Windows 应用软件编程环境的一个关键因素。[78] 像 MS-DOS 及 Windows 一样，微软开发员也是 OLE 的提供者，他们比其他任何人都更理解其复杂性和特异性。只要他们能维持与老产品之间的兼容性，他们在指导 OLE 的演进和率先利用技术中新的能力方面就处于非常有利的位置。这方面最新的例子是尚在开发中的对 OLE 版的网络化。微软计划将其运用到即将出现的 Windows NT 升级版 Cairo 上。

微软为微软 C++库基础类的外部开发者研制了 OLE 的一个早期版本。在 1993 年向公众发布 OLE 2.0 之前，它组织了多次规格检查以及一次 OLE 开发员会议（由 1000 人参加）。另外，微软出版了好几册关于 OLE 技术信息的详细说明[79]，并继续组织开发员开会讨论该技术的未来发展。[80] 不过，毫不奇怪的是，微软作为该标准的提供者首先使用了这一技术，在 1990 年和 1991 年分别将 OLE1.0 结合进 PowerPoint 和 Excel。这在某种程度上说只是一种早期的实验，微软内部对于向 OLE 投资多少意见不一，因为目前的技术还有许多技术方面的局限。总之，直到微软将 OLE 2.0 作为 Windows API 的一个扩展部分，并将其作为新的 Windows 软件开发工具时，外部开发员才广泛运用这一技术。

OLE 2.0 特别重要，因为它能够使软件开发员创造出能以非常有用的方式操作或分享对象和信息的产品。例如，“OLE 启动的”应用软件的用户可以在他们首先打开的原始文件内对不同对象进行编辑（例如编辑一块文本，

一张填充了数据的电子表格或者一幅画）。“主”应用软件的菜单和工具换成了编辑对象的菜单和工具，但编辑场合还是在主文件内。另外，OLE 2.0使用户可以将OLE对象从一个软件“拖放”到另一个软件上。例如，用户可以从 Excel 或 Lotus 1-2-3 中取出一份表格数据并将其插入到在 Word 或WordPerfect 下编写的一份报告中，于是他们可以在文字处理软件下编辑电子表格，而仍使用电子表格的特性。

也许最重要的是，OLE 2.0 对象可以在应用软件之间传送数据和信息。这使得对象可编程，并允许应用软件开发员创造出一套能分享功能（如文本处理或画图）的集成程序。其结果是，应用程序接口和功能对用户来说更为一致，并占用较少的磁盘空间，因为软件生产者无需将同样的特性复制到不同软件上去（例如，Excel 与 Word 都有文本处理和打印功能，它们可以在Office 套装软件内分享这些特性及其他一些特性。参见第 6 章关于 OLE 和Office的讨论。）

OLE 2.0已成为Windows应用软件开发者的标准技术。这不仅是因为其技术上的优点，还因为微软对它的采用和公开，以及将其纳入Windows软件开发工具，还有它目前的可用性（相比之下，苹果公司和IBM等提出的分享对象的竞争标准还仅仅是提出）。它的外部用户中有 Lotus，其产品是 Lotus Smartsuit 以及 LotusNotes（主要依靠 OLE 2.0 产生分散的数据库对象）。 Novell在其5种应用软件中也使用了 OLE 2.0，包括WordPerfect Office 电子邮件、进度表设计产品以及其拳头产品——用于 Windows 的WordPerfect 文字处理软件。为使OLE更大程度上成为行业标准，微软正在开发Macintosh版本，并已与DEC联合起来，争取定义下一代的“普通对象模式”。这种模式包含了公开的对象处理组和公开的软件基础中各种对象的技术特性。

看起来，微软还是 OLE 最热心、最老练的使用者。应用软件的 Visual Basic（包含于 Excel）利用 OLE使用户通过混合和搭配不同功能对象自动完成任务和定做应用程序。Windows 95与Cairo也广泛地运用了 OLE。例如，Cairo将使用户能够通过混合和搭配局部对象（该用户计算机上的对象）与远程对象（联网的其他人计算机上的对象）创造文件和应用程序。类似地，它还可以处理多媒体对象。同样，作为NT与Windows 95操作系统以及OLE技术的提供者，微软在对未来版本的承诺、技术理解和技术影响方面看来都占有优势。即使它要与其他应用软件开发者分享所有基本特性说明和大批文档，这种优势仍然存在。[81] 不过，公正地说，人们也可以把 OLE 和 Visual Basic 看成是微软贡献给软件产业的极为有用的标准和技术。除了微软，成千上万的应用软件开发者正在有效地使用这些工具来创造商业软件产品和定制应用软件。

产品关联

也许最能说明微软保持其标准提供者地位的潜在能力的例子是 Windows 95。正如我们后面要详细说明的，Windows 95将使用户能够从操作系统内直接进入新的 Microsoft Network。这样微软就能够为其消费者提供大量联机产品和服务。另外，微软已经利用了 Windows 95预期的名望来促进Windows NT的销售，同时也使 IBM的 OS/2今后难于竞争。

最初，微软由于前面提到的各种原因在销售NT 3.0方面遇到了麻烦。特

别是由于 Windows 3.1 一直是事实上的大规模市场标准，并且只能运行 16 位应用软件，微软不能说服许多硬件商或消费者在其 PC 上安装 NT（NT 运行 16 位 Windows 应用软件非常缓慢，它需要特别定做的 32 位应用软件才能充分利用其先进能力）。反过来，应用软件开发者又不愿编写 32 位应用软件，因为一开始 NT 销售状况不佳。

为解决这个鸡生蛋还是蛋生鸡的问题，微软在自己的开发小组内和外部公司中都大力推动 Windows 95 应用软件的开发以支持 NT。NT 与 Windows 95 都有基于 Win32 的 32 位结构，因此它们有相同的应用软件兼容标准。这样，如果应用软件开发者为 Windows 95 编写了什么程序，他们也就自动为 NT 编写了程序。这些新产品将有助于微软向那些需要 Windows 95 上所没有的特性和网络能力的消费者推销更多的 NT。为达到这一双重目标（保证 Windows95 的成功，并促进 Windows NT 的销售），微软使用了 Windows 兼容性杠杆和 Windows 标识。同时，减小应用软件开发者为 Windows 3.1 编写软件的动力将有损于 IBM。因为 OS/2 能够运行为 Windows 3.1 软件，却不能运行为 Windows95 编写的软件，尽管 OS/2 和 Windows95 都是 32 位操作系统。

在一个公司的应用软件产品上加上“微软 Windows”标识，证明这种产品与 Windows 3.1 兼容。大约 75%为 3.1 版本编写应用软件的公司为其包装申请了使用该标识。但在 1994 年底，微软宣布只有同时与 Windows 95 和 Windows NT 3.5 完全兼容的应用软件产品才能使用 Windows 标识。之后微软又采用新的许可条件，要求应用软件开发者将其产品升级到 32 位结构、采用 Windows 95 用户界面、证明其与微软即插即用标准具有兼容性并修改应用程序以便可以处理长文件名。1995 年 2 月，微软在标识问题上作了部分让步，允许为 Windows 3.1 编写程序的应用软件制造商在一段时间内使用“旧”Windows 标识。但如果要使用新的“Windows 95”标识，则这些公司必须达到新的兼容性要求。微软还建立了一个由 VeriTest 掌管的独立测试机构，以检查应用软件生产商是否达到了其结构标准。[82]

因为应用软件开发商大多想让其产品在 Windows 最新版本上运行，因此它们只能为 Windows 95 和 Windows NT 编写软件而很少有其他选择余地。另外，WordPerfect（现为 Novell 的一部分）和 Lotus 这些公司在 Windows 3.1 出现时没能及时为其开发产品，所以其市场份额迅速下降。因此现在它们在开发与 Windows95 兼容的产品方面大概不会甘愿再次落后。实际上，它们决定为 Windows 95 设计 32 位版本只是为了与微软保持同步，后者在 1994 年 11 月已开始推出 32 位 Word 和 Excel 产品。[83]

原则五：整合、拓宽并简化产品以进入新的大规模市场

在 90 年代，微软为开发将原本分离的功能集成为一体的、更便于消费者使用的产品而投入了巨大的资源。这些新产品可以以低廉的价格通过简便的运货机制——一个箱子，或通过电子网络打一个电话——到达数量不断增加的潜在的消费者手中。比如 Office 套装软件就是把一度分散于 Word、Excel 和 PowerPoint 上的功能包装在一起，而价格只相当于几年前其中一种产品的价格。Windows 3.1 和 MS-DOS 以及现在的 Windows 95 是把用户原先需要单独购买的许多特性结合在一个价格很低的产品中，其中包括屏幕保护程序、终端仿真器、时钟、计算器、日历和工作进度表、数据压缩程序、反病

毒工具、诊断工具、电子邮件、传真、网络软件和其他特性。这种包装还更多地在其他新产品中实施，例如运用于初级家庭消费者使用的产品中。

这不是偶然的。微软正在办公室设备网络和多媒体产品等领域积累起丰富的专门经验。它还在试验如何将一度分散于操作系统、应用程序和网络通信软件的功能兼并起来。这些努力与其易用性研究和实验一起，将微软与计算机初学者及普通家庭消费者之间的距离拉得更近。潜在的战略意图很简单：如果使用计算机足够简单、购买软件足够容易，那么就会有数十亿的潜在的新的消费者，就会有无数消费者愿意为之出钱的潜在的应用产品、服务和交易。

固然，微软不是第一个在这些新领域提供产品或进行研究的公司。许多实力雄厚的大公司也已经或正在进入诸如办公室网络、消费者软件产品及信息高速公路服务等领域。然而，微软又是相对早地进入了——当然是在大规模市场真正到来之前。而且这一次，它是带着几千万既有的 Windows 消费者、几十亿用于 R&D 和兼并的美元、一个雄心勃勃而又富于变化的战略以及数以千计的天才的科学家、软件开发员、程序经理、软件测试员和营销专家进入这一新市场的。

在工作场所

Windows 操作系统有可缩放能力；理论上，程序的较小版本可以在从微型手提式设备到通过 Windows NT 服务器等产品联网的 PC 机等各种计算机上运行。但在实践中，这种缩放效果却难以达到。不过从 1992 年前后开始，微软就一直在为通过 Windows 用一个微处理器将所有设备都联系起来的工作奠定基础。在将办公设备进行联网方面它已取得了最大的进展。这是一个施乐和 IBM 都试验过但未取得成功的领域，但它仍意味着在将 Windows 技术推广到另一大群用户方面存在着巨大的潜在机会。

主要的一项工作原名叫 Microsoft At Work，它是旨在连结电话、电传机、打印机、复印机和手提式设备并进而通过 WindowsPC 对它们实施控制的一系列网络和接口技术的总称。目前设在微软个人操作系统部的一个独立小组已经建立了一套用于电传机的系统，微软在 1995 年 1 月开始销售这种系统。尽管这一努力的成果不像微软经理们预想的那样好，但已有 70 多个主要的办公室设备生产商（包括康柏、AT&T、爱立信、英特尔、摩托罗拉、飞利浦、夏普、东芝、惠普、NEC、北方电信、理光和施乐）签约要求使用 Microsoft At Work。Windows 95 也结合了这一标准。[84]

另一个正在成长的工作场所市场是“群件”，开拓者是 Lotus 的 Notes。它通过电话线或直接配线将各种 PC 连在一起，使它们能够分享相同的信息数据库，还能自动更新和复制用户已经改变了的文件。Notes 还提供了先进的电子邮件能力，既能处理文本又能处理图形。Notes 有约 140 万用户以及一个新的大所有者 IBM。但是联网计算机的数目—— 1994 年是 3 000 万——为竞争者的进入留下了大量空间和时间。

微软的战略是拓展其电子邮件产品，吸收一些类似于 Notes 的特性，同时在 NT 操作系统和 Windows 95 中加入更好的网络与通信功能。为建造和集成这些产品，微软在其商业系统部有 250 名雇员在为一个称为“Microsoft Exchange”的系统进行开发工作。与更有整体性、更昂贵的 Notes 产品不同，Exchange 采取了一种模块方法。消费者可以逐渐建立起群件能力，根据自己

的需要一个一个地加载诸如电子邮件、电子格式形成、会议日程表等等功能。如果消费者愿意一次性获得这些能力，微软会以优惠价提供给他们一个软件包。微软于 1995 年 2 月发布了一个 Exchange 的β版本，成品版本应该会在 1995 年底或 1996 年初面世。[85]

公司还采取积极姿态，提高其在更加先进的公司软件系统方面的市场份额。另一种称为 BackOffice 的产品将 Windows NT 服务器、微软有名的用于数据库系统管理的 SQL（结构化查询语言）服务器、系统网络结构服务器（一种连结新旧软件系统的产品）、系统管理服务器（一种通过网络等方式远距离安装软件的产品）和 Mail 包装在一起。消费者还可以将 BackOffice 与 Exchange 和 Office 结合在一起以获取群件能力。[86]

在家里

比办公室更有开发潜力的是家庭，这里有几乎无限多的 PC 或更小的设备的潜在使用者。桑贾伊·帕塔萨拉蒂这位高级消费者系统部的组产品经理解释了微软如何拓展出一种从一般消费者获取收益的新（对微软来说是新的）方法。这一想法就是提供产品和服务，然后按每次使用或交易收费，而不是依照一次性的软件销售收费：

> 我掌管一个商业开发小组，大约有 8 个人，他们和我一起工作以决定微软在交互式数字网络方面的所作所为。我们把网络看成是一种与电报、电话、无线电、电力等一样不同的东西……我们正在使公司从对包装好的产品预收货款的时代进入每年有一定收入流的时代，至少在这个你为你的软件收取租金的新领域是这样。你把产品发布出去，然后不论软件被用来干什么，你都按每次使用情况收费……一般一个家庭怎么也得使用 350 至 500 次……我们在生活中的角色就是将计算机传给大众——那些从来没有用过计算机、从来不想用计算机的大众。你确实要采取一种非常不同的方法接近他们：不光以产品的外貌，而且要以产品给人的感觉及其收钱的方式来接近他们。

微软在 80 年代初开始进入消费者软件市场，当时它开始试验录像游戏机和其他多媒体技术。它也是第一个进入多媒体出版物市场的大型 PC 软件公司。微软的第一个产品是 1987 年出品的用于 CD-ROM 的全文本 Microsoft Bookshelf，这是一个参考书目集成。 1988 年，微软成立了一个多媒体系统组，开始开发新产品和一种称为多媒体 PC（“MPC”）的技术标准。这种标准能使所有 MPC 计算机运行所有 MPC CD-ROM 盘。微软还组织了一个国际企业联盟，包括 AT&T、CompuAdd、NEC、好利获得、Tandy、Zenith 和飞利浦等公司，以继续支持并改进这种标准。[87]

多媒体系统组已经演变成了现在的微软消费者部，它在 1994 年的销售额为 3 亿美元左右。按照微软的标准，这一数字非常之小，但这是公司增长最快的部分。该部拥有微软 Home 的商标，每周推出一种产品，从个人财务软件到主要分布于 CD-ROM 激光盘上但很快进入联网的 Microsoft Network 的多媒体产品，无所不包。微软目前有 50 种多媒体产品，在此类资料的出版者中列第 4 位。销量最高的是不足 100 美元的 Encarta 百科全书；其次是 Cinemania，它是对 19000 部影片的回顾指南，售价 50 美元；再次是著名艺术作品集 Art

Gallery。Complete Baseball 是另一种受人欢迎的新产品，提供了棒球运动员和棒球队的最新统计数字。[88]CD-ROM 多媒体出版物在 1994 年的销售额仅为 3.000 万美元左右，但微软显然正在培育能处理声像的有用的家用产品技术，比如交互式电视和联机信息数据库技术。

作为将微软桌面应用软件、设备网络技术、多媒体产品和其他新产品和服务带入家庭市场的基地，微软消费者部还一直在开发一种称为“社会界面”（原代号为“Utopia”）的技术。它为用户提供了一个起居室式的立体环境，而不是 Windows 3.1 中的各种组织的松散图标，也不是 Windows 95 中那种桌面环境。它还有一些出现在屏幕上的卡通式人物（由用户选择）——“智能代理人”。这些智能代理人观察并适应不同用户的需要，并发出带有声音的指令引导用户完成各种任务。它们在很多方面类似于微软在 Word 和 Excel 等产品中运用的自动帮助和“小能人”，不过现在它们有了笑脸、个性和声音。

这种新的社会界面第一次运用在一种称为“Bob”的 Windows 产品中，这种产品从 1995 年 3 月开始以不到 100 美元的价格出售。Bob 将 8 种预备性应用程序集成在一起，其中包括书信帮助、支票簿、通讯录、日程表、财务指导和电子邮件。在一次发布前版本的检查中人们发现该产品是令人失望的：这些应用程序特性有限，Bob“喋喋不休”的建议和其他指导可能使一些人感到厌烦，另外需要的计算机内存量（8 兆）巨大。但就其对智能代理人和图标的使用来说，即使批评者们也承认这种“社会界面”在易用性和初级计算机用户易学性方面似乎超过了 Macintosh。(89)在这种界面的其他版本中，VCR 和电视图标及其他标志都在屏幕上出现，在“智能”人的帮助下比较易于辨认并通过 PC 编程。这些将有助于微软把 Microsoft At Work 风格的技术推广到家庭。

微软还正在开发一种能控制诸如加热和空调系统等家用设备的操作系统。这需要一种新的“袖珍 PC”或普通 Windows PC 作为控制设备。袖珍 PC 还可以通过无线电网络处理银行交易和其他存取信息的任务。另外，由微软与英特尔及 General Instrument 公司开发的用于有线电视预置盒的新软件将把交互式数字电视服务输送到现有的电视机上。(90)在另外一个项目（称为“Tiger”）中，微软研究部正在开发一种在 Windows NT 上运行的网络服务器系统。这将使有线电视或电话公司能够提供点播式（这使用户能够在他们希望的任何时间观看电影）和交互式电视。微软发现建立这种新软件并非易事。但是，它已雇用了一批该技术的专家，与日本索尼和 NTT 等公司达成了协议，并计划在 1995 年与北美、欧洲、日本和澳大利亚的主要有线电视台和电信公司达成协议。[91]

另一种重要的家用产品是用来记录预算、支票簿、税收以及帐单并取得互助基金或股票帐户等投资资料的个人财务软件。Quicken 在这一类产品中领先，但微软在与银行及其他像 Visa 国际集团这样的公司签订协议以促进电子支付方面做得更为成功。微软 Money 也能阅读并转换 Quicken 文件，并包含许多相同的功能。微软本来倾向于兼并 Quicken 及其用户基础，然而，Money 已经能够充当微软向家用财务软件市场扩张的基础，尤其是当微软将该软件的特性与其他畅销产品集成在一起时更是如此。它在将各种功能特性和应用产品与 MS-DOS、Windows、Works、Office 及 Bob 进行集成方面已经有了先例。

在信息高速公路上

将台式 PC、办公室和家用产品市场联系起来就形成了信息高速公路——一个正在成长并加速互联的联网服务、数据库和多媒体网络的混合体，这些服务包括通过专用电缆或电话线传送的电子邮件、家庭金融、交互式电视及点播式电视等，还有无所不包的数据库，从电子报纸和航空时间表到 NASA 发布的哈勃太空望远镜拍摄的图象，应有尽有。信息高速公路目前还不是一个一致的体系，它也许成不了一个一致的体系；它有各种各样的载体及进入独立系统的不同方式。但正如史蒂夫·鲍尔莫在最近的一次谈话中所说，微软的计划是将其所做的一切扩展到信息高速公路上：

> 我们现在有五种业务，我们正努力把它们全部扩展到高速公路上去。我们制造桌面操作系统。我们正努力使 Windows 有效地转化为一种可以在[电视]上使用的东西。我们通过提供 WindowsNT 建造服务器软件。我们正努力使之能够成为这些网络的中枢服务器。我们还制造开发工具。我们最近购买了一个设在蒙特利尔的名为 Softimage 的公司，[电影]《侏罗纪公园》的制片就使用了它的工具。为什么？我们需要把我们的开发工具改进得易于提供这种内容。我们为现代家庭制造应用产品。我们为什么制造这种类似棒球或篮球产品的东西？因为这只是开始。它们现在还需自负盈亏，然而它们正在开始取得特许权以成为提供……信息的公司。每当你将大量信息装载到一个 CD 中去时，你就是在模拟高速公路刚要开始时的情况。[92]

微软的技术专家们已试验过“智能”电传机和电话的设计，他们也注意到竞争对手们加强现有单功能装备（如电话、电传机、有线电视顶置盒和视频游戏机）以使这些设备更像个人计算机的努力。但是他们的结论是，使一台 PC 能够接电话、发电传或放映家庭录像要远比让一部电话机、一部电传机或一台电视机去做文字处理、计算电子表格或执行所有其他 PC 上才有的功能容易得多也便宜得多（新型家庭视频游戏机能够更简单地加入一些类似 PC 的功能。实际上，1994 年微软就宣布与世嘉公司合作开发能够拓展视频游戏机功能的操作系统）。因为 PC 或类似设备的使用在将来会变得更加盛行，所以微软的老总们和研究人员都发现在将各种消费者服务技术包装成 PC 软件方面他们正处于关键时刻。不过正如鲍尔莫所阐述的，微软需要拓展 Windows：

> 当你从技术角度来谈论目前正在进行的功能收敛时，从一个软件公司的观点来看是相当不错的——因为你可以编写标准化软件……我们可以把我们做的 98%的工作从信息传递的某种特定媒介或特定途径分离出来。因此在这么多种不同类型的网络中投资真是一本万利，只要我们采取一种收敛的长期模型……我们的 Windows 商标意味着一套东西。我们在收敛领域工作的人们常说：“不要告诉任何人我们把 Windows 装进了顶置盒。”……但是如果不是 Windows，微软绝对没有理由去打这个赌。我们没有分享，没有合作，什么都没有。除非我们能使用这些软件的 80%—90%，否则便没有意义。[93]

在比尔·盖茨个人对信息高速公路技术的热情激励下，集团副总裁内森·米尔沃德正在领导微软努力推出新的产品和服务，创造目前产品部尚未见过的更加新奇的技术。(94)他还监管着数目越来越多的联盟、合伙和合并企业（参见附录 5 中的一系列最新动向）。公司活动的范围表明，微软正在遵循着一种渐进的而又咄咄逼人的方法以巨大的潜力进入各种各样的新市场：

·建立起开发、营销基本的多媒体应用产品的技术。

·建立起连结 PC 及其他设备与外部网络所需的基础网络与电信软件技术。

·在微软操作系统和相关应用产品中逐渐增加更多的多媒体与网络能力，以期将一大部分微软基础用户转换为微软信息高速公路消费者。

·争取成为内容提供者或来源，比如电影、艺术作品或电影评论方面的权利。

·为实现上述目标而结成联盟或实行合并。

·通过联网的网络以及有线电视线路与电话系统把多媒体产品和交互服务提供给成千上万的微软用户和非微软用户，把各部分联成一体。

这个战略的许多部分现在已在实施。我们前面提到过，Windows 95 中包括与 Microsoft Network 联结的方法，用户可以通过一个调制解调器和一根电话线或某些情况下通过有线电视的线路进行联网。(95)这种新的联网服务（与电信公司合作）将不同于竞争。它有更简便的进入方法，价格非常低（比如每月 5 美元，约为竞争性服务的一半），很少或没有计时附加费（这可能采取按产品个数、服务或交易次数收费的办法），还有大量易于使用的特性。它还将包含普通的电子邮件以及进入 Internet 这扇通向大多数网络的大门的方法。但进入微软系统的方式将使其尤为适用于大规模市场。

目前多数 PC 用户必须退出正常的操作系统和应用软件,然后进入网络通信软件，才有可能进入联网的网络。如果使用 Windows 95，用户只要按一下计算机屏幕上的按钮，操作系统就会自动将他们与微软连接，微软于是提供一个许可的应用程序。如果用户登记了，微软将对他们使用的网络和任何特别服务和产品开出帐单（如查验股票行情或向煤气公司交费）。美国的 AT&T 和 Sprint 公司、英国的英国电信公司、加拿大的联合电信公司都同意帮助微软通过电话线将服务送至 35 个国家（用 20 种语言）。美国电信公司将通过有线电话线路承载网络，并提供一些录像节目编程。另外，通用电器公司的 NBC 电视子公司将与微软合作开发多媒体产品及交互式电视节目编程。

微软产品的消费者还将能够通过网络获得支持（例如对一般问题的解答或软件更新）以及 Encarta 百科全书、Complete Baseball 或 Cinemania 等产品的更新。还会有新闻、体育、天气、商业及技术信息数据库。消费者还能将艺术作品的数字化版本在其计算机上并最终在其家庭内展出(像比尔·盖茨所计划的那样）。消费者们还将能够从其他公司——比如家庭购物网络及几十家另外的公司——购买产品和服务。微软正通过降低价格和提供能将消费者的信息数据库或目录转化成网络格式的价廉而简便的工具来吸引他们进入其网络。为提供家庭银行和联网金融服务交易以及联网股票行情，微软已与大通—曼哈顿银行、芝加哥第一国民银行、波士顿银行及其他主要银行建立了关系。它还正在与 Visa 国际公司合作，开发用于电子购物和防盗信用卡

联网支付的软件。

如果预计的 3000 万左右的用户升级到 Windows 95 或将其预装在新计算机上，而其中哪怕只有百分之几的人注册使用 Microsoft Network，那么微软将马上成为 130 亿美元的联网服务产业中的巨人。这一产业正以每年 30%的速度成长，有着巨大的潜力：只有 5%的美国居民户预订入网，而使用 America Online、Prodigy、CompuServe 和其他主要公司服务的现有消费者在 1995 年总共不超过 850 万。

对有些人来说，微软似乎在太多的方面走得太快了；但是这许多行为背后的想法就是要网罗所有可能的用户基础。尽管人们显然在汇集到一起，成为一个数目越来越大的群体，但没有人确切知道新的通信与多媒体技术将如何发展以及以何种时间框架发展。正如史蒂夫·鲍尔莫所承认的，微软无需靠这些新的事业在短期内赚钱，或制定非常具体的计划。微软老总们只需勾画战略关系和下赌注，以确保其公司收入与技术不会在他们眼皮底下出现滑坡：

> 说到高速公路的收敛，我们确实正在观察、寻找并努力促成那种最具关键性的关系。在我们的公司里，在我们现在的位置上，真正唯一巨大的风险就是出现某种使我们在技术上所从事的一切遭到淘汰的东西，就如同对主机和微机世界的一切构成巨大冲击的东西一样，它将侵蚀掉我们的用户基础……这是一个可怕而又令人激动的观点，因为坦率地说，它迫使我们继续推进和投资于一些你可能会不时感到有点疯狂或有点超越时代的事业。我们常说，我们召开第一次 CD-ROM 会议是在七八年前，我们现在才靠我们开发的 CD-ROM 产品第一次获利——是靠边际获益而非投资获益……
>
> 我们的观点就是，对于人们通常所谈到的关于收敛方面的投资来说，任何人都要等很长时间才会赚钱。我们当然也在期待着……我们进入这一领域，并没有关于事情如何进展的清楚计划。我们只知道，以我们现有的状况，率先站在这一技术的投资边缘并非不谨慎。这不是技术问题。如果有人在这些技术后面进行了基础投资，那么我们就是在为人们如何存取和利用信息设计方案。[96]

微软带给这些新市场的不光是日渐增多的各种应用软件、可拓展的操作系统、OLE 等启动性技术、富于创造性天才的技术人员和组成几十个合伙企业和新企业的资源。正如我们前面所描述的，近几年来，微软一直在引进能够使产品开发组详细分析用户的实际行为，找出对于大多数人来说最重要的产品特性，从而创造出即使对初级家庭消费者也相对易于使用的产品的一些技术。其他关键工具和技术使各小组能够在开发过程中使特性逐渐进化并使其构件经常保持同步并在一定时期内保持稳定。这种方法对于那些程序经理和软件开发员不能完全预测或控制最终产品和各个特性未来形态的应用软件项目特别有用。第 4 和第 5 章将着重讨论组成微软软件产品开发的同步稳定过程的一些专门技术和工具。

第四章
品定义与开发过程

依靠改进特性与固定资源激发创造力

在产品定义与开发过程中。微软遵循着一种战略，我们称之为靠改进特性与固定资源来激发创造力。我们把该战略分成五个原则来讨论：

·将大项目分成若干里程碑式的重要阶段，各阶段之间有缓冲时间，但不进行单独的产品维护。

·运用想象性描述和对特性的概要说明指导项目。

·根据用户行为和有关用户的资料确定产品特性及其优先顺序。

·建立模块化的和水平式的设计结构，并使项目结构反映产品结构的特点。

·靠个人负责和固定项目资源实施控制。

微软定义产品与产品开发过程的方法并非特别新颖。有一系列公司都使用相似的渐进开发法，如计算机硬件销售商（惠普与 DEC），计算机系统集成商（EDS 与 SAIC），航空航天与国防承包商（TRW 与 Hughes）以及电子设备制造商（Motorola 与 Xerox）。然而，微软在制定产品与产品开发过程战略以支持其竞争战略方面显得尤其成功。

微软力求开发面向广大市场的产品，并设立了实事求是的标准。它的产品开发期与生命周期相对较短，这些因素都产生了非常巨大的影响。产品开发计划既要紧凑，又必须灵活而有弹性，开发过程还必须尽快往前进行。最新的工程方法或工具只是第二位的。在变化较慢的市场上，公司可以在开始写程序代码之前，先构想出一份详尽的产品说明，然后“锁定”该说明及相应的工作进度安排。微软的软件开发组更多的是平行展开工作，正如克里斯·彼得斯所解释的那样：“我想，在变化较快的商用软件市场上，开发过程是各不相同的。在整个开发过程中，所有的事情都在平行地进行。每当你有了一个‘产品说明’或开始按工作进度表办事，随着测试项目被重新定义，产品说明马上会被更新，工作进度也会相应更新。所有这一切都以尽可能快的速度往前赶，你永不停息。”

首先，微软的开发小组努力了解用户的需要，再把这些需要分成一个个单独的特性，然后他们给予这些特性不同的优先级，把它们再分配给子项目，这些子项目则将整个开发项目划分成了 3—4 个里程碑式的重要阶段。为了在产品开发过程中激发和凝聚创造力，微软的经理努力“固化”项目的可用资源——限制任何一个项目中的工作人数与可用时间。这样，固化的项目资源就成了项目开发计划中最关键的规定因素，尤其是其中的预期出品时间使得整个开发小组充分凝聚其创造力与努力工作。开发小组必须按照反推法自己规定由出品日回溯的中间步骤和中间阶段，同时与其他微软项目、产品批发商以及第三方系统集成商协调产品的交付。这样，即使预定的出品日期时常改变，项目也能实现这些目标。

然而，对应用软件产品和与之不同的系统软件产品，微软在使用这些定义产品和产品开发过程的原则时有一些不同。应用软件产品，如 Word 与 Excel，或是消费型软件产品，如 Works 与 Complete Baseball，比起系统软

件来（如操作系统软件），往往开发期较短，开发组较小，对出品日的估计也比较精确。而新的系统软件可能有许多新的功能，需要数年和数百人来规定、开发与测试。例如，为了开发 Windows NT 的第一版，微软用了 4 年多时间，最多时开发组里有 400 多人。而 Excel 5.0 只用了 18 个月，开发组也只有 100 到 125 人。许多应用软件产品，像 Excel 与 Word，往往已经是第“ N ”版了（而非第一版），这使得这些产品的开发周期更有规律，也使估算今后的工作进度更为方便。

在功能方面，系统软件往往更多地有一组不可分的“ 核心 ”功能，在产品包装出品之前，这组功能必须被完成并表现得可靠（即“ 稳定 ”）。为了加速产品的出品而从操作系统中删去一些未完成的功能会非常困难。开发和完善一组能正确和有效执行的核心职能，是发行像 WindowsNT 与 Windows 95 这样的系统软件的最强动力；出品日期这个目标被排在第二位。应用软件和消费型软件往往有更多的独立的特性，在项目进行时可以加减这类软件以适应竞争的压力、资源的限制以及出品日的安排。因此，在生命周期的早期阶段，系统软件产品的想象性描述与说明文件要比应用软件完整和详细得多，目的是为了规定好核心职能组以及特性之间的相关性。

系统软件产品往往测试期也较长，包括广泛的 β 测试。这是因为它们需要更强的稳定性以及与众多的应用软件、网络系统、计算机销售商平台，诸如打印机这样的外设的兼容性。应用软件往往有较多的独立功能，产品间的相关性较少。现在由于强调软件之间的共享性（如 Office 套装软件），这一点已有所变化。

微软越来越多地运用共享或通用构件来构造软件，这已经开始改变微软应用软件的结构。应用软件曾是互不相关的产品，但它们正变得越来越组合化与可以共享。举例来说，项目现在可以灵活地往不同的操作或单独的“ 线程执行过程 ”加入内含 OLE 的构件。这些构件可以向用户提供特定的功能包，如绘图、打印或数学计算。用户也可以综合来自不同产品的功能和信息对象——比如，写一封含有图形和电子表格数据的书信。

决定应用软件产品需要哪些功能的往往是市场营销人员和程序经理，而不是具体的程序开发员。他们使用根据用户情况建立模型的技术制定计划。具体开发员在系统软件的功能上则更有发言权。部门内部的各开发组也有差别。虽然微软对小的消费型软件的管理不像对那些作为收入主要来源的大型软件那么管理紧凑，但他们彼此之间却非常相似，如 Excel 与 Word。在系统软件和计算机语言领域，一般说来，Windows NT 就比 VisualBasic 或 Windows95 被更精心地组织和管理过。不过，应用产品与系统产品在微软还是有许多相似之处，部门内的产品开发单位也是如此。我们在以下各节中所写的大部分对这两类软件都适用，对微软制造的所有重要产品也都适用。

原则一：将大项目分成若干里程碑式的重要阶段，各阶段之间有缓冲时间，但不进行单独的产品维护

在今天的微软中——尤其是与 80 年代相比，人们虽然力求更现实些，然而在进度安排上他们仍旧雄心勃勃。一个项目的进度安排具有灵活性，但同时也提供了项目内外互相依赖性以及一种高度组织化的预期交流机制。典型的桌面应用软件项目，如 Office、Word 或 Excel，新旧版本大约只间隔 12

到 24 个月。微软正逐步对应用软件产品轮流使用 12 和 24 个月的进度计划，其中，12 个月的周期只增加小特性，24 个月的项目进行重要的特性与结构的变化。对同一种产品的两种进度安排可以同时开始，在最初 12 个月齐头并进，这样微软就可以每 12 个月提供一种新版本。

然而，重要的新操作系统产品，如 WindowsNT 或 Windows95，可能需要 3 到 4 年的进度安排。微软的管理人员并不是一味地在一个 12 到 24 个月的典型项目中投入得越多越好。他们要努力弄清在一个新的开发项目上，到底有多少时间和人员可用。他们也考虑了未知的变化与困难，确定项目的规模使之与时间和可用人员相符。这样，微软的项目进度表由计划阶段与若干循环过程组成，每一循环包括产品的开发与发布，测试与稳定化以及为意外事故准备的或在重要阶段连接处安插的“缓冲”时间。每个项目都有自己单独的进度表，但微软已经越来越强调产品之间发送安排的协调性。例如，Office 有一个集成的进度表，内容涵盖了 Excel、Word 以及其他作为套装应用软件一部分的产品。

程序经理、开发员与测试员，他们是规定、实现和测试一件新产品的人，同时也是对产品前一版本修改错误、变更并增强其功能的人（这被称为产品“维护”）。并没有一个单独的组从事产品维护。在许多其他公司中，这样的维护工作花去了开发员本可能用于新产品开发的大量时间。与之不同，微软的产品单位创造了产品新版本连续的“发布周期”。MS-DOS 出到了第 6 版，Windows 出到了第 4 版，Excel 出到了第 5 版，Word 出到了第 6 版，如此等等。微软的产品组根据市场营销情况与顾客信息反馈决定什么是产品中应该保持的特性，什么是应该添加的和改变的特性，然后他们把这当作是下一个产品版本的一部分工作来做。下一个版本也可能包含原来项目中由于时间紧迫而不得不砍掉的特性，开发应用软件产品时这种情况尤其普遍。

项目进度安排与里程碑

微软的同步一稳定产品开发法（第 5 章会更详细地描述）始于 80 年代晚期。这种方法集中了里程碑和每日构造这些关键的概念（为做到“零缺陷”）。从前的项目事业单位于 1988 年出品的 Publisher1.0，是第一个在进度表中使用了里程碑的微软项目；1989 年和 1990 年的 Excel 3.0 是第一个采用此概念的大型项目。其他一些小项目，如 Works，大约在 1988 年到 1990 年，也采用了里程碑概念。有的开发组在此时期使用了“每日构造”策略，但他们对使用“每日构造”以不断发现和修正缺陷却只有肤浅的认识。1989 年在采用“休假会”之后，里程碑与“每日构造”以及更严格的质量控制方法，在微软中逐渐变得普遍了。

今天，微软中典型项目的生命周期包括三个阶段。计划阶段完成功能的说明和进度表的最后制定，开发阶段写出完整的源代码，稳定化阶段完成产品，使之能够批量生产。[1] 微软把这三个阶段称为“进度表完成及项目计划被批准”、“代码完成”以及“发布供生产”（用到软件产品上“生产”概念是指磁盘和文档的复制）。这三个大阶段以及阶段间内在的循环方法与循序的“瀑布”式生命周期很不相同，后者是由需求、详尽设计、模块化的代码设计与测试、集成测试以及系统测试组成。[2] 微软的三个阶段更像风险驱动的、渐进的“螺旋”生命周期模型，该模型已日益为软件开发机构所采用。[3]

在微软，计划阶段的产品是想象性描述与说明文件，用来解释项目将做

什么和怎么做。在管理人员拟定进度表、开发员写出源代码之前，这些东西都促进了人们对设计问题的思考与讨论。开发阶段围绕三次主要的内部产品发布来进行；稳定化阶段集中于广泛的内部与外部（β）测试。在整个产品生产周期中，微软都使用了缓冲时间的概念。缓冲时间使开发组能够对付意外的困难和影响到时间进度的变故，它也提供了一种手段，可以缓和及时出品与试图精确估计出品日之间的矛盾。若是对现存产品的更新，计划阶段也许只用 3 个月的时间，但对新产品，计划阶段就可能长达 1 年以上。在以往典型的项目中，开发阶段与稳定化阶段大约需用 18 个月，这 18 个月可以如下划分：[4]

·6 到 12 个月的开发时间，一般包含 3 到 4 个重要的里程碑式产品版本。

· 2 到 4 个月的开发缓冲时间，分配到每个里程碑式产品版本中去。

· 3 到 8 个月的稳定化期，包括测试与缓冲时间以及用 6 个星期为最后的版本和顾客支持作准备（完成最后的测试与联机文件编写的目标日期是最终版本投产前约 6 周；完成印刷好文档的目标日期是最终版本投产前约 4 周）。

在开发和稳定化阶段的所有时间中，一个项目通常会将 2/3 的时间用于开发，1/3 的时间用于稳定化。克里斯·彼得斯，Office 部门的副总裁，曾这样概述通常的进度：“ 一般说来，在总的进度表中，用一半的时间写出产品，留下另一半时间调试或应付意外事故。这样，如果我有一个两年的项目，我会用一年来完成事先想好的东西……如果事情有点麻烦，我便去掉我认为不太重要的特性。 ”这种里程碑式的工作过程使微软的经理们可以清楚地了解产品开发过程进行到了哪一步，也使他们在开发阶段的晚期有能力灵活地删去一些产品特性以满足出品日的要求。

图 4.1 和图 4.2 概括了微软的(a)产品阶段周期，(b)文件或中间产品，(c)复查以及(d)中间的里程碑或某阶段的完成等情况。这些图和以下各节将集中讨论程序管理、开发及测试在产品周期中的角色。图 4.3 提供了一个典型产品周期的综合情况，其中包括另外一些公司职能如产品管理（市场营销），用户教育（创造供顾客使用的文档或手册）等内容。图 4.1 同步——稳定开发法

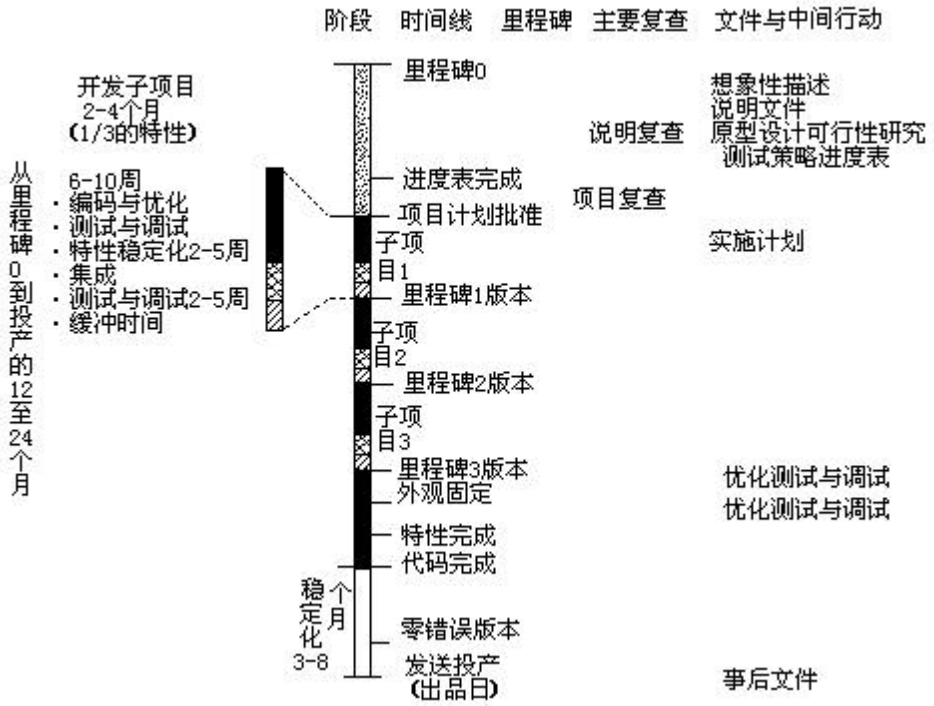
计划阶段：定义产品的想象性描述、说明与进度。
·想象性描述　产品和程序管理部门运用广泛的顾客意见来确定和优化产品的特性。 ·说明文件　基于想象性描述，程序管理部门与开发组定义特性的功能实现，结构问题，以及各部分间的相关性。 ·制订进度表与构造特性小组　基于说明文件，程序管理部门协调进度表，安排出特性小组，每个小组包括大约 1 名程序经理，3 — 8 个开发员，3 — 8 个测试员（以 1 ： 1 的比例与开发员平行工作。）

开发阶段：用 3 — 4 个顺序的子项目，每个产生一个里程碑式的产品发送，来完成特性的开发。
程序经理协调开发过程。 开发员设计、编码、调试。测试员与开发员配对，不断地进行测试。 ·子项目 I　前 1/3 的特性：最重要的特性与共享的构件。 ·子项目 II　中间 1/3 的特性。 ·子项目 III　最后 1/3 的特性：最不重要的特性。

稳定化阶段：全面的内外部测试，最后的产品稳定化以及出品。
程序经理协调 OEM （原始设备制造商）与 ISV（独立软件开发员），监督从顾客得到的信息反馈。开发员进行最后的调试与代码稳定化。测试员发现并清除错误。 ·内部测试　公司内部对整个产品做详尽的测试。 ·外部测试　公司外在的“ β ”测试点，像 OEM ， ISV 以及最终用户处对整个产品做详尽测试。 ·发送准备　为批量生产准备发布最后的“金主盘”磁盘与文档。

资料来源：与图 4.2 相同。

图4.2 程序管理、开发及测试的同步——稳定生命周期



资料来源：1993 年 3 月 17 日对开发主任戴夫·穆尔的采访，以及《进度安排方法学与里程碑之定义》，微软公司 Office 事业单位内部流通，1989 年 1 月 9 日

图 4.3 与产品管理和用户教育有关的产品生命周期模型

		产品周期阶段		
		计划最后的里程碑：项目计划批准（包括：进度表完成）	开发（3—4个里程碑）最后的里程碑：代码完成	稳定化最后的里程碑：发送投产
组织的角色	产品管理(市场营销)	·市场研究 ·营销计划 ·想象性描述	·定价、升级、包装	·β测试点 ·发行
	程序管理	·想象性描述 ·设计＆原型特性 ·写说明 ·基于行为制定计划 ·相关调查 ·创造主进度表	·管理项目的执行状况与联系	·为找出错误而提供输入 ·完成后产品之出品
	可用性试验室测试员	·建立可用性目标 ·考察性测试	·循环测试（原型\早期代码） ·完整产品的测试	·基于行为制定计划的方案测试 ·领域测试 ·为下一版本进行的测试
	视觉界面设计组	·开始用户界面设计 ·对用户界面说明进行讨论	·确定用户界面设计 ·建立图与位图	·视觉界面设计复查
	开发组	·特性的初步设想与可行性 ·特性设计	·写出并优化代码 ·产品每日构造 ·测试与改错 ·发布产品	·测试与改错
	内部可操作性设计组	·通用特性集设计	·支持代码共享·商量测试计划	·对下一次发布之通用特性作计划
	本地化组（为美国外版本服务）	·本地化计划	·建立与批准词库 ·视觉与功能检验 ·翻译文件 ·复查\检查运行时之用户界面	·小组内测试与调试 ·印出生产周期 ·最后的测试
	软件测试	·测试策略	·创立测试 ·运行测试项目组	·内部测试 ·β测试 ·最后的版本测试

		产品周期阶段		
		计划 最后的里程碑：项目计划批准（包括：进度表完成）	开发（3—4个里程碑） 最后的里程碑：代码完成	稳定化 最后的里程碑：发送投产
组织的角色	用户教育（印刷的文档）	·概念上的设计与策略文件	·定义专有名词 ·写出、编辑与创造联机链接 ·技术复查 ·确定内容与图形	·最后的版面定型 ·准备与发布给销售商 ·印刷文档 ·文档发布投产
	用户教育（联机文档）	·与上同	·与上同 ·确定运行时用户界面的帮助	·小组内测试与调试 ·最后的测试、调试
	产品支持	·收集顾客资料 ·预测支持成本	·找出支持问题 ·决定支持策略 ·训练	·产品支持服务（PSS）β与估值 ·对顾客支持资料库的扫描 ·打电话

资料来源：1993年3月17日，对微软开发主任戴夫·穆尔的采访以及微软内部的无标题文件。

计划阶段

计划阶段是在一个项目的生命周期中，所有于开发前进行的计划所占用的时间。计划阶段产生出想象性描述、市场营销计划、设计目标、一份最初的产品说明、为集成其他组开发的构件而规定的接口标准、最初的测试计划、一个文档策略（印刷的与联机的）以及一份可用性问题清单。计划阶段从想象性描述开始。想象性描述来自产品经理以及各产品单位的程序经理；它是对产品作出的市场营销设想，包括了对竞争对手产品的分析以及对未来版本的规划。想象性描述也可能讨论在前一次版本中发现而必须解决的问题以及应添加的重要功能。所有这些都基于对顾客和市场的分析以及从产品支持服务组（PSS）处得到的资料。经理们把长期产品计划问题当作三年计划周期的一部分来考虑，这在第1章讨论过。

说明文件从一个大纲开始，然后定义出新的或增加的产品特性，并对其赋以不同的优先级。就像第2章所说的那样，程序经理负责产品说明的写作，而其他一些人，尤其是开发员与测试员，具体撰写该文件。说明文件只是产品特性的一个预备性概览；从开始开发到项目完成它要增加或变化20%—30%。虽然在生命周期的晚期文件说明变化一般较小，但越是到晚期，开发员就越是必须具备充分的理由来作改变。

程序经理用Visual Basic工具创造出大量的原型。他们也开展设计可行性研究以了解设计中的取舍情况，尽快做出涉及产品说明的决定。前程序管理主任约翰·法恩，现任Excel组程序经理，认为说明文件必须把产品特性概括得使普通用户也能用一句简单的话来描述产品："你发明产品的特性，

一个接一个……所有都是从用户的观点出发的，那是你应做的一切——抛掉任何其他想法，只思考‘用户看到的是什么？’当你用一件产品时，产品所做的应该能被用户充分理解，他应该能用一句话讲出产品在做什么。从我的观点来看，有一些产品不够好，因为虽然它们有一些相当棒的特性，但是没人讲得出产品做了些什么。”

正如第 1 章所述，比尔·盖茨和其他高级董事要对重要产品的说明进行复查。至于不太重要的产品，则将该工作交给部门经理去做。概要的说明文件，开发员的时间估计，为测试及文档制定的策略以及顾客支持，这一切构成了项目进度表的基础。盖茨也对重要产品作项目复查以批准项目计划与进度表。在计划阶段结束时，项目所处的状态是项目计划被批准且“进度表完成”。

开发阶段

开发阶段的计划对三四个主要里程碑版本都个别分配一组特性，规定出特性的细节和技术上的相关性，记录下单个开发员的任务以及对进度的估计。在开发阶段中，开发员在功能性说明的指导下写源代码，测试员写出测试项目组以检查产品的特性与工作范围是否正常，用户教育人员则编写出文档草案。

当测试员发现错误时，开发员并不是留待以后处理，而是马上改错，并在整个开发阶段内使测试不断地、自动地进行。这就改善了产品的稳定性并且使版本发布日期更易估计。当到达项目中的一定点后（例如，执行了进度表的 40%时），开发员就试图“锁定”产品的主要功能要求或特性，从此只允许小的改动。如果在此点之后开发员想作大的改动，他们必须与程序经理以及领头的开发经理商量，也许还要征求产品部门经理的意见。

主要的里程碑版本　一个项目是围绕着 3 或 4 个主要的内部版本，或‘里程碑子项目’来组织开发阶段的。微软希望这些版本能非常稳定。从理论上讲，项目应能做到以之向顾客出品。正如克里斯·彼得斯所评论的：“我们第一次（在大项目上）运用里程碑方法是在开发 Excel 时。我认为现在所有其他组也在使用了……你所做的是把一个项目基本上分成三个部分……然后你假装在每一部分完成后都要推出产品。”[5] 开发员也许需要在每个重要版本发布前做一些预备性发布以达到测试组的验收标准，这样才能完成重要的里程碑版本。前任 Excel 的程序经理，现任 Office 高级程序经理之一的麦克·康特，描述了一个项目是如何基于重要的里程碑版本工作的：“我们实际上将开发分成三个分离的里程碑。它们也许是 6 周的，或者是 10 周的里程碑……在里程碑快结束时，我们的目标是获得所有已经在创造的特性，并且对该里程碑而言是无错误的……这样，当我们达到了‘出品质量’时，我们就能向下一个里程碑前进。关键在于这样一来，我们就永远不会让事态完全失控，不会在项目快完成时由于有成千的错误以致完全说不准何时能真正完成项目。”

项目大约用 2 至 4 个月来开发每一个重要的里程碑版本。每一个版本都包括其自身的编码、优化、测试以及调试活动。项目为意外事故保留总开发期 1/3 的时间，这就是保留给每个里程碑的‘缓冲时间’。举例来说，Excel5.0 的开发阶段中有三个主要的内部版本，每一个都是为时 13 周的里程碑（见表

4.1）。起初的进度表规定在每个里程碑内，用 8 周开发，2 周集成（确保不同的特性能在一起正常工作）； Excel 组还为每个里程碑分配了 3 周的缓冲时间。该组在第一个里程碑处没有用完所有的缓冲时间，于是项目就为第二和第三开发期各加了 1 周，使之有 9 周开发时间。然而，第二个里程碑后，Excel 用尽了缓冲时间，主要是因为在转换 Macintosh 机器上使用的编译时出了问题。在第三个里程碑处，缓冲时间不够了，开发员需要额外的 6 周，这主要是因为 Excel 依赖 OLE 与用于应用程序的 VisualBasic（VBA）。OLE 晚了 6 个月，VBA 晚了 8 个月，这都使 Excel5.0 以及 Word6.0 增加了复杂性

表 4.1Excel5.0 主要内部里程碑预计的和实际的进度表

主要内部里程碑	预计进度	实际进度
里程碑 I		
	8 周开发	8 周开发
	2 周集成	2 周集成
	3 周缓冲	1 周缓冲
里程碑 II		
	8 周开发	9 周开发
	2 周集成	2 周集成
	3 周缓冲	3 周缓冲
里程碑III		
	8 周开发	9 周开发
	2 周集成	2 周集成
	3 周缓冲	3 周缓冲
		6 周延迟
总计	39 周	45 周

资料来源：与 Office 开发经理乔恩·德·沃恩的会谈，1993 年 4 月 15 日与 1994 年 9 月 29 日。

并延迟了出品。正如麦克·康特所评论的：“是啊，我们本打算在 9 月 27 日出品，但事实上却拖到了 12 月 22 日，晚了 3 个月。VB 与 OLE 对我们而言是个大问题。”

再举一个例子。Office 的开发阶段有 4 个主要内部版本，开发阶段与稳定化阶段共占 21 个月。第一个里程碑版本是结构定义子项目，经理们都把这看作一个特例。其他三个里程碑子项目每个长 16 至 18 个周；每个子项目在开发和缓冲上都用相同的时间，然而对每一次集成都逐次增加一周时间。系统软件产品、消费型软件产品以及在研究部门中创造出的新型软件（如 Tiger 视频服务器），都遵循着相似的过程。然而，它们在产品的性质以及面向的顾客上是有一些区别的。布兰德·斯尔文伯格，个人操作系统（Windows/MS-DOS）的高级副总裁，概述了系统产品需要的四类里程碑的完成，并将微软的渐进开发法与他在苹果和 Borland 公司所获得的经验作了比较：

> 当我们到达里程碑时，我们不光是开发了功能，而且要符合产品规格、性能以及质量方面的要求……这就是说，当我们成功地加上了那些

功能后，我们并未到达里程碑，我们是在实现产品性能目标，达到某种水平的质量后才能到达里程碑。如果我们在前一个里程碑处放松了一些产品性能目标，只注重把那些功能开发出来，那么在下一个里程碑处，产品的规格与性能问题就将优先于功能问题。然后在随后的里程碑处，产品规格与性能问题又只是增加功能时附带考虑的问题……

如果说我们在苹果公司中的做法是保守的话，那么这儿的做法与之不同。[苹果的]小组是割裂的，独立的，各自开发各自的东西。在还有 3 个月就要出品的时候，你才会说：“ 好了，让我们把一切集成起来。 ” Borland 公司是把工作分成许多小的部分，并且总是让开发的东西能够运转，这是种渐进式的哲学……看起来似乎这种渐进的方法费时较长，但实际上几乎没有用过很长时间，因为这使你总是能掌握住事情真实的情况。

外观固定，特性完成以及代码完成当对最后一个主要的里程碑版本做了测试与稳定化之后，产品就要进行“ 外观固定 ”。该短语系指产品的主要用户界面构件——诸如菜单、对话框以及文件窗口将不再有任何进一步的大变动，虽然小的改动在所难免。项目采用外观固定是为了方便用户教育组编写产品文档。当产品功能齐全且测试员能在上面运行测试方案的时候，产品就“ 特性完成 ”了。在此时，一件产品还不能说是基本上稳定了，它可能还有许多错误，还需要在速度和内存的使用上大幅改善其性能（称为优化）。当产品“ 代码完成 ”时，开发阶段才最后结束，这意味着所有的特性都能按说明中和用户文档中规定的那样工作，除了调试与随机性的优化之外，没有什么工作剩下来了。例如，当 Excel5.0 代码完成后，乔恩·德·沃恩，以前是 Excel 的开发经理，现为 Office 开发经理，又用了两个月的时间重写完成得不好的代码。克里斯·彼得斯有一个决定项目何时能到达“ 代码完成 ”的粗略的经验估计法：

对一个 30—40 人的项目来说，代码完成不容易定义。例如，如果有人用了三天来修改一个错误，他们……实际上是在写代码，他们就没有“ 代码完成 ”……这样，实际上发生的情况便是，当你赶着完成项目时，错误的数量也增多了。然后，当人们真正注意修改错误时，你会到达某一点，从此点以后到出品日，错误的数量每天都在减少。在那点以后的道路对你而言绝对是越来越轻松的。当这种情况发生时，你就知道你已代码完成了……这是不容置疑的。

稳定化阶段

稳定化阶段着重于对产品的测试与调试，向产品发布不断推进。项目在稳定化阶段尽量不增加新的特性，但如果出现了竞争产品，或其他市场因素改变了的话，也有可能增加一些产品特性。微软的测试过程被组织成一系列的阶段，每个阶段比如说是 4 个星期。测试员与开发员不断地寻找并修改错误以尽可能早地为产品的发送作好准备；稳定化阶段也包括缓冲时间以应付不可预见的问题或延迟。一个项目将其产品最后一个主要的里程碑版本分发给各种各样的测试组，有微软内部的，也有微软外部的。外部测试组（称为 β 测试点）包括个人用户，大的公司顾客，独立的软件销售商以及批发商。

当进行了广泛的测试后，没有发现严重的错误，并且项目经理决定在下一开发周期再修复所有剩下的已知错误时，“零错误版本”就产生了。开发员于是生产出一份“金主盘”，将之发送投产。微软就从这份拷贝上生产所有其他拷贝。产品发送投产后不久，项目小组一般再写一个事后文件——一份广泛的关于项目中什么工作做得较好而什么做得不好的总结。（在所有的项目中，一半到 2/3 的项目实际上写了该报告，长度从 15 页到上百页不等，第 6 章对此有详细论述。）

项目进度表中的缓冲时间

微软使用缓冲计划，以在最高的效率与较好地对未来作预计之间求得平衡。这种应付突发事件的时间在开发和稳定化过程中是每一个主要里程碑的一部分。缓冲时间主要用于弥补由于对特性的不完全理解，或是技术困难或是由于疏忽而忘记把任务写入进度，或是未料到的难题而形成的漏洞。问题常常出在关联项目的拖延上，或是需要时间协调事先未计划而又在不止一个产品间共享的特性改变等方面。缓冲时间有助于一个项目适应意料之外的事件。就像克里斯·彼得斯回忆的那样：“当我们最终开始按时出品时，我们做事的一个秘诀就是终于将缓冲时间纳入了进度表。缓冲并非‘懒惰和愚蠢’，而是为了应付意外事故的发生准备充足的时间。你并不知道未来的一切，只要你承认你有不能完全预测未来的麻烦，你就应该在进度表中加上缓冲时间，这样按时出品就很容易了——如果你有魄力砍去特性或缩小特性规模的话。”

项目必须把缓冲时间完全用到不确定事件上去，而不是在日常的或可预见的任务，像特性开发，测试，或文件复查上运用它。项目小组也使用内部的产品里程碑作为他们的目标，而且不将此算入缓冲时间。这样，由项目领头人与开发小组商定的内部里程碑，像图 4.2 的“里程碑 1 版本”，就反映了主要里程碑减去缓冲时间的日期。内部里程碑名副其实，只供职能组内部使用，并不出现在印刷的小组进度表上；项目小组外的其他单位使用印刷的进度表作出他们的进度安排。

在应用软件产品中，典型的缓冲时间占进度的 20%—30%。彼得斯证实了这一点，“在 Excel[3.0]的开发中，我们把进度的 20%安排成缓冲时间，我认为这也是现在处理该问题的做法。”[6] 因为系统软件产品进度相对较长且特性集相对不可分，这些项目一般需要更多的缓冲时间。说服传统的项目经理，使之明白大量缓冲时间的必要性是困难的，因为他面临着缩短产品开发时间的压力。然而，微软却经常在其项目中运用缓冲时间的概念。布兰德·斯尔文伯格描述了他本人对系统软件产品“50%缓冲规则”不可动摇的信念及来自高级经理的阻力：

> 许多时候开发员天真地说：“那东西用一星期就可以完成，”于是你就相信他，只安排一星期时间。我们已经学到了怎样更好地问开发员问题来控制任务的执行，以及怎样把缓冲时间注入进度之中。在过去，许多人不在进度中包括缓冲时间。我在职业生涯的早期曾遇到过这样的情况。当你将包含一些缓冲时间的进度表呈送给总裁时，他说：“什么，你们这些家伙要出去钓鱼还是怎么的？你们在那缓冲时间里要做什么？”“这个，根据经验，我们需要缓冲时间来修改错误。”“是吗？

确切地讲讲你们要做什么？”“好吧，如果我能确切地告诉你我将要干什么，我很愿意告诉你，但是请相信我——那些缓冲将是需要的。”就许多传统的经理而言，容忍进度表中有缓冲时间，不啻为信念上的一次飞跃。大多数情况下，缓冲时间被开发经理一棍子打死：“不，让我们抓紧些，我们需要出品，我们可以挤掉一些缓冲时间。”于是你的进度表就脱离了实际。我并不知道缓冲时间将被用来做什么，但一次又一次，我知道那是需要的。不管它是用来修改错误，或是你边干边有了一些想法并决定：“我们实在应该有这个特性。”……这样，如果你有两个月，你会将一个月分配作为缓冲。[一个]50%缓冲规则……事实上是准确的，虽然我对此讲不出什么道理。

除了有助于使产品进度安排更符合实际，缓冲时间还给了项目一个机会，使之可以对竞争对手的产品公布或不可预知的大事作出反应：

现在我不知道有完美的说明和丝毫不差的估计，说到底，将来我们也无法完全了解竞争情况。……我们无法了解这个非常非常激动人心的工业将走向何方。传统的进度安排方法只是把你知道的所有东西一股脑儿算进去，说这就是期限——那是我们了解的所有东西。考虑到我们对未来所知如此之少，难道我们就不曾对如此短的期限感到过惊讶吗？……确定正确出品日期的方法是加上缓冲时间，承认未来不确定的时间。未来的不确定性是显而易见的。这并不是魄力不够或胆小懦弱，这只是承认未来就是未来，这是自信。[7]

复查与检验点

为了创造出一些来自管理层顶端的压力使项目处于控制之中，微软也在关键的时候进行每周一次的、每两周一次的或每月一次的项目复查，尤其是在决定实施计划之前。正如第 1 章所述，比尔·盖茨会参加到这个过程，与关键项目频繁接触，有时甚至每周一次。例如，Excel5.0 有四次复查，这些复查通常由两个小时的会议组成。盖茨，麦克·梅普尔斯，皮特·赫金斯及其他关键的开发员和程序经理，测试经理与马克·奥尔森都参加会议。对其他一般的项目，盖茨可能只通过 E-mail 来批准进度表和产品说明。

产品维护与“里程碑 0”

在大多数软件开发机构中，独立的小组——通常对产品开发经验极少——从事产品维护工作。我们在前面提到过，微软的情况与此不同。在这里，对产品下一版本的开发包括了创造重要的新功能与修改上一版本的错误。不光如此，微软还设立了单独的产品支持组以回答用户的电话，该组的大小基于对电话量的估计。对于紧急情况，一些产品组还设有快速修补维修小组，他们对一件产品进行中间的‘点版本发布”以满足顾客需要（例如，MS-DOS6.2 就代替了 MS-DOS6.0）。

微软最近开始采用的另一概念是“里程碑 0”。乔恩·德·沃恩将此描述为在一个产品版本完成后，另一开发周期正式开始前出现的工作：“经验的做法是，每人负责自己编的代码能工作……听起来开始一定很乱，但实际上在最后，人们确实学到了做事的正确方法，我们甚至还腾出了一些时间。

我们现在开始委婉地称之为‘里程碑 0’。这是处于一个版本结束与下一版本正式开始之间的时间，人们在此仔细检查并清除他们所知道的不当之处……如果有的地方我们确实很不喜欢，它就将成为下一进度的一部分，我们将更正规化地修改它。”例如，在 Excel4.0 与 5.0 之间，开发员用了一两个月时间做这种修修补补的工作。

在里程碑 0 阶段，开发员与程序经理一起完善下一版本的说明，完成对进度的估计。这样，项目就有机会往进度中安排时间以在下一版本中改正重大错误。现在，新项目大约用对间的 10%重做已有的特性，包括一些用户界面的变化，而不是一开始就开发下一版本的新特性。也有这样的情况，微软会推出一个填补漏洞的中间版本，如 Windows 3.1 与 Windows NT3.5。然而，对于关键产品，微软越来越普遍的做法是，对小改动（如 12 个月修订）与大改动（如 24 个月修订）平行地进行开发。

原则二：运用想象性描述和对特性的概要说明指导项目

就单独的一个项目而言，微软和其他 PC 软件开发者的主要任务是，既要给出足够的开发框架以使工作能持续进行，又要能容纳开发过程中出现的变化并保持足够的灵活性。为了做到这一点，微软采用高水平的想象性描述和概要的说明来指导项目运转，而不是在一开始就努力写出一份完整和详细的说明。想象性描述是由程序经理和来自市场营销组的产品计划人员共同编写的，它是一份非常短的文件，定义了一组驱动产品开发过程的目标，但不涉及产品要求的细节。例如，Word 6.0 的想象性描述只有一页，Windows 3.1 的描述只有一句话。对一件全新的产品，想象性描述通常会详细得多，实际上，它还包含了一份粗略的说明文件。总的说来，微软的目标是想象性描述越短越好，而且力图讲清楚“产品不做什么”而非“产品做什么”，这是因为决定产品不要哪些功能比决定要哪些功能困难得多。

运用想象性描述，程序经理开始编写功能说明文件，该文件解释产品的特性是什么以及这些特性如何与其他特性及产品发生关系。程序经理开始只写一个概要说明文件，随着项目的进行添加更多的细节。因为他们明白，过早固定产品特性及其细节是不现实的。在一个项目的生命周期中，竞争者的产品，顾客的需求以及市场机会可能不断变化；因此，说明文件必须非常机动灵活，以适应变化和利用新机会。同时，说明文件也应非常具体，以估算项目进度表，并使开发员能顺利工作下去，不必不断地重做特性。此外，开发员需要有这样的自由，即能决定如何用实际代码完成特性目标。

在项目进行过程中，程序经理负责开发、更新和改进说明。完整的说明文件像用户手册一样，是项目的产品之一，而不仅仅是开发中的投入。相应地，完整的说明不只起着产品功能最新描述的作用，它还是在产品投产与出品前进行测试与评估的主要基础。在说明过程的早期，程序经理会建立一个说明文件的正式网络地址，并运用版本控制工具来记录连续的再修订；直到项目结束后说明文件才付印。另外，程序经理们应该与开发员和内部操作组一起对说明提出建议，并用文件和进度表来保存测试软件的运行结果（主要是可用性试验室），以及建立一个跟踪系统来记录信息供写事后报告使用。

想象性描述有助于决定删除哪些特性

Excel5.0 的想象性描述大约有 5 页长。它所关注的是市场营销组希望产品具备的优先次序：特殊的功能，产品运行所需的软硬件，特殊的依赖性（如 OLE），对事业单位计划的总的看法，以及进度表的各个方面。Excel5.0 的想象性描述还规定了“ 领域 ”（用来对特性分组的宽泛的类别），并把这些领域分配给了一个个程序经理。因为 Excel5.0 相对来说是一个复杂的应用软件，有大量的功能，所以想象性描述不仅规定了最近版本的目标，还规定了未来 5 年的产品目标。它还包含了单独地为一些单个“ 领域 ”服务的小型想象性描述。

相反，Office 的想象性描述则强调依靠大量应用软件在一起和谐工作来完成单独一件任务。例如，创建一份复合文件，其中包含文本，数字数据，图形以及自动插入的排版信息。Office 的想象性描述认为 PC“ 只有一个主要的应用软件 ”，而不是认为 PC 有许多独立、各异的应用软件，用户不得不使之互相配合。

在具体说明一件产品时，一个主要问题是，市场营销人员往往希望产品有许多特性，而在一个紧张的进度表内这是完不成的。为了对付这种情况，微软的各个组都用想象性描述帮助细化产品版本的规定主题，然后他们就用这些主题来决定接纳或排斥那些候选的特性。克里斯·彼得斯描述了一个项目若不具备对产品作出的清晰的想象性描述将会发生何种后果，他也讲了好的和差的想象性描述之间的区别。

> 通常的做法是，程序经理会写出一个概要说明，其中包含了许多无法完成的特性。开发员会粗略地估计出在每个特性上要花的时间。当知道了希望的出品日期时，就会召开一个喧闹不堪的会议，人们在此讨价还价，大声咆哮，尖声叫嚷，为的是要努力提前出品。好在有一些关于想要产品做什么的单一的想法，这在永无休止的争论中对大家都大有裨益……
>
> 想象性描述有好有差。好的描述告诉你产品不包含什么；差的则向你暗示产品中包含了一切。为了对产品包含什么与排除什么做到心中有数，你必须做出某种关于这件东西不是什么的解释。市场营销部门经常认为所有的东西都包含进去最妙……困难的部分是需搞清楚不要做什么。在每次发布时，我们总是把清单上 2/3 的特性砍掉。如果我们把想要做的所有事都写下来，那将会是一份长达 1500 页的文件。现在很清楚了，想象性描述可帮你建立选择机制，而非创造机制。

例如，Excel3.0 的想象性描述对产品目标的陈述是：使 Excel3.0 成为“ 迄今为止创造的最有分析力的电子表格。 ”在说明过程中进行关于加减何种特性的讨论时，由这项描述引出了一个决定，不包括三维图形（3-D）和其他一些对象。当项目成员必须决定哪些特性要去掉时，他们更偏爱数学的或数据的分析特性，而宁可放弃那些支持图形功能的特性，虽然 Excel3.0 加上了一项绘图功能。

然而，产品的想象性描述也可能改变。当 Excel3.0 最终出品时，市场营销部门已把主导思想改成了“ 可以更轻易地拥有强大的能力 ”。这个市场营销主题集中在产品的强大能力与易使用方面。实际上产品对这一新的市场营销主题支持得很好，但主题的改变确实也造成了程序经理和开发员的一些混

乱。

编写说明文件

根据约翰·法恩的观点，说明文件应该与做菜的配方非常相似。他给出了苹果饼的配方例子——或者说是最初的说明：“饼的做法因人而异。这里介绍的苹果饼具有当今世界所有饼的优点。它的关键配料是掼奶油、苹果、面粉、糖以及其他一些东西。它应该大得让人人都能满意。比起版本 1，我们会大大改善其规格。制作饼的方法是：混合苹果、糖、柠檬汁，把它们放于面包片中，烤得足够热以确保饼能烤熟。”[8]

对开发员、测试员、用户教育人员以及市场营销人员来说，概要说明文件的作用就像烹调书一样。说明文件在产品小组的所有组员之间，产品小组之间以及产品小组与管理部门之间起着传递产品的设想与要求的作用。这些机构是说明文件的“读者”。因为项目对开发、测试、用户教育以及非英语版本的所有进度安排都基于最初说明中的信息，所以说明文件必须足够清楚地描述特性并赋予其优先级。这样项目才能建立起有意义的进度表。我们也应该注意到，一些程序经理感到越来越大的压力必须在开始前使这些“配方”说明尽可能完备。产生这种压力的一个关键原因是产品构件之间越来越强的互相关联性，比如 Office 的情况（见第 6 章）。结果是对复杂产品，程序经理建立的说明文件可能非常详细与冗长（1000 多页）。

因为说明建立在想象性描述的基础上，它就比想象性描述文件包含了更多的信息。概要说明文件应由一个总经理作出小结，以清楚地概括产品的设想。说明文件也应包含下列各项：能用一句话表示的项目目标，关于产品是什么与不是什么的清单，对顾客的定义，对竞争产品的定义。[9]它应描述出产品对系统的要求，包括操作系统的版本，最小内存要求，硬盘空间，处理器速度以及图形。另外，说明应列出产品对第三方（如打印机驱动程序）、其他组和构件（如视觉界面设计组或 OLE）的任何依赖性。然而，文件主要的内容还是对功能的描述，讲出产品特性是想做什么，而非怎么去做。它也将特性划分，归属于普通的功能领域与子领域，范例可见表 4.2 与表 4.3 描述的 Excel 的情况。

特性描述通常都遵循一个标准格式。在概要说明中，每个特性有一到数页，描述它将如何工作，外观怎样以及从用户的角度出发如何与用户交互。如果特性有一个用户界面（如面板、菜单或按钮），那么说明就应该包含一张示意图，显示出界面该是什么样子。设计一些方案演示出用户将如何使用特性，对弄清如何才能有效构造特性尤其有用。

概要说明也涉及特性努力去解决的那些难题。（特性与特定用户难题及用户行为之间的联系将在下一原则——特性选择与分级处深入讨论。）例如，Word 的 Windows 版有一个特性，即可以很快地存储一个文件，即使该文件包含有其他应用程序所建立的文件；这种存档行为在桌面印刷与书籍印刷中是很普遍的。概要说明是如何描述这一特性的呢？我们举例如下：[10]

表 4.2 Excel 说明中的特性领域

Add-ins 加载宏	Macro 宏
Benchmarks 水准基点	Mail 邮递
Calculation 计算	Name 名字
CBT	Outlining 分级显示
Charting 图表	Print 打印
Configurations 配置	Projects 项目
Data 数据	Q + E
DBCS	R2L (Righttoleft) 右到左
Display 显示	Search 搜索
Edit 编辑	Setup 设定
Format 格式	Startup 启动
Formula 公式	Toolbar and objects 工具条与
Function 函数	Userinterface 用户界面
Help 帮助	Utilities 实用程序
Links 链接	What-if 假设分析
Load/save 装载/保存	Window 窗口
Lotus Compatibility	与 Lotus 的兼容性

资料来源：1993 年 4 月 13 日，与测试经理马克·奥尔森的谈话以及不知日期的微软文件《Excel 测试领域》。

表 4.3 Excel5.0 的特性子领域范例

领域	子领域	领域	子领域
图表	普通	图表	编辑系列
	箭头	（继续）	术语
	坐标轴		文本
	复杂参考		优选
	可拖曳图标点		图表编辑
	主格式与格式		雷达图，系列线
	图形重叠		不限量股价组合图
	格式式样		曲面图
	图表库		三维图表
	数据点标签		三维柱状图
	图例	链接	苹果的事件
	数据系列标识		DDE
	图表建立		OLE
	重叠的坐标轴标题		印刷\签名
	图片		普通
	绘制区		

资料来源：1993 年 4 月 13 日，与测试经理马克·奥尔森的谈话以及不知日期的微软文件《Excel 测试领域》。特性所解决的难题

书籍出版的难题

以专业方式出版的较长的文件一般由一组人共同创造。组员们同时

在文件的各部分展开工作，而出版是一次完成的。因为“书”只出版一次，所以在出版之前，“书”要被印刷许多次以供不断进行复查。这样，对各部分进行合并与印刷以造就一个主文件的工作重复了多次。

开发产品的难题

通过“包含域”的合并形成主文件是不可靠的。合并要求在每一个层次上重新存储数据。这样，主文件就会含有复制的支持文件的所有数据。因为主文件最后会是一个数以兆计的庞然大物，存一次要花很多时间，所以你一定想尽可能少地去存它。但如果不是每次都达到完全再同步，你又冒着使主文件与支持文件失去同步的危险。

概要说明文件中的特性描述对用户教育人员编写文档以及地方化人员撰写非英语版本都是至关重要的。程序经理应尽量避免改变反映特性的用户界面中的术语（如对话框或菜单），因为要在非英语版本中作相应的改变非常困难。当为用户界面写说明时，为了方便非英语版本的编写，程序经理要密切注意单词长度、缩写、数据格式、货币格式、文化差异、排序顺序、纸张大小、左右顺序，以及其他可能在不同语言和国家间有差异的因素。

程序经理负责协调并“写下”说明

很明显，在产品生命周期中，程序经理处在不断转动的信息之轮的轴心上。他们写下说明，帮助定义用户界面以及负责协调内部运作等问题。根据约翰·法恩的说法，他们会考虑以下问题：[11]·这项特性的要点是什么？

·用户如何使用该特性？

·这项特性有意义吗？

·该产品中或微软的其他产品中有类似的特性吗？·开发完成的特性确实是我们所计划的吗？

·有哪些问题被遗漏了？

·组内的交流令人满意吗？

正如第 2 章所述，一位程序经理对开发员而言，是个领导者、协调者、使事情易办的人，而不是他们的老板。因为开发员通常很深地介入到说明的创造之中，所以说程序经理写下了说明而非单独创造了说明是非常准确的。麦克·康特曾这样讨论了程序经理的角色：

当我说程序经理写下说明文件时，我给了你一个错误的印象。也许更准确地说，程序经理只是记录了说明的文字；说明其实是整个组用一定方法写成的。事实上，程序经理的一个困难任务便是力图使每个人都认为设计说明他也有份儿……开发员对设计介入非常非常之深。程序经理凭空造出说明，打好它并交给开发员，而说明居然能付诸实现的情况是极罕见的。更经常的情况是，不断地重复……[以及]争论不休，这可能正是由于开发员对特性应如何实现有自己的看法。一部分是因为他们会给出成本信息：“好吧，如果你要一个单独的窗口类，那要花 64 周。而如果我们不要，如果我们用对话框，那就只用 16 周。也许我们应该用对话框。”同样，如果你与一位开发员交谈，他说：“喔，那会花 1000 周。”也许他真正的意思是，“我认为它是个笨主意。”当对未来进行预测时，开发员对特性的偏好通常会产生惊人的影响。

对重要产品如 Office（包括 Excel 与 Word）或 Windows 的说明，通常都需要 10 至 20 个程序经理，在一名“组”程序经理的协调下完成。一些程序经理工作于产品的某些部分，如购买的拼写检查器或外语版本。克里斯·彼得斯这样描绘程序经理和开发员之间的交往：“[程序经理]力图……使开发员克尽职守。要知道开发员并不是为[程序经理]工作，而且程序经理也不在任何确定的方面对开发员有任何控制，明白这一点很重要。换句话说，如果程序经理在说明中写了什么，开发员可以按也可以不按他写的那样去实现……程序员读[说明文件中的特性描述]，理解代码，然后做出与之非常相近的东西。经常地，在过道中前前后后都有许多人在谈话，许多人在尖叫与咆哮。”

管理说明的工具支持

为了在说明演进期间（对特性的功能添加细节以及特性集的改变）进行联络和对其管理，小组成员们大量使用电子邮件。他们以前使用 Word 来建立和更新一个大的说明文件，但现在通常使用 Excel 的电子表格来管理说明的内容。说明文件体积也不断膨胀，现在即使是每个说明领域或子领域都有一份单独的大文件。电子表格列出下列各项：

- 特性数目。
- 特性标题。
- 与主要程序经理之联系。
- 开发联系，用户教育联系，市场营销联系，国际联系等等。
- 对每个特性的进度预计和真正使用的时间。
- 测试发送文件信息。
- 说明的更改（修订）日志。

当然，说明的用户可以把信息从电子邮件中移至 Excel 电子表格中，再移至用 Word 写的真正文件中。电子表格的一个替代物是微软内部开发的说明工具，专门处理一定结构的 Word 文件。程序经理主要使用这些文件说明及电子表格说明。

在一些较早的项目上，微软开发组利用一种工具来处理对说明文件的‘一致控制’。这种工具在一定时间只允许一位用户更改文件，虽然许多用户可以读它。然而，该工具由于要求每一位程序经理的机器上都有大量硬盘空间从而被证明为不必要。现在，微软处理这种进入权控制的方法是简单地让人们通过电子邮件向“拥有”特定特性的程序经理表达意见。只有拥有该说明的程序经理才能实际上对文件作出更改。

构造原型

当程序经理具体说明一件新产品或一个新版本时，构造原型便成为他们的基本行为。这从许多方面来说都使开发前测试成为可能，尤其是在可用性方面，并且有助于对与用户交互情况作出更好的理解，它也能使产品说明更紧凑。布兰德·斯尔文伯格注意到程序经理即使对系统软件也建原型：“他们绝对会建原型，或者他们会与开发员一起建原型。实际的……编写说明，决定这个或那个特性是否包括进来，正由开发负责转变为程序管理负责。”

Visual Basic 是选择用户界面时的建原型工具。程序经理在说明过程

中，普遍地对大量产品运用该工具。另一个流行的建原型工具，画笔（Paintbursh），使程序经理能创造产品的计算机屏幕模型。这些静态的屏幕图形用于对现存产品作微小改动尤其适宜，比如改变一个对话框的外观。如果程序经理需要从零开始创建原型或有一个实际在工作的原型，那么VisualBasic 更有用一些。（一个与 Visual Basic 相类似的工具是苹果的HyperCard,Excel 的程序管理组在过去曾用之于建原型。）然而,VisualBasic 不能在 Macintosh 机器上运行（虽然一个 Mac 版就要出台了），并且如果改变是渐进式的话，有时会费时费力太多，程序经理因而不愿进行。 Visual Basic 用于对话框原型是出色的，但在其他方面则不是太好，像设计新的象征型图表。

死板的说明变成有生命的文件

说明不应过于详细以至限制了发明创造。微软创造的产品要求有所发明和有创造适应力。正如克里斯·彼得斯所评论的，说明文件必须足够灵活以适应这种演进：

> 说明绝对是随开发过程而变的，说明是有生命的文件。它是对发展中的特性作出的最好的描述。它给出了让开发员作出发明的大量空间。人们希望它是关于重大问题的思考，如这个特性如何与其他特性相互影响。但和大的国防项目比起来，我们在这儿做有些事的方法不大一样，那就是由于该工业变动如此巨大，以至说明必须随着产品开发而变。你在产品开发时得到了如此多的信息：你从你的顾客那儿得到信息，你从你的竞争对手那儿得到信息。你得知有的东西很容易就可实现，或者你实际上已实现了一个特性，并且突然意识到，只需少许扩展，它就可用于产品的全部其他领域。在开初，这是你未料想到的，现在，你清楚地意识到了，你发现了这一点。如果你不知为何仍然说：“好吧，我们有说明摆着呢，我们要把‘写代码的人’——这对开发员来说是个带有侮辱性的词——限制在这个范围内，他们应该按说明来做。”这样一来就会使士气严重不振，开发出的东西会一塌糊涂，并且不会按时完成。

在项目进程中，说明文件的早期版本会有相当大的增加与改变。例如，Excel5.0 的最早的说明在开始编码前是 1500 页，然而当产品推出时完整的说明长达 1850 页。（微软的一些人员认为这太长了。）Word 6.0 最初的说明长约 350 页，完成时的说明大约有 400 页。最新 Office 的说明文件的较早版本长达 1200 页；微软最近没有印刷其说明文件，但可以肯定的是，该说明文件已经大到不能用一个单独的文件容纳了。

不光是规模上变大了，在第一个主要里程碑阶段开始后，30%的说明文件改变了（或“剧烈重组”了）。麦克·康特论述如下：

> 如果你在我们开始开发前读了说明……然后在项目结束时又回过头去读了以前的说明，我敢说你会发现至少 30%的说明是不准确的……一个特性的精神也许还在，但也有可能改变了……当我们开始时，我们认为这个对话框会需要五个选择项。在我们开发了一段不短的时间之后，我们意识到其中三个造价高昂，但我们需要另外两个选项……这样，当你真正完成了工作时，那已经是不一样的特性了。

康特还提到一个项目会削减掉最初的产品说明中 20%—30%的特性。

现在，在第一个主要里程碑阶段开始前，许多项目就力图对说明早期草稿中所有的特性作详细描述。然而，在实践中，微软的各个组看起来只对大约 70%的特性做此工作。在开发的过程中，特性的准确细节以及为实现特性需作出的取舍往往并不清楚，于是程序经理有意让一部分说明不完整。当项目继续下去时，开发员提供关于特性的表现，实施的可选方案，以及表现上作取舍的进一步信息。克里斯·彼得斯强调，非看第一手的源代码不可，这样才能作出正确的特性抉择：

> 说明总是不完整的，作为一名开发员，你也总希望它不完整。程序经理永远不看产品的代码，他们也不想看，那不是他们的工作。但是要想正确做出特性却要求你去看代码，并对代码作出那些工程意义上的取舍。另外，我们工作于其上的特性是极其复杂的，这样，再次地，如果没有实际看见事物是如何运作的，就不可能准确知道特性将会是什么样，我们已经在 IBM 中看到了说明的一次性完成所带来的恐惧，因为没有人聪明到那个份上。应该做的是看看说明是什么样的，修改它，使之更加完美——使之成为人们想要的东西。[12]

项目有时加上在说明早期版本中完全没有的重要新特性。娄·帕雷罗里，Windows NT 的软件工程经理，对 NT3.0 项目过程中作出的变化评论如下：

> 我们要出品的东西必须有以下功能：以窗口方式运行，可兼容，是 32 位平台，是功能强劲的技术，支持网络互联性以及高质量。真正拖长了时间[产品最后版本]的是加入的额外功能。NT 产品现在将能对 Macintosh 机器提供服务，这样你就能把你的 Mac 客户机联到一个 NT 服务器上。你能把 Mac 文件放到其上，把文件夹放到其上，点一点 Mac 机上的图标，就可以看到 NT 机器上的文件夹。这在两年前的产品中是见不到的……然而在此处的产品中却有了。所有通过电话线连接的远端存取服务器都包含在此产品中；我们将 ISDN 驱动程序纳入产品，这样你就可把到达你房屋的 256k 线插入你的 PC,然后通过 ISDN 与你的服务器交谈。我们以前从未动过把此项功能加进去的念头，然而许多压力共同使这项功能得以加入。这样，就有许多“让我们加上这些好东西吧，”的要求，例如 Mail，两年前的产品中没有 Mail，但现在我们正把它推向市场。

然而，开发员需要尽可能早地知道说明中的哪部分是在变化的，哪部分是稳定的。他们不愿投资时间于开发、改进或使用那些在项目发展中会被删掉或改变的特性。但很遗憾，一个大项目的开发员从来无法完全掌握特性所处的地位。微软对此的办法是交错工作，首先集中于那些没有什么用户界面的特性。这些特性不大可能改变，因为在完成开发前不必去了解用户对它们有何反应。

例如，开发员可能首先工作于这样的特性，该特性增加一个字处理程序支持的字体的数目，或“优化”磁盘上储存文件的内部数据格式。这以后，他们可能再工作于常用的特性，如绘图用的工具条。然而，很晚才决定用户界面的外观可能会对文档的及时生产产生不利影响，并且会缩短可以用来做

可用性测试的时间。乔恩·德·沃恩回忆说 Excel4.0 开发中出现了一些问题就是因为他们将说明稳定得太晚。 Excel4.0 开发阶段始于 1991 年 7 月 8 日，第一次发布——在美国的 Windows 版——是 1992 年 3 月 27 日：

> 短期项目的决定做出得非常晚——如此的晚以致……我们不得不马上开始编码。这是一个错误，因为它给整个事业单位带来了许多过度的紧张……从严格的开发角度来说，在进度表完成很久以后，我们才停止作出关于实现的许多决策。这就意味着进度表不像它应该做到的那么准确了。因为我们有些决定作出得非常晚，程序的一些部分就不能与其他新特性同步。例如，因为工作簿被设计和实现得如此之晚，就不可能照最后设计的那样实现 Lotus 式的 3—D 电子表格装载。这就造成了大量与[工作簿] 装载有关的较晚发作的错误，而这本来是可以避免的，通常说来，这些错误倾向于是些概念性错误，而非简单的错误。
>
> 编程组在项目刚好结束处做了大量小心翼翼的改动，他们的活儿干得不错。因为设计决定作得晚，几乎不可能让测试组有时间搞出像样的测试方案……缺乏一开始的设计期在事业单位中的功能组之间造成了紧张。程序管理和开发之间的关系受影响最大。因为要在短期内作出如此多决定，一些程序经理就只能采取专断的态度，这使开发员感到程序经理不是“ 我们这边的人 ”。[13]

文档的外观固定

如果项目的大多数用户界面细节完成得过晚，那么用户文档就可能不完全或不准确，因为为改变文档而留下的进度时间不够了。为了防止这个问题出现，微软设立了“ 外观固定 ”目标，允许用户教育人员与最终调试和测试员平行地编写文档。但如果开发员老是改个没完，或者直到项目结束也未决定用户界面特性，这个目标也就变得几乎毫无意义。

除了付出最大努力在相对较早时就固定产品外观外，开发员还常在较晚时改动那些影响产品外观的特性；用户教育人员别无选择，只能在项目快结束时重做文档以适应变动。这反映出在微软中，开发员比起测试和文档编写人员来，处于支配的地位。克里斯·彼得斯承认了这个问题：“ 保持外观固定是世界上最难做的事……所有的人员通常在最后都没什么事了，而用户教育人员却经常还在叫苦连天。这是典型的战斗……你在编写文档时经常跟不上形势……我的意思是，文档[基本上] 还正确，而这些人却在项目快完成时，不得不一天工作 12 个小时，再搭上许多周末……[他们]必须重做一大堆事情。 ”

灵活的说明文件

在微软（以及其他地方），产品拖延的一个基本原因是在开发过程中不断地对说明文件或产品的特性集进行改动。经常发生的是，一旦一个组同意改动说明文件中的一些东西，那么所有的事情都变得从属于改动了，以前的说明和进度表变得无人理睬。麦克·康特详述了说明的改动会怎样导致进度的拖延：

> 当你回顾为什么[进度拖延]会发生时，你会发现有一大堆原因，但

其中有一些是基本的。一个原因是不断地改变产品定义——如果每次出现了什么新东西你就说：“好吧，我们要改一下计划；我们要加上这个新特性”——这就对产品产生了巨大的影响。每一个改变都要花时间去实现，[于是]你忽略了对整个产品的考虑。它以这种方式影响人们的决定：“好的，如果那是可以讨论的，那就让我们把所有其他一切也加以讨论吧。”另外一个原因是你有一个产品开发进度表，告诉你从何时开始开发，但你对应在何时结束只有模糊的概念，到该结束时你却不得不调试产品了。实际上到现在，这还是大多数应用软件的编写方法。

微软的说明文件相对较灵活。然而，一些项目，当项目进行到了一定点，例如，40%以后，建立了对说明中重大改动的控制。他们允许此后的那些小改动主要是为了让特性正确工作，而不是为了增加新特性。这类控制在诸如 Word 或 Excel 这样的产品上较易实现，因为它们都处于多年的版本更新周期中，每 12 至 24 个月就会出现新版本。如果发现这些项目在开发中应做一个大的改动，他们会把改动延迟到下一版本，而不是去破坏现在的进度表。康特对 Excel 组在一定点后，如何避免了对说明做出大改动颇为自得：

> 它也意味着我们力图做到非常有纪律性，不到说明固定和有了一个包含我们要做的所有特性的平衡的开发进度表后，我们实际上不开始开发。想要在开发过程的中间加上特性，那对我们而言将非常非常困难。这样……如果我们在产品开发的中间知道了新东西——太晚了，等下一次吧。现实情况是，总是有许多小玩意使你滑入机会主义的想法中，想把它们加进去，但那是困难的，因为它们必须经历整个过程，而有许多组要被牵涉。

特性添加太晚不光造成开发延迟，它还压缩了测试可用的时间。然而，愿意让说明演进却有助于产品在市场中获胜。这对有较长开发期的产品尤其有用，因为市场需求与公司需求随时间而变化。要想对说明演进很大的项目保持控制，需要格外强的技术领先性。在一定点，经理必须对任何改变说不，或变得对同意往产品中增加的东西分外挑剔。 Windows NT 为我们提供了一个这种灵活性与强有力控制的范例。

我们在第 3 章提到过，NT 是在 80 年代末开始开发的，用以替代 OS/2，后者是微软与 IBM 协同开发（至 1989 年）用于替代 MS—DOS 的。微软从 DEC 雇了戴夫·卡特勒来管理该项目；他对新的 Windows 类型的界面并不熟悉，但他写了一份说明，然后写了一份为公司市场生产可靠操作系统的详细设计。与 IBM 的分离和 Windows 3.1 面市的爆炸性效果说服了比尔·盖茨、保罗·马里茨、史蒂夫·鲍尔莫以及其他微软的高级人士，要把 NT 带入大众市场的 Windows 家族中。特别地，微软的最高管理层希望 NT 吸收 Windows 3.1 的图形用户界面并与 Windows3.1 的应用程序完全兼容。这些和其他的要求需要产品作出极大的改变。1993 年 NT 出品前，在原始的详细设计后，NT 经历了三次重大设计改变。卡特勒和他的组目睹了所有这些工作并把项目相对保持在控制之中。[14] 在此期间，他们还在抵制对产品作更多改变上展示了相当的纪律性。例如，娄·帕雷罗里回忆起决定不支持 DoubleSpace 数据压缩，因为加上这条特性会使他们没有足够的时间来彻底测试产品：

当考虑像 DOS6.0 那样做 DoubleSpace 时，我们只是会说不。在一个仅测试就要花 3 个月的文件系统中，我们没功夫做那个。想一想，如果花 6 个月开发，加 3 个月测试，然后——你会相信只用过 3 个月的文件系统中的数据吗？我不会。在文件系统中，我们现在用的是像 NTFS 这样的东西，这是我们最近的文件系统，我从 7 月开始就在我的机器上亲自使用它，从没有丢过一份文件。我觉得自己有充分的信心来告诉大家，可以在它上面运行。我永远也不会告诉人们，可以在一个连我自己都不曾用够 3 个月的文件系统上运行。

理解与安排特性优先级的困难

当微软的产品规模变大并且功能日趋复杂时，特性的数量也显著增加了。如此一来，项目就需要用一种方法来对特性赋予优先级。一个办法是集中于下一个产品版本中会使顾客抱怨电话最少的特性，这些电话平均每个要花微软大约 12 美元。程序经理也读市场研究资料。麦克·康特描述了 Excel 组如何对付该难题，即决定什么是用户最想要的特性：

> 特性的阵列让人晕头转向……在产品增加和该工业成熟的同时，出现了几股潮流。首先，这场无意义的，非常像军备竞赛的特性战结束了。在以前，你不停地多加入“ 2% ”的特性，尽管几乎没人用到它们。我们原来就很清楚，人们将不会按特性的多少来挑选和使用产品。然而，软件工业在以前一直着力增加新特性，因为这曾是顾客想要什么的一个重要尺度。但那种潮流在变。我们从五年前起有了市场研究；如果你在那时问人们，他们在电子表格中想要什么，他们会说是功能强大。但如果你今天再问，功能强大就只是第四或第五重要的了；最重要的是易于使用。当钟形曲线延伸出去时，人们对特性的兴趣就减少了，用在电子表格上的时间少了，对其在情感上的喜爱程度降低了。对他们来说，那就像传真：只想让它工作而不想学它。产品已经变得功能很强了，我们必须做的是使它们更好地工作。

程序经理以前常按独立的特性思考产品和组织说明文件，这与他们如何对开发员分组相一致。当说明在长度和复杂性上都大大增加了时，这种特性分割法就成了思考产品的一种坏方法。程序经理和市场营销部门对产品描述作评价也变得越来越困难。康特回忆起从市场营销组得到关于详细说明文件的信息反馈时遇到的问题：

> 我们以前通常按产品的功能领域在说明文件内分类与打印，这样我们就会得到数据库特性、兼容性特性以及分析特性。我们会大致地按开发员如何分组把它们分开。但要评价说明则非常困难……我会将说明交给市场营销部门并说：“ 请给我信息反馈，告诉我这特性集是我们想要做的吗？ ”市场营销部门可能读也可能不读说明，因为那实在是太长了。即使他们确实读了，他们也会迷失在其中，因为读那玩意儿可是件需要超级技术的事情。如果他们对说明确有评论，……他们也是说：“ 啊，我们认为这个对话框排列得不对。你真该把检查框放在左边， ”或是诸如此类的东西。这不是作为程序经理希望得到的信息反馈。你希

望市场营销部门的伙计们回来告诉你：“ 等一下，这产品对律师没用，而他们是我们市场营销策略的关键， ”或是“ 它不会对小企业有用的。 ”……你真正希望从市场营销部得到的高水平信息反馈应是像这样的：“ 这与我们的策略不对路。 ”但读一份像说明那样庞杂琐碎的文件，他们很难达到这样的要求。这就有点像读一份汽车的零件清单，然后试图弄明白这是辆赛车或是其他什么车一样。

程序经理也尝试着召开大型会议，开发员、测试员、用户教育人员、产品支持人员、市场营销人员以及程序经理统统出席，讨论说明与评价特性的取舍。康特回忆了在大会试图弄清问题与作出决定时碰到的困难：

现在你有一个大组，你不能围绕着由 300 个杂乱特性组成的团体来集合。必须有一个总的想法……一个有力的主题……每个人都能明白它……组越大，我们就会在更多的特性上争吵，整个过程就越失去理智。于是我们走进一间大屋子，大喊大叫地讨论事……不管是谁，叫得最响通常就可以把他们负责的特性加进产品。如果特性是……一些新的三维图表或一些很“ 新潮 ”的东西，那它就会被加进产品，如果特性无疑是很重要的但不是那么“ 新潮 ”，就没有人理睬它。打印是非常重要的，但它并不是那么光彩夺目，于是人们就不愿做它。房间里也很容易有放冷枪的。一个聪明的家伙会说：“ 那是个蠢主意， ”于是所有的人都会对那主意失去兴趣。如此一来，对我们设计未来的版本而言，这样的会议过程就不是让我们可以宽心的了。所以我们决定：“ 好吧，让我们稍稍反转一下这个过程，让我们想都不要想特性。 ”

最后，当程序经理把新产品的精髓展示给比尔·盖茨时，他们面临着一个挑战。康特提到，依靠一份数百页长的初步说明是不行的：“ 实际上，Billg[指比尔·盖茨，这是他的 E-mail 地址]的程序复查对说明表示出一些不满。问题在于，‘ 我们的小伙子老板非常聪明和忙碌，我们应怎样与他联系，告诉他在产品下一版本中我们要干什么以及如何干——让他评价我们做的是不是件聪明事？ ’我们能给他说明，为特性辩论，像我们通常做的那样，但那是没有什么价值的。我们很清楚，我们做这类评价工作的体系糟糕透顶。 ”所有这些因素——众多的特性，需要来自其他组（尤其是市场营销组）的关于特性评价与优先级安排的信息反馈，以及需要在产品精髓问题上与组和高级管理层进行交流——都有助于制定基于行为的计划，以及其他将在下一原则中讨论的技术。

原则三：根据用户行为和有关用户的资料确定产品特性及其优先顺序

在微软的早期，像麦克·康特所说，产品包含什么特性通常依赖于谁在设计会上叫得最响。产生潜在特性的主意很多，但在为下一次版本进行开发的时间中，项目完成不了那么多；这样，最困难的问题就成了什么特性应被排除在外。意志坚定的开发员或程序经理往往使那些顾客实际上并不需要或没法学会怎么正确使用的特性包括进了产品。为了解决这个难题，微软采用了一个原则，在开发进度表中将特性选择与优先级安排建立在公司的被称为

“基于行为制定计划”的技术之上。

基于行为制定计划在概念上与会计中的“按活动核算成本法”很相似。后者跟踪公司做了什么（例如在设计新产品时一个多功能组发生的费用）而不是跟踪正式的部门或传统职能来弄清楚费用支出。虽然许多其他公司也以相似的方式深入研究用户，但看起来微软是独立地对自己的计划技术命的名，与成本会计知识没有关系。说起微软中基于行为制定计划法的推行，麦克·康特是三个最早的倡导者之一，另两位是山姆·霍布森与克里斯·格雷姆。（当提出此概念时，山姆·霍布森是 Excel 的一位产品计划经理，克里斯·格雷姆是 Excel 组的程序经理。）

基于行为制定计划法从对用户行为，诸如写信或做预算，做系统研究开始。然后，根据特性在支持重要的或经常的用户行为上的重要性来对其进行评价。这样做的优点是对特性取舍的更理性的讨论，对顾客想要做什么的更好的排序，对某个给定特性是否方便了特定任务的更集中的辩论，可读性更强的说明，以及在市场营销、用户教育和产品开发中更好地同步。

特性选择和优先级安排中的基于行为制定计划

基于行为制定计划法中的关键点在于按用户行为、产品特性以及行为和特性之间的内部联系来分析产品。程序经理和产品计划者把产品试图支持的用户任务或方案分成大约 20 个“行为”，然后他们努力把行为（以及任何子行为）映射入微软的现行特性和竞争对手产品的特性中去。他们也把行为映射到不同的顾客形象——如是新手还是高级用户，或不同的市场部分中去。

当说明产品的新版本时，基于行为制定计划法帮助程序经理和开发员集中他们的精力与创造力。在实践中，像 Excel 那样的项目争取在每个新版本中加入的主要行为不超过四个。绝大多数特性直接映射入这些行为之中。Excel 组甚至列出每个特性支持多少行为，以之作为说明文件的一部分。该做法使项目可以按特性对用户的价值来对特性分级。麦克·康特概述了 Excel 组如何评价行为，使下一个产品版本只集中在少数几个特性上：

> 所有这些要点在于，对什么是在下一次发布时我们要集中于其上的重要行为，一个事业单位基本上要形成统一意见。让我们挑四个行为——我认为四是个好数字，没别的意思——然后让我们安排它们的优先级。它们就是 Excel 下一次发布时我们的工作焦点。我们仍然有那些大会，我们聚在一起，互相叫喊。但现在，不是争吵我们要做什么特性，我们争论的是做什么行为。我们会说：“我确实认为重要的行为是从 1-2-3 中切换，这儿是我的资料，这儿是我为什么认为该行为在战略上重要的理由。”另有人会说：“不，我认为基本的用途更重要，因为更多的人使用它，并且我们应该集中在新用户身上。”……
>
> 当我们有了一份分好级的清单，内容是四个我们想在 Excel 下一版本中集中于其上的行为时，这个过程方告结束……现在我们回到我们有的那份长长的特性清单处，并说：“好了。……我们有特性 1，它对我们的行为支持得如何？啊，让我们看看，它有助于建立模板，它有助于方案——这家伙得了 2 分。让我们看看特性 2，它有助于建立模板与方案，它还有助于打印——这东西得 3 分。”当然，我们从未这样简单地处理过，但我们确实讨论过为每个特性依次打分。

这样“打分”的一个作用是，它促使程序经理和开发员都起来竞争，使他们的特性支持尽可能多的行为。这是良性竞争——对用户有益，而且对说明的生产率及开发努力也有益。打分也使选择特性成为可能，如哪个特性纳入产品，哪个特性被排斥，或哪个特性留到未来也许专门针对某些特定行为的版本去做。康特描述了基于行为制定计划法如何鼓励了项目创造更多的通用特性：

> 它有一些令人感兴趣的效果。首先，人们仍然必须考虑他们想纳入产品的特性，所以他们会想：“好吧，实际上我要骗骗大家。如果我哪怕只把特性改动 10%，它就会支持更多的行为，那么它就可以被纳入产品了。”而那正是你希望人们做出的一种欺骗，因为我们确实希望特性更通用一些。我们希望它支持的行为越多越好。这也给了我们一个对一大堆好主意说不的办法，当然，这实际上是作为一名程序经理最难办到的事之一。

为顾客行为而非产品特性收集资料

基于行为制定计划时，项目在计划阶段首先集中于行为，其次才是特性。程序经理和市场营销人员并不去思考和排除他们喜爱的特性，再围绕它们搞出想象性描述的草案。他们真正做的是列出一份顾客都做些什么的清单，然后把想象性描述集中于支持那些行为的特性。

例如，微软的产品经理与程序经理从人们到底如何使用电子表格开始研究，并且研究人们用电子表格做了什么特定的行为以完成特定的任务。他们注意到很多人用 Excel 来做预算。这些用户也遵循特定的行为：他们也许先做一个模板，发布它，再要求人们填它。然后他们也许会在模板上运行方案，再把结果送往老板处以供审计与评论。也许他们稍后会做一些敏感性分析，最后再合并信息并打印之。人们通常一年做两次预算。另外，还发现大公司和小公司都经历了大致相似的预算过程。然而，在小公司中，通常是一个人做预算；在大一点的公司中，人们在预算过程的各部分有所分工。

当微软的人员仅仅考察这一项任务——预算时，他们就发现了大约 20 种基本的顾客行为。不仅如此，他们还注意到说明的现在版本不够充分，没有包括行为中的几项。他们还发现预算中用到的许多行为与其他行为，与写费用报告或预测销售很相似，结果是，他们创造了一个更基本的叫做“数字累积”的行为。人们做预算时的另一个行为是从 Lotus1-2-3 中切换至 Excel,包括从 Lotus 转移文件和简单地切换程序。虽然经证明有 65%的 Excel 用户以前有使用 Lotus1-2-3 的经验,但 Excel 还是没有因考虑到了这一点就偷懒，仍然使从前的 Lotus 用户可以方便地做转换。

克里斯·彼得斯给出了基于行为制定计划法如何帮助小组集中其创造力的事例：

> 我们有这个叫做基于行为制定计划的技术，它基本上是指到人们和顾客中去，对他们到底在做什么进行研究。如果你问人们他们想要什么特性和改进，当用户考虑了各种可能后，你得到的回答往往就不能反映其真实心声。反过来，如果一家公司试图基于顾客发明特性，他们的想

法往往又拘泥于顾客的需要。我认为我们去试图弄清顾客做事的步骤比让顾客弄明白我们能编写或发明什么容易。于是我们走出去看人们如何做事……观察字处理器的使用，观察打印错误是如何产生的，或人们是如何做邮寄名单的……看起来大多数公司做邮寄名单的方法是：把所有顾客的姓名与地址列到清单上，把名单复印出来，做成标签，最后按正确的顺序一个个地来。如果你只去问人们在一个字处理器中想要什么特性，你得不到这个答案。

你常听人说：“我想要一个绘图包……”“那么，你要用绘图包来做什么？”“我要画一幅机构图。”……这样，只有当你在详细到一定程度上明白他们想要做什么类型的东西，只有当你把这些东西分成可以构造出不同任务的行为，只有当你照这种方法来创造特性时，你才能知道你[是否] 拥有了完成实际行为必须的所有子行为。并且你也能集中于创造具有最大效用的东西；换一句话说：“这个特性为七个不同的行为所用。”它使你能像以前一样有创造性，因为你不曾放弃过创造性，但现在你……用一种相当集中的方式。你不曾说：“告诉我你想要什么特性。”你做的所有事情是对到底发生了什么做基础研究。

以行为为中心对产品进行全面考虑

1991 年初，Excel4.0 的计划阶段开始使用基于行为制定计划法。微软也在 Excel5.0 中运用该法，并且在其他组中推广，尤其是在一些行为影响了或对特性有所要求的产品中推广。Word 与 Mail 组，以及一些系统软件组和语言组，都采用了该方法论下的一些形式。然而，为了让这项技术能更容易地演进，微软没有写下这套方法或试图把它变成一门科学。就像康特评论的：“它当然是完全突然地出现的……我们有意不把它写下来和使之非常科学化，这样它就可以在公司内传开的过程中有所演进……我们认为，如果行为[基本行为] 稍微超过 20 个，你就无法保持与行为的联系了，它就会变成一个大杂烩。它不是个非常科学的东西。”

基于行为制定计划法看起来已经在组之间传开了，因为它有助于项目人员以更全面的方式来考虑产品。这种着眼于整个产品的观点有助于在不同职能上工作的项目成员理解产品做什么，以及其他产品的相应特性如何可能支持那些需要或不需其他应用软件产品的行为。康特对这些技术的好处评论如下：

我们努力做的一件事便是对其他组大讲该方法论带来的福音。这样做有一些出于自私的理由。该方法本是设计成基本上只在“说明执行文件”——说明实际代码自身——上帮助我们的东西。但结果是，许多时候你看到了这些行为之一，如从 1-2-3 中切换，然后说“嗯，你知道吗？我们需要一个市场营销方案来支持它。”于是它就变成了这样的东西，我们现在突然可以由此出发，用一种更全面的方式来构造产品。我们也发现有一些跨应用软件的行为，如创造一个复合文件，那儿需要一个特性——不是我们的产品需要，而是我们要求开发 Word 的伙计们在他们的产品中加进去的特性……传统上，这类跨应用软件的特性会被归入到某些附录中，以方便砍去。

做市场营销研究以支持基于行为制定计划法

为支持基于行为制定计划法，从市场营销组来的产品经理与程序经理、开发员一起开展一些联合的研究，如指导对用户的研究工作。然而，一般说来是产品经理做大多数的研究。正如康特所述，这种研究现在一年到头都在进行：

> 传统上，产品计划研究是一件很专门的事，只在我们固定说明前五周内发生……当我们完成了，比如说是 Excel4.0 后，程序经理想马上做 Excel5.0。他们会说：“好，我们需要你们所有的产品计划研究，我们将在八周内固定说明，你们要做好研究给我们。”但是搞市场营销的伙计们正为发布新版本忙得不可开交呢，他们没有时间，当他们有时间来考虑时，太晚了，因为我们已经固定说明了。由于基于行为制定计划法的研究是更通用的，所以我们事实上一年到头都做此研究。

基于行为制定计划法也导致微软拓宽了用以支持产品开发的市场营销研究类型。开始时，程序经理对大家应做何种研究来收集关于行为的信息不加限制，虽然他们确有建议。康特就偏爱定量研究，不太喜欢定性研究或很小的样本研究，因为从定性研究中做推断最困难。关于大型研究项目有个好例子。研究中有一项市场各部分调查，做法是向美国家庭随机拨号，电话询问。市场营销组用这种调查方式问人们使用产品时有些什么行为，并且由此得知 Excel 需要一项新的列表管理行为。（第 6 章对顾客信息反馈与产品改良有深入讨论。）微软的组从那时起开展了花样百出的研究；现在，他们的研究从直接访问顾客到请求顾客使用能记录下他们所有击键的“工具版”产品，无所不包。康特说到：

> 研究从访问顾客到专门的研究项目，变化多端。我们确实对用户做些纵向的研究。我们会每三到四周访问他们一次，看看他们使用文档与产品顺手与否……我们有一种行为叫基本用法，它是一种假行为。它是翻滚查找、选择和进入公式，以及那些人们在做一个预算或其他什么事时老在做，实际上并非必要的行为。为此，我们有一个 Excel 的专门版本叫工具版，它可以记录下用户的每个动作，这样我们就能回溯过去，分析这些动作。

一般说来，基于行为制定计划法给了市场营销组一种机制，使其可以更明确地影响微软的产品演进。正像康特总结的：“我们[程序经理]发现能从他们[市场营销]那儿得到比过去好得多的信息了。我们现在得到了真正的信息，‘喔，这个行为不是那么重要，不要集中在它上面。这个行为对我们很重要，集中在它上面。’而不是仅仅得到一张表，上面列出了他们希望在发行展示会上看到的为其所喜爱的特性。该方法也使我们得以从一些不具备资源进行美国市场营销的合作者[微软的组中]，像[微软的]国际部那儿得到更好的信息。”

做相关调查以研究群体行为

虽然基于行为制定计划法可以抓住不管是个人还是群体的行为，微软的

项目还用一种叫“相关调查”的技术来帮助理解在组或小组中工作的用户的行为。相关调查是在 20 世纪 80 年代晚期，在 DEC 中逐渐发展起来的，而最早则是人类学研究者用以研究群体和社会动力的技术。Office，Money，以及 Cario 这些产品，都用了相关调查来帮助完善现存的说明与特性，同时也用它来探索自己应支持的新的行为类型。

相关查询包含一系列的纪律、训练及态度，比基于行为制定计划法复杂得多。基本的模式是个浸入的过程。首先，项目小组选择一个焦点，比如用 Office 产品创造一个复合（文本与图形）文件。其次，小组成员访问顾客点；这不是真的采访，而是更偏向于仅仅观察顾客及其行为。第三步发生在顾客点采访中或其后，观察者做大量的笔记，绘制图表或模型，从物理环境、工作过程以及行为流程、依赖等方面记录下行为。

最后，小组成员回到微软开一个数据提炼会，一打左右的成员一起讨论这次顾客点采访。在数据提炼会上，人们用成千上万标了号的不干胶便笺做成一幅“姻亲图”，试图抓住他们观察到的行为及行为间的联系。顾客点采访通常持续 2 到 3 小时，其后的数据提炼会开 2 到 3 小时。项目在 3 到 4 周内指挥进行一系列的顾客点采访与数据提炼会。麦克·康特总结了该过程：“这简直就像‘不去不明白’的事：他们达成了一致，但他们自己也不完全清楚是怎么达成一致的……没有去做[相关调查]的人就达不成[一致]。”

康特还提到该技术的其他一些限制：“问题在于相关调查没有真正的产出……没有快速得能用一页纸写出所得结果的产出……其他人想吸收它很困难。所以想要把它推广是费力不讨好的。”另一个问题是相关调查非常费时间，还可能要求项目代表在 3 到 4 周内参加一系列的访问。一些结果也是模棱两可的；例如，对 Office 组的相关调查研究导致在关于“找到东西的地方，打开东西的方法，做新东西的地方”的用户界面中出现了概念分离。然而康特指出，在可用性试验室中，用户在区分打开与找到对象时有困难。

原则四：建立模块化的和水平式的设计结构，并使项目结构反映产品结构的特点。

微软产品设计中的一个关键概念是产品的基础结构——尤其是生命周期短的应用软件——应随项目的进展变得更加单一（而不是错综复杂）。当开发组构造产品的第一版时，他们更多地使用分级式结构，好为产品设计规定出一个最初的架构。然而，随着时间推移，他们向单一的结构迈进，以使项目能集中于特性开发。项目需要逐渐地增加和删除特性，随着时间改变和发展特性（比如从基于字符的界面向图形用户界面过渡），以及增加产品间特性表现和运作的一致性。就像第 6 章将要详述的那样，微软越来越强调不同产品间的特性共享。共享有助于使不同产品的“性能与感觉”都统一协调起来；它也方便了需要不只一个应用软件的用户，减少了代码的重复书写，缩小了单独一个应用软件的规模。

正如第 2 章所述，微软力图使项目结构反映产品结构的特色——它用特性组织产品，用特性小组组织项目。这种方法使得每个人都容易明白小组是如何与整个产品相关联的。项目从规定概要说明开始。概要说明的形式是一份已确定了优先级安排的内容清单，涉及产品下一版本将要开发的相对独立的特性；例如， Office 有一些诸如编辑文件、操作表格、格式化数据的特

性。这些特性并非完全独立，因为编辑文本特性依靠打印特性来打印，依靠文件管理特性来存储文本。不过，相关的特性组应是足够独特的，这样项目才能由分开的特性小组加以开发。

程序经理和开发员把项目分成特性子集，再将之分配给每个特性小组，让他们在 3 到 4 个主要的内部项目里程碑中进行生产（见图 4.2）。对于 Office 软件，微软还把项目分成 Excel 特性，Word 特性，PowerPoint 特性以及对所有这些产品通用的特性（通常由 Office 开发组来构造）。第 5 章分析了开发员从整体上协调与演进产品的技术，如每日构造与同步。这种产品组织与开发法使微软能靠简单地增加开发员和创建一个大的小组来渐进地增加产品的功能。正如克里斯·彼得斯所说的那样：“我们[在 Excel3.0 上]有一个非常大的开发小组……可能会让一些人感到惊奇的是它确实有用。你实际上可以写许多特性。让我们设想有大约 30 个人。我们做的特性是 10 个人做的三倍吗？不，我们大约只能做 10 个人做的两倍。比起 Excel 产生的收益，Excel 的开发成本是微不足道的。事实是我们用 30 个人做 20 个人的工作，实际上得到二倍于以前所能得到的特性。这使我们相对于 Lotus 有了巨大的比较优势。”[15]

把特性（与函数）当作建筑材料

微软产品的特性是用户最终可见的相对独立的功能单位。它们就像建筑材料，对应用软件产品更是如此，例如打印，自动选择一行数进行加总计算，或为一个特定销售商的硬件装置提供接口。系统软件产品，如 WindowsNT 或 Windows 95 的特性，对最终用户通常并不直接可见；微软和其他公司有时简单地称这些不直接可见的特性为“函数”。

程序经理承担开发一组特性（或函数），努力使其实现从说明经测试、文档化直到最后完成的过程。他们必须与开发员合作，后者负责估计进度表与完善每个特性。开发员还要在一台联网开发计算机上存储一到几个文件，用以保存特性的程序源代码。

大多数特性的开发与改进只需要一名开发员，而有的大型特性则要一个小的小组。造成这种差别的原因是一个特性的复杂程度是主观的，取决于开发员对它的认识。一个特性只需要一个人工作的例子是为 Word 增加字体。在最近的产品中，一些特性是如此之大以致个人要工作整整一年。大多数特性需要用时约三人周——即一个人工作三周。最小的特性也许只花三天左右。一个非常受欢迎的特性，Excel 的“自动汇总”特性，只用了一个人一周的时间。

产品结构——为特性服务的产品内部的灵活构架

产品结构是产品内部的基干，它规定了重要的结构构件以及这些构件如何组装到一起。产品结构及用于组装结构的构件，提供了实现产品特性（即做详细设计与编码）的支柱。产品的结构对最终用户而言通常并非直接可见；只有结构要实现的特性是可见的。产品结构也是决定产品长期结构完整性的基石。产品功能的任何改变都不应造成潜在的产品结构散架。

微软倾向于使它的产品结构非常灵活，以利于渐进性的添加、删除、完善或集成产品特性。好的产品结构减少组与组之间，包括微软内外开发员之间的内部依赖性。比尔·盖茨在我们的采访中强调了这一点：

如果有 100 人卷入了[Windows]NT 的开发，那么实际上是有 10 000 人卷入了，因为所有在公司外写 NT 软件的人，以及那些写实用程序和用户界面程序的人，都在工作。于是有一系列关于内部互相依赖的工作怎样进行的问题。好的结构甚至能减少开发组内部的依赖性。

产品结构层

我们也能从“层”以及层之间接口的角度来描述一个结构。低层为结构中的下一更高的水平层提供一系列的功能与能力。软件公司通常把最低层叫做系统“内核”；最高层提供功能与能力——即最终用户看到并使用的特性。有时最高层下面的功能和能力也能被最终用户看到。做一个粗略的类比，层化结构就像一个洋葱。一个洋葱有一个内核（或核心），它有许多环绕核心的相连的层，但顾客只能看到最外层。创造和规定功能层之间的完全抽象的接口有助于把产品的功能与下面的细节隔离开。这些细节包括对特定硬件平台，如 Macintosh 或运行 MS-DOS 和 Windows 的 PC 的依赖性。

在软件结构中，主要的规定性概念是对每层间接口的定义。第 3 章把这称做应用程序编程接口（API）。 API 规定了结构中一层的公开进入点，这使得处于高层的产品部分对其调用成为可能。产品层由上千的“子程序”或计算机代码指令集组成；开发员通常只让其中的少数能为程序的其他部分运用或调用。典型地，当一个项目把产品分成许多部分让不同的小组去开发时，他们同意每个小组去定义应给其他组提供什么类型的功能与能力。这些定义通常以 API 的形式给出，它可能是关于过程名、每个过程应接受哪些参数以及对每个过程应干什么做简要描述的清单。定义良好的结构层与 API，有助于对产品特性进行灵活的增加、删除与改进。

Excel 的结构通过使用通用层获得可移植性

Excel 结构中的每一层都有一个应用程序编程接口，它定义了本层的功能与能力。对不同操作系统平台上使用的不同版本的 Excel，前四层 API 是一样的。Excel 最底层的 API 随基本平台（如 Macintosh，Windows，OS/2 或 WindowsNT）的不同而不同，但由于微软对通用可移植层使用了与平台无关的 API，Excel 代码的外壳便与这些差异隔离开了。

Excel 产品结构展现了水平产品结构层的概念。在这儿，我们从高到低列出 Excel 的五个结构层：

- 公式与公式操作。
- 格式表（创建格式或展示数据的方法均经过良好定义）。
- 单元格（数据放入单元格，单元格计数等方法均经过良好定义）。
- 通用的可移植层（专用于 Excel 的小型“操作系统”或核心）。
- 操作系统（诸如 Windows3.1，Windows95，Macintosh 或 OS/2）。

Excel 的通用可移植层是产品结构的一个重要技术策略。这大大方便了在不同硬件和操作系统平台上重复使用应用程序代码的重要部分。在一定意义上通用的可移植层像一个小型的“操作系统”，可在 Excel 应用软件的特性与基本操作系统软件，如 Windows 或 OS/2 之间专门作为接口服务。正如乔恩·德·沃恩所详述的那样：“我们按〔通用〕层写 Excel，它是我们的操作系统版本。我们在 Macintosh,Windows,OS/2 上实现了它,现在又在Windows

NT 上实现了它，最后这个工作做起来非常轻松。通用层几乎是 Windows 版的，刚刚又在 Win32 上通过了编译。它是一个相当棒的对图形环境概念的实现。我们利用这个层，在 Mac 和 OS/2 上做了 Excel2.2。”

并非每个微软的应用软件产品都有一个像 Excel 那样的通用可移植层。例如，Word 就没有。 Word6.0 用 WindowsAPI 来近似地作为一个通用层，尽管初次尝试的结果就像最初的 Macintosh 版那样表现不佳(见第 3 章)。Excel 组在产品开发中较早地编写了他们的通用层,以便于在 Macintosh 和 PC 机器上使用相同的代码。德·沃恩总结了 Word 的通用可移植层的开发情况：

> 例如，Word 努力使自己有一个通用层，它们将用 Windows API 作为它们的通用层。我们〔Excel〕早就做出了通用层。以前这纯粹是一种试验，因为当我们考虑该问题时，我们内部都对此存在明显的怀疑：你能不能用这种办法写出勉强能够满足要求的程序呢？于是我们必须在 Excel 上做实验。在那时，我们认为无法用 Windows API 作为通用层写出一个好的 Macintosh 程序……部分原因是由于我们了解两种 API，我们在 Mac API 上做了 Mac Excel1.0，在 Windows API 上做了 Excel2.0。这两者很相似，同时又很不相同。我认为在很多时候我们对差异估计过大：“哇，这些差异是没法直接克服的，我们认为自己做不了。”但当你对事情更熟悉之后，你会对此有不同的认识。

Windows 95 的产品结构

作为产品结构的另一个范例，Windows 95 用专门的层来支持大范围的应用软件，包括 32 位，16 位以及 DOS 应用软件。这些被称为“虚拟机器”的层提供独立的处理线程和被保护的区域；每个“虚拟机器”看起来都像是一台让程序运行于其上的单独的计算机（见图 4.4）。所有的 Windows 应用程序，包括 16 位的 Windows3.1 应用程序与 32 位的 Windows 95 或 Windows NT 应用程序，都在系统虚拟机器层上得到执行。在系统虚拟机器内部，每个 32 位应用程序都有其私有的“地址空间”。私有的地址空间可以防止应用程序的数据干扰其他应用程序的数据或使其他应用程序异常中止(或叫“崩溃”)。所有 16 位应用程序在系统虚拟机器中共享一个单一通用的地址空间,这样它们就仍然可能互相干扰（就像它们在 Windows 3.1 中那样）。 Windows 子系统支持系统虚拟机器，而 Windows API 则规定了 Windows 子系统的能力。每个 DOS 应用程序在其各自的 DOS 虚拟机器中执行,使用各自私有的地址空间。

Windows 95 的基础系统提供了产品结构中的最底层，它支持系统虚拟机器，Windows 子系统以及 DOS 虚拟机器。基础系统包括一个文件管理子系统（磁盘文件，CD-ROM 文件等等）、网络子系统（包括远程计算机存取与数据共享协议）、操作系统服务（即插即用结构能力、日期/时间功能等等）、虚拟机器管理器子系统，以及设备驱动程序（键盘、显示器、鼠标等等）。基础系统中的虚拟机器管理器子系统构成了 Windows 95 操作系统的心脏。虚拟机器管理器子系统实现基本的系统功能，如处理调度、内存操作以及程序的开始与终止。

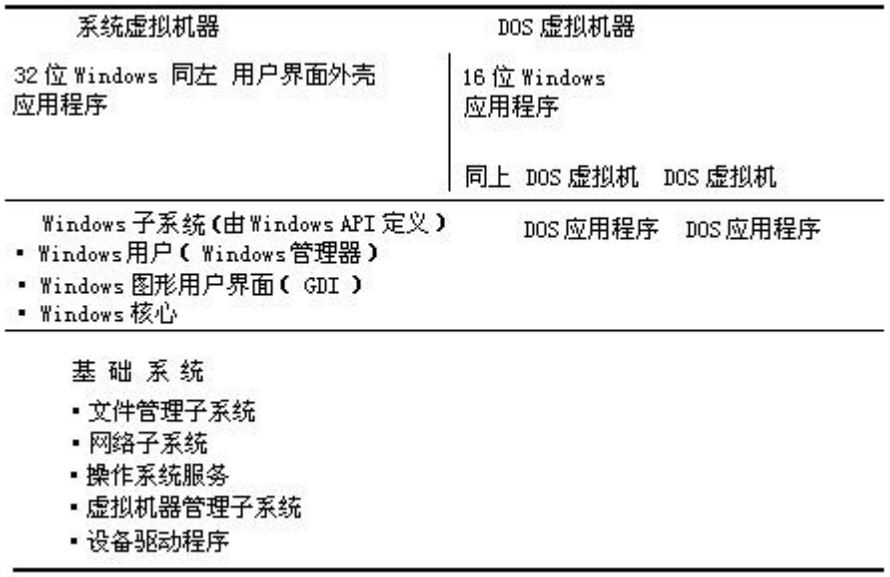
产品子系统

产品结构的水平层中含有一些各不相同的子系统或是功能与能力的子

集。每个子系统有一个 API，该 API 是子系统所属结构层的 API 的直接子集。例如，Windows3.1 开发出了“通用打印子系统”概念。这里的 API 很简单，它使第三方销售商能用更简单的方法来编写打印驱动程序（打印机驱动程序是一个随销售商不同而不同的小程序，它对销售商销售的特定的打印机进行控制）。布兰德·斯尔文伯格描述了通用打印子系统的好处，以及微软如何在通用视频子系统与其他地方应用同一概念的：

对 Windows 3.1，我们使用了新的打印机驱动程序结构，我们称之为通用驱动程序。它是一个功能强劲的通用打印机驱动程序，起着引擎的作用，使我们运用微型驱动程序

图 4.4 Windows 95 产品结构



资料来源：艾德里安·金：《深入 Windows95》(Redmnond，WA，微软出版社，1994)，第 64，105 页。

的可拆卸的模块成为可能。在简单的情况下，他们〔指编写打印机微型驱动程序的人〕基本上只需写出与打印机通话用的转义序列表，所有的光栅操作〔如计算机对线的定位〕以及所有的设备管理就都由我们提供的部分〔子系统〕完成……如此一来，编写打印机驱动程序就从原来要花一个非常有经验的人 6 个月时间才能写出一个难懂难改的庞然大物，变成了一件只要读懂打印机手册，查查转义序列表，花上两到三周时间就能完成的事情了。结果自然是对于 Windows3.1，我们有很棒的打印机驱动程序，并且可以驱动更多的打印机。打印机驱动程序过时问题也不再困扰大家了，我们从 PSS 处得到了许多这样的信息……我们现在正在对大多数其他驱动程序运用相同的基本原理。由于我们还没有对 Windows3.1 的视频使用通用驱动程序，于是视频驱动程序就成了一个难题，视频驱动程序常有错误，从而导致系统崩溃。于是，对芝加哥版[Windows95]，我们使用了通用视频驱动程序结构，为各个显示卡附上小的可拆卸的微型驱动程序。我们对通用调制解调器驱动程序也在做同样的事，安上小的可拆卸的调制解调驱动程序。我们对硬盘和其他东西也在这样做。

特性优先的产品结构与表现优先的产品结构

许多现存的微软产品，如 Office 与 Excel，均为“特性驱动”的产品结构。然而，对一些刚出现的产品，如用于“快速响应视频”的 Tiger 项目或其他的多媒体项目，要求有“表现优先”的产品结构。结构不同是有原因的。例如，传送视频和音频信号，要求即使在负荷很重的条件下也要有可靠的实时系统表现。（“实时”指产品必须在一定期限前瞬时地进行某种操作，如传送视频音频信号，否则信息就没有用了。）根据研究部副总裁瑞克·罗歇德的说法，这些不可调整的表现优先的要求与微软传统的产品特性概念不符：

> 在像 Word 和 Excel 这样的产品中,对在某个时间内完成某个操作可能也有目标规定，但是它们不是硬性规定，不须严格照办；它们只是讲应该做到差不多这样。但是当你考虑视频信号传送时，例如通过有线电视网或是电话线或甚至是标准计算机网络传送，你就必须给出一个非常非常严格的保证，讲明信息何时到达，必须以多快的速度处理才能使之正确发挥功能。系统在重负下的表现必须被完全地规定下来。绝不能让系统在重负荷与轻负荷的情况下表现得很不一致。要确保系统在重负荷下能够按规定表现，并且负荷超过一定点系统就拒绝接受进一步的负荷增加，这样才能维持住前面的保证。

微软开发这些表现优先型系统的过程包括，在所有可能的负荷情况下，对经保证的表现进行数学与算法设计。这个开发过程强调“错误容忍”功能，这指的是在操作中，系统能从失败中自动恢复。例如，第 3 章讨论的 Tiger 视频服务器项目开始时只是一个研究活动，经历了广泛的最初分析与模拟阶段。现在它转入了高级消费型系统部，由罗歇德充任交互式电视及视频系统开发主任。项目花了 6 个月时间来定义基本的系统思想以及生产最初的工作版本。在开始编码前，研究组必须指明系统的重要表现元素。研究组也使用了诸如经常构造、里程碑等老过程，并且允许说明不断改进。首先强调表现，而非强调特性或功能，同时需要预先分析与模拟，这或许就是这些新系统类型与微软通常构造的其他产品类型相比，所具有的主要特点。罗歇德继续讲道：

> 我认为那就是重大差异可能存在的地方，更传统的方法会是，比方说，首先搞出功能，然后再解决表现问题……而强调表现则是当有重要的强制因素时，你必须首先通过分析和建模型来从算法上解决那些强制问题，然后考虑还想在它上面增加其他什么特性或想做其他什么事……我认为差异可能是，一个做法在过程的早期就提出了行为中的表现与质量分级问题，而不是像任何计算机项目中可能更传统的做法那样，把这些问题留到过程的后期。

小的结构文档：源代码是唯一文件

与“黑客”传统相符，除了 API 文档，微软不对其产品结构生产什么文档。有时高级开发员写下高层结构，但这最终还要取决于开发经理。对复杂

特性，许多开发员在某些点记录并复查特定于他们所负责特性的结构细节，但做不做由他们自己决定。除了源代码文件与特性，为数不多的组为新程序员准备了描绘某层结构的文档。例如，一份 8 页纸的文件就把 Word 的内部结构解决了（主要的数据结构，它们如何工作，等等）。然而，人们不常更新这些文件，经理们也不要求项目生产这样的内部文件。

相反，说明不涉及实现问题；开发员应该知道如何去实现，或者能够去学会。克里斯·彼得斯解释道，记录的关于结构的文档如此之少是因为“一个开发员的工作是编写我们要卖的代码，而不是花时间写高水平的设计文件。”比尔·盖茨支持这个观点，强调设计文件不应该与源代码分开：“〔有〕一些方法使你拥有独立于源代码的文件。〔如果你认为〕靠走弯路和在那些〔其他文件〕上花大量时间，你会获得更高的效率的话——那简直荒谬可笑。自然界中很多相当刻板的法则都是非常可笑的。这样，许多讲实效的人就边构建很棒的设计边创造很棒的软件……一个文件。那就是源代码。那是你去的唯一地方。你在那儿学到所有的东西，你也知道那儿的所有东西。”

分割代码与“保持事情的简单”

微软不是依赖有关产品结构的文档，而是靠开发员来分割代码。这通常有效，但并非百试百灵。乔恩·德·沃恩描述了对开发员的依赖：“我们不做——我们经常陷入这该由谁负责的争论中——许多的全面设计工作,像 那个特性应放在这个模块中，这个特性应该是那个的接口’这样的事我们不做。我们依赖人们的判断力，他们正确地分割事情。有时人们做不到，那就有麻烦了。”德·沃恩强调，微软也力图使设计和代码尽量简单：“我们的另一类指导原则是‘保持事情的简单’，除非万不得已，事情就没必要搞复杂。我不确定是计算机科学那么教的还是别的什么原因，令人吃惊的是，人们喜欢把事情搞复杂，即使那样做是不必要的。”

分割与简化的一个例子是德·沃恩在 Excel 5.0 上参与过的一个“回到存盘”过程。原先的过程很复杂，影响到程序中至少 20 处不同的部分。因为人们不大用到该特性，开发员经常会忘记了它的存在，在做一个与此特性无关的改变时，就“破坏”了该特性。结果便是该特性一直有很多错误。德·沃恩用简单得多的设计替代了先前的特性，把功能集中在代码中的一个地方，于是在系统其他部分工作的开发员就不用担心弄坏它了。本·斯利夫卡，MS-DOS6.0 的开发经理，给出了另一个例子，描述了糟糕的分割以及他的一个产品部分中过度复杂的设计：

> 之所以把 Double Space 管理器的用户界面与压缩工具库（CEL）分开设计，是为了能在不需要修改 CEL 的情况下写出基于 Windows 的 UI[用户界面]。CEL 能理解所有的 CVF〔压缩后的卷文件〕细节，如格式、重启性等等。不幸的是，这个概念未被执行，于是重启性，牢固性，以及 CVF 的限制与性质的重要方面都被分散于〔DoubleSpace〕管理器与 CEL 之中……在将来，我们应确保所有小组成员都用面向对象的设计方法，集中于数据隐藏和接口最小化，以保证代码是模块化、分离化和可维护的。搞一个两到四周的教学期，讲清在何处设计与实现项目将会是一件非常非常有益的事。

产品规模增长

微软和其他公司运用产品规模度量制使开发员明白他们的产品从一个版本到下一个版本有多大改变。不断增加新特性和图形能力会造成源代码行数和可执行文件长度的极大增长。例如，MacWord 从 3.0 版到 4.0 版，源代码行数增加了 68%，从 152 525 行变到了 256378 行。规模出现极大增长的其他产品还有 Word for Windows（从 1.0 到 2.0 版本增加了 31%），Excel（从 3.0 到 4.0 版增加了 31%），ProjectforWindows（从 1.0 到 3.0 版增加了 85%）以及 WindowsNT（从最初的开发员工具包 2.0 到β1 增加了 109%，见表 4.4）。

表 4.4 产品规模度量制

产品	源代码行数	美国发送日	增长百分比
Mac Word3.01	152525	1987	-
Mac Word4.0	256378	1989	68 %
Word for Windowsl.0	249000	1990	-
Word for windows2.0	326000	1991	31 %
Excel 3.0	648531	1990	-
Excel 4.0	851468	1992	31 %
Project for Windows l.0	134225	1990	—
Project for Windows 3.0	248025	1992	85 %
Windows NT PDK2	1821719	1991	-
Windows NT β 1	3800000	1992	109 %

资料来源：克里斯·梅森：《Mac Word 4.0 开发事后分析报告》，1989 年 5 月 19 日；乔恩·德·沃恩：《对 WinWord 2.0 初步的事后分析资料》、《微软 Excel4.0 事后分析报告》，1992 年 6 月 8 日；格伦·斯雷登：《微软 Project for Windows 3.0 开发事后分析报告》，1992 年 3 月 16 日；路易斯·欧：《WinProj3.0 项目事后分析报告》，1992 年 4 月 2 日；大卫·安德森等：《NT 测试 PDK1 与 PKD2 评论》，1992 年 4 月；大卫·安德森等：《Windows NT 测试 PDK 与β1 评论》，1993 年。

表 4.5　产品可执行规模度量制

产品	源代码行数	可执行规模（字节）
Mac Word 3.0	152 525	349 000
Mac Word 4.0	256 378	668 000
Word for Windows l.0	249 000	775 000
Word for Windows 2.0	326 000	1 268 000
Project for Windows 3.0	248 025	1 629 568

资料来源：与表 4.4 同。

增加的特性会证明规模变大是值得的。然而，开发员和测试员却不得不受折磨，去理解、修改、测试与调试不断变多的代码。当代码增加时，开发

员通常力图分割代码并确定各部分之间的依赖性。这些方法有助于使修改、调试以及测试特性的时间长度保持在可管理的范围之内。在有些情况下，项目组不得不改变他们的开发风格。例如，Office 就曾采用过两步制构造过程，以适应巨大的代码量和减少产品构造所用的时间（见第 5 章）。

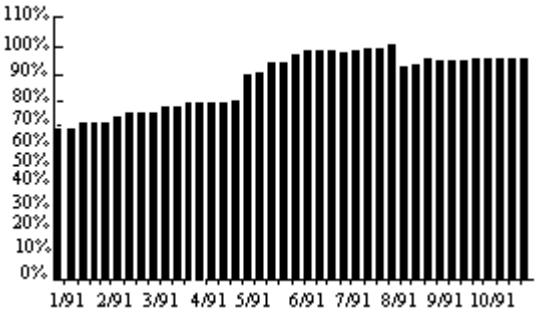
开发员还应尽量减少产品需要占用的磁盘空间量。该空间量是产品的“可执行文件”的大小。它会影响到产品在软盘上的存储以及内存的占用情况。然而，产品可执行文件的大小不容易管理或预测。它依赖于特定于产品的代码和数据的数量，以及一个产品中或从其他产品“链接”到一个产品上的可复用或可共享代码的数量。例如，Word、Excel 和 PowerPoint 作为 Office 套装软件的一部分共享某些特性，Office 还直接与 Windows 操作系统的某些文件链接。若比较单个应用软件的可执行文件的大小，单独安装就比把三个软件都作为运行于 Windows 上的 Office 套装软件的一部分一起装上时大。因为这个原因和其他原因，在代码行与可执行文件大小之间没有直接的对应关系。另一个实例是 Word for Windows1.0 与 Project for Windows3.0（见表 4.5），两者的源代码行数相近，但可执行文件的大小则差距悬殊。

微软的项目同时使用实际数字与图表来记录可执行文件的大小，然后开发员花一些时间使代码简洁紧凑和有效率，以力图缩小可执行文件的规模。例如，图 4.5 显示了 Word for Windows2.0 可执行文件规模的增长情况。1991 年 5 月到 6 月的跃升和同年 8 月到 9 月的缩减反映了特性的增减、使用和去掉调试代码、代码优化，以及其他一些因素。

特性小组和作为“内容专家”的小组领导

在第 2 章中，我们把特性小组描写成这样一个小团体：它由一个领导和 3 到 8 个开发员组成，工作于相关的特性领域。小组的规模常常视小组领导的经验和能力而定（更有经验的领导带更大的小组）。特性小组领导向项目开发领导汇报，并负责项目的全部开发工作；而项目开发领导则拥有对产品的更为全局性的观点，从而最有可能发现内部互相关联的问题。乔恩·德·沃恩认为特性小组领导是微软开发过程和吸纳新人能力的关键：“特性小组领导非常重要，我认为他们是使整个过程有效运作的最重要的因素。因为当工作要求很急而人员又需要指导时，他们就是那些居于此位而有责任提供指导的人。他们还负责编写大量的代码。”

在 20 世纪 80 年代晚期，几个组都差不多在相同时间开始使用特性小组。Excel 是带头人之一，在 1989 年和 1990 年间制造 3.0 版时率先采用了特性小组概念。该组当时只有几个小组，然而从那时起，这个数目就大大增加了。例如，Excel5.0（1993 年晚期出品）有 10 个特性小组：8 个工作于基本的 Excel 产品，一个工作于单独的 Graph 产品，另一个工作于 QueryTool 产品。Word6.0 图 4.5 Word for Windows 2.0 可执行文件规模增长度量制范例



资料来源：微软内部文件《Win Word 2.0 初步事后分析资料》。

（1994 年早期出品）有六个特性小组。

Windows 95 项目也以特性小组为中心。考虑到它的规模（超过了 100 个开发员），经理们一开始就把特性小组又组合成了三个大的独立的组：一个组负责 Windows 95 基础，一个负责网络，最后一个负责可移动的构件（项目后来合并为网络组与可移动构件组）。正如布兰德·斯尔文伯格所解释的：“可移动构件组下有两个特性小组，网络组下有半打，基础组下约 10 个，基础组最大，人也最多，完成的产品的部分确实也最多。他们构造文件系统，外壳，底层的内核，设备驱动程序，所以有 10 到 12 个不同的小组。每个小组有一名特性小组领导和一名程序经理来负责该领域。对每一个领域……如果特性小组领导能力很强，他带动全小组前进。如果程序经理能力很强，则由他带动小组前进。如果他们能力都很强，他们就共同带动。”斯尔文伯格把 Windows 95 中负责主要的独立组的人看成是“内容专家”：

> 在我的组中，我把他们分成……三个主要组。这三个组在我的组中是相当独立的，他们内部各自有程序管理，开发和测试——所有有关开发问题的部分。一个组负责基础部分，一个组负责网络部分，一个组负责可移动构件部分，因为这样可以使这些组仍然相对较小并保持等级制度的平缓。负责这些部分的人将是内容专家……如果你去看成功的软件产品，那些我在 Borland 与在这儿，微软都与之打交道的产品，你会发现处于做决定位置的人——程序经理，测试员，开发员，市场营销伙计们——都是内容专家。他们了解那些产品，他们了解如何使用产品，他们了解竞争对手的产品，他们了解未来将向何处去，他们也了解技术。他们是产品人；他们不只是管理人员……他们是内容专家。当你了解产品时，你就能作出正确的决定。当你不了解产品，当你仅仅是力图照着说明，依葫芦画瓢地构造产品时，你就抓不住什么是重要的。你没有关于优先级的感觉，也感受不到什么确实是重要的和激动人心的，什么是确实重要得马上就应开始做的。

斯尔文伯格也强调要保持特性小组的小规模，使工作联系紧密，因为产品有与创造它们的组织相似的倾向：

> 软件倾向于映射出构造它的组织的结构。如果你的组织庞大而迟缓，你就很可能造出庞大而迟缓的软件。如果你的组小而灵活，交流充分……那么它写出的东西也会是互相配合得非常好的。如果你有两个

组，它们最后应交付密切配合的软件，然而两个组之间相处得不好的话，你最后得到的通常会是两个组或两个构件之间非常正式的接口……正因如此，我经常只要看看一个软件就能告诉你造它的公司的情况，准确性还颇高。你让 15 个人去写编译器的话，往往
会得出须经历 15 次通过的编译器。经历 15 次通过的编译器一般是慢不堪言的。

每个人都是某个功能领域的专家

许多微软的产品现在是如此之大，以致经理们鼓励开发员成为部分代码的专家。结果，开发员常与其他开发员——“ 当地专家 ” 商讨——以进行可能影响到其他特性的编码与结构决策。例如，Windows NT 有成百上千的功能领域。一些个人在各自的地盘内成为专家。像 Excel 和 Word 这样的产品小一些，但它们也有成打的功能领域，每一领域亦有专家（见表 4.2）。娄·帕雷罗里对此评论道：“ 我们做的一件事是确保每人都有其专门领域……我认为这有两个作用。一，当你有难题时有地方可去，你可以找该领域的专家。二，它让人们自我感觉良好，因为他们是这个领域的专家。我们并不鼓励人们只成为一个领域的专家。 ”

像在所有的软件公司中一样，一些开发员比其余的有经验得多，技巧高得多。微软的项目利用这种差异带来的好处，把最困难的任务和领域分配给最能干的人，通常，只有更有经验和天赋的开发员才能作同时影响到产品许多部分的改变,如更改 NT 内核的代码或 Excel 的重复计算工具箱。克里斯·彼得斯评论道：“ 不管你处在开发的哪个层次上，老的模式——一些开发员比其余的好 10 倍——总是正确的。总有一个家伙比你强 10 倍。如果你是这个牛气十足的家伙，你就会担负巨大的责任了——责任总是按一个实际的特性涉及面的大小来分配的。如果只是个相当小的独立的特性，我们会倾向于把它交给一个经验有限的人。但如果特性有各种副作用和衍生分枝，它就会被交给一个更老练的人。 ”

下一个原则描述基于特性的产品组织与基于特性小组的项目结构如何也形成了项目进度安排的基础。

原则五：靠个人负责和固定项目资源实施控制

估计产品的开发与交付进度是一项富有挑战性的任务，尤其是在软件项目中。有许多因素会影响到进度，并且测量开发过程也很困难。微软解决这些困难的方法是把进度安排和工作管理的责任推到最底层，即单个的开发员和测试员那儿去。这保证了每个人除了作为小组的一部分来表现外，还负有个人的责任。单独的开发员设立他们自己的进度表，程序经理把单独的进度表加总起来，再加上缓冲时间，以制定出一个全面的项目进度表。顶层的总经理也固定基本的资源——人员与时间——以确保项目集中并限制其努力与创造程度。

关键的目标，尤其对应用软件，是指明产品的目标出品日并争取尽可能长久地坚持它。程序经理和开发员然后从出品日回溯，规定中间的项目里程碑的日期。这个 “ 固定的出品日 ” 法的中心在开发员身上；动机是减少这样的情况：项目没有定义结束点，因此在最终是无用的设计、再设计和测试的

循环中消耗一年或更多的时间。这些看起来无休无止的循环及相当不准确的进度表在微软中曾是普遍现象。当经理们对项目控制不严时，这些情况还不时发生。这种失控的项目使员工、经理和顾客都感到灰心丧气。用布鲁斯·雷恩的话来说，一名程序经理“捏造进度，无论是有意还是无意，对谁都没好处。”[18]

开发员做出他们自己的进度估计

在过去的 10 年左右,微软靠让那些将为特性编码的开发员估计他们需要的时间来为项目做进度安排。高级经理一般不交给人们进度表，然后告诉他们要在特定的日期前完成工作。许多美国、日本、欧洲的公司都用这种专断的态度做进度安排，虽然它们通常也要与开发员商量并浏览其他项目的历史资料。比尔·盖茨强调，微软让开发员和小组设定他们自己的目标：“所有这些日期都是小组定的日期。没有其他人来试图设定这个日期。我们在大约 10 年前就抛弃了那种自上而下的日期设定方法。”

基于开发员的估计也随之带来了问题。经理们不得不对开发员过分乐观的程度做出猜测，并相应调整项目进度表。与之类似，经理还必须增加缓冲时间来解决可能因信息不完全而出现的问题。这类进度安排就不如另外的一些那样精确。例如，对每个开发员在特定类型项目上做什么来保持详细的记录，编制不同行为的统计平均值或“标准时间”，然后再用这些资料做项目估计与控制。（这种更工厂化的风格在大的日本软件制造商那儿很普遍。19）然而微软的基于开发员的估计方法有两个主要优点：它从人们那儿得到更多的合作，因为日期是自己定的，不是经理定的；进度总是富有进取性，因为开发员不可避免地会低估他们真正需要的时间。

在过去几年中，微软的项目改善了他们调整估计的能力，并加上了适当数量的缓冲时间。现在，许多开发员对过去的项目保存了他们自己的记录，以观察他们的估计有多准确。项目成员、小组领导以及开发经理经常接触以前项目的资料从而帮助调整他们自己的估计，虽然经理并不要求项目保存和分析这样的历史资料。虽然微软的进度安排方式给了开发员一种更大的“自由幻觉”——这是克里斯·彼得斯的叫法，但如此一来，就要在准确性上作出牺牲：

> 从心理学上讲，让写特性的开发员估计特性的进度是必要的……如果你想要按时推出产品的话，这是你能做到按时的唯一办法。因为如果你把估计写好交给一个人，那么它并不是真正的出品日，它是你所以为的出品日。于是，如果他们做一个特性花的时间比你规定的时间长，你就被证明只是个傻瓜。他们会认为你并不了解事情如何进行。你永远不用担心开发员产生的估计会过于消极被动，因为开发员总是弄出太乐观的进度表。这样，仅仅靠给人们选择权，让他们估计自己的特性的进度，你就能造出自由幻觉，同时仍然有一个非常雄心勃勃的出品日期……至少在这个地方［微软］，你永远不用担心……进度表是由开发员完成的，并且开发员完全拥有它。

对细致的任务的进度估计

微软的第二个进度安排方法是，对要完成之任务做非常详尽的考虑，在

此基础上请开发员给出他们对“实现”的估计，以此力图“促使”更加现实主义并避免过度低估（20 世纪 80 年代曾发生在 Word for Windows 和许多其他项目上）。通常把任务细化到 4 小时（半天）到 3 天之间。有些组，如 Excel，便在这个范围内。正如乔恩·德·沃恩所说：“当我们做进度安排时我们力图把进度细化到任务不超过两天。对不同特性而言，有的可能只有一个任务，有的则可能有 20 个任务。”这股朝向更细致的进度安排的力量很重要，因为开发员不擅于具体地考虑特定的任务要花多长时间。克里斯·彼得斯曾在 Excel 和 Word 中推广更准确的进度安排，现在则在 Office 组中推广。他解释道：

> 任何任务只要超过一星期，那人们就一定没有充分地全盘考虑它。任何任务某人估计只用少于半天就可完成……他对它则考虑得太多了；他应该用更多的时间去编程，更少的时间来考虑。经典例子是，你问一位开发员要花多长时间来做某事，他回答说一个月。一个月其实就等于无限长的时间。于是你说：“好吧，一个月有 22 天［工作日］，你在这 22 天中要做的 22 件事是什么呢？”他会说：“喔，这个，也许它要花两个月。”即使把任务分成了 22 个任务他还认为，“喔，它比我认为的要难得多。”于是我们通常力争把任务弄得小于 3 天。

为系统软件产品做进度安排更困难。砍去特性不再是容易的了；而且，产品的可靠性与功能通常比按时出品重要的多。不过，像 Windows NT 这样的项目在早期强调了细致的进度安排（几天或半周的工作），以后则把注意力更多地放到了对构件进行集成与全力完成产品上。正如娄·帕雷罗里所回忆的：“当我们做最初的进度表时，我们确实力求把事情的进度安排在几天内或半周内……然后我们力争有一个集成期。很明显，估计软件设计简直如同巫术。我们的一个大问题是，你永远达不到人们想要的功能水平。你做出了你许诺要交付的东西，然后人们说：‘哦，看看，你没有处理这个。’或‘在这台机器上他们的接口有点不一样，你必须处理这三种情况。’于是我们在最后就要考虑许多额外的因素，并通过测试把它们集成进产品。”

安排开发员与小组进度时的心理学

我们已提到过，当项目变大时，微软把员工分成小组。然后经理把进度的责任和所有权尽可能低地分发下去，直到小组和个人；这使二者都产生了一种拥有工作的感觉。它还在小组中，个人中，尤其是小组领导中造成强烈的跟上预计进度的同事压力，因为经理可能再平衡进度，从落后的小组或个人手中拿走工作。这样，同事压力使经理不太需要努力就可以对个人或单个小组的进程实施严格控制。

例如，Excel 3.0（及最新的 Word 版本）就使用了这种进度安排心理学。几个特性小组做出了自己的进度表，他们相对自治地工作，互相激烈竞争以保持进度。结果是甚至一个大组（整个 Excel 项目）也像自己设定目标与时间界限的自治小组那样工作。Excel 组并没有单一的、协调的进度表，然而项目推出仅仅比预计晚了 11 天。克里斯·彼得斯回忆了这次经历：

> 为了把责任分到尽可能低处，我们让每个特性小组制作他们自己的

进度表。不光是出品日对每个人来说是重要的，它还意味着每个特性小组领导希望他们能确保按时推出……实际上并没有协调的进度表，换句话说，如果你有 5 个特性小组，当人们犯了错误并且特性尚未完全造出时，你实际上会有五个不同的出品日。于是在每一个里程碑时间，你将再度平衡进度表：你会或者砍掉特性，或者将它们在特性小组之间重分，直到事情重新变得平衡，这时你就能向下一个里程碑迈进。最后的情况是，你得到了各种各样极好的东西。此时，一个特性小组中的伙计甚至会更不愿意犯错误，因为在这个管理者被挑出来做特性小组领导的公司中，在他的直接同事可能做得更好或更差些的情况下，犯错误使自己显得不像个很好的新管理者。比起以前所有这些人只是跑来告诉一声“ 啊，我有点晚…… ”的情况来，这对出品日所有权的影响甚至更强。基本上，我们以前一直在试图想出一种办法，使我们可以产生出与更小的组相同的行为模式。

“ 固定的 ”出品日

当然，当经理把如此多的权力委托给了个人与小组时，潜在的危险也增大了。麦克·康特，他曾对 Excel 5.0 的进度情况做过记录，回忆了微软的进度安排有多糟：“ 软件开发的大问题之一是你的进度表很不确定。那总是个大问题……在有的令人尴尬的情况下，微软中曾出现过本应花两年左右时间的项目竟花了 8 年来开发。 ”根据康特的说法，微软中项目进度打滑的一个基本原因是，开始一个项目时没有固定的出品日，这样一来，对项目试图做些什么就没有限制，对怎样确保开发中产品的质量也没有认识。我们在第 1 章提到过，这种情形会导致微软的人们称之为“ 无穷错误 ”的状态——即使在开发员认为错误已被消灭了之后，项目还持续地产生数量巨大的错误，因为每一行由开发员写出用以修复错误的代码往往又产生另一个错误。于是开发员面对的是经年累月的调试与返工，无法预测何时结束。

有鉴于此，为了把创造力约束在时间限制之中，微软现在在新产品或产品新版本开始前争取固定出品日——至少是有出品日的内部目标。这给人们施加砍去特性和集中在一个项目上的压力，逼迫他们去苦苦思考那个绝对应该进入一件新产品的关键特性。虽然最终产品的交付目标可能由高级执行人员设定，但开发员与小组仍然设定他们自己的进度表。克里斯·彼得斯详述了固定的出品日的优点：

> 我们努力做“ 固定出品日 ”发送的原因是，这逼迫人们拿出创造性，并做出在浮动出品日下无法完成的困难决定。你也许会想，“ 我应该做 30 个还是 20 个特性呢？ ”如果出品日不固定，答案常是 30 个。“ 我们是做大而复杂的版本，中等版本还是小而简洁的版本呢？ ”“ 啊，我认为我们应该做复杂的版本，我们只要移动一下出品日 ”……这样，固定的出品日通常被采用，以强迫人们提供那产生了 80%收益的 20%必不可少的〔产品〕……记住，问题不是缺乏主意，而是主意太多。这样一来，你靠做什么来给自己约束以找到那绝对重要的主意？

彼得斯把 1990 年 Excel 3.0 的几乎准时推出看作他的“ 最高成就 ”。这个项目保持着微软中一个相对较小，但相当复杂的应用软件按时推出的进

度安排记录：“一般说来，我们推出产品往往延迟半年到一年，微软的有些部分肯定还在常这么干。但数年前我们努力创造了一个信息反馈环路，了解是什么造成延迟，并力求对其进行改进。我们那时就能做得不错了；现在，甚至是一些灾难性事故〔通常〕也被控制在只比原计划晚 60 天以内。但还没人打破只晚 11 天〔的记录〕。”

固定出品日概念用在系统软件产品上的程度要低得多。一个大的负面因素是限定出品日一般会造成在项目末尾处，测试时间和质量保证行为的时间不足。考虑到一个操作系统必须工作于其上的复杂多变的用户情况（设备与应用软件的不同组合），相对于应用软件，测试和质量保证行为对系统软件往往更为必需。本·斯利夫卡总结了对 MS－DOS 6.0 项目而言，固定出品日方法的有利面和不利面（我们将在第 6 章对此深入讨论）：

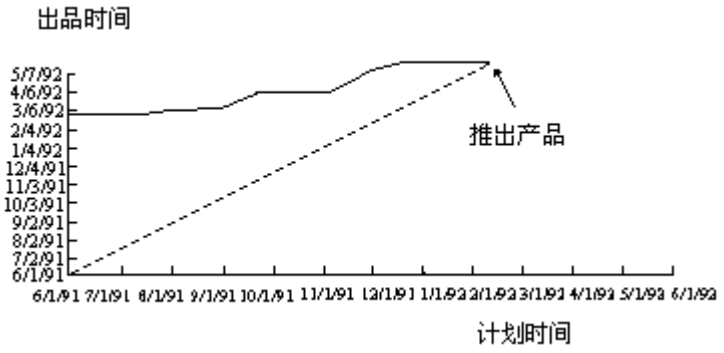
> 产品组确实在整个项目期间都时刻关注着出品日。我们都对我们要去哪儿有明确的认识，并记牢了要到达目的地就必须做出的那些取舍。共同的目标是把小组团结在一起，使之发挥巨大力量……不幸的是，在一定程度上我们成了出品日的俘虏。如果在创造性的测试以及查找尚未发作的错误上多花点时间，也许对避免顾客正经受着的一些问题会有所帮助。虽然不清楚我们当时是否会找到并纠正那些发布以来遇到的问题，但结束时更为从容的气氛会给我们更多的机会。[20]

跟踪与宣布出品日

与 20 世纪 80 年代的做法比，组程序经理，开发经理，产品单位经理以及高级执行人员现在都对主要产品在开发进度表上处于何种状态保持相对较新的记录。即使如果直到里程碑交接处前经理们都不干预或再平衡工作，他们也在做这项工作。许多组现在也有记录估计时间与真正进程数据的相似的度量制,并使用或是微软的 Project 或是一份 Excel 电子表格来收集和显示数据。（比起 Project 来，Office 组与其下的产品单位更喜欢 Excel，这主是要因为 Office 的人对 Excel 更熟悉。克里斯·彼得斯也说，比起管理像软件开发这样的单独任务来，微软的 Project 更适合管理对飞机和建筑物的设计。）

例如，图 4．6 展示了 PowerPoint 3.0 的项目进度表。原定的交付日期是 1992 年 3 月；在同一年 5 月产品推出前，该组几次修改了这个日期。对目标日期的一次重大修改发生在 1992 年 1 月 1 日左右,项目大约已到了代码完成里程碑时。当一件产品到达此点时，往往四五个月后就可以发行了。

图 4.6 跟踪 PowerPoint3.0 预计的和实际的进度表



资料来源：微软的内部文件。

直到产品确实已经准备好了进行商业发布前，微软不太喜欢宣布一个计划的出品日。然而，正如第 3 章提到的，为了战略上的原因，在有些情况下，微软和其他公司也非常早就宣布其新产品计划。早期宣布（软件行业内称之为“蒸汽品”）可能是发出要进入新市场或有意向一位特定的竞争对手挑战的信号，从而防止顾客去买另一公司的产品。然而，一般说来，过早宣布计划的交付时间减弱了变化开发计划的灵活性，给了对手再反应的机会，并且在出品日错过（经常如此）时引起负面的新闻报道。麦克·梅普尔斯对此评论道：

> 我们也已走向保守，在产品没准备好推出前就不宣布出品日。除了系统软件，至少其他所有产品都是如此。对为什么早宣布是有益的，市场营销人员总是有很好的理由，但几乎每次的结果都是，从某种市场营销的意义上讲，早宣布确实未带来益处。其次，你常给竞争对手一些信息和更多的时间来做出反应。第三，你完全搞乱了自己的业务模式。第四，如果你偏离了进度表和决定改些东西，你会惹起整个世界的关注和对你的指手划脚。我最不希望《华尔街月刊》来告诉我如何推进一个开发过程。如果在 6 月推出，我希望他们永远不知道那本是打算在 4 月推出的。如果由于一系列原因我们决定要改变出品日，我不希望为此必须向 500 个人作解释。这是一个我们的多数顾客和竞争对手都尚未领会到的因素。

许多公司为其产品宣布出品日，然后又“滑过”了这个日期以纳入额外的开发或测试时间。微软的进度表正变得更为准确，但是，正像看到的 Windows 95 或 Microsoft Exchange 这样的全新产品的情况一样，很明显，它们继续超过出品日。我们的印象是，微软的经理们现在希望在推出产品前保证其产品的质量或可靠性。交付延迟是要冒着一些使顾客疏远自己的危险，但有错误的产品对此则冒着更大的风险。在第 5 章我们将深入讨论制定产品出品决策的基础，以及检查微软实际上是如何构造与测试其产品的详细过程。

第五章
开发与出品

所有工作都平行进行，并经常保持同步

在开发和出品方面，微软所遵循的是一种我们称为“所有工作都平行进行，并经常保持同步”的策略。我们把这一策略细分为五项原则：

·在平行的小组里工作，但要保持同步和每天调试。

·永远拥有一种理论上可以推出的产品，并拥有针对每个主要平台和市场的版本。

·在同一开发场所使用一种共同的开发语言。

·在构造产品的过程中不断测试产品。

·使用度量数据来决定重大的阶段成果和产品的发布。

这些原则刻划了一种相当简单的开发产品的方式，包括许多既是协调的和自发的，也是不断增长的和保持一致的步骤。但是没有一条原则（或微软使用的任一工具、任一方法）本身是高度创新性的。许多软件开发组织采取促进小组或个人分阶段以及平行工作的措施，数不胜数的公司和实验室展示了更为先进的软件设计和开发技术。在定义产品方面，微软的独到之处首先在于其实施：产品开发支持了一种依赖于产品特性的不断演进的大市场战略。同时，人们在一种沿袭自 PC 软件的组织松散的世界的合作文化中工作。微软的小组致力于快速地工作并将大量的产品和特性以低价提供给一个范围广阔的顾客群。

有一些原则为微软项目提供了框架，促进了其内部协作和问题的解决，同时也为在通常是迅速变动的时间表下产品的循序渐进保留了足够的灵活性。也许最重要的是那些我们用以标示同步化和稳定化过程的方法、程序及工具。微软运用这种方式处理不仅是大的，还有小的项目，并且仍然保有灵活的很小的小组，并保持联络轻松快捷的优越性。关键在于如果开发员可以通过经常的“构造”和定期的“稳定”来保持他们工作的同步化，一个大的小组就可以像许多小的小组一样地工作。这种渐进式的构造也使得微软的组可以永远拥有在理论上已做好出品准备的产品。

还有一些其他因素改善了微软项目的框架结构和灵活性。不同的组使用共同的程序语言和工具，并在同一地理位置工作。这样，个人和小组面对面地交流或转换项目就变得相对容易了。经理把开发员和测试员配对，这样他们在进展过程中就可以一起工作，来测试、调试以及对特性进行集成。由于项目经理是依靠度量（即数量测量）数据和统计趋势来做出诸如何时转向下一个里程碑或何时出品产品的决策，项目也变得更容易控制了。

反之，更具顺序性的软件开发方式可能会要求非常长的时间，因为它们是按顺序而不是平行地计划工作。经理人员也会发现很难精确地控制进展情况，因为他们在开发周期中往往把主要的测试安排得很晚——通常是太晚了。顺序型的过程也不利于促进频繁的产品构造。这样，顺序型开发对微软而言是不适宜的,特别是在那些开发产品的后续版本,如 Excel5.0 或 Word6.0 的组里。如果一个项目是创造一种全新的，毫无已有特性基础的产品，这个小组通常会采用一种混合的开发方式。开始的时候它采用较为传统的顺序型步骤；而一旦特性的第一部分完成了，小组就会迅速转向一种经常构造过

程以保证下一步所开发的构件的同步化和稳定性。例如，Windows NT 作为一种新的操作系统产品在其四年的开发周期中就遵循了这种类型的混合方式。开始时它采用的是说明和细节设计的步骤，然后是渐进的里程碑和每日构造过程。

另外，微软对应用软件产品和系统软件产品所采用的开发原则是有差异的。系统产品由于其代码规模之大和组成部分之多，其构造往往不那么经常。系统产品的运行可能需要一昼夜的时间，而应用软件产品只需几小时。这样，“永远拥有可推出的产品”的概念更多地运用于应用软件而非系统软件产品。

微软同时也更经常地为应用软件产品构造满足不同硬件平台（PC 和 Macintosh）和不同语言要求的产品版本。系统产品往往只瞄准一个平台（如 Windows），最终用户所见到的特性中较少包含母语（如英语）及文化惯例（如货币的版式）方面的内容。应用软件项目更重视实用性测试，因为它们有很多要让非专家用户也能够理解的特性。系统产品则更广泛地使用非常巨大的β测试，因为它们必须能与品种繁多、情况各异的硬件设施和应用软件程序协调运作。

原则一：在平行的小组里工作，但要保持同步和每天调试每日构造过程

每日构造过程

由不少步骤组成，因为会有许多不同的人平行地对同一源代码（产品的最微小的构件）作出不同的改变。例如，Excel3．0 项目在其巅峰时期共有 34 个开发员积极地每天改变同一源代码——一个 Excel 可执行（.EXE）文件。[1]表 5．1 勾划了微软开发员创造每日构造及保持每个人“同步化”的步骤。

表 5．1 每日构造过程

1.检查

检查从源代码的中央主版本而来的源文件的私人拷贝。只有检查了这些私人拷贝的开发员才能改动这些源代码的私人拷贝。其他开发员可以检查同一源文件的不同的私人拷贝。

2.完成特性

通过对源文件的私人拷贝的改变、增加或删减代码来完成特性。开发员使用一种交互式源代码编辑器工具来进行改变。代码完成可能耗费不到一天或多达数天的时间。

3.构造私人版本

构造产品的一个私人版本，对其源文件的私人拷贝作出改变，以完成新特性。开发员会为每一个目标平台，如 Windows 和 Macintosh 构造一个产品的私人版本。这种构造通常一夜之间就完成了。

4.测试私人版本

测试产品的私人版本以确保新的特性工作良好。

5.同步化代码改变

把包含了用以完成新特性的改变的源文件的私人拷贝与源文件当前的中央主版本进行比较（这种比较的过程被称作同步化或改变同步化）。开发员使用一种“差异”工具来比较文件和自动确定差异。源文件的当前中央主版本可能已与开发员在步骤 1 检查时的源文件的私人拷贝有所不同，因为从那时起可能有某些已完成其他特性的其他开发员已改变和录入了他所检查的源文件的某些部分。开发员应该在下午晚些时候把他的源文件的私人拷贝与当前的源文件的主版本进行比较。然后他就可以肯

定他是在使用最新的源代码的中央主版本。每天的某一特定时刻之后没有人可以再改变源代码的中央主版本。例如 Excel 源代码改变的最后时限是每天下午两点。如果 Excel 开发员在特定的某一天想要把他的代码同步化，就应该在当天下午两点以后进行。

6.融合代码的改变

随时更新源文件的私人拷贝，以使它们不仅包括其他开发员对相同文件所作的改变，还包括开发员自己的改变（这种更新过程被称作“融合”）。资源库管理程序（SLM）自动完成这种融合。它还会提醒开发员文件之间存在的任何不一致性（称为“融合冲突”），而这需要开发员动手去解决。以 Excel 为例，同步化和融合通常要花 5 到 20 分钟的时间，这取决于改变的性质和程度。

7.构造私人版本

通过代码改变的融合，将其他开发员所做的改变也融合到源文件中。对这样的源文件做一私人拷贝，使用此私人拷贝来构造产品的一个私人版本。开发员可能需要为每一个目标平台，如 Windows 和 Macintosh 构造一个版本。构造通常在一夜之间完成。

8.测试私人版本

测试产品的私人版本，以确保开发员新完成的特性工作良好。开发员在完成步骤 5、6 和 7 后那天的上午进行这一特性的测试。

9.进行快速测试

在绝大多数组里对开发员的产品私人版本进行一种高度自动化的被称作“快速测试”（其他名称包括“烟雾测试”）的测试。这是为了确保在添加了他的特性后产品的基本功能仍运行良好。快速测试并不直接测试他的新特性；它只是确保他的改变不会与产品的其他特性冲突，也不会间接地引起错误。开发员在完成步骤 5、6 和 7 后那天的上午进行快速测试。以 Excel 为例，进行快速测试通常需要大约 30 分钟。

10.记录

如果通过了特性测试（步骤 8）和快速测试（步骤 9），开发员可以正式地把他的源文件的私人拷贝记录入源文件的中央主版本。记录过程的第一部分是再次运用“差异”工具把自己的源文件的私人拷贝与源文件当前的中央主版本同步化（就像他在步骤 5 所做的）。记录过程的第二部分是运用 SLM 工具解决私人拷贝与主文件之间的任何融合冲突（就像他在步骤 6 所做的）。记录过程的最后部分是完全更新源文件的主版本以使它们包含开发员用以完成他的特性的改变。其他开发员可能已在同一天的早些时候改写了同样的源文件；这可能发生在此开发员把他的改变记录入那些文件之前。这也就是为什么开发员除了在步骤 5 和步骤 6 中从事同步化和融合工作外，还需在此步骤中再次进行同步化和融合，并把它们作为记录过程的一部分。一天之中当开发员记录改变时，他必须密切注意同一天中晚些时候的其他记录，以确保它们不会间接影响到他在同一天的改变。如果确实存在相互影响，他也许需要撤回或“退出”他的一些改变，直到开发员们可以解决这两套改变之间的互斥性问题。每个项目都有一个每日记录的最终时限，这样开发员每天必须在一个确定的时间之前完成所有对源文件中央主版本的改变。例如如果一个 Excel 开发员在某一天希望记录他的改变，他就必须在那天下午 2：00 之前完成。他在完成步骤 5、6 和 7 之后的那天进行记录。这与步骤 8 和 9 是同一天。在 Excel，记录约需 5 到 60 分钟，这取决于文件的数目。

11.生成每日构造

每天在记录的最终时限（比如下午 2：00）后，一个被指定为项目“构造主管”的开发员运用源文件的中央主版本生成一个完整的产品构造，由这一构造形成的新的内部产品版本被称为“每日构造”。这是在产品发展和不断提高功能的过程中拍的一张相对稳定的快照。不管有多少开发员在一天内记录他们的源代码改变，项目都必须在当天生成每日构造。构造主管管理构造过程，直到所有的代码编译成功构造结束。这也许要求他们工作至深夜甚至凌晨，这取决于完成构造所需的时间长度。构造完成之后，构造主管执行一系列自动测试。这些保证了产品的基本功能运作良好，构造基本稳定。然后构造主管把每日构造提供给所有项目人员，包括项目经理、开发员、测试员以及用户培训人员，

供他们使用和评价。

每日构造过程有一些关键步骤。首先，为了开发某一产品的某一特性，开发员“检查”源代码的中央主版本的源文件私人拷贝。他通过改变他的源文件私人拷贝来完成他的特性。然后他把自己的私人拷贝的改变记录下来，把这些再返回到源代码的主版本。记录过程包括一个自动测试，以帮助确保对源文件的改变不致引起产品其他部分的错误。一个开发员通常每周至少两次把他的代码记录回主拷贝，但他也可以每天记录。

无论单个开发员多久向源代码记录一次他的改变，一个专门指定的被称为项目“构造主管”的开发员每天都要运用源代码的主版本生成产品的一个完整的构造。生成产品的一个构造的过程由执行一系列良动的被称为“构造计划”的命令组成。这创造了一个产品的新内部版本，还包括许多“编译”源代码的步骤。自动编译把产品的源代码翻译成一个或多个“可执行”文件（可执行文件可以直接在计算机上完成特定的操作，而不像那种也许只包括一些文档或一些数据的文件）。每天所构造的产品的新内部版本就是“每日构造”。乔恩·德·沃思，Excel 的前任开发经理和 Office 的现任开发经理，曾经这样解释 Excel 组所遵循的每日构造过程背后的规则和逻辑（亦可见于表 5.1）：

> 思路是我们总希望所记录的代码拥有尽可能高的品质。为了做到这一点，我们建立了一些规则。第一条规则是：
>
> 如果你今天想做记录，在两点以前做。你所记录的代码必须被编译和链接。对于 Excel5.0，它必须为 Windows Excel，Macintosh Excel 和 日本版的 Windows Excel 这样做。我们还在大量 Excel 源代码的基础上做了一个叫作 Graph 的程序。这样 Windows Graph 和 Macintosh Graph 也必须编译和链接。 Windows 和 Mac Excel 必须通过一个快速测试宏，这种宏是用来测试那些经常会出问题的部分的。人们必须把他们的版本下载到他们的机器并启动宏，但这之后它就是自动进行的了，所以你可以在两点以前记录完毕。当我说到你必须能够编译和链接，我的意思是你必须在前一夜已经与项目同步化，解决了你所有的融合冲突，以及已能够从一个清洁的状态构造我所说的一切。很多人每天保持同步，但你的确只需在记录的前一天做这项工作。我确信我们几乎每个人都不止每两天做一次。

每日构造——保持小组之间协调的严格法则

每日构造工作给项目小组提供了关于产品进展状况的迅速信息反馈。Windows NT 的软件工程经理娄·帕雷罗里认为每日构造是痛苦的但是有用的：“每日构造工作是世界上最痛苦的事。但它也是最伟大的，因为你可以得到立即的信息反馈。”微软的法则是很少的，但是项目必须严格遵守经常构造的过程，因为这保证了开发产品中每天之间的稳定性，并把所有的开发活动组织到了一起。就像 MS-DOS 和 Windows 的前任测试经理戴夫·马里茨所评价的：“每天五点整必须有一张关于构造的快照，这绝对是一条严格的军事化的纪律。我坚信总是会有什么东西要出乱子。所以你永远应该每天拍一张快照，无论发生了什么，即使第二天是假期，你知道不会有人要用它。每

个人所需要知道的是节奏并感觉到项目是受控制的。即使你知道从现在起两周后的构造也仍将是不稳定的。星期型的构造也还是引起剧烈的低谷或高潮的波动，但那并不重要。”

Word 的开发经理爱德·弗莱斯认为每日构造过程对于那些想有效运作的开发小组而言是至关重要的。一个大的微软项目努力像一些小的自发的小组一样工作，即使是分组也不会有几个人——就像构造 PC 软件的早期。但是大的项目不再只是小的小组之和。由于对产品构件的需要，他们也非常依赖于许多其他的组和个人。即使一个特性小组里的单个开发员也需要利用其他人所改变的源代码来工作。弗莱斯描述了每日构造过程如何保证了小组之间的协作：“那确实非常重要。我们不可能使这样大的一个组没有每日构造过程，因为我们需要合作。我们努力像小的小组一样工作。但我们不是小的小组。我们需要别人所做的工作。我们也需要产品基本上在任何时候都能运行，否则它会干扰你的领域。你不能让从事绘图工作的人暂停打字工作，否则没有人能打字，那么他们就无法打字以进入他们所要从事的领域。”

每日构造过程使得许多项目成员可以像一个整体一样地工作，因为它提供了一种代码控制机制，可以几乎一直产生出产品的能工作的一个版本。为了追踪代码的改变以及发现冲突，微软使用一种内部开发的被称为 SLM 的工具作为源代码库管理程序（微软人充满情意地叫它作“Slime”）。微软项目在采用自动测试的同时也使用 SLM 来帮助保证所开发产品的持续性和稳定性。就像弗莱斯所概括的：“我们依赖于广度测试或快速测试来做同样的事。我们依赖于我们的源代码控制系统；当它不运行时我们就会有麻烦。它使不同的人可以改变相同的文件，并且会处理和协调冲突。在你记录之前，你必须先与已记录的版本相融合，而冲突的部分会在你的当地构造上立刻显示出来。然后你必须在记录之前解决这一冲突。”

保持与产品的每日“同步化”

我们已经强调，每日构造过程保证了所开发产品的基本功能在绝大多数时间里运行良好。为了确保这种高度稳定性，微软希望开发员们每天都同步化和融合（但不一定记录）他们的源文件（见表 5.1 中的步骤 5 和步骤 6）。例如，弗莱斯相信，形成每天晚上离开办公室之前“同步化”的习惯十分重要：如果一个开发员耽搁了较长时间，同步化和融合中的问题就会增加。

开发员经常把自己检查过的源文件与其他开发员在调校后记录下的已完成的源文件进行同步化和融合。结果是开发员总拥有一个变化的最新版本。每日同步化和融合的步骤帮助协调不同开发员之间的源代码的改变，防止项目晚些时候的较大的有错误倾向的融合冲突发生。弗莱斯解释说：“我们依然认为每个人每天晚上同步化和每个人与项目始终保持同步化是重要的，这样他就可以了解其他人所做的改变……如果你试图停下来等待……在你回来与每个人同步化的时候，你将有可能遇到巨大的融合冲突……所以我们努力做到几乎任何时候，所有的人都在同一套源代码上工作。”

事实上有很多原因可以解释为什么人们经常做记录。首先，一个开发员越少做记录，其他开发员就会越多地改变他们工作于其上的源文件。这样，较迟记录就意味着开发员在融合他的文件时可能面临较多的问题。其次，做记录创造了源文件的一个中央支持拷贝。这在开发员的机器出现技术故障或其他情况使得检索文件很困难时尤为有价值。第三，做记录保存了开发员所

改变的所有文件以及它们之间差异的历史。这种“差异史”对于未来发生问题时的寻根溯源很有用处，正如弗莱斯所说的：“你可以看到是谁在什么时候改变了什么。”人们经常做记录只是为了利用这一工具的这一特性。

构造暂停

开发员的一条重要规则是确保他们所记录的改变的编译正确。改变不能与其他特性冲突或引起产品的结构性不协调而导致自动构造过程的停止。“暂停构造”对于那些需隔夜执行构造过程的大项目而言是特别严重的问题，因为直到第二天上午才会有人发现这一故障。小组因而失去了宝贵的时间。它必须修正引起构造失败的原因，重新开始构造，并且同时必须回到前一个构造进行测试。由于开发员们不能辨识最新变化的影响，他们也许会推迟他们的记录，而这阻滞了整个进程。

每天下午的中间，构造主管开始使用源代码的中央主版本进行产品的主要构造。他通常要等到构造成功之后才回家；所得到的版本即这一产品的每日构造。应用软件组一般把构造主管的职务交由下一个所记录的代码导致了构造无法成功的人来担任。Excel 的测试经理马克·奥尔森解释道：“你不可以暂停构造，这是根本性的规则……如果你在某一天阻止了构造，你就成为进行构造的人，直到有其他人又阻止了它。所以如果你把过程弄乱了，你就会受到惩罚。”

各组通常都对暂停构造的人施以惩罚。在应用软件组，由有罪的人来担任构造主管的职务。但在系统产品组，通常会有一个由 2 到 4 人组成的专门的构造小组；在这些组里开发员也许得付 5 美元或更多的罚金。微软人还喜欢讽刺一下暂停构造的开发员，经常给他们一顶特制的帽子戴，比如在 Word 组里给他戴一顶 WordPerfect 的帽子。娄·帕雷罗里回忆在 Windows NT 3.0 工作时记录代码的仪式：

> 我们过去的习惯是用羊角。规则是：如果你暂停了构造，你就戴上羊角，然后当另一个人暂停了构造，你就把羊角给他。……现在你只需付 5 块钱。某天有一个人做了一项涉及 150 个文件的巨大记录，然后他就把一张 50 美元的钞票贴在旁边，这是颇为幽默的。我告诉我的调试人员们，如果他们有一大堆错误要修改，还有一大堆记录要做……他们可以一石二鸟。他们应该把一张百元钞票贴在板上。如果运行没有问题，他们可以把他们的钞票拿走；一旦出现问题，他们就丢了这笔钱。这样他们就会进行所有他们需要做的测试，因为要构造调试器尚需约 40 分钟的时间。

尽量缩短总的构造过程

同步化、融合以及记录的过程对于开发员而言是相对非生产性的时间，因为他们使得机器负载相当沉重，而开发员在这段时间内不能从事任何其他基于计算机的活动。所以开发员会尽量缩短总的同步化、融合和记录过程。他们需要有效地使用他们的时间、迅速地完成改变——特别是在项目后期测试员需要快速修改错误的时候。在项目的后期，开发员经常做“一天通过”：改变代码经常是对错误的修改——并在改变发生的当天做完记录。乔恩·德·沃思谈到缩短总的构造过程和迅速完成错误修改的必要性：“其实

人们并非总需要每天记录，因为他们所从事的特性领域并非“一天通过”。时间变得昂贵是在项目结束阶段你做“一天通过”的时候——而且实际上也没有其他什么会像“一天通过”的错误修改那么昂贵。你想要依靠那些能尽快修改错误的测试员，因为这种修改不可避免地阻滞了特性的其他领域。这是总的记录过程真正开始有作用的时刻……一旦我们进入错误修改阶段，依靠测试员是很重要的。”

同步化、融合以及记录源文件所需要的时间与所改变的文件的数量和改变的程度严格成比例。在 Excel 和 Word，每日构造所需的时间相对较少，通常不到一个小时。德·沃恩作出分类：“至于总过程所需要的时间，同步化约需 5 到 20 分钟，取决于你检查的文件的数量。编译是在夜间完成的，所以我把其成本记为零。下载和进行快速测试大约需要半个小时，这取决于你检查出了多少事。一个小时是很长的时间。”

微软在 Office 项目上没有采用完全一样的每日构造方式。这是因为开发员必须同步化、融合和记录的文件数增加了一倍以上，所以所需的时间也会增加一倍以上。更快的 PC 机和开发工具使得对于每日构造而言产品的规模不那么重要了。不过对于一个像 Office 这样的大项目而言，缩短总过程的办法之一就是不要那么经常地做记录。由于这可能损害到较经常做记录的好处，微软重新构造了 Office 的源文件，使得开发员完成其特性时只需改变几个文件即可。于是开发员无需对整个产品进行再构造，而只需再构造 Office 中与他的改变有关的部分。

除了减少开发员在添加特性时所需改变的文件的数量，Office 组还建立了一种两阶段过程来进一步减少同步化、融合以及记录源文件所需要的时间。这一过程的第一阶段允许开发员把源文件的改变记录进被称为“石灰岩”的系统的试验台版本。石灰岩试验台系统的构造无需对整个 Office 产品的重新构造；这样，它既不需要对每一个单独的应用软件进行快速测试，也不需要对整个产品进行完全的“回归测试”（回归测试是用来确定以前工作的特性或功能是否仍正常工作）。Office 组也每天构造石灰岩试验台版本，并进行其自身的自动回归测试。

在第二阶段，构造主管定期把文件的石灰岩试验台版本中的改变转移到 Office 源文件的主版本。他们在开发员完成特定的特性之后把石灰岩的改变转入 Office 版本。这通常至少一周一次，并且在里程碑版本之前完成。对于里程碑版本，他们把石灰岩上的所有源代码改变转移到 Office。这一过程要求构造所有的构件应用软件（Word、Excel、PowerPoint、Access 以及 Mail）和执行所有的快速测试以及应用软件专用的和 Office 专用的自动测试。

在做记录之前公布

Windows NT 3.0 的有些记录非常庞大，一次就包括 800 个改变的文件。这些改变可能只是构件名称的改变或构件之间附加数据的交换。许多更常见的较小的记录则包括对产品的复杂改变。于是 NT 组在每日构造过程中又多加了一步。在记录源文件之前（表 5.1 中的步骤 10），开发员走到项目布告栏前写下他要记录的文件名。他把这一意向先公布出来，这样其他人就可以作出相应的预期，因为他的改变可能会引起不协调或影响源代码的其他部分。过一会儿，开发员就会接到构造主管的电话，告诉他可以记录他的代码了。开发员做好记录后向 Windows NT 构造主管发一份 E-mail，特别说明产品的

哪些部分有赖于新记录的代码并需要重新构造。在记录之前加入的这一步骤使其他开发员有了额外的时间来准备应付这些改变，使构造过程的统一协调成为可能，并把构造暂停的机会降到最低。

大的系统产品如 Windows NT 的构造主管，通常有一个小组帮助他的工作。 NT 3.0 的构造小组有四个人；领导这一项目的戴夫·卡特勒也经常加入。 Windows NT 还把不同的构造要求交错开，使之平行进展并连夜处理，以此来处理这些多样化的构造要求。娄·帕雷罗里解释道：

> 这是每日构造过程。你接到一个电话，说：“好，我们准备好了接受你的记录。”你做记录，然后你向一个叫“NT 构造”的名字发 E－mail，精确描述要使之进入源代码群所需的操作。这样，你把文件同步化，构造这些东西，到这一步你再同步化这一文件，构造它。现在到目录上链接它，你会得到一个新的核心，一个新的 Windows 可执行文件或不管什么东西。你要一个新的编辑器。而且，你当然是在一天之内做完这么多事的。你拥有一台一天可以编译 8、10、12 小时的机器……所以很多时候我们所要做的是在晚上八点时说：“去记录你的所有东西。”有人会在全天过程中做记录，但是我们直到晚上才进行构造。我们回家时应该有五个窗口在运行不同构件的清洁构造，就像整个核心的一个清洁构造。可以说是我们改变了过程目标的规模；重新构造一个核心约需一个半小时的时间……我们不喜欢在白天做这些事，凌晨两点时它会工作良好。

构造周期的频率

微软根据项目的特别要求和成功完成一个构造所需的时间来决定构造周期频率。系统产品由于它们的规模及包含的文件数目和相互依赖性，一般会需要较长的构造时间。微软每天构造 Excel、Word 和 Office 的试验台版本；至少一周构造一次 Office 的完整版本。 Windows 3． 1、WindowsNT 和 MS－DOS 在它们开发周期的早期每月和每周构造，然后在发布前的数月内每日构造。WindowsNT 在产品出品前大约 12 个月的 1992 年 7 月开始完全的每日构造。微软在 Windows 95 的扩展测试阶段每周构造。

一些其他公司也采用经常构造。比如曾是 DEC 公司 VMS 操作系统项目开发小组成员的娄·帕雷罗里说，他那个开发小组每周和每两周进行一次构造。现在 IBM 和 AT&T 的一些主要软件开发项目每两周或每月构造一次。大的操作系统或顾客软件项目在项目的绝大多数时间里没有定期构造，然后在项目后期的集成和系统测试阶段进行每两周或每月一次的构造[2]，而 Borland（Turbo Pascal）至少一个小组是每日构造。曾经在 Borland 管理这一组的布兰德·斯尔文伯格认为他的前任公司在开发方法方面最像微软：

> 在 Borland，产品和小组都要小得多，所以事情进行得极为迅速……我们在常规基础上构造产品， Turbo Pascal 几乎每日进行构造，虽然那里只有 6 个程序员，很容易保持同步。当他们做了一个构造，那就是它了。那一产品的β测试周期……只需 6 个星期，因为领头的开发员是如此之棒。Borland 的工作也是非常出色的。他们比微软小，更灵活一点，但组织得不够那么好。在最高层次上他们也许比其他人更像微软，

但我还不知道有谁做得比微软好……我不会看着任何其他开发组织说：“嘿，我希望我们可以做得像他们一样！”

下一项原则描述了每日构造过程如何帮助微软做到始终准备好推出一件产品。

原则二：永远拥有一种理论上可以推出的产品，并拥有针对每个主要平台和市场的版本

从每日构造中产生的一个主要好处是项目总能拥有一件稳定，然而又是不断进化的、有一套灵活特性的产品。不断增进的特性开发、经常性的同步化以及持续的测试帮助迫使项目努力建立和保持产品的一个达到或接近顾客所需的质量水平的版本。当然，一个拥有广泛的新增特性的每日构造的可靠性是不确定的，但是理论上微软可以推出其三四个主要里程碑版本中的任何一个（就像每周构造中的大多数一样）给顾客。这种进化中的产品构造可随时出品的性质使得微软可以在产品开发周期相当晚的阶段还增、删特性。这使按时出品或改变特性以响应顾客的信息反馈或竞争压力都成为可能。项目也有很大的灵活性来决定何时发布其产品给顾客。在开发过程中，项目为其产品同时构造针对不同的平台（如 Windows95、Windows NT 和 Macintosh）和不同市场（如美国、欧洲和亚洲）的不同版本。这些同时发展的版本使得微软可以在极短的时间内向不同的 PC 平台和市场推出产品。

了解你的位置

渐进的里程碑方式和每日构造过程使得微软小组在开发产品中可以紧跟其进展。它们还使得总的产品开发过程更可见和可预期。所以，一个项目小组总能了解相对于完成它所想构造的产品特性和功能而言，它的位置何在。在完成了产品的开发阶段之后，微软项目也会在产品实际发送给顾客之前插入一段相对较短的稳定期。但这与常规的软件开发方式有所不同，常规方式通常把他们时间表的 50%花费在完成开发后的最终测试阶段。麦克·梅普尔斯概括了里程碑如何帮助微软人评估他们是否作好了发布一件产品的准备：

> 旧的把“编写说明、编写代码、测试代码、维护代码”作为一个生命周期的概念极易误导你关于离完成还有多远的认识。你会看到大批的项目在完成最后的 20%的阶段花费了 80%的时间。所以我们发展了这样一个里程碑过程的思路，你努力找出比较困难的事去做，你 100%地完成它们，然后你重新估价你的位置……如果这套特性就是产品，你就做所有推出它的准备工作……如果你做的是对的，则从你结束产品到你可以提供整个产品的时间非常短……这确实使过程容易预测得多了。在过程较早的时候你就知道你是否已陷入困境……你知道你的位置。

迅速发布一件比小组最初的设计有更多或更少特性的产品的能力是微软的一项极具竞争力的优势。不过，其他公司如 Lotus 和 Borland 也采用同样的原则。至少 Borland 曾用这一方式把一种新产品比计划提前推向市场，以此来与微软竞争（我们没有任何微软由于采用每日构造过程而比计划提前出

品产品的例子，却有很多项目砍掉特性以保证按时出品或缩短延误期的例子）。如布兰德·斯尔文伯格所言， Borland 是分多阶段开发 Turbo Pascal 的，并不得不至少把其出品提前了五个月以面对竞争性的微软产品 Quick Pascal。Borland 在 Turbo Pascal 的开发过程中一直使之保持高度的可出品性，这使它能够迅速变化并成功地与 Quick Pascal 竞争。斯尔文伯格从 Borland 的角度回忆了这场两大产品的竞争情况：

> 如果你总有一些正在开发的东西，你在竞争市场上的步伐就可以更为轻捷。如果你的竞争对手推出了你没有预料到的东西，你总还可以拿出你的中间产品并出品它。实际上，当我还在 Borland 的时候就有一起极好的例子。我们在开发 Turbo Pascal，那确实是 Borland 的基础语言产品。我们有特许经营权，而公司声望中的很大部分是与 TurboPascal 连在一起的。我们听到传言说微软正准备推出一种叫作 Quick Pascal 的产品……于是我们组织起这个战斗小组，说：“好吧，如果我是比尔，我会做什么来与 Borland 构造的超级语言产品竞争呢？”于是我对我所认为的 QuickPascal 的样子做了说明，然后用这一说明来引导我认为 Turbo Pascal 的下一个版本应该是什么样子。
>
> 能分阶段开发 Turbo Pascal 真是太好了，因为我们本来计划下一版本要在九个月后才出来。但是我们听到 Quick Pascal 四个月后就完成了，我们需要到那时也有一些东西出来与他们抗衡，并且我们可以做得到。我的预测是对的，我对那件产品有什么功能的说明 98%是正确的。我所在的位置足以抗衡他们自认为将令人震惊并将夺走 Borland 在 Pascal 方面的领导地位的产品发布。我们事实上推出了一种十分优秀的产品——而且毕竟，有多少人听说过 Quick Pascal 呢？不太多。我们赢了，因为我们能够永远处于有产品可推出的状态。

“培育”而不是设计软件

微软现在完成特性是通过创造原型、由内部和外部用户进行测试以及调试这一个连续过程进行的，而不是在开始构造产品之前就试图创造出产品的完整的说明和细节的设计。项目在开发一个新的产品版本时也经常用新构件替换掉产品中旧的部分。开发员们经常把这种先初创单独的产品功能，再把它们系统地发展成完备的特性的实践过程称为“培育”软件（IBM 的前成员弗雷德·布鲁克斯在他 1987 年关于软件开发的著名文章中也使用了这一说法[3]）。娄·帕雷罗里这样描绘了项目如何初创和发展特性：“我相信这种理论，即你应该培育软件，而且如果你看一下应用软件组，你会发现他们正是这么做的。你得到一些开始工作的东西，然后，缓慢而毫不怀疑地发展它，找出它的弱点和优势。它能发展完善是因为人们会关心它并提出建议说：‘嘿，这东西要是能这样就太棒了。’此时你就‘培育’这一部分。或者，人们说：‘它糟透了’。”

微软的各组广泛采用这一重复初创的方式。例如帕雷罗里曾描述 Windows 3.0 项目在最终发布产品前是如何数次重新编写主要的次级系统，如文件系统：

> 这是我确信无疑的东西。你希望尽快初创出原型，这样你就可以在

深陷其中无法看清之前找出愚蠢的和错误的地方并修改它。有例为证：我们的文件系统人员曾经奢侈地做了四个文件系统，他们还更为奢侈地重做了其中两个。我们的这两个文件系统之所以被重写只是因为当我们在编写更多文件系统的时候我们发现了些做得不太好的事情。然后我们把这些概念综合进较早的文件系统。我们的串行端口驱动程序被编写了两次。对于整件事，我们只是回去并说“这是它的瓶颈。我们要重新编写它。”然后你并非连夜重写它。你逐步改进它，而当你完成时它就是全新的了。

短暂的代码半周期

许多开发员指出小组应持续地重写软件，而不应过份精雕细刻代码的某一部分。戴夫·穆尔估计微软一个软件的“半周期”只有 18 个月，这意味着小组每 18 个月左右就改变或替代他们产品代码的 50%。由于微软大约每 12 到 24 个月发布某种产品的一个新版本,这一半周期概念表明产品代码在每两次发布之间改变 50%。实际上，克里斯·彼得斯指出微软并不太重视编写可共用代码，因为它会很快过时：“你可能花费了如此多的时间制造一种可共享的东西，然后看着它两年之后就过时了。在一个变化如此迅速而猛烈的世界上，几乎没有什么是要与人共享的。”

改变的速度，或者说代码的“翻腾”，在有些组内变得极端迅速。比如在 Excel，爱德·弗莱斯声称开发员们至少一周一次，至少是部分地改变每一个源文件。他提醒道，频繁的代码改变使得要保持与源代码分离的、涉及细节的设计文档是很困难的：“Excel 程序非常庞大，但是它改变得极快。一个星期内每个 C 文件的相当多部分都会被触及。如果你等一个星期的时间来同步化，整个项目中的几乎每个文件都将变得不同，因为有如此多的人在忙碌地做出不同的改变。有时他们所做的改变是巨大而全面的，会影响到很多文件。而这正是文档的问题。在此过程中取出一些东西，然后它就很快过时了。所以当我取出一些东西并说这就是关于如何做一些事的范例时，我总是很担心。”

要想改变特性，必须做到加倍的好

久而久之，微软的项目学会了约束自己，不去进行产品上相对小的改变，因为那只给用户带来很小的改进，却要求大量的努力或可能导致不一致性。比如 Windows 应用软件的目的是为了让产品与 Windows 共同工作。这目的并非要求改变 Windows 本身——当然，除非改变可以带来较大的改进。

微软是在研制 OS/2 时得到这一教训的。就像克里斯·彼得斯回忆的：“OS/2 是他们试图改变什么的尝试……他们试图制造出仅改善 10%但变动极大的东西，而没有人想要这 10%的改进。我们有一条经验法则——如果你想保持一致性，那么在你把它变得不同之前，你得保证它有两倍的改进。”彼得斯又提及“还原按钮”的例子，这是在 Word 里首先出现的特性。Excel 组开发了一种工作完全不同的“还原按钮”并在可用性实验室测试出它有 10%的改进。这没有好到足够使 Excel 与 Word 使用不同的“还原按钮”，因为非常多的人要使用这两个程序；它也没有提供足够的改进，以使 Word 也采用 Excel 的格式。就像彼得斯所说的：“很明显，如果它好了很多，也许我们应该考虑它。但如果它只改进了一点，那么缺乏一致性也会成为问题的。”

重新编写代码的 20%的成本

保持产品随时可出品的另一个方面是投资于项目之间的代码重新编写（这类似于其他软件公司里的“维护”或产品“再工程”工作，虽然微软并无专门的组来专司此职）。作为一条非正式的原则，开发经理努力将他们的开发员的 20%的时间分配于重做产品较弱的部分。如果他们在多个版本上持续地这样做，产品也将持续地提高；如果他们不这样做，则当开发员添加不同的特性以及用户要求产品的更多功能时总体质量会恶化。克里斯·彼得斯指出：“我们把这些东西称为 20%的成本，它与重新编写或重新构造产品的某些部分有关。如果你不付这 20%的成本，你的结局将会更糟。所以你应该努力找出最坏的部分。”通常，项目是用这部分时间重做程序的重要然而不是用户所容易看到的部分。这样的例子包括 Excel 中的保存名,Word 中从 256 种字模转为 4 000 种字模，重新编写程序的打印方式，或对特性的运行或互相影响的途径做更多的基础性改变。

微软人把这种工作视为“成本”，是从这样的角度考虑的：项目把开发员一周时间中的一天用于提高产品的基础性结构而不是添加新特性。开发员经常在他们发布完一件产品（在第 4 章里被称为“里程碑 0”）之后到正式开始下一个产品版本的开发阶段之前的这段时期内使用这一时间。

同时构造产品的多个版本

对于微软的竞争性策略而言还有一点也是十分重要的，即不仅对世界上不同的市场，也对不同的硬件平台同时准备好每种产品的不同版本。项目在资源分配上权衡，以期最大化不同产品版本之间用来进行调试、最终发送、服务于不同的硬件和操作系统平台以及服务于不同最终用户语言的可共享的代码的数量。这种共享使得不同项目可以同时开发和测试所有这些版本。比如 Excel 组构造了 Excel4.0 的 20 个不同版本,以适应五个不同的语言区——美语、德语、法语、西班牙语和远东语区（日语、汉语以及韩语），适应 Windows 和 Macintosh 操作平台，并且同时有产出（或出品）版本和“调试”版本（它包括了测试和发现错误的额外代码）服务于每一个语言/平台组合。（微软只向其顾客发送了其中的 10 个版本,因为调试版本是供项目在内部使用的。）

Excel 为项目如何准备好推出这些不同的版本提供了很好的范例。在一个项目中，开发员把产品的语言专用部分（如错误信息的内容以及对话部分）分离成一个单独的文件。这种分离使得他们可以通过仅翻译一个单独文件的内容就把这些部分译成外语了；微软各组把这种翻译过程称为“本地化”。于是开发员们可以集中注意力于代码中与语言无关的在所有语言版本中都可共享的部分。马克·奥尔森对这种独立于语言的代码与产品中依赖于语言的部分的分离评论道：“美语版本是我们花时间最多的版本。我们产品中的所有语言专用的部分，就像所有被本地化或被翻译的 UI（用户界面）一样，被组成了一个独立的 DLL（动态链接库文件），我们称之为语言 DLL。所以我们只需构造一个 Excel.exe（可执行文件），然后为每种语言配一个 DLL。法语版本的代码与美语版本是一样的，但是 UI 的特色被归入 DLL 了。”

对依赖于语言的本地化文件的翻译在开发过程相当靠后的阶段进行，这时用户界面已经稳定了。项目希望特性首先要稳定，这样在特性改变时他们

无需翻译用户界面的内容。对本地化文件的真正的翻译其实是在项目马上要发布美语版本之前或刚刚发布完之后进行的。依赖于语言的本地化文件与独立于语言的代码的分离使得测试员可以通过主要集中测试一种国际产品的本地化部分而有效地利用他们的时间。奥尔森解释了这一过程是如何工作的：

> 那使我们可以分离产品的本地化，对翻译的测试和在对话、包含字符串的菜单以及排序顺序方面的本地化。我们最后再做那一类测试。而且我们在过程中所做的测试是为了确保类似国家设置这样的东西不会产生干扰，于是完成国家设置的代码成为了核心代码的一部分。在法语版本上我们也没做什么特别的；它也是同一代码基础的部分。所以对于每一特性，人们都必须了解改变代码页或在特性上改变国家设置的含义。对一些特性而言它一点儿也不起作用。但对于一个像格式数据这样的特性，你会看到货币格式或日期分隔符或名单分隔符，你就必须确保你是从正确的地方得到的信息。没有一成不变的代码。

Office 项目为其产品系列创造了差不多同样数量的版本。Office 构造了一种 Win32（ 32 位）版本和一、两种 Macintosh 版本（为 Power Mac 和更早的基于 Motorola 68000 处理芯片族的 Macintosh 产品）；他们为每一个平台构造了一个产出和一个调试版本，以及为每一种语言准备了不同的本地化文件。Office 的远东版本最终会要求一个独立的构造而不仅仅是一个本地化文件，因为微软要采用单一代码标准（UNICODE 标准）。单一代码对每个字符使用两个字节（这不同于现存的每个字符使用一个字节，然后以特别的扩展来完善它的方式），以对汉字字符提供广泛的支持。

向多个平台和市场发布产品

同时构造一种产品的多个版本的真正好处在于微软可以在很短的时间内向不同的平台和市场推出各自的产品版本。公司的目的是对每一种主要产品在每一个主要的平台上和在每一种语言的市场上都拥有同时的出品(称为‘ 同出品 ”）；这意味着在约 30 天内发布完各种版本。在很短的时间间隔内发布不同的版本对于成功营销产品而言是至为重要的。当国外顾客听说在美国已有一种新版本上市，他们就会停止购买产品的旧版本，就像 Macintosh 的顾客，如果听说一种新的 Windows 版本问世了，他们就会停止购买旧版本的 Excel 和 Word。

虽然目标是在 30 天内出品不同的版本，然而，微软在推出有些小语种版本时会慢一些。比如：Excel4.0 首先在美语的 Windows 版本中出品，法语和德语 Windows 版本在 30 天后推出,但是西班牙语 Windows 版本在美语版本之后 60 天才推出。在美语 Macintosh 版本出品的 Excel4.0 比美语 Windows 版本晚大约 20 天，而国际 Macintosh 版本的推出也像 Windows 版本一样有 30 到 60 天的时滞。对于 Windows 95 的目标是在美语版本之后三个月内推出汉语版本（Windows 95 代码也为中东版本和远东版本提供了基础）。这些交错的发布在很大程度上反映了逻辑上的延迟和测试，马克·奥尔森认为：“ 每件事都被铺开了，而这部分地是由于一些最后一刻的逻辑细节的需要。总有些事你没有考虑到。软件作好了充分的准备：文档做好了；装到磁盘上的‘ 只读 ’ 文件做好了；安装程序做完了；所有的示范文件和所有的我们出品的添

加项最终也完成了。现在只需保持这份文件名单以确保它们是正确的，以及它们都会有机会被看到。”

在有些国际市场上的销售量并不太大，但是如果经理人员相信这些市场最终会变得可观，微软就会投资于这些外语版本。比如 Word 的希伯莱语版本所售出的拷贝相对很少。然而微软人希望在许多中东国家里有良好表现，他们也把中国视为一个巨大的目标市场。微软产品开发董事克里斯·威廉姆斯对这一投资策略进行描述：

> 中东组居于产品的业务单位之外是因为我们想分离出我们在那一地区的投资。如果我们把希伯莱语 Word 的开发置于 Word 组中，它的全部仅 1200 份的拷贝或它所能售出的量将使 Word 成员完全地忽略它……或者也许他们会投入过度的努力。问题在于你无法判断向它投入了多少努力。所以我们把那些人专门分配到那个市场空间，他们的任务是在我们投资的基础上，为我们在那个市场空间占据一席之地而提供所需的产品。
>
> 现在，他们的知识产权法律令人震惊的糟糕，所以人们在 90%的时间里是偷窃软件。但是我们知道世界的 20%说阿拉伯语，迟早我们会在那个市场空间赚钱的。我们现在的策略是，花掉与所赚同样多的钱。我们有一个非常可观的投资，它几乎花掉了我们在那个邻近地区所获得的全部收入……在日本和在远东的人们也在努力打开中国市场，这是我们在成本基础上投资的另一个重要领域。我们软件的拷贝正在涌入那个布场……目标是当人们最终不得不购买软件时，他们已经知道了我们的软件，并且那是法律生效后他们所愿意去买的软件。我们基本上得到了市场份额。一旦我们开始从那项投资上获益，那将是惊人的。

原则三：在同一个开发场所使用一种共同的开发语言

为了利于人们在组际和组内交流，微软人在西雅图郊外的公司总部从事他们的开发工作。他们还使用一种共同的“语言”——即他们使用一小套开发语言、惯例和工具。这种对如何创造产品的普遍共识有助于微软的个人交流和相对迅速有效地解决问题。

单一场所开发

微软一直鼓励那种在微软总部完成所有主要开发的传统。这一规则只有极少数的例外。有一些海外适应项目（如 Windows 的日本版本，微软稍后也把它转移到了美国）以及某些应用软件产品和工具开发工作（如 PowerPoint 和 Softimage）是在这一场所之外进行的，因为它们一开始就是微软的附加产品。但是正如第 1 章所提到的，比尔·盖茨特别希望组与组之间能面对面地解决问题和在技术上互相依赖。虽然 E-mail 有很多好处并已在微软广泛使用，单一场所开发使得项目中个人确实可以经常性地聚集在一起，并互相启发，发现新思路。经常性的和方便的交流可以防止问题的恶化。戴夫·马里茨强调了这些好处：

> 位置——单一场所开发——是如此如此的重要。Win[dows]3.1 还没

有离开日本的原因之一是因为它是在日本开发出来的。而那里有一种努力不让我们了解进展情况的日本式心态，因为那是他们所想要的工作方式，他们的经理也不让两个人在一起交谈。这就是它现在为什么要被带到这儿来的原因。在芝加哥的 Windows95，我们能做到在走廊里蹓跶一趟就看到了进展情况。如果你不能随时看到别人的进展情况，你就遇到问题了。而这正是 OS/2 这个多场所开发项目（微软—IBM 在 80 年代联合完成的项目）所遇到的问题之一。

微软经理还愿意投资于建设在微软总部工作的人们所需的基础设施。几乎每一个人——除了暑假实习生和偶尔的新雇员——都有自己单独的办公室。很多办公室都有窗户，因为建筑的平面图是“ × ”形的。产品单位还向项目个人慷慨提供他们需要的任何计算机和工具，包括一些不同的台式机器（通常包括一台 WindowsPC、一台 Macintosh,还有一台任何其他类型的 PC），已完全装载好软件和具有网络能力。

C 程序语言

微软项目除了使用有限的 C+ +和汇编语言外，主要使用 C 程序语言完成他们的产品。许多公司在商业性软件产品中使用 C 语言，因为它在不同的硬件平台上非常容易移植，且运行效率高并提供了很大的补充完善的灵活性。C 语言本身独立于任何硬件平台——不像汇编语言，但是 C 程序所运用的内在系统函数（称为“ 库 ”），特别是用户界面库是平台专用的。所以用 C 语言来编写程序可能会有助于但并不能确保可移植性；微软在移植它的代码到许多平台（Windows，DOS，以及 Macintosh）方面取得成功，是因为它还使用内部开发的工具来估价代码的可移植性，并对某些产品开发了在不同的平台上无需做巨大改变就可以再编译的“ 核心代码 ”。比如 Excel4.0 包含了可在 Windows 和 Macintosh 平台上共享的 69%的核心代码。[4]（核心代码对于开发员而言有很多好处。但正如我们在第 3 章所提到的，在 Word 6.0 中使用 Windows-Macintosh 核心代码的初次尝试，不仅带来了 Macintosh 上的“ 观感 ”的问题，还引起了一些其他现象。）[5]

虽然 C 语言有可移植、有效率及灵活的优势，软件开发区里的有些人却觉得它太原始了，并鼓励开发员使用一些不必要的技巧。比尔·盖茨也指出了 C 的劣势，但坚持微软开发员应用 C 来生成（或抽象）他们的代码：“ 像其他人一样，我们还没有找到一种不用 C 而用其他东西编写代码的方法。现在 C 是一种非常低层次的语言，我们在很多东西上使用 C++。我们抽象出很多东西。 ”C++语言鼓励开发员编写高度组织的代码，抽象出一般概念而隐藏细节。微软在有选择的适当的项目上运用 C+ +（例如 Office 和 Windows NT 的一些新构件），但 C 仍然占统治地位。

盖茨还强调，微软远在像 C+ +这样一种语言支持他们之前就已经使用了目标抽象和复用的概念（称为“ 面向对象的程序设计 ”）。这证明了微软开发员的普遍技术水平。但他也承认微软没能促使这些方法上的细节成为其产品的优势：

很多人会说：“ 面向对象的程序设计会引起包含 10 项改进的因素 ”……是的，它好处很多，因为工具会更多地支持它，而且它会带来

更多的共享，甚至更多好处。但我们十年前所编写的代码就使用相同的抽象。我们广泛使用过程性的指针，这样你就可以拥有不同的数据结构而使用共同的子程序；这只是程序问题。公平地说，就对效率的理解以及清楚地知道自己分解东西的方式而言， C++会因它们在未来更适于被共享，而存在着被滥用的危险。很多人跌入这块希望之地的陷阱。这也就是为什么你也没有看到我们跳出来说“ 啊，顾客一定要来买我们的产品，因为它有针对性。 ”即使我们已编写了大量的 C+ +代码，如果它运行更迅速，如果它有更多的特性，你会购买它。我们是否用心理专家来创造它并不重要。所以我们并不吹嘘我们在考虑到产品的最终用户时采用了或没有采用什么方法。

“ 匈牙利语 ”命名的惯例

许多微软的应用软件项目以及一些系统项目，运用一种所谓的“ 匈牙利语 ”命名惯例，这个名称源自它的创始人查尔斯·西蒙尼的出生地匈牙利 [6]（微软的匈牙利语与在欧洲使用的匈牙利语那种语言没有联系）。一种命名惯例是一个开发员给源代码中的程序和变量起一种明确的名称时所使用的类型或风格，它应该可以向那些试图调整或复用这一代码的开发员提供较高的可读性和可理解性。（软件开发中的一个普遍问题是源代码经常难被初创者以外的其他人理解。然而很多人还必须会读、理解并调整产品源代码，这样开发员们才有可能做到既会复用、改变或修改别人的代码，又能平行地工作。）

匈牙利语命名法开始在微软使用是在 1981 年西蒙尼从 Xerox PARC 转到 Multiplan 工作时（见第 3 章）。他和其他人，如前任高级开发员董·克兰德，在他们 80 年代工作的应用软件组里极力促成它的使用。[7] 匈牙利语命名惯例使得开发员们可以相对容易地阅读其他人的代码，只需源代码上最小量的注释。就乔恩·德·沃恩估计 Excel 产品代码中仅 1%条是用来注释的，但由于使用了匈牙利语命名法，代码仍是非常容易被理解的：“ 如果你看到我们的源代码，你一定会注意到注释非常少。匈牙利语命名法赋予我们只要进入即可阅读的能力……熟练掌握匈牙利语命名法会使你成为希腊学者一类的人物。你拿起一些东西就可以读它……我记得微软对此最初的反映是：‘ 这是什么？这是疯狂！它不会带来任何不同。 ’但是它确实带来了。 ”

在匈牙利语命名法中，变量名由三部分组成：前缀、词根以及一个修饰语。[8] 举例来说，变量名 pch 是指字符的指针；p 是前缀，表示“ 指针 ”，ch 是词根，表示“ 字符 ”。名称 pchFirst 是指字符数组的第一个元素的指针；First 是修饰语，表示是第一个元素。虽然这个例子看起来可能很简单，但还有一些相当复杂的名称。变量名 mpmipfn 是下标为 mi 的表示菜单项的函数指针。开发员可以用这样的数组来分派命令。mp 是词根，表示“ 数组 ”，mi 是这一数组的下标类型，pfn 是数组中所包含的元素的类型，它是函数（fn）的指针（p）。在匈牙利，人们先写姓再写名，就像微软开发员在他们的匈牙利语命名法里先写词根再写修饰语。[9]

Office 应用软件产品的绝大部分持续地广泛使用匈牙利语命名法。只有从 Multiplan 的早期版本而来的少量旧 Excel 代码不使用它。事实上几乎所有的 Word 代码都使用了匈牙利语命名法,除了一些暑期实习生所编写的代码（而其中一些人最终也修改了他们的代码并用匈牙利语命名法重新编写）。

爱德·弗莱斯提到，有些人特别讲究要恰当地使用命名惯例：“这和任何语言一样。它从某种意义上讲也有方言土语，而有的人对他们的匈牙利语命名法使用极其严格，其他人可能没有那么严格。但是它已是我们文化的一部分，我甚至不会意识到我在用它。”

但是系统产品并不大量使用匈牙利语命名法。这也许是因为匈牙利语以及查尔斯·西蒙尼未曾成为系统分支文化的组成部分。娄·帕雷罗里赞同不要过于广泛地争论这种语言惯例的好处，但他在谈及他的小组中某一开发员的经历时也谈到了他的疑惑：

> 匈牙利语命名法会给他添麻烦，根本就没有帮他阅读代码，也没有使代码更易维护。他所看到的代码与其他代码有同样多的错误……应用软件的人认为它很棒。我们并不反对人们使用匈牙利语命名法，但在我的 Windows NT 小组里没有人能在系统上很好地运用匈牙利语命名法。这是由于它并不像它看起来的那样有较好的描述性。描述性不是我们所关注的问题。我们的问题是同步化……
>
> 我认为即使由我来做一个应用软件，我也不会用匈牙利语命名法。我认为这是一个信仰问题，而在信仰问题上是不必争论的。我个人是在非匈牙利语世界里长大的。我们编写过庞大的系统，我们在 NASA 里所做的系统有两百万条代码，且都工作得很好。他们通过一种自上而下的方式和非常小的程序来做它，即使它是用 FORTRAN 编写的。

支持项目的通用工具

微软人可以很好地交流还应归功于各项目使用一套通用工具来帮助管理和开发产品。许多工具像微软产品一样可供出售；还有一些特制的内部工具或商业产品的补充，由于数量少而不在市场上通行。一些内部工具也为微软提供了一种主要的竞争优势。例如，微软能迅速为不同的硬件平台和操作系统，如 MIPS Altair、CP/M、AppleⅡ、DOS、Macintosh 以及 Windows 开发出不同的语言和应用软件（见第 3 章）。比尔·盖茨强调了创造和使用内部工具的好处：

> 我们可以控制我们的工具，因为我们使用的是我们自己的工具……在这一领域有很多值得称道的地方。为什么你能为 Macintosh 编写软件而别人不能？因为我们编写了我们自己的工具……为什么公司处于第一的位置？因为我们编写了我们自己的工具。没有其他工具能做得这么好。这是一个巨大的竞争优势……你只需做简单的事，就像我们为保护结构而在我们的编译器里处理内存指针一样。微软公司的优势在于我们是为真正的机器而编写。如果那是 8088，我们为就 8088 编写。如果它是 286 保护模式，我们就为那种机器编写。如果它是 AppleⅡ……
>
> 我们使我们的软件起作用。就规模和速度而言，没有任何一家软件公司能够接近我们的水平。而控制我们自己的工具正变得日益重要。即使在今天，我们还有工具来做这些工作。我们只是不把它们作为产品发布出去，因为那是我们的竞争优势。

微软的通用工具并无正式名单，而且任何名单都会随时间而改变并在不

同产品组之间有所差异。然而还是有大多数小组使用我们可以在这里描述的类似的工具。程序经理经常使用 Excel 来管理产品说明，使用 Excel 或 Microsoft Project 来追踪日程信息。如果需要，他们使用 Word 来编写和打印一份项目文档。程序经理一般使用 Visual Basic 来初创产品特性的原型。（它的一个版本在应用软件产品，如 Excel 中被用作一种宏语言，这样用户可以根据自己的需要来改变产品。）

开发员经常使用微软的商业化的 VisualC + +编译程序来从事 C+ +的编写，微软也拥有它自己的 C 编译程序。有些项目有特制的内部代码调试器和分析器，帮助理解和评估代码。不同项目和产品的编辑器也不相同。比如 Word 使用一种被称为主编辑器的工具，它包含许多特制的宏，用于浏览代码、分辨变量用途以及函数调用的联系。开发员目前使用 SLM 进行源代码管理、同步化以及集成；微软出售它的名为 δ 的商业化版本（微软还从 1994 年买入的“ 一棵树 ”软件公司获得了 SourceSafe 配置管理工具。有些项目用这个来代替 SLM 或 δ。）过去很多开发员在进行开发工作时使用 OS/2 作为他们的操作系统平台，这主要是由于它具有多任务处理能力（使不同的进程可以同时执行）。但是现在很多项目改为使用 Windows 95 或 Windows NT。例如 Office 就是在 WindowsNT 上构造的，即使稍后它要在 Windows NT 和 Windows 95 上同时构造以促进自动测试。

测试员使用内部的 RAID 工具来记录和追踪产品错误的状态。要进行广泛的用户界面测试的产品，经常使用微软测试来自动地执行和再执行键盘动作。有一些项目使用一种内部测试案例管理程序（TCM），它把 Visual Basic 作为“ 前终端 ”而与一个结构化查询语言（SQL）数据库组合起来。这一工具组织了测试脚本，记录了执行测试的信息。

使用一套通用的工具和开发方法使人们在公司内部调动时能迅速适应产品的变化。盖茨认为这特别重要：

> 我们努力使各组使用工具不要差别太大，因为我们希望人们能够在组际间相当容易地调动。我们确实鼓励大量的组际调动，这样人们可以从事新鲜的工作并且可以了解别的组进度如何。今天人们所使用的工具已差别不大了。过去工具之间存在差别，系统有某些工具，应用软件有另一些工具，现在这一问题已基本被解决了。

虽然微软在工具开发和使用方面实力很强，但仍存在着提高的可能。一些开发员继续使用过时的工具只是因为他们已经熟悉了它们，而且他们宁愿把时间花在开发产品上。盖茨承认在有些类型的开发工具上投资不足：

> 我们有一些在工具上投资不够充分的情况，也确实有些时候外部工具更好一些……很多年以前我们的调试器甚至不能达到正常水平……而我们现在拥有了一些与动态信息反馈相联系的工具，你观察情况，然后返回并改变基于它的编译和存储组织。为什么我们不在多年以前就这样做呢？由此我忽然想到我们的源代码控制系统已相当破旧了，然而人们还在使用它。许多人还坚持用它，而它确实已很糟糕了。人们只是习惯了那些东西，即使是像我们这些拥有正确想法，诸如应该投资于工具、应该在工具投资上采用长远眼光的人也不例外。在资产负债表上它

曾是一项巨大的资产。但是我们没有完全按照应采取的方式去执行它。所以现在它是一个巨大的负值。

但是即使是微软的相对不成熟的工具也有其优势。乔恩·德·沃恩指出他们的有些工具，比如编译器和调试器，与大学里学习计算机科学的学生所使用的相比是相当落后的。但他也提出，更复杂的工具可能不那么可靠。更进一步，他还发现微软的工具生成了就商业目的而言最好的代码：

> 我们的工具与学校里习惯使用的工具相比是相当原始的，信不信由你。事实上我们在 Excel4.0 才第一次进行源代码层次的调试。我深深感到这是微软工具在某种意义上滞后这一事实的写照。我们使用微软工具是因为与我们所评估的所有其他工具相比，它们生成了最好的代码。而对我们来说，关键是我们所出售的磁盘上的代码。这就是我们我行我素的原因。因此有很多像我这样的人，宁愿使用汇编语言调试器，因为它更快，也工作得更可靠。

在汇编代码层次上理解产品

绝大多数微软开发员还有另一项共同的能力，像乔恩·德·沃恩一样，他们能够在一个非常深入的层次上理解他们的软件工作的状况。但是他们不会对所有用户都利用这一能力。特别而言，在 Windows 和 Macintosh 版本之间共享代码经常意味着微软开发员优先考虑 Windows 代码，而尽量不要为其他平台编写更深入的代码。但是一般而言，开发员如果能在尽可能深的层次上理解代码，这将是极其有用的：这样他们就可以弥补微软编译工具的局限，确保在给定的项目目标下出品效率最高的代码。德·沃恩强调，至少在 Office 和 Excel 组中，高级人员坚持开发员必须能够阅读汇编语言。经理不仅让开发员使用较高层次的语言，如 C 或 C++ 来编写代码，而且还随机地在编译器上运行它。（编译器自动把高层次代码翻译成汇编语言代码，然后译成一种低层次的机器代码以使计算机可以直接理解。）比如 Excel4.0 中，语言的组成是 75.7%的 C 代码，10.8%的汇编代码和 13.5%的其他代码（如包含文件和标志文件）。[10] 微软也曾在 Word 中用 80%的 C 语言并将之在 PC 和 Macintosh 平台之间传送，另外 20%的代码采用汇编语言。

增加汇编语言的使用可以加快执行速度。例如在 WindowsNT 3.5 上使用 Excel 和 Word 的新 32 位版本的自动宏测试的计算约需 74 秒的时间；而旧的包含更多汇编代码的 16 位版本只需 70 秒。[11] 除了加快产品的速度，一些开发员还需要理解早期的 MS-DOS 代码，它是微软和 IBM 用 Intel 汇编语言编写的，其目的是为了减少内存要求和加快处理速度。

这种对产品的深层次的理解也包括掌握编译器生成不良代码的结构，然后编写代码避免或替代那些结构。德·沃恩估计他的开发员们对 50%以上的源代码都阅读由编译器生成的汇编代码。他还抱怨微软不得不教会开发员在这一低层次上理解系统。大学里的计算机课程一般并不充分强调如何保证对存储和执行时间的有效利用，而这对于出售软件产品给个人计算机用户是至为重要的。德·沃恩详尽阐述了这些观点：

> 我们确实鼓励人们在非常低的层次上理解系统是如何工作的……不

幸的是你的工具常常不能充分地发挥作用，Windows 和 Mac 系统也不是非常理想。所以你必须深入那一层次，找出它不能很好工作的原因。这是推动力之一。另一主要动力是我们对确保产生最好的产品代码兴趣浓厚。

当你看机器指令时，你就能知道代码的质量。你还会去学习——这是学习的唯一途径。你可以用 C 编写出好代码，但为了做到这一点，你必须了解你的编译器是如何生成代码的。如果你永远只在源代码层次的调试器里走来走去，你就找不到任何线索。

这是我们必须教给人们的态度。人们在学校里学习时，那没什么关系。但是当你制造一件人们想买的产品时，那绝对是有关系的……思路是去掌握编译器所不擅长的构造。

我们可以要求编译器方面的专家来修改它，但是从某种角度上讲那是一件长期的事情，我们要求的是编写不存在这一问题的代码。

原则四：在构造产品的过程中持续测试产品

微软里有不少原则指导着测试，而每一条原则都帮助小组在构造产品的过程中持续地测试它们。这种在开发同时测试的概念使微软与众不同：因为太多的软件开发员基本是在开发周期的尾声才强调产品测试的重要性，而这时要修改错误可能是特别困难和耗时的。而且，微软组现在是从一个相当广泛的角度来测试他们的产品的。

比如，项目创造了开发员可以每天运行，并且必须在他们把代码改变记录入源代码主版本之前运行的自动测试。测试系统通过使用产品的宏语言或模拟键盘敲击而自动执行。开发员们创造他们产品的“调试版本”，包括特别规则，不仅为了检查特别的条件，还可帮助发现和确定错误的位置，他们还实施代码复查以在阅读代码的过程中发现错误。可用性测试邀请“街边”的人们来到专门的实验室评价产品特性使用的方便程度。项目还把测试员与开发员配对（在微软，每一个开发员差不多都有一个测试员）。测试员通过阅读说明而在早期就开始准备测试方案以及测试策略；开发员在记录他们的代码之前向他们的“测试伙伴”提供他们代码的私人版本以供测试。

测试员然后使用多种方式。一种包括高度结构化的测试脚本（称为“基于方案的测试”）。另一种是“大猩猩测试”，测试员并不遵循任一脚本，而是试验他们所能想到的每一件东西来试图使产品出错。另外，微软项目还举办“臭虫聚会”，这是不同的人聚到一起共同寻找程序错误的集会。项目成员还使用他们所构造的产品，在内部分发初期的版本，以及向重要用户发出数以千计的β拷贝——所有这些努力都是为了在向市场发布一件产品之前找到错误。

为了尽可能快地测试并鼓励测试员在性能领域中获得专门知识，项目把一个典型的产品测试组重组成一些平行的小组，每个小组侧重于一个特定的特性领域。比如 Excel5.0 测试小组由 45 个负责测试 Excel、MS Graph 和 Query Tool 的测试员组成；他们还开发和测试内部的 TCM 测试案例管理工具。项目把这 45 个人组织成 6 个测试小组，每组有 6 到 9 个人。

当然，由于软件产品的复杂性、时间安排上的约束及人力的限制，测试不仅是一门科学，也是一门艺术。不幸的是，不能完全地测试一件产品也是

确凿无疑的。微软的产品在不断地进步，但它们当然并非完美无缺，特别是在它们的早期版本中或当它们包括全新特性的时候。过去，微软也曾向市场推出过未经充分测试的产品；而随着新开发方式的采用和对决定何时一件产品准备好了推出的标准的日益精确化，这种现象已变得极为少见了。对那些在目标出品日特性未能完全工作或错误数量高于期望水平的产品，微软还会进行扩展β测试和延迟最终发布，特别是对系统产品。

（Windows 95 是这类延迟的一个较新的例子，它包括大量的新代码和复杂的新特性，诸如即插即用以及对应用软件程序的多任务处理。）

快速测试

促使每日构造过程有效运作的一个关键工具是一套被称作“快速测试”的高度自动化的测试。当一个开发员完成了一项新特性而尚未把源代码改变记录入源文件的主版本的时候，他构造产品的一个私人版本（见表 5.1 中所描述的每日构造过程的步骤 7）。然后他在其私人版本上执行快速测试。这确保了当他加入新特性后产品的基本功能仍保持良好（见步骤 9）。

一件产品的宏语言除了为使用者提供按自己的要求改写和扩展产品功能的途径外，还为自动执行这些快速测试提供了有效机制。快速测试并非直接测试一项新特性；它只是确保一个开发员所作的改变没有与产品的其他特性发生冲突并间接引起错误。如果快速测试成功，开发员就可以记录他的代码改变了。微软把快速测试自动化了，这一点十分重要，因为许多开发员要执行很多遍这种测试，而且它会检查许多开发员也许会遗忘的特殊情况，然而开发员也不能完全依赖于快速测试，他们还需与他们领域里的其他专家交谈，正如爱德·弗莱斯所描述的：

> 从测试的角度看，大量的自动化是完成它的一条途径……从开发员的角度看……你做一些真正全新的事的情况是非常少见的。但愿情况是这样的：当你要把一些东西放进去时，你在代码中寻找一些类似的东西，然后你模拟它。于是你会看到所有这些特别的情况，如代码的有些部分说：
>
> “如果录音机是开着的”，我会对录音机做些特别的事，然后你想“如果不做又会怎么样？”然后就有了一些不做的结果。这确实是捕捉那些对你不明显的情况的最好途径。另一件事就是我们非常依赖组内的专家。在不同的领域有不同的专家。只要人们交谈，就能产生许多素材。

测试“伙伴”和私人版本

持续和同步测试的原则还意味着测试员必须在开发员编写代码的时候紧挨着他们工作。同时，为保证与代码保持一定的距离，微软的方法是测试必须保持与开发相独立的功能。如我们在第 1 章和第 2 章所描述的，测试员并不向开发组或程序经理报告。他们向测试经理汇报，再由测试经理向产品单位经理报告。但是项目通过在特定特性的小组里指定开发员和测试员一起工作而实现这种紧密的联系。而且测试员不仅测试新代码，还有机会在说明发展的过程中复查它以及在代码进化过程中平行地编写测试方案。克里斯·彼得斯强调这种开发员与测试员之间紧密联系的重要性：“在开发和测试方面能有独立然而是平等的组织，这使你可以避免出品坏东西。开发和测试需要

深层次的融合，否则你会碰到这种糟糕的“检查员 12”的现象，即开发员把东西扔过墙。我们现在所做的是测试员和开发员，至少是测试小组和特性小组，通常被配对，这样他们就会倾向于彼此一起非常紧密地共同工作。”

如我们已指出的，许多微软开发员对进展中的产品每天生成一个个人的每日构造，无论他们是否计划在那天记录代码改变（见表 5.1 的步骤 3 和步骤 7）。所产生的“私人版本”因而包含了开发员所作的尚未记录入源代码的主版本的最新版本的改变。开发员与他的指定的“测试伙伴”共享他的私人版本。这使得测试员可以估计特性的进展情况，也防止了许多错误进入源代码的主版本。马克·奥尔森指出这一技巧本身就创造了开发员和测试员之间的紧密的工作关系。它也为开发员提供了更多的时间来编写代码和改正问题：

> 私人版本测试的目标是在这一版本为小组中所有其他开发员所共享之前，在缺点进入真正的产出代码或进入主源代码库部分之前找到这些缺点。其他开发员不得不处理由于你记录的代码而引致的问题。所以关于私人版本测试有一些相当严格的规则……这里只有开发员和测试员，他们形成一种工作关系……根据他们所发展的信任的水平，他们可以是一个非常紧密的小组。开发员可以致力于编写代码和修改问题，测试员则关注于使这一特性迅速地稳定化。

开发员在记录他的代码之前要至少有一次使特性通过私人版本测试，而且经常是每天都这么做。爱德·弗莱斯相信非正式的私人版本测试缩短了测试员发现错误和开发员改正它之间的时间：“我们喜欢在从一个特性转向下一个之前作一次私人版本测试。这意味着编写一份测试版本文件和为你的特性专门做一份用于测试的特别版本。这是为了在保证事情的稳定的同时在不进入错误数据库的情况下先行测试，它使你能搜索到许多迅速产生的错误。这使我们比那种搜寻更正式的错误数据库的过程可以更快地调整方向。”

调试代码

我们在关于多种版本的讨论中曾提及在产品的开发过程中，微软开发员为他们的产品创造“调试版本”。这来源于把不同的特定测试的表格，或“调试代码”，插入产品的源代码。当开发员做一个构造时，他把调试代码打开作为调试发送的一部分。而在产品发送时他把它关上，这样出品给顾客的版本就不会有额外的代码（开发员通常是通过一份在产品的用户界面上不可见的菜单来访问调试代码。比如 Word 的不可见的菜单紧挨着“帮助”选项）。

开发员必须把编写他们自己的调试代码作为他们日常工作的一部分。弗莱斯提到他们不得不计划开发调试帮助；否则人们不会做这些：“无论何时，当我们在计划产品的所有新特性的时候，我们努力弄清：‘那么，什么是我们还需要做的基础工作呢？让我们确保我们节约了一些时间并且为那项工作安排了一些时间，否则它是不会被做的。’……我们努力为每一版本计划一些东西，一些会被加入整个程序的基础性的东西。我们同普通的特性一道计划它们。”调试代码主要为管理产品的执行和检查例外条件提供自动化的内置测试。这些帮助开发员改进存储器的使用和提高产品的执行速度。当使用调试代码时，开发员也可使用源代码或一个汇编调试器工具在程序执行中观

察其内部的数据结构。弗莱斯描绘了 Word 中一个双重的、冗长的特性完成例子。小组用 C 语言完成了一套例程，又用本地计算机的低层次汇编语言完成了另一套（他们这样做是为了达到效率与可移植性之间的平衡）：

> 我们在我们的应用软件产品中所做的某些肯定正确的事是我们给它们构造了大量的调试代码。在 Word 和 Excel 中都是如此。由于应用软件是如此巨大，一个好的框架里就必须包括你想构造来支持它和使得追踪问题较为容易的部分。一个例子是 Word 有 C 语言和本地语言两种版本。对其例程的所有汇编版本都有 C 版本。所以如果我们有一条用手动汇编语言编写的例程，我们也有 C 版本。而在我们的调试版本我们两种都包括；我们可以飞快地在使用 C 版本和使用汇编版本之间转换。而你拥有同一件东西的 C 版本和汇编版本的一个普遍问题是它们可能不同步。如果能保持它们同步会很有帮助。事实上你可以启动一种检查方式，依次运行这两种版本，然后确保结果是相同的。

确实有很多关于微软开发员使用调试代码提高他们构造的产品的质量或效率的例子。总体而言，即便在一件小产品里这些也可能达到成千上万条代码。特别重要的是存储器和数据结构检查、断言和工具化（工具版）。调试代码在产品构造向测试组的每周发布中也扮演了特别重要的角色。存储器和数据结构检查“存储器检查”说明了一个程序使用或分配的内存的每一个字节。比如，如果内存的一次自动检查失败，一个“MIF”（存储集成失败）的信息就会出现。一种类似的称为“数据结构检查”的技术使用特别的例程来保证产品中几乎每一个数据结构的一致性。这些保证程序已经正确地存储数据并可检索该数据。它们是一类运行于“空闲时间”的缺省的进程，在计算机处理器有空闲时间的时候自动地进行检查。

其他相似类型的调试代码模拟资源分配中的失误。例如 Excel 可以模拟几乎所有对操作系统功能的要求的失误。它通过给程序增加重复的“循环”和利用一个命令发送者“阻碍”大部分的 Excel 功能来进行模拟。这种循环给资源分配失误设置了域值，然后要求操作系统的一项特定功能。这项功能最终会在某一点上失败（由于所定义的域值）并返回一种失误状态。循环于是就完成了存储配置检查和数据结构一致性检查。如果没有问题，调试代码就提高失败域值到下一个最高水平并继续测试。如果操作系统功能失败了，则调试代码发现了一个问题并向开发员显示一个错误信息。其他调试代码的例子包括可移动内存检查、动态在“堆”中分配的内存再配置（称为“摇摆”）以及强迫的失败（“堆”是项目在执行过程中为保存数据而分配及回收的内存范围）。

断言　调试代码的最普通的例子是“断言”。这些是代码中“如果——那么”测试，并且不管用户输入了什么数据，它都必须为真。比如一件银行应用软件产品可以在一次交易之后使用一个断言来测试一个支票账户的余额。它可以说：“如果余额大于等于 0，则通过；否则失败。”这种断言的陈述应该永远是真实的。许多微软产品倾向于在程序中共享信息的“全局状态”。开发员经常使用断言来检查他们的关于程序中全局状态或传送进、出一个函数的特定“参数”的假设。如果人们在编写代码时没有充分考虑程序的全局状

态或这些参数，就有可能产生很多错误。所以许多项目都有一条非正式的规则，即在开发员对存储在程序其他位置的数据作出假设的时候必须包括断言。乔恩·德·沃恩解释了这些断言会使追踪错误相对较快：“断言相当重要，因为……错误的一项巨大来源是人们不了解全局状态或在对全局状态作了一些假设的情况下编写代码……我们试图建立这样一种习惯，即如果你要对全局状态作出假设,你必须在你所编写的代码中断定它。对参数也一样……如果你要对你函数的参数作什么假设，你必须断定它……有了充分的断言，你就可以马上发现错误。”

工具化　开发员还把调查加入代码之中，记录用户选择了哪些命令及源代码中哪些相应语句被执行。微软把包含这些调查的产品版本称为“工具版”。经用户同意并送回运行结果，这些版本可以成为内部测试、可用性测试以及区域β测试的有价值的补充。从工具版获得的数据使得开发员可以很容易地再现用户的行为，理解他们的问题，并且避免了不能重新模拟错误的情况。比如说，开发员重做出了 Windows 的 Works 2.0 中所报告的错误的 96.6%[12]，以及 Windows NT PDK 2 中所报告错误的 91.4%。[13]

产品的工具版所能生成的数据的数量是极大的。事实上，比尔·盖茨就炫耀过微软的数据中包含用户所执行过的数以百万计的命令：“开发员和测试员也采用工具版。它是记录人们所使用的命令、所发生的错误以及所造成的结果的版本……我们可以对事态的进展作大量的分析：人们为什么得到错误信息？什么是他们确实普遍使用的命令？什么是他们顺序使用的命令？也许我们可以在任务层次上捕捉到它？在工具化数据库里我们确实拥有用户所提供的数以百万计的命令。”[14]

工具化还表明了源代码的哪些部分用户使用较多、哪些部分使用较少；软件开发员把这叫作“执行范围”。在整个执行范围内进行的测试可以帮助发现那些藏在源代码的较少使用的部分里的错误。德·沃恩观察到项目普遍在开发的后期使用自动化的测试（如一小套用一种宏语言编写的联结了很多操作的程序）来进行执行范围测试：“最终调试时间的一部分会被用来使开发员能帮助测试员进行覆盖更多代码的自动测试。”

每周测试版本　在开发阶段，开发员每周一次把产品的每日构造提供给测试组。这种“每周测试版本”倾向于使用调试版本，因为这种版本中的方便之处如断言可以更快反映出问题。当项目接近出品日期时，开发员的每周版本会更多是产出版本（它不包括调试代码的数千条额外条码）而非调试版本。如弗莱斯所言，测试员还会编写他们自己的用以测试代码的例程并把它们加入程序：“我们有一些水平相当高的测试员。有一个基本上是程序员的组。他们通常十分清楚自己想要什么并问道：‘我们可不可以随意进入你们的代码并任意插入一些内置测试，这样我们可以更好地测试它？’当然我们会说：‘好。’”

代码复查

在完成特性以前，微软组有限地做一些正式的、高强度的设计复查。因为今后的测试要求有能运行的版本，所以代码复查是为了在此之前就发现遗漏的代码或一致性问题。但是一般而言，微软组不为它们的产品做深入的、

广泛的设计或代码复查，因为它们在开始编写代码之前对产品功能只做了大致的描绘（见第 4 章中关于概要说明方法的讨论）。更由于在项目过程中说明和代码也会有很大的改变，确定复查的时间也是困难的。

在短开发周期的压力下，微软人没有时间去研究是在最开始就多做一些复查和设计工作比较好，还是迟些时候再为这些问题设计更好的检查方式比较好。比如乔恩·德·沃恩会在要完成一项特性时试图考虑所有可能引起问题的偶然性，但在他刚刚开始设计这项特性时就不必这么做：“我们从某种意义上讲是放弃了认为我们可以预见所有情况的那种想法。这就是为什么我们为实际的特性完成的阶段留下了相当的灵活性……我们也许可以通过进行更多的、正式化的设计复查来减少案例的数量，但是没有办法可以根除它们。所以问题就变为你怎样更快地找到它们。你是通过做测试来更快地找到它们呢？还是通过事前的思考？”

高级开发员倾向于通过复查新人或从事产品的一个新领域的人的代码来了解他们的素质以及帮助他们学习。通常，最好的专家——有时是小组长，有时是新人的指导者——会在新人记录之前复查他的代码。代码复查提供了一种有效的同伴对同伴的机制来学习所完成代码的技巧，就如德·沃恩所说的：“对于新人，我们复查他们的代码。当有人开始进行新任务时，也许其他人曾是这一领域的专家，那么他们在记录代码之前会让最好的专家复查他们的代码。他会告诉他们‘这做得不错’或者‘这的确是较好的做法。’思路是人们应当在实践中学习，而他们确实如此。指导者通过回答问题和做其他事务来帮助他们前进。然后，当他们认为他们已做好了代码时，他们能通过一次代码复查来获得技巧。”

测试版本和测试版本文档

前面我们提到，项目定期向产品的测试组提供一份演进中产品的测试版本。测试员把测试版本中发现的错误记录在一个数据库里，以汇总各种出错情形。这个数据库是用 RAID 工具管理的，而经理们利用这些出错数据进行进程追踪。项目过程中，一个组通常每周构造一份测试版本，并把一份叫作 TRD 的测试版本文档附加到版本上。比如 Excel5.0 在每星期一晚上构造它的每周测试版本并在星期二提供给大家。

测试版本文档描绘了初次完成和能被移交去测试的每一项特性。开发员为他们的测试员生成这一文件，并且开发员一般是在测试他们的私人版本之前编写这一文件中他们有关的部分。文件的规模因特性的规模而异，它向测试员解释特性是如何工作的，以使他们可以对之进行有效地测试。测试员还把说明文件作为测试信息的另一来源。爱德·弗莱斯解释了测试版本文档所扮演的角色：“有些文件是我们要求程序员必须生成的，这关系到程序员如何与其他组协调行动。当他们创造了一项新特性并准备转向下一项时，就必须把它提交去测试并配合测试工作。他们需要提交一份测试版本文档，向测试员解释这一特性工作得如何，他们应该看什么东西、应该测试什么东西。”

可用性测试

微软项目主要在产品开发阶段进行的一种完全不同类型的测试，是在一个正式的实验室环境里进行的对新特性的可用性测试（也可见第 6 章中我们对此的讨论）。这类测试尤其适用于应用软件产品中的用户界面和特性，因

为它们都应强调使用的方便。微软拥有一个专门的由 30 到 35 个专家组成的班子，运用不同的可用性标准对几乎所有的应用软件产品和一些系统产品进行评估。由于这一测试是持续进行的，它代表了一项对时间的巨大投资，可在项目期间，它生成了价值难以估量的顾客信息反馈。比尔·盖茨这样评论 Office 的广泛的可用性测试过程：“我们得到基于所有这项工作而构造的原型，然后把它送入我们的可用性实验室，在这里我们观察那些真正坐在应用软件前面的人们。再一次我们获得信息反馈——我们用这个来一次又一次地改进。对于 Office 4.0……我们在确定已把它做好并准备把它投入市场之前进行了超过 8 000 个小时的可用性测试。”[15]

其他公司的可用性实验室仅用来测试新的产品思路或者改进已有的产品。然而微软把这一思路推到了极至，把评估特性的可用性作为了开发过程的一个常规部分。当开发员认为他们已经或接近完成一项特性，程序经理就安排这一特性进入可用性实验室。有时如果开发员尚未完成特性，但希望在实验室里测试一下其原型，就为特性做一个特别版本或一个实体模型。一般而言，应用软件中每一项会影响用户的特性都要通过可用性实验室。每一项特性的小组长管理小组的日程安排，整个项目还有一份主日程安排；程序经理利用这一进度的信息来安排特性进入可用性实验室的时间。

在一个典型的可用性测试中，实验室人员招募一些外部人员，每十人组成一组来代表不同类型的用户。他们坐在特别的单人房间里。实验室人员要求他们用一种产品来完成一项特定的任务（如创造一份新文件或把金融数据组织成季度简报），然后可用性专家从后面的单向镜中观察和拍摄用户的行为。经常人们得不到什么指导，所以如果产品不容易理解，他们就难以完成任务。可用性测试员记录用户的问题和那些无需手册指导就可以成功完成每一项任务的用户的比例。测试员对一种特定产品每周做两次实验室测试，在每次的半天会议上评估大约三项特性。开发人员阅读有关实验室中遇到问题的报告，并与程序经理和可用性测试员共同讨论解决办法。

可用性实验室一般不从整体上支持产品计划，比如帮助项目对行为作出主要决策，那是第 4 章所讨论的基于行为制订计划技术的目的。实际上微软主要是利用这些实验室提供的几乎是立刻的信息反馈，以帮助开发员决定如何使新的或已有的特性更容易理解。比如开发员经常不得不为了某种特性而牺牲掉不少外观或行为。他们还必须解决用户界面设计上的细节问题，或在一小套有类似功能的特性之间选择最容易使用的那种。绝大多数开发员会亲自访问实验室去观察处于实际使用中的特性。如爱德·弗莱斯所解释的，这使他们掌握第一手资料：“程序员倾向于当他们个人的特性正在进行可用性测试时到可用性实验室去。他们观察人们对它的使用，看他们有什么样的问题，从中得到如何做得更好的想法。所以可用性测试也许比其他测试对我们所生产的东西有更大的影响。”

然而实验室也有一些局限：它们对新用户的考虑远超过对有经验用户的考虑，而且它们在定义清晰的问题时最拿手。不过可用性测试毕竟有重要的优势。它不仅使微软可以保证新增特性容易使用，还使微软可以不断添加新的有力的特性和避免不太有用的特性（有时被苛刻地称为“特性爬行”）。可用性实验室比 β 测试有优势，后者通常很晚才提供信息，并且经常缺乏细节。如克里斯·彼得斯所描述的：

在那种实验室里，可以得到你要改变特性所需要的数据，而且还可以在尚在开发一项特性时经常这样做，而β测试则不行。在推出可用性实验室之前我们从未有过这些数据。经常有人来抱怨有些东西工作得不那么好；但是你不会知道为什么……我们得经常注意那些自认为理解了问题的地方。我们会做好一项特性，然后没有人会用它。每个人都会对我们极其极其的愤怒，因为它只能完成任务的80%，却就此停止无法继续。这是因为我们从顾客的要求中过滤了太多东西……在人们所说的他们做了什么与他们实际的所做之间经常有非常大的差距——这是我们注意到的另一件事。

虽然在开发过程中开发员会进行可用性测试，程序经理和开发员还是用了更多的时间来再定义：如果他们在开发的早期就进行这项测试的话，特定的特性会是什么样子？会如何工作？但是很多微软产品有大量的图形式用户界面，测试这些产品经常导致改变发生在生命周期中较晚的开发员较好地理解了用户如何与产品相互作用的时候。麦克·梅普尔斯观察到：“我们并不在项目一开始的时候就说，编写好说明，然后照着说明构造它。我们知道事情会改变。任何时候如果你拥有一件非常交互式的用户界面驱动的产品，事情在最终有时并不像你所想象的那么好地工作……你最好还是修改它。”

性能的基准测试和大猩猩测试

系统产品应该具有极高水平的性能（执行功能时对用户的答复时间、对内存的使用和对磁盘的使用都达到尽可能的小）和高水平的可靠性（执行功能时失误的频率或数量降为0或非常小）。性能测试的一个例子是使用行业标准化的评价标准——称为基准测试——来评价产品的执行速度。不同的计算机行业组织在某些类型的操作或计算顺序的重要性上达成了共识，并把它们设置成标准的基准测试以促进不同产品之间的细节的比较。布兰德·斯尔文伯格概括了在系统产品上基准测试的使用：“我们拥有完整的、非常广泛的性能测试以测试一个非常宽泛的范围，不管它是中断的等待时间（为进程之间的低层的转换）还是首尾相接（为完整的用户命令）。可以在一个非常微观的层次上——像中断的答复时间之类——或在一个非常宏观的层次上，运行这一整套应用软件。那里有Babco测试或Winstone测试，或一些著名的应用软件测试组。我们运行整个测试组和性能组，然后看我们做得如何。我们追踪它们并衡量我们的工作配置如何（内存的配置），然后我们就有了需要达到的某些配置的目标。”

系统项目所采用的一类可靠性测试是大猩猩测试。这里，专职的测试员设置并执行自动化测试,目标是试图通过在极高要求的层次上运作特性来‘打垮”产品。大猩猩测试员作为常规测试员的补充，在系统产品中就像在应用软件产品中一样被经理人员以一比一的比例指定给开发员。娄·帕雷罗里回顾了在Windows NT3.0上的测试：“测试员对开发员的比例可能是非常接近一比一……也许你还会有两或三个测试员测试安全性或测试文件系统，或测试存储管理功能。然后，你有大猩猩测试员或系统功能测试员，他们的工作是找出会被打垮的东西。所以他们会编写一个命令程序在一夜之间安装和拆卸同一驱动器，检查它在第二天是否仍正常运行。”

系统产品可靠性测试的另一个例子是对低层次的应用编程接口（API）的

测试，API 为应用软件产品提供了一种直接访问操作系统函数的机制。操作系统 API 函数的可靠性对于一个应用软件的可靠性非常重要。许多应用软件产品的卖主都希望像微软这样的公司在发布一个操作系统之前就做好可提供给他们的 API 产品。那么微软测试员必须在项目的早期就使操作系统 API 函数稳定下来并且可靠，这样，应用软件卖主就可以使用 API 来“转换”其应用软件。Windows 95 的测试经理乔纳森·曼海姆描述了他的产品的测试过程：

> 测试操作系统是一个多阶段的过程，并取决于你当时在观察哪个方面，可能在不同的领域有不同重点。比如，在开发过程的早期，你非常强调直接测试 API。你对内核以及应用软件程序进行低层的测试，这部分地是由于这样就可以在它们出现在操作系统的较高层次的函数中之前找到那些低层次错误，部分地是由于这样你可以使开发接口被很好测试并可以提供给 ISV 们（独立软件开发员），以使他们能够在一个操作系统发送之前就开始为它开发应用软件……
>
> 然后，在过程的较晚阶段，再转向较高层次的测试、用户界面测试、应用软件测试——所有这些你在过程早期不可能真正做好的事……在过程中的这一点，我们基本上处于较高层次应用软件测试的阶段，直接的 API 测试已不能再找到更多错误了。系统已经被很好地调整，我们的绝大多
>
> 数错误是通过运行应用软件并在用户界面上找到的。

内部使用和自产自用

持续测试的另一种极其重要的形式是在公司内部尽可能快地让各项目使用新产品，或尚处于开发中的新产品。微软人非常信赖这一做法，有时把这种内部使用的行为戏称作“自食其果”（见第 6 章）。对产品的第一轮测试而言，内部使用还被证明是极其节约成本的机制。微软有成千上万的每天都要使用计算机的雇员，所以为什么不让他们把使用产品的最新版本作为他们工作的一部分并报告他们所发现的任何错误呢？当项目使用它自己的产品来构造它自己的作品时，微软人把这称为“自产自用”。这一做法与操作系统特别有关联。例如 Windows 95 的开发员在他们的机器上运行最新的内部发行版本，所以他们是用 Windows 95 来构造 Windows95 的更多特性的。

布兰德·斯尔文伯格解释道：自产自用使开发员可以与他们的最终用户保持紧密的接触。他把这与“交叉开发”相对比，后者是指项目使用与它的目标平台不同的平台来开发产品（比如，用 OS/2 来开发 Windows 95）：“我们是完全自产自用的。我们确保开发员使用他们在构造的系统，而不是交叉开发。我认为，我所看到的由于交叉开发而引起的产品失败多于任何一种其他开发技术所引起的失败。你失去了与目标环境的完全接触；你失去了与用户所看到的东西的完全接触。”但是正如我们在第 6 章所进一步讨论的，自产自用和利用微软雇员进行内部测试的好处是有局限性的，因为他们并不总是最具代表性的顾客。即使是比尔·盖茨也承认这一点：“微软并不是所有微软产品的最典型用户。并且，如果你只在这些组内进行测试，你会遗漏某些东西。”

β 测试

由于产品可能用途的组合数量庞大，所以微软采用的用以在发布前评估

产品的多种测试方式仍不足以发现所有的错误。不同的命令、数据输入和所依赖的系统结构可能引起几乎无限的组合。比如，作这样的假设：一件系统产品有 100 项特性，每项特性有 1 条正常的执行完成，以及所能生成的 2 条错误信息。这一产品在 20 个不同卖主的磁盘驱动器上运行。它将运行于 4、8、 16 或 32 兆字节的内存中。产品要与 15 台不同卖主的打印机以及五台不同卖主的视频卡相联。它要支持 100 个最流行的应用软件，每个应用软件有 50 条用得最多的命令。仅仅是为了开始测试这一产品，测试员就必须建立一个可以支持超过 90 亿种用途方案组合的实验室（因为 100×3×20×4×15×5×100×50 大于 90 亿）。即使这样一个实验室是可行的，它也将是很不合算的——而且这份组合的清单也远非完全。

一种被称为β测试的大区域测试被用来测试那些具有多种可能的用途组合的产品。系统产品特别常用这种类型的测试，往往会花费 6 个月到一年，甚至更长的时间进行β测试。布兰德·斯尔文伯格回忆了微软如何开始认识到对系统产品进行大范围β测试的必要性：“另一件我们在系统产品组里做的真正的事是我们的β测试程序，它改变了软件开发的完成途径。我们从 Windows 3.0 开始，然后从 MS-DOS5 以及 Windows 3.1 认识到系统软件需要在如此多的机器和如此多迥异的配置上使用。在这一行业你根本没有办法预测人们会把什么插到 PC 里。我们不可能在内部测试所有情况下的所有配置。我们只能极度依赖非常宽泛的β测试。”

MS-DOS 6.0 的β测试员有 7 000 人，Windows 3.1 有 15 000 人，而 Windows NT 3.0 有 75 000 人。 1994 年 6 月 Windows 95 的β-1 版本的β测试员的人数为 15 000 人，而微软计划到 1995 年 8 月以前使这一领域内 Windows 95 的β测试拷贝达到 400000 份。斯尔文伯格回忆起使他的微软同事们提高β测试的规模是多么的困难。但是由于大型测试的好处如此显而易见，微软扩大了这一实践，至少是在系统软件上：

> 过去以及现在的应用软件中仍是如此，一个β测试有 300 点、 500 点，也许 1000 点。对一个应用软件而言，有 1000 点的β测试程序就是一个庞大的不可思议的β了。但是我们意识到为了达到一定的覆盖，我们必须真正依赖于外部的β测试员……在 MS-DOS 5，我们达到 7 500 个β点。当我初到委员会时，最初的β测试计划是 300 点。而这时充满错误因而是巨大灾难的 MS-DOS 4 刚刚出品。我向史蒂夫·鲍尔莫建议……也许我们可以做到 2000 或 3 000 点，然后这就像……“什么，你疯了吗？——3000β点？没人曾这么做过。你到底在想什么？”而它工作得如此之好，对β有如此多的需求，最终我们结束时有 7500 点。结果是产品绝对的稳定。在 Windows 3.1 上它也工作得非常好，我们最终达到 15 000 点。结果产品非常非常稳定，而且与硬件和软件的兼容性都很好。

很多软件公司都使用β测试，当然由于其产品销售规模的庞大，微软动用的β测试人数远大于其他厂商。微软还在随时间流逝增加β测试员的数量。由于β测试的技术上的好处，这种情况并不奇怪。而且β测试还在用户中激发起对一种新产品的狂热，这有助于在行业的主要业者中获得支持——例如鼓励应用软件生产者提早开始为像 Windows NT 或 Windows 95 这样的新

操作系统开发产品。许多微软的批发商、二手商以及独立的软件开发员，包括像 Lotus 这样的公司，都是微软系统软件的常规β测试员。

在β测试中，参加者收到一份发布前的产品版本（称为β版本）并自己使用它。这些β测试员成为测试小组的非正式扩展人员。他们评估和测试β版本，然后向微软报告任何错误或问题。一般而言微软的β测试员没有报酬，也对软件不付或只付很少的费用；他们还需订一份合约，许诺不把产品的特性透露给竞争者。Windows NT 的 75 000 个β测试员无需签订合约，但必须为β版本支付一小笔费用（所以他们实际上是购买了软件）。 Windows 95 的β测试员签订了合约，而只有后期的测试员才必须为软件付一小笔费用。

急切的 PC 用户是积极的β测试员,因为他们希望在购买之前试出最新的产品特性并对软件在他们的特定机器上工作的状况作出评价。然而微软努力使β测试员的范围更为广泛，以便能够充分代表众多的不同硬件、软件和网络的组合。测试员在微软完成其正式的产品版本之前提交错误报告。在最近这几年，β测试员使用电话拨号账户和 CompuServe 上的 E-mail 来接收β版本，以及向微软发回他们发现的错误和建议。

微软项目指派专门人员来监控 E-mail 邮路交通并直接答复β测试员。斯尔文伯格指出：“我希望确保论坛是被妥善管理的，这样，每一个从事β测试的人员都在 CompuServe 上的一个私人 E-mail 论坛中有一个账户。我们任命很多人去管理和回答问题，并且非常鼓励开发员在论坛里花些时间，这样他们既可以自己看到被提出来的问题是什么，又可以亲自回答问题。”斯尔文伯格自己也会花些时间来阅读从β测试员信息反馈回来的 E-mail：“而且我自己在一件主要产品出品前的最后两三个月里，不管它是 MS-DOS 5、MS-DOS 6 或 Windows 3.1，差不多每天要在 CompuServe 上花两到三个小时的时间，只是回答问题，只是获得对产品的一种感觉。你可以把它想象成系统其他部分的最新消息发布。我从我的测试经理和开发经理那儿得到错误报告。但是当我上到 CompuServe 上，我能看到报告的第一手资料。所以如果它们之间没有联系，我就对我们该做点什么心中有数了。”

Windows 3.1 的数据显示了β测试可以非常有效。最终的β测试发现了产品发布前所找到的全部错误的 22%。[16]Windows 3.1 所有的外部测试，包括最终β和所有较早的β版本，发现了全部错误的 27%（“外部”这里指任何微软外部的人员，如β测试员，或微软内部非 Windows 3.1 项目的人员）。[17] 然而微软在对某些产品的测试上仍有麻烦。由于顾客用来与一个操作系统共同使用的不同的计算机品牌、应用软件以及外围设备（特别是打印机）的组合极为庞大， Windows NT 和 Windows 95，甚至还有 MS-DOS6（特别是数据压缩特性）要在发布前充分测试是极其困难的。结果就是微软项目还要依赖于多种方法进行测试。他们还持续地评估自己的工具和技术，以找到更好的测试方法。

比较不同测试技术的效果

微软报告展现了项目用来发现错误的不同测试技术的效果。比如，对于 Windows NT 3.0 项目，效果最好的技术是用途测试，它发现错误的比例最高（15.0%）。[18] 用途测试通过模拟产品的日常使用来发现错误。应用程序编程接口（API）测试发现错误的比例次高（12.8%）。这种测试通过直接调用而执行一个特别的 API 函数。API 测试对项目早期错误的发现贡献颇大，但其

贡献会随项目进展而日渐削弱。应用软件测试，特别是 16 位应用软件（Apps-16）测试开始的时候对发现错误的效果不大，但它们的效率会在项目后期随着测试员更倚重运行应用软件而提高。有些测试技术是非常有用的，因为它们发现了相对较多的严重的错误；比如压力测试仅发现了全部错误的 3.8%，但这些错误一般都是相当严重的。表 5．2 定义了 Windows NT 3.0 所使用的测试技术。[19] 表 5.3 概括了这些不同技术的效果。

表 5.2　Windows NT 3.0 中所采用的产品测试技术

用途测试	在对系统的每天使用中发现错误。
API 测试	被编写来通过直接调用而运行一个特定的 API 函数的测试。
其他专门测试	不与其他任何标准相吻合的测试。
Apps-16 测试	使用 16 位应用软件的测试。
大猩猩测试	人们通过在极高要求条件下用来使系统崩溃的测试。
用户界面测试	对用户界面的测试，包括命令行、文件管理器等等。
压力测试	压力测试方案，不包括也是压力测试的 API 测试，但是包括那些使用它们的方案。
Apps-32 测试	使用本地的 32 位应用软件，无论大小。
NT 证明测试	在定期地构造以试图达到可用质量层次的过程中发现的错误。
小应用软件测试	对不同的系统小应用软件的测试，它们是很小的应用软件，特别是控制面板小应用软件片。
非 NT 测试 NT 测试	组外的测试，但仍局限于 NT 项目内部。包括开发员单位测试，文档编写员整理内容，开发员发现代码被分割，以及程序经理确保产品被正确包装。
“臭虫聚会”测试	通过举行发现错误的集会而进行的测试。
SGA 测试	用 SGA（合成图形用户界面应用软件）工具发现的错误。SGA 自动记录 16 位应用软件与操作系统之间的函数调用并转化它们，以使 Win32 位操作系统函数的执行可以用来代替 Win16 位程序的执行。
RATS 测试	用 RATS 工具发现的错误。RATS 是一种自动的测试‘机器”（工具），它运行 API 测试并把结果存储在一个集中的地方。
非特定技术	在错误报告表中未明确指出所使用的发现错误的测试技术的部分。

表 5.3 Windows NT3.0 中测试技术的效果

测试技术	发现错误的百分比
用途测试	15.0%
API 测试	12.8
专门的其他测试	8.0
Apps-16 测试	7.6
大猩猩测试	6.8
用户界面测试	5.5
压力测试	3.8
Apps-32 测试	2.8
NT 证明测试	1.7
小应用软件测试	0.7
非 NT 测试	0.6
“臭虫聚会”测试	0.3
SGA 测试	0.3
RATS 测试	0.2
非特定技术	33.9
合计	100.0

注：此表为 1992 年 12 月 19 日 Windows NT 项目中各种测试技术发现错误之百分比。β—1 版本于 1992 年 10 月公布。

资料来源：大卫·安德森等，《Windows NT 测试 PDC 及 β—1 评论》，1993，第 17—18 页。

原则五：使用度量数据来决定重大的阶段成果和产品的发布

与如何评估测试技术的效果相类似，微软项目通常使用被称为度量数据的经验测量和统计数据来理解、追踪和显示项目的进展情况。例如，程序经理和开发员必须决定是否要转向下一个里程碑，或按原定的发布日期还是延期发布一件产品。为了作出这些决策，他们不仅依赖于累积的关于错误（Bug）的趋势和它们的严重性的数据，还依赖于特别的度量数据。他们还依赖于用以解释这些数据的个人经验和判断。错误数据引导着里程碑的估计、完成、以及最终出品的决策，测试经理还必须同意开发员、程序经理和其他人所做的决策。依赖于度量和数据可以防止一个人——比如营销经理、开发组长、甚至是产品单位经理——在产品准备好之前就推出它。目的是为了避免或尽量减少那些曾在过去发生过的灾难，那时产品存在致命的缺陷，以至于微软不得不收回它们。

使用错误数据来评估里程碑的完成

微软经理使用错误数据，一般被称为“臭虫数据”，做基础来判断一件产品何时达到完成一个主要里程碑所要求的质量标准。任何时候，当一个可执行程序没有达到说明书中提出的标准时，一只“臭虫”就出现了。错误的出现是软件开发过程的一个自然部分，开发员只是在发布产品的最终版本之前尽可能多地修改它们。错误数据与产品的整体质量之间有一种正向联系，也与开发和测试过程中产品质量提高的程度和速度正相关。项目人员把错误

分为已找到（称为“打开的”）、已改正（称为“修改的”）、已解决（表示推迟的、重复的等等）、以及活跃的（表示错误已被找到，但还没有被修改或解决）。开发员修改测试员报告的错误，并遵循每日构造过程来把他们的改变综合进源代码的主版本。每日构造的使用非常鼓励开发员每天完成新的错误修改。

图 5.1 展示了关于错误情况的一张典型的柱状图。错误数据反映了 Excel5.0（也包括 Graph 5.0）在第二里程碑的综合阶段所打开的、修改的、解决的以及活跃的错误的情况。图中显示了项目在它们接近一个里程碑的结束时会持续地测试产品、找到错误和修改错误。有斜线的柱反映新找到的错误，实心的柱反映已修改的错误。经理人员和开发员密切注意活跃错误数目的稳定减少，以之作为进展的标志。有实心小格的连线显示了活跃错误的数量的下降趋势，这一数量即使在所进行的测试仍在持续找到新错误时也是减少的（在图 5.1 中，活跃错误的数量并不是一个简单的总和，因为它还可能包括或排除从其他来源获得的信息）。为了评估里程碑的实际完成情况，可能引起系统中止或破坏数据的活跃的错误的数量应该在测试仍在继续时就降为 0（或非常接近 0）。

专职发现和修改错误

开发周期中在代码完成里程碑之后和所计划的出品日期之前的这一段，对于项目小组而言是一段紧张的时间。他们试图发现和改正尽可能多的缺陷，然后还要按计划的日期发布产品。爱德·弗莱斯回忆了 Word 6.0 的最后阶段：

> 为了完成 Word6.0，我们还有一大堆错误要修改。这永远是一段紧张的时间。你努力想在出品前减少错误的数量……你有两个坚定不移的目标。首先，你有一大堆错误放在那儿，它们必须被修改——而你在这时除了接受现实努力修改外，已别无他法。然后，你还有一个希望实现的发布日期。你希望那两件事能一起做好，因为你所能做的全部就是：尽你所能，迅速地修改错误并希望进展顺利……但它会好的，我们等它好了才发布它。我们很有希望能在想发布它的时候就发布它。

在项目过程中，测试员持续地向开发员报告他们所发现的错误。在代码完成里程碑之后，开发员专事修改错误的工作。有些开发员可能会修改掉相当大一部分错误，因为他们更有经验，或因为错误影响着他们个人所开发的代码，或只是因为他们是优秀的错误修改员。比如 Excel4.0，37 个开发员中的 8 个修改了全部错误的 50%，而其中一个开发员修改了几乎 10%的错误。[20]（但是并非所有的开发员在项目上都是专职的）。 WindowsNT3.0 的开发员在 1993 年春天专职修改错误，而每处错误的修改平均需要改变三至五个文件。许多错误源于与 Windows 3.1 以及已有应用软件的兼容性问题。娄·帕雷罗里作出如下描述：

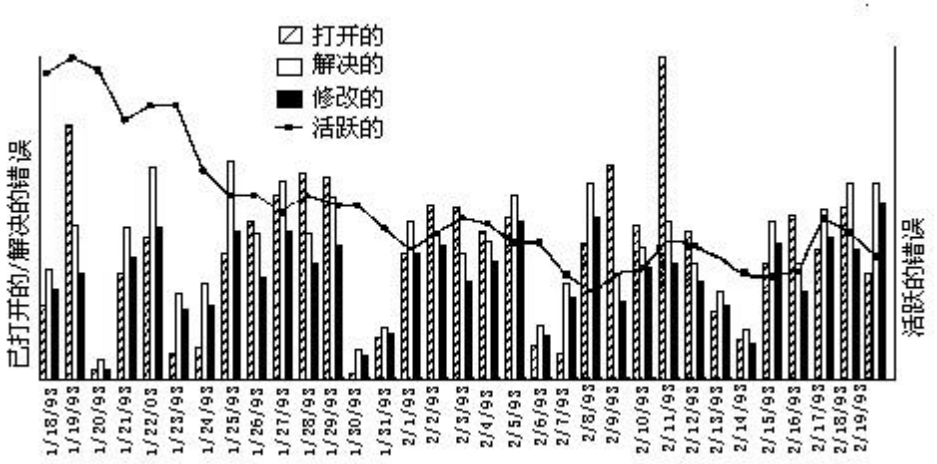
> 我们在好的情况下一周可以修改 1200 个错误……你知道，修改一处错误平均需要改变三到五个文件，但我们还有一些大个的……你得记录对 16 个、20 个也许 30 个文件的小改变……现在坏消息来了。大约 1000

处错误被打开了。问题是，怎么可能有人开发出有如此多错误的软件？这被证明是所谓的“兼容性”的问题。如果我们不是必须要与 Windows 3.1 兼容，我们不会有这么多错误。所以如果你的打印结果看起来不完全像 Windows，那么就有一处错误，因为不管怎样， Windows 总是正确的。如果这个 WoW16 位应用软件——我们称之为 WoW，因为它是 Windows 上的 Windows——不能运行，那么就有一个错误。

特性和产品完成的基于度量的检查清单

微软项目还编译基于度量的检查清单以帮助确定特性和产品的完成。比如，Excel 有一份长达 6 页的关于评判标准的清单，名为“我做好了吗？”，它把完成的标准分为 26 项内容，如菜单命令、打印、与操作系统的互联，以及应用软件的互操作性（见表 5.4）[21]。一个程序经理或一个开发员，或一个测试员，利用这份检查清单来帮助评价一项特性是否完成了。另一份基于度量的检查清单帮助确定一件产品是否已准备好发布给顾客。表 5.5 概括了 Excel5.0 的准备出品的标准的检查清单。这份检查清单列举了诸如不同的测试类型都必须 100%地完成，以及最优先考虑的错误必须被改正和至少再测试两次（这被称为回归测试）。还有，发现错误的速度和

图 5.1 Excel/Graph 5.0 第二里程碑的错误数据和每日构造



资料原来：微软内部文件。

表 5.4“我做完了吗？”检查清单的范例部分

与操作系统的互联：

- 激活/不能激活 Excel（用键盘和鼠标）？
- 非标准的控制面板窗口的色彩？
- 当应用软件运行时是否出现色带？
- 系统的 6 个单探测器和复探测器？
- 操作系统文件名长度的约束？
- Macintosh 的桌面附件？
- 无色彩的 QuickDraw？
- Macintosh 上的双显示器？

· Excel 在本地化的操作系统上是否工作？

资料来源：WesC：《我做完了吗？》，微软内部文件。错误的严重性都应该降低，而且在发布前最后一周的支持测试阶段不应再发现必须修改的错误。[22]

除了检查清单之外，项目有时使用特制的完成标准来评估产品的可出品性。比如在 Windows95 的例子中，产品在微软认为它已准备好之前必须正确地使一部分公司用户升级。Windows 95 的即插即用的特性被认为可以自动识别用户安装在计算机上的硬件的类型。操作系统必须能够在成千上万的机器上成功地发现差异很大的硬件特性。正如乔纳森·曼海姆所解释的：“芝加哥（Windows 95）发展了硬件发现特性的功能。这是我们现在所侧重的一个重要领域，所以我们花了很大力量着重于设置测试和硬件发现测试。除了在这里明确测试设置之外，我们现在所做的事情之一是我们走到外面的点去，把那些自愿加入 Windows 95 的公司点升级。我们的目标是在出品之前把一万家公司升级。”

系统产品有一个通用的决定发布的公式，基于产品可保持不变的时间长度。首先，一件系统产品必须达到它的不同功能、质量、速度和规模标准。然后，项目经理估计测试工作的彻底性（特别是β测试），已知错误的数目和严重性，以及当前版本保持不变的时间长度。一般而言，在出品一件产品之前项目应该有两个星期的时间保持人们专职测试而找不到新的严重的错误，并且无需对源代码作任何改变。比尔·盖茨对他们在 Windows NT 3.0 项目中所考虑的复杂的问题结构评论道：

> 保罗·马里茨和我讨论了：他该在零错误状态上呆多久，什么是关键的标准。当然，我们花费了大量的时间来讨论什么特性是必要的——OS/2 该走多远， Windows 兼容性测试又该走多远？速度该如何？它比 Windows 3.1 慢，是不是慢得太多了？在做这些事上它与 OS/2 相比如何？所以我们花了大量时间来思考速度标准和规模标准。要求 8 兆内存行不行？早期的时候我们认为 8 兆或 12 兆内存就可以了，最后我们觉得应该多到 16 兆。这行不行？我们不断讨论，阅读了大量数据以作出决定。
>
> 当进入到发布阶段……没有人会在晚上打电话到我的家中问：“我们可以推出这东西吗？”每个人都知道应该把所有的错误赶出去，然后等待也许一周或两周时间——这取决于项目的复杂程度如何，这期间你不动代码，也没有什么进展。你只要确保内部测试和β测试非常广泛，然后你就可以继续往前去出品它了。所以这期间没有人与我联系。老实说，出品一件产品并没有那么大的压力。我们是一家在银行里有很多钱的公司。我们可以用一种非常长期的方式做事。所以出品的压力何在呢？如果有人认为我们不该出品，往往是我们本身就不打算出品这件东西了。

错误数据和出品日期的经验和法则

在产品准备好发布以前，程序经理、开发员和测试员专职测试代码，发现和估计错误，并且改正错误。虽然这些人都测试代码，但只有开发员改变代码，而且开发员几乎每天记录他们的代码改变。

表 5.5　Excel5.0 为确定可出品性的基于度量的检查清单

已完成的测试

- 自动进行的自动测试运行无错误
- 手动的测试案例已运行
- 在主电子表格（刻画了产品特性）里定义的每个测试区域都标注“已完成”
- 初级和二级测试员对每一区域进行的专门测试都完成了
- 所有的错误都被回归测试和结束
- 最后 200 处优先级为 1 和 2 的错误被再次回归测试
- 在发布至生产（RTM）的出品日期前一个月内，设置和所有构件（除了 Excel．exe）都被冻结（不得改变）
- 一直很流行的“主观感受”调查表明测试组觉得我们已做好了出品准备

错误发现/修理数据：

- 错误发现速度在零错误发布（ZBR）之前呈现出下降趋势并在我们的 ZBR 之后保持这一趋势
- 错误严重性的分布情况变成有较多严重性等级为 3 和 4 的错误，所报告的严重性等级为 1 和 2 的错误应持续地减少
- 在第一次发布候选（RC1）之后所报告的所有错误都通过一个“也许会议”（在这里每一处被报告的错误都要被划分为“是”、“不是”或“也许”，用来表示它是要在当前发布中被改正还是推迟到下一次发布中），而且在错误被解决后，错误报告中应加入详细的解释以帮助进行错误回归测试
- 在发布至生产（RTM）日期前一星期的支持测试中不再有“必须修改”的错误被报告

资料来源：Excel5.0 测试计划，1993 年 4 月 13 日，以及《微软的测试》，微软内部报告。

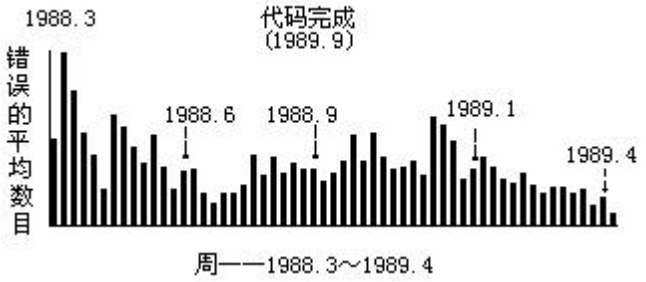
每日构造过程（见表 5.1）是保持产品稳定和稳步提高的支柱。如果没有这样一个系统性的过程，在这一阶段，仅因改变代码的开发员的人数如此之多就会引起混乱。当开发员与测试员完全同步并改正了所有已知的错误（或至少是所有已知的严重错误），微软人就把这一项目状态称作“接触零错误”。一个应用软件项目初次接触零错误应当是在出品产品之前大约六个星期。克里斯．彼得斯概括了他用来确定产品可出品时间的经验法则：

> 我们对确定代码完成时间并把它作为预料我们的出品日期的标准非常感兴趣，我们不仅仅说：“你不能改变什么了”……过去我们习惯于说出品日期应该是在我们完成代码后两个月，但从过去的经验中我们认识到它实际是在四到五个月之间，所以在现在的日程计划里我们通常是安排四个半月。而这是保证我们可以按时出品的一件事……另一件事是你在出品前大约六个星期初次接触零错误……标准情况是，当你第一次

绝对肯定你可以去生产了，你距离真正去生产大约还有两星期的时间。

可出品性的经验标准包括发现新错误的速度。当项目接近目标出品日期时，这一速度应趋近于零。图 5.2 是 Macintosh 的 Word 4.0 的图例，它显示了项目过程中每个测试员每星期发现错误的平均速度。发现错误的高峰时期主要集中在项目的早期以及发送前大约 5 个月。[23] 即使这些速度差别很大，经理人员仍可以利用它们来判定测试来源要求。比如在 Money 2.0 项目，每个测试员每天平均发现错误的速度为 1.9 处。[24] 项目有 10 个测试员，而测试员人数在项目过程中随需要而增减。单个测试员测试产品所需时间从 21 天到 203 天不等。在为 Windows 开发的 Works 2.0 项目中，错误平均发现速度是每个测试员每天 1.4 处。[25]

图 5.2Mac Word 4.0 错误发现速度



资料来尖：菲尔·弗西特：《Mac Word 4.0 的事后分析报告》，1989 年 5 月 25 日。

无情的市场

吸取了过去一些项目过早出品它们的产品的教训之后，微软各组学会了利用错误数据作依据判断产品是否作好了卖给顾客的准备。当然，还有许多有关市场的考虑会影响出品决策。但是最终，微软经理必须从产品更新或收回的机会的角度来权衡其决策。

如果产品有一处致命的错误破坏了用户的数据或阻碍了一些重要特性的工作，微软就必须发出一份产品更新版来纠正这一问题。这种产品更新版很少见，并且在某些方面成本很高。它们引起另一种产品周期的受阻，从下一个主要产品版本中抽取资源，并且重复整个磁盘生产和销售过程。致命的错误以及由之引起的更新还会损坏公司的信誉并可能导致法律诉讼。用户有时为收到更新版本而支付一小笔费用，但是像微软这样的公司常常按成本价发行，这取决于问题的严重性。就像比尔·盖茨所评论的：“我们处于一种独一无二的处境。当我们送一些东西去生产，我们只是构造大量的拷贝。DOS6 被发布去生产，然后他们构造了两百万份拷贝。我们把这一产品以大约 45 美元的价格卖进销售渠道。如果我们必须把它更新，那么所有利润会‘呼’地一声烟消云散。如果不得不经历一次产品收回，那是很荒唐的。”

主要是由于来自顾客和竞争者的压力，质量对于盖茨和其他经理而言越来越重要。然而微软人仍倾向于从一种相对的和实用主义的角度而不是绝对的角度来看待质量。例如戴夫·穆尔坚持质量必须包括对市场的重视和灵活性：“我们的观点是对质量的追求必须让顾客知道。它必须有效用。有时质量要求更多地考虑灵活性、对不断改变的顾客需要及市场条件的适应性。所以你的质量标准不能与之相违背。”

同时，如同 Intel 公司在奔腾芯片方面的经历一样，PC 顾客不愿再容忍产品的缺陷，这是很危险的。乔纳森·曼海姆在解释微软为什么延迟了 Windows 95 的发布以完善特性和更透彻地进行测试时承认了这一点：“ 当它还有错误的时候我们不会发布它。我们感觉市场会原谅我们的迟到，但不会原谅我们的错误。 ”盖茨提到了另一点关于微软信誉的更深考虑。他看来更愿意推迟像 Windows NT 或 Windows 95 这样的重要新产品的出品，如果它们在预定的出品日期还没有准备好：“ 你永远不会希望向市场发布一件坏产品。我们的信誉是一项巨大的资产。所以，我介入关于产品是否足够激动人心或好得可以发布的最终决策是很重要的。我们有强大的销售力量，他们认为我们给他们的所有东西都很棒……所以我们必须确保不要给他们臭弹。 ”

第 6 章讨论微软如何建立一个组织来向它自己和向它的顾客学习。人们不仅思考过去的错误，还学会了如何把顾客的信息反馈直接联入开发过程以及在不同的产品组之间共享更多的构件。

第六章

创建学习组织

通过自我批评、信息反馈和交流而力求进步

为了建立学习组织，微软制定了自己的战略，我们称之为“通过不断的自我批评、信息反馈和交流而力求进步”。对此，我们分解为四个原则予以讨论：

- 系统地从过去和当前的研究项目与产品中学习。
- 鼓励使用数量化的测量标准和衡量基准进行信息反馈和改进。
- 视客户的支持为产品的一部分和进步依据。
- 促进各产品组之间的联系，实现资源共享。

在近年来的管理文献中，有关组织学习的内容频频出现，已成为一个很普遍的问题。我们对这一概念的阐述则选用很实用的术语。其实无论做什么，组织自我改进的机会很多：他们可以从经营中进行反思，研究产品，倾听顾客意见；也可以鼓励组织的不同部分像分享各自努力的成果一样实现知识的共享，比如在产品和构件的设计上。所有企业都有这些机会，只是很少有公司能充分利用它们的优势罢了。当一个公司中的个人和各组都试图抓住这些机会，取得了可衡量的进步时，我们便认为该公司“创建了学习组织”。

至于学习的原则，微软与其他成功企业并无不同。无论何时，好企业都会对生产过程和产品吹毛求疵，以求从过去的成功中获得经验、从失败中吸取教训。他们会对自身行为制定标准、进行衡量，再广泛地研究客户；并且，他们试图促成不同组织部门间更普遍地相互合作，共享构件，并交流产品与生产过程的知识。然而，就个人计算机软件公司而言，我们感觉微软尤其强调这些原则，与众不同。之所以这么说，是因为我们相信该公司正孜孜不倦地（虽说仍不够）努力着，要建成一个更具凝聚力的组织。

例如，并非有很多软件开发组织制度性地进行项目后期分析与审计，虽然这已得到软件专家们的首肯，被视之为有益措施。而微软的各组却在事后检查与项目审计两方面都做到了制度性管理。和众多软件制造商及服务业企业一样，微软在充分衡量产品开发过程的各要素之后，竭力在进行更有效的管理和避免过度官僚化之间寻求一种平衡。当然，这种衡量与那些中央处理机和微机世界中的一流软件制造商并不完全相同，比如 IBM、惠普或日本的计算机制造商。不管怎样，微软正在培养一种习惯，即对其发展过程中的关键因素进行数量化测量。而经理们正是使用这些资料做出关键决策的——例如，何时推进某一个研究项目，或何时发布一个产品。他们还在创造一些用来评判绩效的标准，从而推动进步。

与世界上任何其他软件公司相比，微软可能有更多的客户接受过电话采访或收到电子调查表。究其原因，当然也可归结为它拥有最庞大的用户网——如果没有其他理由的话。不过我们觉得微软公司比任何其他公司更善于通过彻底地分析与客户的联系而获益，并且所得信息能迅速地直接传输到各产品开发组。自 1990 年以来，产品组还日益重视自己设计的构件能在其他各组间通用。这种共享制在影响着公司文化和发展战略的同时，也使微软的公司组织发生了巨大变化（如办公产品一体化的建立）。

过去，微软的产品组能集中全力关注一个产品，一个竞争对手和一个目标。比如，他们集中力量推出了 Windows 或 WindowsNT，或者在电子表格方面赶超 Lotus，在文字处理上与 Word—Perfect 较量。但是，即使像微软这样资金力量雄厚的公司也必须对各种产品进行标准化，使其具有通用性，以便减少开发、测试和获得客户支持的成本。由于产品竞争的客观现实，微软也承受不了因重复设计错误、漠视客户抱怨与建议或纵容项目进程失控所付出的代价。作为一个组织，“学习”如何使公司不再是缺乏控制与合作的各独立项目小组的集合体可能已成为微软必须进行的最困难的转变。

我们认为，微软比起那些全球大企业来，更易于从学习中获益。如其他章节所述，微软几乎把它所有的产品开发组都设在里士满和华盛顿总部。除了通过电子邮件建立的广泛联系和信任外，盖茨和其他经理人员还坚持主张人们保持密切接触，以便面对面地解决问题。就我们的观察，微软公司内进行的大量学习活动都是非正式的——包括在走廊里常见到的谈话以及在个人办公室里、午餐桌上、休假会期间甚至更偶然场合中的各种交流。

原则一：系统地从过去和当前的研究项目与产品中学习

如何对单个的软件开发者和项目作绩效评估历来是个令人头痛的问题。这些任务颇像畅销书的写作（生产），极富创造性。人们很难给他们的生产效率或质量制定一套客观的衡量标准。而微软公司聘用的都是些天赋颇高的独立编程人员，想给他们提出建设性的信息反馈和批评意见自然更困难了。正如在公司的一些事后分析报告和内部备忘录中所表现的那样，微软职员是愿意进行自我批评的。但在 70 至 80 年代间，各组的保守倾向日益显著。麦克·梅普尔斯认识到了公司需要高瞻远瞩——各组应就其所作所为与微软内的其他组或其他公司进行比较,或许别人在软件开发管理上更有高招呢。1988 年，梅普尔斯把在软件开发中鼓励深入学习和资源共享这项工作交给了戴夫·穆尔。

穆尔 1954 年出生，曾就读华盛顿大学的数学专业。1976 年毕业后进了波音公司，最初在 CAD/CAM 系统工作，还制定过工程技术标准，因此，对于一个管理出色的工程技术公司应该是什么样子他颇有体会。1981 年，穆尔作为一个开发员加入了微软，当时公司大约有 100 名雇员，几乎还未曾建起什么运作规则。他工作勤勉，不光是 Multiplan、Word、Chart、Works 和其他一些产品开发中的楷模，也因成功地推出了这些产品而享有盛名。1988 年，梅普尔斯请穆尔做了应用软件部门的开发主管。1992 年，又晋升为全公司的开发经理，并于 1992 年至 1994 年间担任测试与质量担保方面的代理经理，同时还身兼数种临时职位。从 1994 年起，他就向产品开发部主管克里斯·威廉斯汇报，后者再向保罗·马里兹报告。

作为开发部主管，穆尔把注意力集中于几大领域。他一直鼓励各组写事后分析报告，至少能就项目进程开会讨论。他还实施了过程审计以帮助各组分析和解决问题；又组织起正式的休假会活动，届时有关重要人士会就软件开发与质量控制的相关问题相互切磋；再就是在相同职务的人员间极力搓合一些非正式会谈以鼓励知识共享。这些努力弥补了微软公司另一惯例的缺憾，即所谓“自食其果”——内部率先使用微软的产品和工具，以直接体验它们对顾客的好（或糟糕）的程度。穆尔还开始设法制定公司各组的评估标

准，这个理想现已为全公司所强调和追求。

事后分析

自从 80 年代后期以来，有一半至 2/3 的微软项目已写了事后分析报告；其他项目大多也举行过事后分析讨论会。事后分析的文件在自我批评时的坦率令人惊异，原因恰恰在于公司的最高层会传阅它们。甚至比尔·盖茨也饶有兴趣地阅读了有关主要项目（如 Word 和 Excel）及一些新领域（如多媒体）的事后分析报告。盖茨承认：“人们很自然地会对进展得一团糟的事情表示不满……我们在编写软件方面很成功，所以对那些没能做好的部分就可能过于苛刻”。克里斯·彼得斯也同意，并认为：“这些文件的目的就是击败自我”。梅普尔斯则视事后分析为微软人喜欢自我分析的一个自然结果：“在某种程度上，那是我们文化的一部分。我们总是拧紧了发条，我们担心竞争失利，我们习惯于思前想后……其表现情形之一为我们特别喜欢自我批评，而事后分析以及一系列此类方式都在试图做到这一点，这都是对所作所为不能尽善尽美的永不妥协。我们总是奋斗不息。”

各组一般 3 到 6 个月汇总一次事后分析文件。文件从 10 页以下到 100 页以上不等，且有篇幅增加的趋势。比如，Word 3.0 的事后分析报告从 1987 年 8 月起只有 12 页；Excel4.0 的事后分析报告从 1992 年 2、3 月起，有 41 页；始于 93 年 6 月的 Encarta 的事后分析报告有 84 页；而 Windows NT 的β鉴定竟长达 112 页！

由于微软的各项目已不仅仅是可简单描述的开发、测试活动，说明文件也随之增加了。其中添加了详细的针对每种功能的章节，一般由各组负责程序管理、开发、测试、产品管理和用户培训的主要人员分别执笔。最常见的格式是对最近项目中哪些做得好，哪些做得不好以及本组在下一个项目中应如何改进展开讨论。事后分析报告还包括了描述性资料，涉及成员（按职能分的小组规模、工作天数），产品（编码规模、与前一版本的比较、所用语言、生产出的平台和语言版本），质量（每 1000 行编码的错误数、在四个等级中的错误程度、按产品领域分的错误类型、发现和修正错误的记录、由上一项目遗留下来的以及新产生的错误数目），进度表（实际的以及计划的阶段和出品日期）以及过程（比如所用工具、涉及到与其他组及提供产品附件的外部供应商的合作问题）。通常，职能经理们会准备一份草拟文件，经 E－mail（电子邮件）在组员间传看，各自添加意见，再由作者综合检查以最后定稿。之后，文件便发送到组员、高级管理员和产品开发部、开发部及测试部的各位主管处。接着，职能组，有时整个项目组会开会讨论事后分析的研究结果。有些组（如 Excel 和 Windows NT）还养成习惯，在每个阶段结束后召开分析会议，做出中期的工作修正，检查特性表并重新调整进度表。

事后分析文件可提供的是一个粗线条的记录，告诉人们在开发 Word 等产品时微软是如何在对困难的应战中取得进展的。Word 组从 1987 年起在复发性问题文档化上独树一帜，而 Excel 组则从 1989 和 1990 年起，在为相关大项目寻求解决方案方面堪称楷模。自此，微软的经理们开始把 Excel 等组或某些项目的技巧或经验教训在全公司内广为宣传。他们所依赖的一系列机制有：散布主要备忘录和事后分析报告，创作有关进步安排及项目管理方法的书面文件（这未能受到欢迎），发行克里斯·彼得斯 1990 年关于 Excel 组做法的谈话录像（这被证明颇受欢迎），戴夫·穆尔所推广的非正式审计，休

假会上讨论质量和进度问题以及按彼得斯的录像所授的经验，人们总结出的软件按时出品的一周程序。下列的这些综述在事后分析报告中均已载明，从中可知微软从各种教训中曾体尝的痛苦滋味。

在 80 年代中期，微软公司推行过一种 Word 小组称之为“广度优先”的方法。先由程序管理员写出一个简要的思路陈述和功能具体化提纲，然后开发员在把各种新特性编入新的产品版本时做首次削减——直到各项工作均已告终，人们才会把精力投入测试和各特性的集成上。这一方法显然在很大程度上依赖于项目结束前的大量集成工作。

“广度优先”法对那些各种特性相互独立的小项目很有效。但是，当微软将之“升级”，用于 Word、Excel 以及 Windows 和 Omega （Access）数据库这些大型的复杂产品时却未见效。到 80 年代后期，微软产品不仅在各特性间相互影响方面有缺陷，各特性本身也是问题重重。（一个相互影响方面的例子是，当有人使用制表功能时，打印功能却来凑热闹，将它打印出来。）开发员总是不得不返工，又研制大量代码去修改错误，以使各功能运行无误；于是，返工会完全撇开了进度表。在有些情况下，发布可靠产品还会成为难题，或根本就不可能，因为修改程序的使用可能招致更多的错误，直至出现所谓“无穷错误”的状态，结果项目虽已近尾声，有的错误仍得不到修改。

查尔斯·西蒙尼在 1987 年 8 月 13 日的会议基础上写出了一份 Macintosh Word3.0 的事后分析备忘录。这个项目之所以重要，全因为该组的种种不凡遭遇。 Windows 版本的产品始于 1984 年，预计进度为一年。虽然开发组试图重复交叉使用 Macintosh 和 PCWindows 平台，并力求适应 Windows 程序的变化，可还是不可避免地延期了。Windows 版最终是 1989 年 11 月推出的——迟于原进度 4 年之久。微软在 1987 年 4 月确实推出过一个 Macintosh 版，却错误百出以致公司不得不回收产品，再向 7000 用户提供了免费替代品。[1] 尽管组员们经历了这些挫折，人们仍能看到他们在反思整个过程。Word 小组在事后分析中建议项目组从“广度优先”法中摆脱出来。看来，更明智的做法是把产品特性分成不同的阶段，从“深度优先”着手，即把一小组人员配置到某一组选定的特性上，进行更加深入完整的构建，并在开始其他特性之前彻底予以检验。西蒙尼对此做了如下解释：

> 我们在 1987 年 8 月 13 日召开了一个 Word 3.0 的事后分析总结会……Word 3.0 的问题是由各种情况共同引起的，包括缺乏早期的深入测试（这在很大程度上又是由“广度优先”法导致的），不能完全理解产品的复杂性，也未能充分变动 1.05 数据库以驾驭新的复杂情况。我们打算避免上述问题再度滋生，方法是从更深入细致的开发起步，组织——如果可能的话——“攻坚战”，每 2、3 或 4 个人一起在同一特性的不同细节上开展工作。到战役快结束时将详加测试，这次工作由开发员们进行，未加入战役本身的程序员们也参与其中……至于如何改变编程组织和编码操作以减少错误数目，我们也有了些主意，多数情况下，这些想法同样适用于其他项目。[2]

Word 组还有过其他一些不妥举措，如在完整提交以前不检查开发员的代码。西蒙尼抱怨道：“先是由于盲目乐观，后是基于进度表的压力，Word 3.0 的许多新代码没接受检查。”[3] 而几乎所有的公司都发现：代码检查是尽早摒

弃错误的有力途径。当然，这份事后分析也揭示出微软的开发员们已在使用先进的抽象方法和基本是目标导向式的设计技术，这使代码有更广的适用性，并能支持大量的新的特性。微软开发过程和编码技术的其他许多优势在 Word 3.0 项目中也崭露头角，包括强调编码中对各种“主张”的插入说明，好让其他开发员、测试员知道原开发员在此做了怎样的假定；使用自动化测试发现错误；引进“工具版”产品记录下用户的每一步操作，以便分析错误。

糟糕的是，Macintosh Word 4.0 项目并没有更佳业绩；小组成员在 1990 年推出产品之前又重犯了许多错误。下面这段评论节选自克里斯·梅森所撰的开发事后分析报告，暗示了问题的糟糕程度。麻烦源于小组盲目地想在 Macintosh 和 Windows 版之间搞代码共享，没有固定进度表，开发工具毛病多端，缺少特性设计的各方面合作，倾向于一次性修改纠缠不清的错误以及具体化不够充分。[4]

> Mac Word 4.0 的开发事后分析会议 5 月 10 日在美丽的贝尔维尤市中心召开……若说这个项目中有失败之举的话，那就是制表以及和 Win Word 的合并。其中当然也不乏幼稚之举和时间的浪费。我们天真地认为自己能在显示、PLC 和日常规则设计上做些基本变动，写出大量复杂的新代码且错误寥寥。Word 史上恐怕永远不能实现这一奇妙构想。我相信，和 Win Word 共享代码是在浪费他们和我们的时间，完完全全就是一个失败……最终，Mac Word 被树立为经典案例予以研究，告诫人们制定项目进度表时不能怎么做（第 1 页）。
>
> Mac Word 4.0 的进度安排总体看来是不充分的……我们的工作其实就没有什么进度表……真是太愚蠢了，事后看来，我们从未能按时出品实在不足为怪。我们对自己正在做什么其实并无主见（第 5 页）。
>
> 最大的问题莫过于我们被告之开发系统可以推出了。实际上，所有部分，尤其是汇编程序和解释程序里，错误云集（第 8 页）。
>
> 我们倾向于独立设计特性，从不考虑他们和程序其他部分的相互作用……错误，尤其是集中在某一领域的大量错误，能指明特性间的相互作用以及被忽视的设计问题。而我们通常没去领会 Word 4.0 所给的启示。我们始终一次性修改错误……我们也不能做到透彻理解表格与 Word 其他部分的相互作用（第 9 页）。
>
> 我们三次将 Win Word 代码合并入 Mac Word 中。第三次之后，两个项目已共享了产品的大部分代码。换个角度来看，Win Word 至少做了 6 次合并……每一次都严重破坏了他们的产品……当 Mac Word 正在彻底地重做基础设计，而 Win Word 却欲向前推进工作时，这些合并更成为远非一个方案可解决的难题（第 12 页）。
>
> 攻坚战未能解决问题，事实上也未如他们设想的能推动深度优先……说明存在缺陷，表格只用 3 页半就描述完了（第 15—16 页）。

不管怎样，Macintosh Word 4.0 项目仍是检查了较多代码，做了较好的测试尝试。测试的事后分析由菲尔·福赛特执笔，反映出该项目已引进了更为系统化的测试程序；并已超出了“专门测试”范围，演变为有着更多的计划和具体化分析的文件。福赛特还申请更多的自动化测试工具，呼吁进行更合理的项目进度安排以确保充足时间被用于测试。[5]

Mac Word3.0 发布之后，有一点已确信无疑，即仅仅使用专门测试，以此作为确保产品质量的手段实在漏洞百出。为了避免盲目的行动，也为了有助于测试员的培训，我们决定将部分测试时间用于创作特性说明和特性测试的案例文件。一旦这些完善了，测试的尝试就可组织为一系列完整的工作循环，并可为充分进行非系统化测试配置时间。……特性说明写在项目早期，其基础是出于对某些特性如何运作的初步设计考虑……我们发现最能发挥这个文件的效力的是把它作为对某种非系统型方法的补充（第 2 页）。

在所有影响测试效率的因素中，缺乏充足的工具最为突出。由于缺乏足够的自动化工具，我们只得求助于人工测试方法（第 6 页）。

不正确的项目进度安排和赶进度产生的压力会在整个测试过程中减少连贯性。不太可行的进度表迫使测试在 11 月（1988）就开始进入出品状态，并一直延续到 4 月中旬。这意味着测试中 40%的努力都花在了出品状态上了（6 个月）。在这令人精疲力竭的过程中，我们注意到了生产效率的损失惊人……而一个现实的进度表必是基于对代码变化、每 KLOC 错误数及代码的复杂性衡量的较准确的估计的,并且其他一些通用标准应在开始进入出品状态前多给出 3 个月时间，用于更系统、更彻底的测试（第 7 页）。

Excel3.0 和 4.0 项目完成于 1990 年至 1992 年间，利用了本书第 4、5 章所述的过程。 Excel3.0 被证明是高质量的，且只延期了 11 天。虽然克里斯·彼得斯在他 1990 年录像的谈话中详细阐述了所用过程，这个项目还是一直未写出事后分析文件。Excel4.0 的开发周期仅为 9 个月，比 3.0 版迟一个月便出品了 ;它的事后分析很具体地将两个项目组的经历作了比较。由于 4.0 项目略去了大量重要的具体化阶段，直接进入了 3.0 版中未涉及的添加特性，因此它不很典型。不管怎样，开发事后分析报告（由乔恩·德·沃恩执笔）还是可以反映出微软公司内部的奋斗历程。关键在于能维持这么一种平衡，既不在那些会变动或终被放弃的特性的具体化上浪费时间，又能写成充分详细的文件以便了解什么特性应优先考虑，并估计出给它们编码时的难度如何，进而合理安排进度。该项目小组还从其他组及销售商处引进了一些构件，故而导致了一场关于“ 相互依赖 ”问题的讨论，这一问题在以后的项目中还会越来越严重。很显然，微软已开始接受更高的项目管理与质量标准：虽说与老项目相比，新项目的实际进度与进度表的吻合好多了，为什么还是延期？虽说顾客的抱怨已减少到了最低限度，为什么还是有这么多错误？[6]

Excel4.0 的开发始于 1991 年 7 月 8 日，第一版（Windows， US）于 1992 年 3 月 27 日推出……Excel4.0 的开发过程总体上堪称成功。产品广为接受，来自 PSS 的报告也暗示我们没有质量问题。不过，项目的急促迫使我们砍掉了开发过程中的一些细节，其后果在我们看来是会引起一些进度上的损耗，某些内部和外部的设计决策不当以及较晚终会显形的错误（第 1 页）。

在进度安排上我们遇到的主要困难是估计失误。由于重要设计的不足导致了估计不够准确……虽然 Excel4.0 和 Excel3.0 的零缺陷规定

几乎一模一样，可在每个开发员的优先表上却未给予同等重视。人们总的感觉是 Excel4.0 错误太多。按照记录，零缺陷对 Excel 意味着：

· 所有变化必须予以收集，建立联系；

· 所有变化必须同时在 Macintosh 和 Windows 平台上通过“快速检验”；

· 任何开发员，一旦错误记录超过 10 个就必须在修

· 改之后才能转向新的特性（第 4—5 页）。

Excel4.0 程序管理的事后分析报告由迈克·马蒂厄执笔，描述了开发开始之前计划不足，对具体化重视不够等问题。马蒂厄写到想象性描述“是个很好的参考，凭此可把注意力集中在产品上”，但“它比我们真能做到的有更好的蕴意”。他把活动基础上的计划描写为“一种通过将特性合理化并给出重点从而能更好关注于设计目标的好方法”，但他声明“我们并不事先删减说明，所以在那些终被砍掉的特性上白花了很多时间……活动小组未能组织有序，成员变动频繁……工作中与开发员紧密配合以解决设计问题比说明不离手要更有成效。这会节约周转时间，减少繁文缛节”。另外，马蒂厄还指出需要较早地、更为方便地使用实验室测试，并用更好的方法控制说明的改变。[7] 测试部分事后分析的作者马克·奥尔森则抱怨了具体化过程（说明过时、没有说明和说明不完善是三大难题）及一大堆其他问题，诸如在测试员中平衡工作量，加快改错的周转率。[8]

多数项目都描述过相似问题。例如，新近的 Encarta 多媒体百科全书出品于 1993 年 3 月，项目为期 22 月，成员 10 来人。该项目的重要性在于它混合了正文、视、听形象的运用。其事后分析文件表明了该组在采用 Excel 开发方法方面很下了功夫，然而，项目并未做到善始善终地学习 Excel 的成功经验。它居然遇到了微软一些项目组在 80 年代曾有记载的同类困难。思考和计划不足导致了该组在特性的优先安排上不很得力。结果，项目直到最后阶段时还有一些特性未攻克下来，又不能删除，终于延误了进度。半途停止的特性（和早期的广度优先法一样）将被带到下一阶段，给实际进展抹上不大现实的色彩。过于乐观的进度安排迫使每个人都得拼命赶，从而将项目本应配置在代码检查、测试和改错上的时间压缩再压缩。不过，微软还是通过了它第一个主要的多媒体项目，并为将来的同类产品开发打下了经验基础。

虽然在 Encarta 组的努力下，该产品很有竞争力，令人引以自豪，但小组仍觉得我们能够并力争在下一次做得更好。工作的关键部分将是产品效果、进度安排和协调工作量。从管理的角度来看，因为存在着相当的进度拖沓，Encarta 项目算不上典范。事实上，要研究在多媒体项目的软件开发进度安排上什么不可做时，它倒是个例证……Encarta 的多数问题可归因于一点：在开发这类规模和复杂程度的产品时，我们还缺少对多媒体出版产品（MM-PVB）的经验。现在有了 Encarta 的经历，以后在项目中面对这些困难时，我们再不会不知所措了。[9]

过程审计

梅普尔斯交给开发主任穆尔的任务之一就是实施不定期的过程审计，尤其在项目进展遇到困难时。例如，1993 年，穆尔实施了 5 次审计，每次都集

中力量花一周时间。他经常查看那些能弄到手的项目和质量方面的资料，尽可能多地和项目成员谈话，并试图鉴定他们什么地方表现不错及哪里可能做得更好。总而言之，穆尔是想找出一个过程典范以促使小组的工作更为成功；并且，对于核心的项目人员他还会写成文稿或口头上予以详细信息反馈。他把自己的任务描述为建设性的：

> 麦克[梅尔普斯]已让我着手对小组做审计。一旦哪儿出现问题或他们需要帮忙，审计者总能应需而动。我在那里扮演的可是警察或打手。因此人还没到，他们自己就开始整顿起来：工作暂停，做点变动……这就是我要去做审计的地方。一般我会这么来个开场白：“ 我给你们帮忙来了。我会记录下最好的做法，广为传播。我来这儿就是要抓你们的最好典型。 ”然后，当他们叙述着自己最好的做法如何如何时，我会问有些什么困难。如果有的问题或做法我相信能在小组内改进，我就给出改进建议。我总把审计视为我到那里去帮助他们的建设性步骤……我几乎可以说自己现在做的工作是个专职审计员。

尽管审计是有压力的，穆尔更像一个福音传教士般工作着。他通过逻辑的力量和来自其他项目的资料传输，亲切引导小组朝着更好的做法迈进。我们已注意到了，对于来自管理层、官僚性规定或公司质量保障（QC）组的命令，微软内部存在着一种文化对抗，这使得告诉人们去做什么很困难。为适应这种微软文化，穆尔让人们把他的审计视为一种“ 技术交换 ”：

> 这大概要算是我的个人奋斗吧。这里没有一个层级间的管理命令，只是一种技术交换。我不会走入什么人的办公室对他说：“ 麦克已吩咐我了，要高举条理规范之旗，所以你们就遵守这些标准吧。 ”我会说：“ 大家工作进展如何？有没有试过这种方法？或许它在解决那个问题上能有效。 ”那才是讨论方式。这些小组会选择出最适合的解决问题之道。你们大概已注意到了，根本没有什么质量保证组来围着这些工作组转，我们没有那种传统的质量保证。有个组曾在穆尔的帮助下渡过危机，开发出了 VisualC + + 语言产品。麦克·梅普尔斯在一个程序检查例会上遇到了该项目组成员。辨明问题所在之后，他请穆尔去做审计。该组有三四十名开发员，同等数目的测试员，另加 10 个程序管理员，但采用的做法却是心血来潮式的。程序经理杰夫·贝勒曾这样描述小组在与穆尔打交道后的变化：“ 戴夫花了约两周时间采访产品组的主要成员……带回来了一系列建议……关于开发、测试及程序管理问题的大部分建议得到了支持。但有时我们也不得不强制执行：‘ 你们应该能做到这点。戴夫·穆尔已说过我们组的确不错，我们要在这个项目上当之无愧。 ’至于其他方面，建议都通情达理，我们只要照着做就行了，当然也有些从未执行。 ”

最重要的是，小组采用了每日构造法，即按照各个阶段和细分的任务来安排进度，在将代码纳入构造之前先予以检查。克里斯·威廉斯对这种审计结果大加喝彩：“ VisualC+ + 组……完全乱了套，C7[C++第 7 版]……是为期 12 个月的项目，可出品却推迟了 13 个月……他们打电话给戴夫说：‘ 戴夫，快来帮帮忙吧。 ’戴夫来了，告诉他们：‘ 好吧，你们要遵循这些方法，你们要做这些事情。 ’……于是这些家伙完全改变了办事的路子，结果准时、

无误地推出了产品。”

休假会

80 年代早期起，微软开始对其主要成员每年至少组织一次“休假会”。目的之一是促成不同部门就手头工作交换信息，再就是对付一些难题，诸如怎样才能在特别产品领域更有效地竞争（1983 年的一次“休假”活动曾瞄准了 Lotus，以对新的 Excel 产品的明确界定而结束了会议）以及如何提高开发产品和获得客户支持的技巧。在有些会上，组织者会要求与会者读有关软件工程及相关领域的经典文献（比如菲尔·克罗斯比强调质量，而汤姆·德马科关注项目进度），然后讨论这些作者的洞察力何在以及微软意欲着手进行哪些改进。领导人物会首先就所读发言，引导讨论。

最有名的且业已载入克里斯·梅森的“零缺陷”备忘录并将流传百世的“休假会”恐怕要算 1989 年 5 月的那次了，会晤集中于那些旨在减少错误，提高质量的技术和工具上。大约 30 人参加了这次休假活动，由戴夫·穆尔组织并视为其新任开发主管工作的一部分。穆尔把与会者分为四组，要求他们全力关注零缺陷的方法、工具和其他相关问题。他曾回忆起这次会议，谈到了麦克·梅普尔斯所起的作用：

> 我和杰夫（哈贝尔斯，后来成了穆尔的老板）谈到了我们需要如何做些变动，可我就是毫无进展。我正和组员们努力工作，力求改进。杰夫也在尝试一些事情。但是“我们需要了解全过程，我们需要零缺陷推进这类观念”之类的内聚力还没形成。到 1988 年 9 月份麦克·梅普尔斯走马上任之时，终于开始了旨在真正理解全过程的根本性变革。那时，我和麦克一起工作，规定了这次“休假会”的组织中心是着手实行零缺陷。我考虑这个问题已有些时候了；我敢说其他人也都在这个问题上想了许久。可我还是把大家召集到一起来。事实上，查尔斯·西蒙尼是零缺陷小组当中的一员，不过，选他领导这个特殊小组纯属巧合。所幸的是，那种安排，那种组织系统化以及对各种问题的见识使我们能抓住……始终需要向应用软件开发小组揭示的各种观念。

微软坚持每年就不同问题召集“休假会”。1993 年至 1995 年，会议集中的焦点是迎接挑战，以更为集成的实体运营；还有对联机网络和多媒体这类新技术备战。举个例子，1993 年 6 月的“休假会”就讨论了不同产品、不同的业务单位之间相互配合的改进问题。而1994年2月的一次则讨论的是‘互相依赖”的管理问题：即当项目依赖的关键构件是由其他项目开发的或从公司外部引进的时，如何处理日益严重的进度安排和质量保证问题。克里斯·威廉斯回忆起这次来自公司各部分的开发经理和测试经理参加的历时三天的“休假会”：“麦克[梅普尔斯] 和戴夫[穆尔]是实际上的主办人，他们一起在合计这事。我想议题的形成原因在于大家相当广泛地关注着日益实出的合成化、相互依赖问题，并且对相互依赖和进度的有效管理的能力欠缺问题也有所考虑。所以，有的组被派去察看管理依存度，去巡视‘我们是否缺少一些重要工具，怎样着手减少那些测试时暴露无遗的错误’这类事情。”

目前存在几个关键的相互依赖问题。例如，Visual Basic forApplications（在语言领域中开发）和 OLE（在商业系统部门开发）是

Excel 和其他使用这些技术的应用软件中的关键部分。Offfice （办公室）产品系列必须在开发 Excel、Word、PowerPoint、Mail 和 Access 时相互配合，这就需要将其一起推出。威廉斯解释说 Office 现在问题少多了，因为它全部由克里斯指导：“ 为什么 Office，比如说，工作效率很高，原因之一就在于它们都在同一个组织之中。只有当牵扯到不同部门时，事情才会变得问题成堆。 ”1994 年 2 月的“ 休假会 ”上提出了一个很特别的建议，就是在两个相互依赖的组间创造出一个合约样本。这也可作为一种最佳方案清单，用于相互依赖的小组间的协调管理，并详细说明在开发和测试阶段的责任划分和截止日期。

小组间的资源分享

和许多大公司一样，微软目前也是部门林立，以至于某一领域的人不一定知道另一领域的人正在干什么。“ 休假会 ”有助于发布信息，但这种信息交换并不频繁，层次也太高。于是，戴夫·穆尔这些经理人员鼓励中级经理和其他人员在不太正式的场合更经常地交流各自所知。相同职能部门内经理们的午餐会已固化为一个主要的机制。Excel 组的测试经理马克就解释了他如何定期与其他各组的同僚们交流，这甚至比 Office 产品单位的形成还早些：

> 在测试水平上，我和 Word 的测试经理经常交谈，我们做了大量努力去创造同时适应于 Word 和 Excel 的特性。我们让各组间工作在相近特性上的测试员紧密联系，共享各种主意和信息。我们反复地分享着工具……我们每月还有一次两小时的聚餐会——到会的包括 Word、Excel 和 Project 的测试经理们。我们谈论“ 正面临着什么问题 ”，“ 如何予以解决 ”，“ 我正在考虑这个问题 ”，“ 你这家伙在忙什么呢 ”……我们每月还和公司内所有的测试经理会谈，甚至是全世界范围内的产品组……我们做个介绍，然后大家分享各组的业绩。

克里斯·威廉斯是另一位力倡和维护这些经常的非正式会谈的健将。负责用户培训和测试的经理们似乎对此最为热衷。然而在穆尔组织的大部分会议上，开发经理们都没露面，这使穆尔和威廉斯受到了挫折：

> 如何让大家更多地进行交流是我们碰到的有趣问题之一。用户培训主任举行了所谓的“ 博览会议 ”，每周聚一次，每次谈上一小时。每组就各自产品及其走势发表意见，使大家都清清楚楚。然后话题转入多种领域，他们谈 HR 问题，技术问题，有时也引荐一些外部人员做发言……测试员们每月开一次会，到时测试经理们济济一堂，提供有长远价值的议题加以讨论。看来他们收获颇丰。戴夫也曾搞过每月一次的开发经理人员会议，可只能请到其中 15%的人出席。

程序经理们在所谓的“ 蓝色托盘 ”午餐中定期会晤，人们就一些特定项目交流经验，沟通信息。微软公司将其中的一些录了像广为流传。某些小组的开发员也定期聚会。比如 Excel 开发员每周一次“ 自带酒食 ”午餐会，人们借此机会谈论不同领域的代码，既对正在了解产品的新人有所裨益；也会

使自己熟悉一些经验丰富的开发员，他们往往可以填补自己在某些变化和领域内的空白。有时，开发员还会就有关代码的细节问题发行备忘录。[10]

即使没有上述这些定期会晤，“点子”在微软内的传播也是很快的。例如，人们可以通过电子邮件“别名”或小组地址去联系相同职能领域的人员。举个例子，戴夫·穆尔可以向所有的开发经理发布信息，而罗格·舍曼可以联络所有测试经理。尤其是穆尔，他常用电子邮件，借此与别人分享一个好主意，综述一下自己在一些新书或文章中的发现，或者就阅读提出些建议等。人们还会带着各自的好主意在公司内转来转去。正如 Windows 和 MS-DOS 的前任测试经理戴夫·马里茨所言：“我们可不是循规蹈矩的家伙，我们常在各部门间东走西瞧。”

自食其果

微软还有一种学习机制，即所谓的“自食其果”。我们在第 5 章解释过，这会意味着创建一特定产品的组将尽可能在自己的工作中使用该产品。如果产品性能太差，构造者和其他每个组员不得不“自食其果”。比如，经理人员坚持要求研制 Windows NT 或 Windows 95 的小组只要有了稳定版本，就得先“款待”自己，在自己的运营系统中使用产品；这样，小组就能自己测试出错误来了。Word 和 Excel 组将在开发阶段使用这些产品的新版本，以写出备忘录或保存项目资料。

这一机制的更广泛应用是微软各组在工作中使用微软自己的工具和商业产品，通过亲身体验，见客户之所见，并向相关小组不断反馈信息。因此，比尔·盖茨命令程序经理使用 Visual Basic 来做个榜样；而史蒂夫·巴尔梅则指示微软内部信息管理系统（MIS）的职员在早期开发阶段就使用 Windows NT。相似地，微软各组一般只用公司自己的商业语言编译程序，而不会制造或购买更多的专用开发工具。NT 组的开发经理戴夫·汤姆森评论了盖茨和巴尔梅的工作：

> 这些人所做的是试图确定让所有的固有力量能适得其所，发挥效力。而告诉 MIS 组使用这些产品本身就是把各种自身力量安置妥当，从而产品能做得更棒。要知道，你有一长串用户，只有你的产品很出色，他们才会谢绝一长串的其他供应商，随你而来。另一个例子是，当 Visual Basic 还在开发之中时，比尔就命令我们组进行网络管理工具开发的成员将它用于工作中……他们根本不喜欢它……不管怎样，他们还是这么做了，结果，推动 Visual Basic 进行了一系列有益的改进……比尔已达到了他的目的，他迫使 VisualBasic 的成员做了改进；而后者在发布产品时也承认，开发小组在使用这些产品中提供了大量的信息反馈和意见，着实贡献不小。所以这些家伙们以后会欢迎一些压力的合理介入的。

瑞克·罗歇德在 Windows NT 中扮演了一个相似的角色。作为 Mach——一个在 Unix 基础上的微型内核操作系统，对 NT 的设计有影响——的主要设计者，他能使用和测试 NT，并与自己的系统作比较。比尔·盖茨聘请罗歇德可不是为了检查 NT，可他非常倚重罗歇德的评价。换句话说，盖茨让另一种“自然力”适得其所，详审那些关键性新产品，并帮助他引导这个项目：“举

NT 的例子，我们让创造出了 Match 的人管理着小组……他总在挑 NT 的毛病，一发现哪儿比不上 Mach，他就直言不讳。于是，这个组出现了许多好建议……这的确利于 NT。我们并没在聘用瑞克时说你是 NT 监视委员会的……我是和 Word、Excel 一起发展起来的，我知道那里的规则系统，也认识所有那些开发员。NT 则相当新，所以我会不时问瑞克‘你认为这事怎么样？’这里有许多人巴不得对周围的其他各组批评一通呢。”

“自食其果”的另一内容就是开发员应使用普通客户所用的计算机——而不是那种带着许多 RAM 和硬盘存储空间的“高性能”机型。Windows 组的负责人，高级副总裁布兰德·斯尔文伯格详细叙述了他对这一规则的感受（微软并未能一贯遵守它）：

> Windows 对安装的支持系统非常依赖，因此，产品能够在一个通用的 4 或 8 兆内存的机型上运行良好就显得特别重要。因为，如果你给你的开发员用 16 兆内存的 486 机型，他们所见的结果就会完全不同于你的客户们的使用结果；在 16 兆时运转自如的到了 4 兆条件下没准会摇摇晃晃，抖得厉害。你不能说：“没问题，我有测试机器呀。我会不时在我 4 兆机型上试用一下。”所以我的做法是，确保我的开发员用的机型和我所预期的客户所用有可比性。这样，我的开发员基本上只用 4 兆或 8 兆的计算机。有时这又太保守了，因为大多数开发员想用最快的计算机、最新的技术。他们不是总能想得通，但还是得这么做。如果你看我的保存记录，只要在哪一个项目中开发员用的硬件设备比客户的要优越，产品推出后准保会出现问题。

最近的一个例子是对 MS-DOS 6.0 中 Double Space 特性的内部（和β）测试。[11] 微软主要依靠使用自己的 1000 名员工进行测试，他们一般用 Windows 和新应用程序。微软对该产品的 5 000 名β测试员和内部的微软用户有相似的特点——他们倾向于运行 Windows 和新的 Windows 应用软件。不过，在一个老式的或有点毛病的硬件上与某些旧版的 DOS 应用软件一起运行时，DoubleSpace 会遇到一些问题。而且，在和 DOS（而不是 Windows）并用时，它还会与 MS-DOS 6.0 中一个叫 Smart Drive 的记忆存储功能相抵触。[12]

在微软推出这个产品前已有了上述问题的征兆，可对他们而言，返工太少见且太困难了。不管怎样，由于看中了信息压缩和记忆增进功能，数以百万的还未升级到 Windows 的 MS-DOS 用户买了 MS-DOS 6.0；又因为机型太老、太小，没法升级，他们还继续在用旧版的 DOS 应用软件。这类用户中很多人都碰到了 DoubleSpace 的问题，从而迫使微软进行一次 MS-DOS6.0 的新的发布（这只花客户很少的费用）。[13] 如 MS- DOS6.0 的开发经理本·斯利卡夫回忆的：

> 还在出品之前，我就知道有两个问题，一是有一些特定的复制保护应用软件不能用，而我们的感觉是没有几个用户会超出预期范围。[用户] 一定是那些用旧版应用软件、还未升级的人，但他会将操作系统升级的。我们很难知道那会是些什么人……我们的一个理论是……买了 Windows 3.1 的人们更偏爱买 MS-DOS 6.0，而机器是和 Windows 3.1 一起推出的。将买 MS-DOS 6.0 却没有 Windows 3.1 的人似乎

微乎其微……从营销人员那儿我了解到有一批人正要升级，可之后的说法又是也有一批人不会升级……后来我们就做了随机抽样的电话调查，以便弄清楚谁正在买 MS-DOS 6.0；结果是，购买 MS-DOS 6.0 升级的人当中 80%至 90% 拥有 Windows 和一台 386 机器。这个消息符合我们的预期……[而且] 我们感觉到 Windows 3.1 是我们的保护伞……可是一反思，我们也不能确信自己的考虑已足够谨慎……在微软内部我们有超过 1 000 人在运行 DoubleSpace，却没有看出任何问题。我们的β测试员和 1 月份的β论坛一直在叫：“推出去吧，推出去吧！一切都就绪了。”

虽然错误率只是 20%，我们还是觉得微软本应发现并解决这些问题的。公司测试员、内部用户和β测试员应该每日都在一个旧款的 DOS 机器上对 Double Space 和其他一系列 DOS 应用软件做试运行。当然，MS-DOS 6.0 和 Double Space 还提供了一个吸取教训的机会。正如我们在第 5 章所讨论的，对于它的下一个主要的操作系统发布对象 Windows 95，微软延长了内部开发、测试过程。它还把一系列β测试场所增为 400 000 个。虽然产品延期一年半才推出，而且一些特性有严重问题，上述的和其他一些措施还是减少了因急于出品或β测试不充分而发现不了重大错误的可能性。

原则二：鼓励使用数量化的测量标准和衡量基准进行信息反馈和改进

许多公司业已发现，在产品开发这些管理和改进活动中一个关键因素是创造出数量化的测量标准和衡量基准。测量标准是一些统计性的量度体系和资料，可以此对质量加以评判，并表明产品或进程中关键因素的特征。微软的项目最终逐步采纳了一小批测量标准并坚持使用下来，这也算是对事后例分析报告中披露的问题的一种对策吧。经理们主要利用这些做项目的重大决策，比如何时结束某一阶段继续推进或推出产品。另外，数量化的测量标准和资料被作为一种信息反馈系统用以帮助项目间共享信息，一起改进。微软的经理们现在还正在试着利用测量标准创造一套全公司范围内的衡量基准，以期摸索出最可行的开发方案。这些衡量基准将会使所有小组更方便地展示各自的技巧并互相学习；而为确定衡量基准进行的资料收集过程也会延长项目事后分析文件的制作。

前面我们提到过有一些公司在利用数量化测量标准和资料进行软件开发方面比微软先进得多。微软人也承认，他们在明确测量标准和积累项目资料上可做的事很多，而这么做就能把握——以一种能相互让渡的方式——干练、成功的经理们做决策的思路。不管怎样，测量标准和资料在微软已有长期的特殊地位。要是缺了它们，人们可能会企图将决策建立在关于感情用事或主观性的争论上，或者会在是否容纳一个特性，是否发布一个产品，是否采纳一个不同的做法或工具问题上争论不休。微软各组现在还存在这些争论，但感情化的方案从来不会在比尔·盖茨或微软其他高级经理那里获准通过。比如，戴夫·穆尔对运用资料支持人们在程序检查中的建议的重要性发表意见到：“如果人们不能利用技术性材料说明那些建议，就有可能受到忽视。这些人都知道这点。只要是以感情化的或政治化的背景引发的方案，就不用理它；就当什么也没发生。但是，如果在一次程序检查中有人向你提出

建议，只要它是有技术支持的，而你却置之不理——你没有充分利用所有资料去做决策，那你就会在那项程序检查中被完全撇在一边。那将是你的终生大错。”

数量化材料在微软似乎很受重视。这种公司内的上下一致使我们深信，微软在更好利用测量标准和衡量基准上有着相当潜力。

测量标准的类别

在第 4、5 章以及在事后分析报告的概述中，我们已看到了微软拥有的各类测量标准的例子。在本章以后的部分，我们将讨论一些关于微软利用实验室和客户支持组织的实例。总的来说，这些测量标准分为三大类，它们具有下列可资度量和追述的特征：

- 质量——包括错误的发现和每日每周的修改率；错误严重程度；错误的解决；错误群分析；每 1000 行源代码发现的错误；实验室使用结果；客户使用中满意程度；客户使用中出现问题的频率。
- 产品——包括特性类型和数目；按照源代码行数、储存字节和可执行文件字节来考虑的产品规模；上、下两版间产品规模的变化；代码重复使用程度；进度和内存使用情况； 代码的测试范围。
- 过程——包括特性小组规模；估计的和实际的各特性、各阶段和可推出产品的完成日期；进度拖延程度；各阶段完成情况一览表；发现错误的有效方法。

在一个项目中，经理们会自始至终运用这些质量、产品和过程的测量标准。在一个项目开始前，他们利用测量标准帮助预测人员需要情况（比如开发员和测试员需要多少人）；他们还试图就进度要求进行估计（比如集成和缓冲所需时间）。在项目进行当中，经理们——以及开发员和测试员们——会使用测量标准来获得有关进展情况、稳定性和效果显示上的信息反馈。项目完成之后，他们又用这些测量标准来标明项目的特征，弄清问题所在领域，评估错误发现技术的有效性，探索多元化项目的趋势，并强调可改进的机会何在。然而，对于某些由微软各组和其他公司收集的测量标准的使用价值与普遍性问题，公司内仍有争论。人们在解释一些专门的测量标准数字时的确容易出错。因此，微软职员（及其他人）需要很仔细地描述为什么以及在何种背景下，他们选用了这些测量标准。比如，在微软我们就看到，人们特别努力地区别应用软件的各种特性，而不是系统产品或项目上的。

在微软，无论是应用软件还是系统软件人员都广泛关注着错误的测量标准。不过系统项目人员更重视那些表明一个产品的执行速度和内存利用特性的成果标准。在系统化的产品里，较低层次功能的有效运行极为重要，因为较高级的应用软件产品会重复执行这些功能。用户还要求应用软件解决快速进行人机对话问题。事实上，一个系统产品的内存使用，比如它要求的是 4、8 或 16 兆（MB）内存，直接会转变为其潜在的市场销售规模。举个例子，许多老式的或低价的 PC 只有 2 兆或 4 兆内存，因而不能运行诸如 Windows NT（它要求 16 兆内存）这样的系统产品。相反，应用软件产品正倾向于强调产品推出时及中间阶段期限上的测量标准。应用软件项目一般每隔 12 至 24 个月发布新产品，为了实现时间目标就会削减特性。而系统项目则多偏向于推迟发布，只要有可能的话；因为他们必须发布单独一套产品特性用以支持

许多不同客户的使用方案。

例子——关于错误严重程度的测量标准

测量标准最广泛地被用于微软项目中以处理各种各样的错误和缺点。这些错误是软件产品的“常客”，软件行业的多数人称之为“臭虫”。无论在微软还是其他什么公司，对错误的发现和改正活动从来都是软件开发的一部分。由于微软开发员编写了大量代码，创造的产品的每日构造日渐增多，他们也就会产生和修改大量错误。因此，针对错误的测量标准如此之多是不足为怪的。如第 5 章曾讨论过的，错误情况反映了产品的整体质量和开发测试期间产品质量的改进程度。微软还使用了一些测量标准用以表明那些发生了并可区别出不同的改进领域的各类错误的特性，估计进展程度以及对发布产品的准备情况做出评价。最终看来，一个项目的错误数目是许多因素的一个函数：人员技巧，人员数目（尤其是新雇员），具体化中的变化，新的、有更动的代码量，代码理解难度，测试的彻底程度以及和其他产品的兼容性等。

项目在给错误分类时使用了一系列不同方案来辨明它们的严重性，它们是怎样、何时被发现的，又是何时解决的，以及对产品特性的影响如何。多数组依靠一种四等的严重程度分类方案，这与软件业的其他公司大同小异：

- 1 级严重度：错误导致了产品失败（“崩溃”）或没法操作。
- 2 级严重度：错误导致了一个特性不能运行并且不可能有替代方案。
- 3 级严重度：错误导致了一个特性不能运行，但可有一个替代方案。
- 4 级严重度：错误是表面化的或微小的。

表 6.1 错误按严重等级的分布

发布日	产品	严重等级（%）			
		1	2	3	4
4/89	MacWord 4.0	31.5	19.1	38.1	11.3
	MacWord5.0	28.7	14.2	34.5	22.6
9/91	Works for Windows 2.0	21.1	25.8	39.0	14.0
3/92	Excel4.0	17.3	24.1	42.7	15.9
3/92	Project for Windows3.0	17.0	34.0	40.0	9.0
8/92	Money2.0	10.2	21.6	47.5	20.7

注：每行加起之和约有 100%，代表对一特定产品的所有错误。来源：《Mac Word4.0 开发事后分析报告》；《Mac word 累加错误比率》；《Works for Windows 2.0 开发事后分析报告》；《WinWord 2.0 测试事后分析报告》；乔恩·德·沃恩：《微软 Excel4.0 事后分析报告》，6/8/92；《微软项目 Windows3.0 版的开发事后分析报告》，3/16/92；《WinProj 3.0 项目回顾分析报告》，4/2/92；《Money2.0 测试总结报告》，所有微软内部文件。

表 6.1 概括了在不同产品的开发期间各种严重等级上发现的错误比重。这些错误都是在一种产品的“正式的”每日构造中以及在 β 测试这类的测试发布中被抓获的。（微软是在一种产品的最终的公开发行之前发现并更正这些错误的）。

对于一组特性项目，错误率资料表明在逐年的产品发布中，一级错误比重呈现下降趋势。比如，Mac Word 4.0 在 1989 年 4 月出品时一级错误率为 31.5%。[14]Works for Windows 2.0 在 1991 年 9 月出现，其一级错误率是 21.2% 。[15]Money 2.0 是 1992 年 8 月发布的，其一级错误率只有 10.2% 。[16] 软件开发方法的改进为这一趋势提供了一种可能的解释。

经理们也愿意看到较高级别的严重错误率能通过产品的旧版换新而下降。例如，较高级别的错误率——1 级、2 级或 3 级——在 Mac Word 4.0 和 Mac Word5.0 间就略有降低。如果该项特定产品的这一势头持续下去，那就说明开发者们以一种错误倾向较少的方式增加或改变了特性。

现行测量标准的局限

微软的现行测量标准只是一个最小的（或可能是超小型的）集合。它们需要包括用于更深层过程与产品的测量标准，那是有助于在发生之前预测和了解多种问题的，从而项目就能有更多的前导时间以预防问题的发生或至少为应付它们做点准备。一些公司，诸如摩托罗拉、惠普（HP）、NEC 以及东芝，已成功地把测量标准运用于这些更深层目标的完成上。然而，麦克·梅普尔斯认为，与他们那些正式计数的公司相比，微软发现并修改了更多的错误：“ 对于 KLOC（上千行的代码的测量标准），我唯一的保留意见是我们并不像那些追求它的人那么严谨。在每 KLOC 的错误上，我们并未记录下开发员在设计检查以及写代码中发现的所有错误。现在，作为一种观念，我们要在项目中推动更多的识错尽早完成。零缺陷以及许多方案在促使项目的错误提前出现，可我们的计数方式又使得发现的错误看起来少了。 ”

而且，即使微软收集、使用了一系列的测量标准资料，各项目仍依赖于许多公司并未数量化为测量标准，而是在实际经验中形成的主观规则。戴夫·穆尔曾试图使用测量标准和对历史资料的收集来抓住这些主观知识，但他遇到了来自一些开发员和经理人员的阻力，他们以各自产品或项目的独一无二或存在区别而予以辩驳。克里斯如此评论了这一挫折：

> 我们所遇的问题之一是虽然掌握了大量经验性信息，可那在[个人]相互间的传播却难乎其难。你看看克里斯（彼得斯）和其他一些推出了许多知名软件的人的各种事例，他们只熟悉自己的那一摊……要他们把知识变成一种能广为传播的形式真是太难了。所以我们只好以给克里斯的谈话录像而告终，可那毕竟不够完善。克里斯，说吧，到底什么事启发你考虑到这还不能出品？那是测量标准，那是“ 我看过[错误] 表了，在我看来它有问题 ”。为什么看起来有问题？……这就因为有人只给出纯粹的经验性[信息]，却没有作为测量标准加以指明。

以测量标准为基础的过程改进

1994 年，麦克·梅普尔斯创造了一种新的组织结构，把大家召集起来集中全力于过程的改进和功能组内部的训练。他推选了克里斯·威廉斯担任产品开发主管和新结构的负责人，后者曾在谈及软体开发需要更加系统化时直言不讳。威廉斯 38 年前毕业于保龄·格林大学的计算机科学专业，在微软收购 FOX 软件公司时他是该公司的开发经理。赴新任之前，他在做编译程序的开发工作。

组织重建后，所有的功能主管现在都向威廉斯汇报，其中包括测试、开发、程序管理、用户培训和内部工具的各位主管。培训经理（他负责的组共有 14 名微软职员，另加一些外部承包商）和使用测试经理（他负责一个 30 至 35 人的组），也需向威廉斯汇报。（向威廉斯汇报的总人数大约有 100 人，包括需要用特殊语言的中东产品开发组）。威廉斯认为此次重组旨在改进知识传播，自我检查和不同组间的互相学习。由于产品组合正试图共用更多的构件，因此这些目标已在微软占据了新的重要地位。项目现在需要预测进度，并生产出高质量产品，这既是为了外部客户，也是出于彼此互利。他们再不能只为各自所需或自己的那部分用户创造尽善尽美的产品了。

> 我想，戴夫[穆尔]、我和其他一些人均已承认的主要一点是我们的开发过程一直对创造卓越产品的过于关注……以至于在某些情况下，我们被驱使着，付出的代价是没能力求采用一个利于控制过程的行动步骤。有些组在这点上显然比其他组要好，可多数却缺乏连贯。我们没能促进组间的相互支持或共同吸取各自教训。所以我的工作焦点已转向试图促使人们所获的知识尽可能多地在小组间流动，并让人们更多地自我检查各自的行为方式，这会使他们干得更好。我们这儿聪明者云集，只要指明最可行的行动步骤，他们就能妥善行事。但在过去，这种优先次序的考虑一直是力争在最合适的时间把最可能好的产品打入市场。鉴于公司日益变得构件多元化、相互依赖紧密化，我们逐渐认识到掌握一些通行的做法和行为步骤是何等重要！

以过程为中心（而非以产品为中心）的事后分析

作为过程改进这一创新的部分成员，克里斯·威廉斯、罗格·舍曼、戴夫·穆尔和另一些人正着手改进为技术开发员和项目经理们设定的测量标准。尤其，他们看出需要建立更多的过程测量标准以解释为什么一个项目在进行中出了问题，而不仅限于说出了什么问题。另外，威廉斯和其他经理计划改变的一个情形是：各项目有时不能使用同样的术语或不能在定义上保持前后一致。（比如，微软并不真有一个通用的定义能解释“代码完成”这一关键的阶段性术语。）威廉斯和他的小组还试图调整和扩展项目事后分析过程，以便并入更丰富的测量标准资料，包括对新资料的收集以及相关的现存资料的共享。最后，他们希望能用测量标准资料在全公司范围内定义和交流衡量基准，从而取得最好的开发方法。

威廉斯尤为急切地要把事后分析过程提高到一个新的水平上。各组在把各自问题编入分析报告方面工作出色，可事后分析经常只是很简短地分析一下为什么会出现问题，哪些解决方案是可行的。而且，单个的各组并无一个持续的合适过程能确保在将来项目中不重蹈覆辙，各项目间也不存在一个系统方法可用于互相学习。比如，威廉斯观察到几乎每个组的每份事后分析报告都提到了由于小组成员不断加入新特性致使进度失控的问题。

> 你去他们中间总会发现一句耐人寻味的“我这里失控是因为我没能关紧特性的闸门”，一贯如此。我们总是一遍遍地吸取同一教训，这很惨痛。我想在某些标准下，事情就是这样的。而用另一种标准衡量时，我们做的估计工作很糟，导致了我们实际行事时要花些时间在额外添加

的特性上，这同样是在做糟糕的工作……

戴夫[穆尔] 做的很不错的事情之一……是让人们看清自己的事后分析过程以及他们是怎样行事的。人们在对如何做和什么能做得更好进行检查的事后分析阶段一向工作出色。我们唯一的问题是他们没有把再评估、建立新的优先次序和工作推进拢入一个循环……他们说我们将来会改变这一切，可没人说从现在起我们就要改变它，也没人停下来思虑一下6个月以后——到那时你改变它了吗？……事后分析只是人人脑海中都已有的短语罢了。每个人都知道其含义是什么，每个人又都指望花些时间这么做。我们只想……较少关注发生了什么，更多注意怎么发生了。你怎么会失控？并不是“当我需要一份错误表时，该死的家伙从不给我，”而是“为了确保你需要错误表时就能得到，哪个环节出了问题？”……事后分析的问题就是……在你自始至终地按照自己的方式工作于开发过程中时，事情处理并非有条不紊。而那正是我们希望去控制的……现在从一份事后分析中所能获得的大多是无病呻吟。

戴夫·穆尔同意这种对事后分析的意见：“关于事后分析问题的全部就是这个东西是好的，可……人们……一遍又一遍地抱怨着同一个问题。大多都是……这类事情。许多情况下只要做一件事：一吐为快。好了，把心里话说出来了；再去试着做些改进。那就是用在事后分析中的一些做法。无疑，我们也要改进事后分析的其他环节。在向下一次发布的过程中理应做出改进的。”

最佳做法的衡量

这一想法与包括过程审计的事后分析的观念是有联系的。比起写一份事后分析报告，它花的时间要少些，但能创造出一系列易于共用的技巧清单。为了做到这点，威廉斯、舍曼和穆尔一直在设计一套有关最佳做法的衡量基准。（例如：“开发中随每次发布会产生一份详细的测试发布文件或相似文件吗？”“你是否在第X阶段估计了将检查出的错误数目？”“你检查了你的特性表了吗？”）这些可用作过程审计的基础，并在某一项目——比如第二阶段后，为进行衡量基准的部分分析提供一个实施框架。这还能较早地提供一些过程中的或“先导”性暗示：这个项目是否遇到了麻烦。完整的衡量基准方案能检查每一功能的内部运作，比较不同组的做法，并把微软的绩效和外部过程的衡量基准绩效标准做对比；这可能包括软件工程协会和国际标准组织，或行业客户的满意度资料。威廉斯这样对目标做了解释：

我们正在努力开发一种过程评价系统，那非常适于我们的开发过程。这是截然不同于你和那些卖主或供应商的普通咨询关系的。我们把它叫做我们的技巧衡量基准程序。我们正在发展一套开发方案，覆盖全过程……人们应非常关注……我们想把事后分析作为自我测量的一次机会。你怎么能与我们已获得的迄今为止的最高智慧背道而驰呢？我们认定这会演变成一套标准。我们的目标是到那时展示一些相当具体的测量标准……我们可能最终会形成250行左右的[指导纲领]。

威廉斯将强令各组进行衡量，但他是从那些表现出一定兴趣的项目开始

的。如果衡量基准的努力成功，威廉斯将会有一套他正想用于小组的测量标准,以使项目管理更有效,相互的学习也更系统化。然而和在他之前的戴夫 穆尔一样，威廉斯也料定会遇到阻力，尤其他成立的新组正尝试一种更广的，试图对项目如何组织和管理自身产生影响的创新：

> 我们最感兴趣的并不是把这变成个急转弯，而是要鼓励自我检查……我最后想的事……是人们普遍参加进来，就像都要进行一次牙齿检查。所以我们要做的第一步是针对那些对此有兴趣并已明确表示的组。之后，我的计划是向各组兜售。而且我认定到那时一些组已能发现其价值所在，并在和其他组展开讨论，于是其价值将有目共睹，迅速增加……我内心的感觉是，75%至 80%的组会判断出这是有价值的，并想这么做。在 18 个月或两年时间中我们将实现目标……
>
> 如果没有能力把该方案和测量标准联系起来，我想它必遭惨败。我绝对相信“ 你所不能衡量的，你就不能控制 ”。在我看来，我们并不必急功近利，对所有的事都一追到底，可我想我们在以一种连贯的方式集中追踪一些事情上做得并不好——比如每开发员每周的错误……我现在正力争改进的事情之一就是检查工作太低劣，无论是估计的还是实际的，尤其时间耗损不起。所以，举个例子，一个开发员会说：“ 我想那要花我三周时间。”可 4 周半搭进去了，我们照说不误：“ 你做好了吗？ ”而他也照答不误：“ 快了。 ”终于他完成了，我们只会叹一声“ 唉！ ”却不会停下来问：“ 哎呀，那可花了你 4 周半。为什么？哪儿出了问题？你遇到的是哪方面的困难？ ”——太糟了。

罗格·舍曼在 1993 年成为测试主管，他尤其关注写出过程文件，获得好的做法。在 1988 年至 1989 年间，只有几个组遵循梅普尔斯初入微软后的要求（见第 1 章）。舍曼在欧伯林大学学习了计算机科学和音乐之后于 1988 年加入微软。他先是在波音工作，后才作为一个用户软件测试员进入微软；他的组在 1988 年曾就他们的开发和测试过程写成文件,但对材料的处理却未跟上事态的发展。舍曼现在也正在留心要明确规定一些通用术语和测量标准，以便微软各组能更持续地从不同项目中学习，并制定一个标准过程好让他们能一起工作，共同改进：

> 我们正做的是要用最通常的术语写一个文件，包括所有这些产品的最新版本。然后我打算把它带到微软的开发圈中，说：“ 我希望大家都同意，以此作为将来我们开发产品的标准途径。 ”现在有两件事相当重要。一是，我最想搞出一整套标准化术语，这样，即使我们在多元化产品上放弃测量标准时，还能做合理比较。我认为这是微软拥有的一个战略优势。我进行的开发项目很多，由于这些项目的案例丰富，我们能更迅速地了解到什么是有效的，什么做起来却不灵。但如果两组在“ 代码完成 ”的定义上都完全不同，那么所有你统计的资料……都会很糟。第二点是我想有……一个标准的现成的过程。我不想这个过程一成不变；而希望人们用它去实验。当他们做不同事情时，我希望他们记录下做的是什么，以便我们能有区别地予以追踪。如果这工作起来更有效，则广为宣传或变为标准供大家遵守。我想能看到这变成技巧文录……它不是

官僚化的、负担一样的东西，而能紧跟事态发展，把我们所知晓的最好的做法都收录进去。

原则三：视客户的支持为产品的一部分和进步依据

除了尽力从内源——公司项目——多加学习外，微软最近还千方百计地从外源学习：即它的数以百万的客户们。客户可能成为信息的无价源泉，所以现在他们有很多机会直接向微软的开发组提供信息反馈（见表 6.1）。例如，程序经理们所要分析的既有以行为为基础的计划资料，又有来自客户“希望热线”的信息和每月对客户电话分析整理而成的“电话分析报告”。程序经理和开发员在项目期间需要在微软的可用性实验室里测试样品；各项目还提出各自产品发布一个α版，以供内部使用和提供信息反馈。紧接着会对选定客户进行一次对外的β发布以作为实际测试。之后，开发员将在这些反馈信息基础之上，赶在最终产品向制造商或市场发布之前，做进一步的完善。

甚至微软的产品支持服务（PSS）部门也利用可用性实验室来做它所称的“支持能力测试”。这是要检查支持一项新功能的难度有多大，并用诊断问题的不同方法来试验。当一个新产品推出后，微软就把开发员和测试员们安置在 PSS 的热线电话旁，并与来自公司不同支持中心的 PSS 人员一起参加“情景屋”电话会议。这些做法都给开发员和测试员提供了机会去亲自听听什么问题让客户抱怨不止。PSS 还会就客户对微软产品、客户支持及公司整体的满意状况作实际调查。除此之外，它也和营销组合作研究客户使用微软产品和竞争对手产品的情况。微软甚至发布了其产品的“工具版”，它能跟踪每一次键盘敲入和鼠标的按动，从而作出用户实际使用产品的电子记录，这个循环会不断重复。

这一系列连贯的活动可清楚表明——微软在 1990 年以前并未系统遵循——客户支持和客户信息反馈已和产品与开发过程的改进紧密结合起来。其好处有两个：减少支持成本，并通过创造出更多的对产品满意的客户而增加销售额。微软 1993 年提交的一份质量奖励申请书描述了这一客户支持哲学：

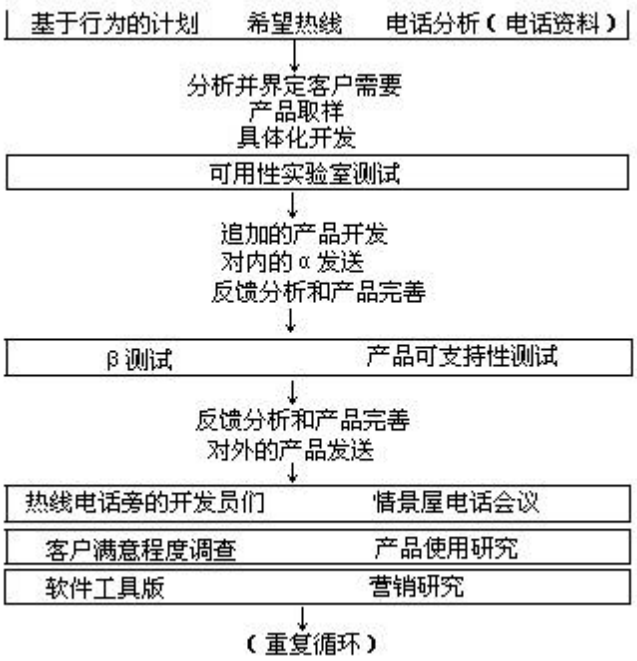
> 微软的支持哲学强调每一次客户支持活动都是改进产品设计的机会——相应地，我们会用更便于使用、更无需产品支持组织关注的产品回报客户。正像微软已在实践着的，这种客户驱动法在产品设计上的应用改变了软件开发循环的目标。以前，开发员们关注的焦点是“计算机设计”目标——如果没有额外编码就能完成很出色的设计，他们会感到心满意足。现在，只有当他们完成的特性制作使客户初次使用就能理解时，他们才会满意……在微软，完成可用性改进首先要做的是了解用户面对的工作任务是什么，以及他们是如何使用软件来完成它们的。这些资料能帮助决策在新的或升级的软件上设计哪些特性组。PSS 成员组在信息收集过程的每一个阶段都和产品开发员们一起工作，利用那些通过与客户接触得到的资料，并在特殊的测试与分析中提供协助。[17]

产品支持战略和组织

微软曾有着一种对客户反应较为冷淡的公众形象。它发布过的一些产品很不好用（比如 MS-DOS 与苹果的 Macintosh 操作系统相比），对某些类型

的用户来说既慢且制作粗糙（如 Macintosh Word 6.0）。但它总算已从不太注重客户支持的坏名声下逃出来了（至少敢于和 WordPerfect 比较了，该产品以免费电话线提供的无限制的且频繁而出色的支持而闻名）。理查德·巴斯——Windows NT 的高级产品经理——曾回忆起往日的情形：“如果你是个寻求支持的客户，那么想想 3、5 年前你所忍受的一切真是可笑。你只能在规定的几小时内给我们打电话；你不大可能或根本不可能找到几个人，谁知道他们究竟在干什么。我们是有些人非常有天赋，可这对本行的任何人来说都算不上是优势。”

图 6.1 产品开发期间的客户投入



在微软，这种状况随着 1990 年里 Windows3.0 的引入开始改变，而 1992 年 Windows3.1 则强化了这一变化。新的 Windows 操作系统和新的 Windows 应用软件的出售量数以百万，其中数以千计的客户开始带着各种问题每天给微软打来电话。另外，微软当时正处在从主要对计算机硬件制造商出售软件向通过零售商主要销给单个客户的转变过程中，而个人用户购买软件后更有可能需要软件制造者的支持。市场研究还表明，和以前的大型计算机与小型计算机客户一样，PC（个人计算机）用户越来越多地视来自软件供应商的支持为软件产品的一部分。为了保住这数百万的新客户，微软不得不改变做法。

一种变更方案是循着 WordPerfect（它现在为 Novell 所有）的路子，建立起一个庞大的组织以便更快、更有效地回答电话询问。另一种变更方案是着力于更加系统地从电话交谈中获取信息，并把这些信息和其他客户材料传给开发组织，从而制造出更易于使用的产品——以此减少客户打电话的需要。第三种方案是要提高处理数目日增的电话的效率，比如通过使支持过程尽可能自动化或以收取不同的费用来打消打电话的念头等等；还要制定出更确定的打电话期间，并帮助为支持运作过程提供资金。

事实证明，微软对这三件事都愿意做。比尔·盖茨带了头：他曾在一次“每周反思会”中（后形成了一份备忘录送往微软执行经理部门，并被泄露给了新闻界）总结认为，微软再也经不起因忽视客户而付出的代价了，它必

须改进其支持政策和组织。他还向公众许诺他的公司将为新的 Windows 产品提供更好支持。与此相适应，微软增加了回答电话的员工人数，此举实在意味深长。其增幅是从 80 年代中期的 20 几人到目前的仅美国国内就有 2000 多人（差不多 5 个微软雇员中就占了 1 人）；另 1000 个支持人员分散在海外 36 个国家工作。微软还通过产品组组织起电话技术员，总的说来，他们要支持大约 200 种产品。毕业于威斯康辛大学的 MBA、产品支持服务部的前任营销主管特瑞希·梅曾回忆过微软经历的这一转变：

> 我要说的转折大约发生在两年半以前……其表现有两点。一点是[当]比尔在一次论坛上公开陈述说支持是我们事业很重要的一部分，我们将保证在你打电话咨询微软 Windows 产品时——即使此刻是 Window 3.0 的发布时间——你也能确切得到高质量的支持。另一点……是比尔关于他的“比尔每周反思”所写的内部备忘录，不过最终还是到了新闻记者手里。其论点之一是他要在产品支持上进行投资，那曾是事业的重要部分，并将继续扮演重要角色……我们的研究表明那的确是不可或缺的一部分。所以大约两年半前，我们增加了投资数目，并建起一个管理组织以关注我们如何才能推进这一转变，这些是很有意义之举。

梅承认这一新战略有两大基柱：一是使产品更便于使用，另一个是建立起支持组织。后者包括采取积极行动、尽可能地实行自动化；并通过向客户提供更多信息——比如以 CD-ROM 盘的方式，使他们能自己解决更多的问题。梅说微软仔细研究过 WordPerfect，但并不想模仿它免费奉送式的支持组织：

> 这一战略有两个部分。一是……减少我们最终接收的电话数，方法是设计一个用户界面以及可提前输入问题的产品，并利用支持和文件的提供来减少问题数目。而另一战略是用颇有意义的投资来确保我们所提供服务的高质量。所以我认为对此要同样强调。例如，我们已经看过了 WordPerfect 管理支持的办法，并正从中学习。但我们还要研究一下你怎样才能最有效率、 最有效果地提供高质量支持。促成满意的标志是什么？因此，用一下投入——产出模型，让我们看看影响有哪些方面，你能从可处获得客尸满意度测试最有效的信号，然后对模型加以微调。我们的目标是，对花出去的每一美元，你怎样才能做到最有成效地利用电话？……
>
> 对比现在的一对多式电子化服务，你来看看两年前被称为一对一的事故处理法吧，我相信我们所遇事故的 80%是由员工在电话上处理的。而今天只有 50%的事故由员工解决。其他 50%是通过专题讲话、我们的 FastTip（快而通）服务、布告栏及其他电子媒界以电子化形式解决的，这使得我们以较低成本扩展了接触客户的范围。所以我们在投资于解决如何提供服务，什么是接近更多客户的最有效益方法的问题方面是很有意义的。

电话信息及其处理

正如比尔·盖茨在他关于产品开发中的公司力量一览表中所说（见第 1

章），微软接到的数不胜数的电话为分析客户需要及困难提供了绝好机会。微软每天大约会收到 6000 个“事故”询问——4000 是通过计算机或按钮式电话发来的电子质疑，另 2000 个电话是由支持工程技术人员予以答复的。平均来说，每售出 3 个产品单位或“盒”就有 1 个电话，这比起几年前微软所经历的每 1.5 或 2 个盒就有 1 个电话可谓是进步显著。打进 PSS 的电话平均每个历时 12 分钟：支持技术员花 5 到 7 分钟诊断问题所在，2、3 分钟讨论一下解决方案，另花 1、2 分钟结束谈话。回答问题的耽搁时间在 1991 年是 4 分钟左右；到了 1992 年—1993 年间，则下降为 1—2 分钟，虽然当时电话数目翻了一番。而现在微软对 80%的电话能在不到一分钟内做出答复。[18] 对电话模式的详细分析为如何改进产品，最终减少电话提供了许多有用信息。这些信息还启示人们如何给服务定价以打消打电话的念头，或至少让电话需求更为确定些。

电话分析　向 PSS 打电话的是客户中的少数。微软的信息资料表明，70%的软件用户从来不打电话来，而另 15%的客户却打进了电话总数的 70%。大多数打过电话的只就每种产品打过一两次，并且一般仅限于购入之后的前 90 天里，因为在此期间电话可免费打入。历史资料还显示，在前 90 天里打入的电话中 40%到 60%是有关设备安置或软件安装的；其他常见问题涉及打印、新的或有变化的特性的使用、操作环境及与其他产品的共同操作性。[19] 很多客户在遇到问题时会首先求助于邻居。然后他们会利用产品的帮助文件、说明或去找卖给他们计算机或软件的零售商。由于向微软的客户支持热线打电话是大多数用户的最后求援之道，理解客户这种选择组合的需要对减少电话数目就显得至关重要了。对这些用户的研究结果激发微软要使某些特性更易于使用，并提供出更好的帮助文件和产品说明。微软还决定对计算机零售商、软件咨询商及乐于帮助周围人的熟练用户提供更多的信息和培训。

微软也知道哪类产品、在什么时间段产生的电话最多。例如 1993 年间，70%的电话是来自于高居销售额榜首的 5 种产品，其中：Windows 占 31%，Word 占 15%，Excel 占 12%，FoxPro 占 8%，MS-DOS 占 4%。微软甚至仔细跟踪了解每种产品的平均电话持续时间及与此直接相关联的每种产品的平均电话费用数。从微软总体上看，平均每个电话耗费约 12 美元（低于 15 美元以上的同行业平均费用）。[20] 但是，产品不同时，费用、电话模式和长度也大相径庭。比如，1993 年中，花在 Word for Macintosh 上的平均电话费只有 7 美元，而 Word for Windows 则要花 12 美元。这种差异似乎可以说明 Macintosh 产品还是要稍微好用些。或许是从 DOS 程序上转移出来的人数过多之故，Windows 用户呈现出电话咨询更频繁的趋势来。

过去，微软是要每个部门等额交纳产品支持组织费。但自从 1991 年 7 月以来，微软已改为向单个产品单位收取支持费用。这意味着如果一个组的产品用起来很费劲，设计的特性有许多都是“电话发生源”，那这个产品单位就必须承担所有这些支持费用。“基于活动的成本核算”是微软征集的一个术语，凭借这些征集活动微软才能核定每个电话的准确费用，从而决定如何向单个产品单位收费。麦克·梅普尔斯曾解释过他怎么又被牵扯到对 PSS 的管理上来，这不仅是出于盖茨新近关注起 PSS 的驱使，而且出于他在消减成本方面的一贯努力：

两年半以前我开始看人员统计。那时我们一直忽略了它们，突然之间我认识到比起开发员来，那里的人员数目更多，于是我说：“这是个不妙的势头”。所以我们做的第一件事就是开始把所有的产品支持费用直接记到产品头上。现在，每种产品会得到一份月报，告诉他们有多少电话、占收益的比重、存在哪些问题，这一做法已扩及全球。我开始这么干的原因是我刚来时，没有人注意产品成本问题：你们想，500 美元收益中 20 美元可以扣除出去，谁还在乎产品成本。所以我们召集了个工作组，提出了一些简单的准则。我们能从产品成本中削减 8%至 10%——即 20 个百分点中的 8 或 10 个百分点。这是很大很大的变化。从利润计算来看，这简直是直逼成本底线。

很显然，当你有了问题时，要做的就是让人人都引以为忧，这也正是我们处理 PSS 问题时首先做的；我们开始向所有的人报告发生的问题。然后，我们有了一些实际的简单准则，就像说，在你成为我们的工作对象之前，为什么不对每次产品发布都设法减少一半原有每单位产品的电话数或一半的电话时间呢？于是你会想，好吧，怎么解决这一问题？

其实很明显。你必须处理最集中的难题。所以，如果打印信封是处理电话的用费中最突出的问题，就派些人去动脑子把它做好些。

梅普尔斯把产品收益的 8%设为总支持费用的上限（相比而言，微软估计 WordPerfect 的此项费用为 16%）。同样，各种产品间有了很大差异：Excel 和 Word 的收益中 7%花在产品支持上，6%则花在用户产品中。相比来看，Windows 的这类支出占收益的 20%，而 Windows NT 则占到 25%。微软很快发现，公司诸如 Windows NT 和 LAN Manager 这类客户服务器产品支持起来开支更大，因为它们引起的电话比大多普通 PC 软件要多而且长些，需要更老练的支持技术人员。因此，向这些新产品领域的成功推进要求在支持组织上进行有意义的同步投资。

定价机制以及对产品类型和所收到电话类型之间的关联分析对于控制和预测在新的、现存产品上的电话支持需求是至关重要的。有的产品会在前几周或数月内便有大部分电话打进来，可其他产品可能要很长时间后才有电话。微软和其他 PC 软件公司一般在客户购入产品后至少前 3 个月内免费给予支持，但电话费自付（除了 WordPerfect）。这个行业的趋势，包括 Novell 的 WordPerfect 部门的新产品，也已从无限的免费支持转向有偿支持。为了处理好那些预计在好多个月——如果不是几年的话——都有电话来的产品，微软也和其他行业的一些公司一样，制定了价格政策。对 Word 和 Excel 这些基础产品，客户可在工作时间里通过单独收费电话接受免费支持。90 天之后，对这些及大部分的微软产品，客户可通过支付每分钟 2 美元的电话费拨打“900”这个号码接受 24 小时服务。希望每天 24 小时接受支持——比如针对公司产品——的客户还可以从支持方式中进行选择：每分钟 2 美元，一个电话 25 美元，一年 195 美元，或者每年 20000 美元以享受高质量的支持服务。随着这个定价表的实施，平均看来，一个新产品在其使用期内（包括所有版本）50%的电话会在前 6 个月内打进来。

支持技术　微软利用计算机和通讯技术使得对客户支持的成本大大降低，并

且能更有效地分析流入的信息。它主要采用了三种形式：支持技术人员使用的计算机系统或“工作台”，电话系统和客户能利用的自动化信息服务。

“PSS 工作台”包括一组一体化工具，由微软的内部信息组创造，在高能 PC 机上运行。这些工具能跟踪客户提出的问题（电话数目、长度、等待时间、每个产品的电话数、每类客户的电话数及每类问题的电话数），然后他们把客户特定信息储存在系统配置和发行之中。他们还制定了特定做法一览表以供技术人员据此采取行动；记录个人工作量；在工作台信息的基础上生成报告；方便客户对免费条目的查询（比如磁盘替换）以及提供一些微软知识库的使用方法，这是一个电子化信息库或技术支持数据库。[21] 支持技术人员能使用固定查询词语检查整个数据库；他们也能以写入文件回答以往未有答案的问题而补充数据库。微软在 CD-ROMs 上或通过联机网络向客户散布这个知识库的部分内容，包括 CompuServe 和 Genie。

电话系统拥有一种所谓的存储功能。微软通过某种技术把电话内容数字化并存储下来，从而对接收的每个电话都能记录少量信息。PSS 产品开发顾问、在微软干了 9 年的老兵马克·辛登沃格曾表扬这一技术是电话系统的“卡迪拉克”，它包括了一个通往 PSS 工作站的软件“桥”。工作站和电话系统的目标是要尽快分析客户电话，并制定出按重要程度排序的问题表递交产品开发组织。辛登沃格曾对从一个客户电话中采集信息的过程与好处作了如下描述：

> 每一次和客户的接触都是改进产品的好机会……我们手头掌握着每个电话的一定量信息。现在每种产品都有个标准模型……附有该产品着手进行所依据的一些标准类别。我们想看出一点主要领域的总体特性，比如多少电话是针对安装的，多少针对打印，多少关于普通用法，多少是有关操作环境的，多少又是涉及共同可操作性的……然后，再深入下去，我们跟踪了解了一些特性或问题，最基本的，就是电话都涉及了什么。今天这一任务是通过电话上的“卷藏”机理完成的。你有一张一览表，于是你可以把一些能代表问题出在哪方面的信息和电话联系起来。这些当然是概要性信息。你可以获得一个 3 位数的产品代码，再加上 UCC 代码——就是我刚才告诉你的那个水平，也是一个数字。之后我们就有了问题代码数字。所以，如果它是有关安装的特定问题，我们就能知道有多少电话与此相关。
>
> 这与以往的支持截然不同：你和手下的支持人员要去告诉某个人物某个问题，他们要问：“你在这方面接到多少电话了？”你只能说：“哎呀，我们不知道。”于是他们又问，“好吧，如果我只能解决 10 个问题，那么我应从哪 10 个最重要的问题着手呢？”。你又得说：“哎呀，我不知道。”而现在我们能给各个问题排出先后次序了。

在自动化支持领域——这省去了要人回答电话的麻烦，最重要的是 Fast Tips（快而通）技术。这种通过按钮或电话可以实现的全天服务能就微软产品的关键领域提供信息，并回答一些普通问题。另一种自动化支持系统 MS Down-Load 是一个电子布告牌，在上面微软会展示新的驱动器（运行打印机、图像屏幕和点设备等外部设备的软件）、产品修改（权宜修补）及产品解释；客户利用调制解调器以进入该系统。PSS 还管理着微软信息网络。它包括微

软开发网络，该网络在 CD-ROM 磁盘上向开发员提供技术信息，每年更新四次。它也包括微软技术网络，这是每月更新的又一个 CD-ROM 信息库，能提供详细的产品信息、培训信息，以及 PSS 知识库——PSS 支持人员使用的技术支持信息库——中的一部分工具。[22]

产品改进的信息反馈

对微软而言，一个主要的挑战是要去组织和分析每天从不同渠道收到的、来自客户的大量信息。然后再把这些信息以一种有用而精简的形式传输到各开发组。经过正确的处理，这些信息将有助于开发组在修改错误时确定先后次序，使产品更易于使用，并提供出客户真正想要的新特性。另一难题是统计概要只能传输有限数量的信息。

为了尽量利用好客户信息反馈，微软最近建立起一个产品改进组，并引进了其他一些机制来分析客户信息并传输到产品组。由于出自客户的特性想法和改进实在太多，这里不可能一一列出；主要例证包括产品安装，Windows 里对打印和图像显示的驱动背后的设计原理，以及贯穿于各种产品的工具条和特性的标准化。经理们还要求开发员们投入更多时间将产品制作得易于使用些；而且，一旦微软推出一个新产品，便要他们去回答客户电话。这些措施是对其他类型的客户研究和产品适用性测试的有力补充。

产品改进的积极创新　1991 年，微软组织起一个囊括了 6 名 PSS 高级技术人员的产品改进组。每名技术员都是某一产品上的专家、担任着该产品开发组织顾问的角色。Excel 专家马克·辛登沃格和皮特·希金斯、杰夫·雷格斯及麦克·梅普尔斯一起讨论了成立这一小组的想法，并付诸实施，称之为“微软的秘密武器”。“我们被称为产品改进组，我们是产品开发顾问……我们的使命是帮助微软更好地满足客户需要，通过客户的信息反馈和建议而改进产品。为此，我们的战略是去发展一些方法、系统和过程以便获取信息、加以量化，再以一种易于检测的形式传送出去，这颇具意义。它其实意味着开发真正的有效途径，能把五花八门的信息加工整理为统计性的确凿无误的资料。”

产品改进组建立了两种分析客户信息的宝贵机制：所谓“电话分析”的月报，及一个独立的客户建议数据库。报告主要是按产品里界定的功能组织起来，它按产品细分，分析上月从客户打到微软的电话里收集到的各种主要信息。尤其重要的是关于“客户就每种产品与全球各地的 PSS 联系的‘十大’原因”的一览表。[23] 微软把电话分析写在纸上并通过电子邮件传送各产品组的个人和经理处。建议信息库的重要输入源是微软的希望热线，这是一个电话号码系统，客户能打进来对特性的新功能提些建议（客户也可以写信陈述）。PSS 工作人员负责把建议抄录下来输入数据库，以供全公司使用。电话分析报告里会包括每种产品的主要 15 种建议。

在电话分析报告中得以鉴定的问题和建议广为传看；盖茨和其他高级主管们也都是热心的读者。因此，产品组对于报告中反映的问题一般会在 3、4 小时内就能做出反应，并让 PSS 知道他们打算怎么处理。正如辛登沃格观察到的：“交流圈发挥了效力，所以每个产品组都有人对我们的刊物逐期予以信息反馈，一般是说：‘问题的确是这样的。’然后他们会说出对此他们打算怎么办或何时着手解决。有时他们已开始处理这些问题了。”通常，测试

组会有一名人员负责把 PSS 鉴定的问题录入产品组的数据库，并确保开发员对仍存在的问题加以解决。程序经理既要关心本产品的新版中哪些特性有待修改，又要注意到客户就新特性或特性变化所做的建议。任何月份中单个产品都可能从客户那里收到几百条建议，因而辛登沃格组的整理排序工作是重要的：

> 现在,所有这些建议也——对 Excel 而言它每月的数目在200到600条间——进入了数据库。我们不仅每月检查一下它们，而且把它们积累在数据库中了。然后在计划过程期间，所有建议再次整理分流到各程序经理处，他们仍要再仔细检查一遍。我们不能把这些资料给丢了。我们确实感觉到这是种有竞争力的优势……你的程序经理们各有一套特性，他们就会问你收到过些什么电话是与此有关的……于是我们的工作就是以一种确实高质量的数量化的方式告诉他们什么有待修改；如果互相需要协调的话，什么是达成平衡的最好办法，因为一切都不是很清楚的。

产品改进组还组织过研讨会，与会的有来自各产品组的营销、程序管理、开发、测试和用户培训人员。这些会议更深入地分析了对某一新产品来说，为什么会有人打电话找到 PSS 以及在特性或提供上哪些变化可能有助于减少电话。辛登沃格把这叫做一种新产品的“事前分析”：

> 我们在两天中花了 8 小时和来自 Excel 业务单位的 80 人一起检查了客户在各种常规原则上打来电话涉及的所有问题。每年我们都要做几次这类调查。这么做可以是对一种产品的“事前分析”，也可是“事后分析”……在我们仔细检查事情过程时，可以收集到许多信息，包括人们如何使用我们的产品，他们有哪些问题，他们希望看到哪些情况发生，并且我们会提交一份长达五六十页的文件和一个数据库……然后我们开始想象性描述和看说明。在仔细查看说明期间，当我们检查到特定性质说明时，他们会说出这么做是否由于 PSS 的职员的帮助。之后他们还会问我们，这能解决你告诉我们的问题吗，客户会理解这一方案吗，如何把它制成文件——这一类的问题。所以我们就完成了特定性质说明方面的检查以及代码检查……再接下去是进入文件提供的仔细检查阶段，到时将有一名工程技术人员，他是我们组员之一，将代表客户参加每一条文件编写——他们需要知道什么，什么是他们亟需理解的关键概念。

开发员时间的再分配　来自 PSS 的数据已帮助微软改变了开发员的时间利用方式。麦克·梅普尔斯曾回忆起他加入微软时，开发员大部分时间都是用在新特性上了。他们很少注意去使现有特性更好用或在一些其他方面做改进：“我们想做的就是让开发员把 1/3 时间用于改进现已存在的，l/3 用于新特性，另 1/3 则用于兼容性或和世界的其他产品保持一致性。那只能意味着运行 Lotus 的空白表格程序或与某人的网络相配合，诸如此类吧；并且（或）只是和你以前的特性保持连贯或兼容。在那之前，我们可能还想过要花 80% 或 90%的时间在新特性上，而不是去使已有的特性更简易、更好用……PSS 真是帮了我们大忙，让我们搞清楚了那些领域都是些什么。”

梅普尔斯和其他经理们还主张开发员在每推出一种新产品后能花些时间在 PSS 帮助其员工处理电话。各小组的管理助理承担起了安排进度的工作。有些开发员在进度紧时会设法躲避这一职责；但一般来说，开发员似乎也觉得花时间在 PSS 工作是他们时间安排中的重要部分。对于开发员而言，这既能培训和帮助 PSS 人员，也能使自身接触到那些颇为受挫而直接就给微软打电话的客户，从而获取第一手资料。布兰德·希尔文伯格回忆起 1990 年他所在小组推出 Windows3.0 后他开始在系统组中进行的这一实践：

> 无论何时我们推出了一种新产品，在随后的一周、两周或三周中，无论需要与否，开发小组都要被派去做技术支持……他们有这么多障碍，并且不能处理各种询问，所以要[开发员]去那儿培训他们……自我到微软后那可能就是我干的第一件事吧。Windows3.0 刚刚开始推出，非常成功，于是他们简直要给工作压垮了。所以我把整个开发小组派到产品支持部那儿做两周的协助支持工作。这真是让大家对客户有了丰富的认识。那也是我的另一种哲学，就是真正让开发员、让每个人都以最终用户为着眼点，记住他们是我们的老板，我们成功的唯一原因是他们喜欢我们的产品。

梅普尔斯是对软件组实行这一政策的强烈支持者，那里诸如“ 邮件合并 ”（Word 上的一个特性，允许用户把一系列地址和一个普通资料来源合并起来，比如制式信函）这样的问题已是声名狼藉了。把开发员送到 PSS 就有助于使他们对客户更敏感些，也是给 PSS 一个信号——你们的工作也很重要。梅普尔斯这样来描述他的一次访问：

> 我们尽量让每个开发员每季都在 PSS 呆一天，专接电话；为了鼓励这种做法，我也去这么做。有一次我在 Word 组中转了转，发现在这帮家伙后有张沙发……我说：“ 嗯，那是干什么的？ ”他们说：“ 那是邮件合并沙发。只要我们接到一个有关邮件合并的电话，我们就知道要打上半小时，所以有人会接过电话走过去躺在沙发上，再和客户谈。 ”一个需要改掉的有趣的问题！但把开发员放在那会使 PSS 的员工感觉好一些。这就是古老的霍桑效应吧……同样重要的是，他们会听、会看什么事情是导火索，还有哪些领域需要改进。
>
> 另外，PSS 建立起在每种主要新产品发出后每周举行两次的“ 情景屋电话会议 ”。这些会议召集起整个开发组，并把他们和微软不同支持地区的支持人员联系起来，就客户所遇到的问题有哪些类型做另一种审视。微软还会发行会议记录，以便让其他组也知道这些会上都谈了些什么。和其他过程创新一样，又是 Excel 组首先采纳了这一做法。

客户研究和可用性测试

除了电话和其他的 PSS 联系方法外，微软还针对不同场合利用了一系列研究用户的机制。这些研究包括对用户行为和该领域内产品的分析，以及开发期间的产品特性测试——换句话说，即可用性实验室分析。

PSS 的研究 微软每年支出 50 万美元就产品支持服务和客户满意度做市

场研究。一个主要项目是每年一次的最终用户满意度测量调查，由一家外部调研公司为微软做的，旨在通过鉴别为了获得并保住一个客户什么是必要条件从而衡量客户的满意度。这项调查覆盖面超过 1000 用户，按微软的和非微软用户粗略分成两组。对微软客户的问题是要分析他们对微软产品、微软公司及微软产品支持的满意程度。在上述三方面都满意并且一定购买和推荐微软产品的用户被定为“可靠”客户（他们不会转向其他公司的）。调查还就微软与其竞争对手及各自产品和支持服务做了满意度水平的比较。数据表明产品设计与客户对支持和公司总体满意度之间高度相关。正由于此，为 Macintosh 设计的产品——它有个更为简易的界面可用——在客户满意程度上就比 Windows 产品更高，虽然产品用起来并无差异。特瑞希·梅描写了这一发现：

> 在[客户满意度]与什么产品及该产品是怎么设计的之间有相关关系。比如，我们做了些交叉型表格，结果显示如果某人对产品满意，那他们将会对服务也非常满意。如果你有一个交互作用的用户界面，那你一般会对服务更满意了。这里有着一点光环影响效应。一般来说，我们让同一个技术人员回答一个关于 Mac Excel 和 WindowsExcel 的问题。所以你要把人员固定下来。Macintosh 的界面有一点容易，或说在过去是这样的，于是我们在一些 Mac Excel 的支持问题上就比在 Windows 开发早期阶段里对 Windows Excel 的问题上得到了更高的满意度级别。

PSS 的另一主要调查项目是向微软打来电话者的客户满意程度。PSS 人员需要请一小部分打进电话的人参加一个 20 分钟的调查，然后他们每月分析一下这些资料，观察趋势。这些调查数据表明微软的努力是有效的：过去的满意评价相对较低，而最近的一个表格分析结果是 78%的客户对 PSS 非常满意，64%的人愿意再次购买，61%的人一定推荐该产品，另有 54%对产品非常满意，结果是 41%的“微软产品可靠”评价率。(24)表 6.2 给出了 1993 年 6 月在一周内对 163 个客户电话所做的满意度分析的例子。

表 6.2 按产品区别的客户满意度数据（1993 年 6 月）

	非常满意	基本满意	总体满意
总计（加权的）	74.1%	16.1%	90.2%
Word for Macintosh	83.3	12.5	95.8
Excel for Windows	85.0	10.0	95.0
Windows 3. 1	84.2	10.5	95.0
Fox for Windows	72.7	18.2	90.9
MS-DOS	61.1	27.8	88.9
Word for Windows	65.2	21. 7	87.0
Windows for Workgroups	65.2	17.4	82.6
Access	68.0	8.0	76.0

资料来源：微软公司，产品支持服务部门：《ITAA 奖励申请》，1993 年 6 月，第 13 页。

产品使用研究 在对通用产品进行研究以了解用户习惯和需要方面,微软依靠着几种途径。例如，在 90 年代早期对 Word 加以改进时，微软就挑出了 200 名 WordPerfect 用户，通过每月采访对他们进行了一年的研究。微软还在美国不经常地做过些“ 分割式研究 ”，即从全国不同地区随相选择电话号码，挑出 800 名左右愿意回答 20 分钟调查问卷的用户（约 50 个问题）。这可以就某种特定产品提供有关典型用户情况的信息。微软也研究过各产品的用户登记库,但认识到这些人未必是有代表性的。另外,产品经理克里斯蒂 维特莱斯特别提到了营销组还对特殊客户（尤其是大公司客户）做过详细的个案研究，从而收集到了客户需求方面的信息，并为营销宣传提供了数据资料。

克里斯·彼得斯回忆起从产品使用研究中获得的一些重要启示：空白表格程序虽然神通广大，可 20%的使用只是记录一列单子；所有文件的 40%只是普通信件；所有文件的 30%是备忘录，只有 5%是业务通讯。彼得斯得出的结论是：“ 基本上，我们所发现的事实是，现实生活中这些产品的使用比任何人能想象到的要简单得多——这很有意义，因为现在我们正把这些产品向 Safeway 商店里数以百万的人出售……这项研究是少数几项极受信任的事情之一……我们视之为富有意义的有竞争力的一大优势，因为只有我们真正了解到了公司正向哪一类人销售产品，没有别人做到了这点。 ”

信息的另一来源是微软的产品工具版。产品的这些复制品都有一个独立文件，可以记录下用户每一次按动鼠标或敲下键盘，以及每个动作所花时间。这对研究人们实际上如何使用一种产品及他们在要完成特定任务时做出什么选择是极重要的。微软将这些特殊版本交给那些愿意作为试验点的公司，然后研究收回的数据。自从 80 年代末以来，微软对 Excel 和 Word 的历次新版本都做这样的研究，一般在首次推出后几个月内发布出去；各组利用这些数据对产品加以改进或对下一版重新设计。（Word 的工具版对鉴定邮件——合并特性以及寻找文件格式化的不同捷径起过重要作用）。

客户信息反馈的另外渠道是 β 测试员,他们通过联机 CompuServe 网络向微软汇报情况。 β 测试员主要提供有关哪些错误要在产品最终发布前修改，还对产品下一版出谋划策。正如第 5 章所讨论的，被选为 β 测试员的用户要签一份私下协议，然后就可得到软件及需填写错误数据的表格，并从电子渠道交还。

可用性实验室测试 我们在第 5 章描述过微软的可用性实验室如何就新产品的信息反馈提供了另一种宝贵渠道。其他的软件制造商，从 IBM 到 Intuit，许多年来都在利用可用性实验室研究用户怎样对潜在的新产品、新特性做出反应。然而，我们相信微软与众不同，它已到了把这种测试形式集成到固定的软件开发过程中的程度。其重要性在于，不像其他的信息反馈渠道（如 PSS 数和 β 测试报告），产品组能在开发过程中——而非之前、之后——就收到这些关键信息。

建立起固定的可用性实验室测试的主意源于 1989 年和 1990 年间微软正在开发 Excel3.0 的时候。一些开发员已在用典型用户测试典型产品。但是做的并无规律，微软也没有实验室设备来分析和收集数据。麦克·梅普尔斯回忆实验室的起源说：

可用性实验室的小组是从很少几个人起步的……他们是个服务性组

织，试图对每种产品都帮上忙。在某种程度上，那里是真正的突破口。如果你回到 5 年前，当我们开始搞可用性组时，该组总是对开发组说：“ 10 个人中有 6 个不会这么做。 ”而开发组则反驳：“ 你们从哪儿找了那 6 个傻瓜？ ”所以我们就逐步让开发员们也去可用性实验室观察并参与进去。然后我们做出了工具版。再后来我们有了这么个主意，就是在他们开发特性时，找来 6 个、8 个或 10 个可用性实验人员，让开发员就在中间巡走，和用户谈谈什么是能用的，什么不行。结果这成了开发员们严谨对待的一件事——“ 你已经对那个特性进行可用性测试了吗？有什么发现？ ”我想……他们认识到了自己和用户想的就是不太一样。

克里斯——1989 至 1990 年间 Excel 的开发经理——对实验室的产生和普及也有类似追忆。他强调，即使微软最好的开发员，要使特性易于使用，光有自己的那些常识还远远不够。而且，微软的“ 机智的 ”开发员们在去可用性实验室看到用户与特性“ 奋力斗争 ”的事实之前是不会认识到自己的局限性的。从那以后，使用实验室成了家常便饭：

我不记得确切是在哪天了，但开始是发生在 Excel3.0 那段日子。碰巧有些开发员在看可用性测验。要不是亲眼所见，你肯定不以为然，并且会立即有无数想法涌入脑海。首先，你会马上想到人的因素，常常有毫无益处的看法——“ 嗯，如果他们不知道怎么用，可以查用户手册嘛 ”或“ 我的点子很棒；只是你找了 10 个笨蛋才导致可用性实验室里的运行失灵 ”——一旦你亲眼见到了，这类观念便都不会复生……

它是让你的产品更好些的有效途径。显然你能使产品更棒……人们经常认为凭借常识或聪明就能搞设计了。但是人们对软件做反应的方法复杂之极。我们最好的设计员，我想对任何人也都一样，只能做到 60% 是正确的。然后还需要第两次通过，在首次通过实验室测试后，正确率可达 80%，到第三次通过时，已能达到 90%了。

要达到“ 60%正确 ”指的是微软采用的衡量实验室结果的简单测量标准：没有参阅用户手册第一次就正确完成了工作的人数；或者人们在首次试用就能正确无误的任务比重（比如占总步骤的比率）。举个例子，一个开发员可能让一些测试员输入一段课文，然后随便查找一个词（如 Programme）并用另一个单词（如 Program）予以自动替换。这个工作会涉及到“ 查找并替换 ”特性的好几个步骤。

彼得斯观察到，开发员们为了改进特性的可用性，往往把大部分的特性不止一次地通过实验室检验。不过，有一些在待检验目录单上排在后面的特性根本就不经过实验室。一个例子是对 Excel 中的宏进行排错这个特性。几乎没几个人会实际用到它，并且微软预测那些用到此特性的人必是非常有经验的；因此，开发员们倾向于做实验室检测时略过这类特性。相反，像 Word 中的查找并替换这类特性非常受重视，因为几乎人人要用它。在有些重点考虑的特性测试中，由于总是只有 50%左右的分数，开发员们可能会五六次地把这个特性送往实验室。在这种情况下，除非他们对如何修改问题另有高见，否则就要对特性或它的用户界面重新设计，或干脆删掉。

虽然应用软件组是可用性实验室的最大受益者，其他所有的组其实都能

用它。彼得斯解释了其中的原因：“相信这种做法的组会比不相信的要用得更频繁……Word 和 Excel，再就是大多数（其他的）应用软件组，都有这种信任……系统组和语言组可能远远靠后，因为他们一般有更为熟练的用户，因此本来麻烦就要少些。系统开发员倾向于只对那引起普通用户会接触的界面特性或功能做可用性测试，而不考虑那些幕后操作的功能或只由其他软件开发员可能接触的功能。”

甚至 PSS 也参加进来了。它利用可用性实验室测试用于支持新产品和特性的不同方法的有效性。马克·辛登沃格描述到：“我们实际上在用可用性测试可支持性。我们走进实验室说：‘在给定某客户遭遇这一问题的情况下，我们会给这个客户用户手册和一个电话’。我们还会给他设计一个问题，然后他们就会给实验室另一边的技术人员打来电话。于是我们把两边拍摄下来，看他们试图解决这个问题的过程。”

微软请到可用性实验室测试产品的可以是任何人,比如用户组人员或‘从街上拉过来的’人员。如果开发员想检查一下与竞争产品的兼容性情况，如 WordPerfect，他们就去找个该产品用户。（“他们进来会得到免费提供的一大杯咖啡，”彼得斯如是说。）彼得斯解释道：“我们开始扩大范围做这类非常规用户的测试，是基于这么一种观念，即一个人从未见到某特性，一旦给了他一个工作，他一定会找点事干。所以现在一个开发员去找隔壁的另一个开发员说：‘坐在这儿，试一试这个。’”微软人还发现，测试一个特性并不需要一大批用户。彼得斯评论了这一发现：“这是基于一种少数几人仍能做到同样事情的有趣假设，但它看来完全正确。你可能想到 10 个人中只用一个人不会得到很多信息，然而事实证明，如果真是有问题的话，就是让 10 个人来做还是零分。对它加以修改之后，你会得到 70%的正确率……人们倾向于犯同样的错误。”

彼得斯把他从可用性实验室中观察用户得到的“教训”列了出来。其一是开发员在一方面的常识并不总能导致易于理解和使用的产品的出现。他举了在 Macintosh 中被称为“工具偏好”的工具选择的例子。微软决定采用 Macintosh 用的这个术语，把它放到 Word 2.0 for Windows 中去。可用性实验室测试的目的是要看，如果人们从未见过‘多层对话’菜单，他们能否理解：“因为他们不理解‘偏好’这个词，故只有 1/10 的人激活了对话框。多数人一扫而过。这项工作最后致使关闭水平方向滚卷条。”在小组把“偏好”换成了“选择”之后，十分之五六都做对了。小组还发现只要在对话框中加一些解释性文字——比如直接了当地说：“要看更多选项，请在目录左边按动鼠标”——这非常有助于人们通过菜单去找到操作办法，文字说明的添加也并不会放慢操作速度。现在，彼得斯表明：“在更现代的对话中，至少是微软的产品里，它们一般包含了更多的文字信息。而那是基于——如果你为提高可用性已是用尽了办法，在对话上加些文字说明一般会行之有效。”在加上说明后，彼得斯组的适用性分数提高到了十分之六七。后来他们又把彩色头像放在了左边以引起对该功能的更多注意，结果成绩提高到了十分之八九。“所以这个例证表明，哪里的正确率从 1/10 直达到 4/5 的话，只是需要做一点变动而已。我坚持认为那都不是常识性变化。”

彼得斯吸取的第二个教训是，他认为用户手册基本上不太必要，因为人们多数用不上它：“一个更重要的观念，就是用户手册可有可无。因为那是过去我们总可利用的一种逃避方式。”当用户不能理解一种特性时，只有 30%

的人会首先阅读用户手册。大多数用户有问题时会首先想到向邻居请教；要是不起作用，他们一般就向产品支持部门打电话。多数客户只是最后才向用户手册求援。

彼得斯承认，其他公司的人以及某种程度上的微软，都在抱怨在开发期间对每个特性进行可用性测试可能太慢，且耗资巨大。如果对这种测试不加适度控制的话，那些过于热情的开发员可能花上数年时间改进他们的产品特性而再也推不出产品了。他解释了成本目标：

> 我已告诉了大量各式各样的公司，他们说那似乎太昂贵了，我们支付不起。我说你至少能承受得起的是首先承认没人愿意去翻用户手册，或人们至少应努力创造一个不需要用户手册的特性来。你应该关注用户怎么操作不灵了，应该让一个对设计不熟的人看到它，哪怕他就是隔壁的程序员。所以我想，即使一个 10 人或 5 人的小组在麻省理工学院（MIT）干些什么工作，如果他们能有“人们能没有用户手册就做完这项工作”的较好观念，那必将受益，随后再观察他们的进展，多加关注。

为了减少成本，微软各项目一般会集中那些能一起运行的特性，同时进行测试，并且他们力图避免在实验室对特性的运行超过两次。像 Excel 和 Word 这些组还尽量把实验室工作限制到每周两个半日集中处理，每次测试大约 3 个特性。然后实验室人员组会把报告送往开发员和程序经理处。被分配到某个特性组工作的程序经理要和实验室工作组一起负责安排测试进度。梅普尔斯尽量也将实验室成本包括进来，把工作组限制在 35 人左右，实验大约有五六间。（每个房间一次只能有一两个可用性测试员。）虽然梅普尔斯承认存在着创造多元化设施以供不同设计组利用的这种压力，可微软还是坚持只有一个中心实验室。他对为管理好一个经济合算的可用性实验室而做的努力评论道：

> 实际上没有任何指导方针，可用性实验室是 200%或 300%的超过预定工作，因为它成了产品如此重要的一部分，以至于人人都想用它来干一切事。可用性实验室希望有 100 人，有 50 间实验室。可任何事儿一旦放开，就很难说会变成什么样子了……我们在做的过程中是把设计组化整为零，对于可用性问题我们可能也将这么做。一旦它大到一定程度了，我们就把它分成多个组，和客户更加接近，以使他们更明智些，我们会这么做的。集中的另一个原因是它是从两个人开始做起来的，确实存在着设备的问题。

PSS 的数据表明，实验室在使产品易于使用和易于支持方面收效显著。事实上，彼得斯把卖出的每“盒”产品的客户电话数减半之功归于了可用性实验室，他说：“没有人会很高兴打电话以获得产品支持。所以这是个两边受益结局的例证。你能做得这么好以使他们用不着打电话，你成功了。而他们首先想到的也不是打电话，他们也赢了。”彼得斯还宣称实验室有助于防止程序经理和开发员添加太多不必要的和复杂的特性。在许多因 PC 软件产品及其要求储存空间的增加而受到过挫折的客户中，“特性的衍生”是令人忧虑的焦点：“我们想的是，利用可用性测试我们能确实增加更多的特性，并

制造出越来越易于使用的产品。所以如果你做的对，我们并不认为特性的衍生会真的有用。这暗示着在能力和易于使用之间存在着一个平衡。只不过在这个行业中一直在灌输的是人们正像行政命令的加工人员一样开始做事儿——即，我们不能使产品造得更便于使用，所以我们就要删除一些特性。但我认为正确的方法是利用可用性测试。所以你能同时有能力和便于使用。我并不认为它们对立。”

原则四：促进各产品组之间的联系，实现资源共享

建立学习组织的最后一个问题是：不同的产品组要学会如何作为一个完整的公司成员共享构件和一起工作。微软对集成的、特性丰富的产品日益强调，比如 Office 和 Windows 95，这就要求它要有一种系统化方法以设计界面、分享构件，并在项目进度安排和产品质量方面提高一致性。微软和其他软件制造商都再也经受不起那种维持几大相互独立的产品组，并让他们随心所欲，“重新去发明轮子”的做法了，因为其代价必定是利润的降低以及客户对同一公司产品组合里的不一致和不兼容的抱怨。例如，微软经理们断定他们在各自产品中共有 14 种各不相同的文本处理代码，他们还有几种版本的代码，分别用于数学计算、绘制图表、帮助功能和其他一些任务。

微软目前在各产品组间已有了一系列联系、交流网，便利了各产品之间的构件共享和特性的标准化。除了对设计和代码明确进行重复利用外，项目组正在界定以构件为基础的整齐化设计。他们还正在制造些单个构件，不同产品能通过诸如动态链接库（DLL）或对象链接和嵌入（OLE）的机制予以利用。这种“以构件为基础的整齐化设计”方法和微软早期的主要制造各成一体式应用软件的方法是相对立的。应用软件产品，尤其是 Office，需要重复使用共享型构件（如一个工具条，绘图工具或窗口框架）以便向用户提供一个稳定的用户界面。这种重复使用还减少了向单个应用软件添加新的兼容性所需的努力。同时，标准化和构件共享会减少由于不同产品中相似功能的操作略有不同而需要向客户提供支持服务的诸多麻烦。

共同操作性小组

学习如何共享并不容易。微软实际上花了几年时间才在它的不同产品单位间实现了构件的系统化共享，要做的还很多。比尔·盖茨当然是其中的主角，他下令新的产品版本一律并入 OLE 技术体系，还要尝试其他办法力求共享。几年前，微软在这条路上的首批措施之一就是创立共同操作性委员会，然后是一个共同操作组。后者包括 OLE，共同操作设计组，初期的 Office 组（该组集成了 Office 并作为一个产品推出），适用性测试组，以及想象界面设计组。1993 年，共同操作组发展为了现在的 Office 产品单位。

共同操作组的工作是劝说独立应用软件组采用普通用户界面（如标准化菜单和工具），然后在主要产品上使普通功能同一化。按戴夫·穆尔的说法，共同操作组——1993 年共有 26 人——是被指定的“第三派”而和 Excel 组、Word 组一起工作的。如果两组就如何设计一个普通特性而意见冲突或陷入僵局，共同操作组就要做决策：“这是凳子的第三条腿。你要做出决策。所以如果 Excel 组和 Word 组不能拿出主意，你就作为第三方，对两种产品间的任何常用特性——文件打开，文件关闭，复制，粘贴，只要是你在两个产品上

都见得到的同样功能——他们的工作是确定这个特性是否以完全相同的面目出现。如果有一个打开的对话框，那么对话框要完全相同，对话操作的控制也要一模一样。

共同操作人员强调一致性是从最终用户的角度，并非指代码重复使用意义上的一致。共同操作组并不检查人们的代码去确保共同特性，而是关注于具体化，确定开发员把共同特性已设计在其中了。当然，开发员可以不同的方式给特性编码，但它在两种程序中必须以相同面目出现在消费者面前。如果不存在这么一个视促成特性的外形和使用的相同为中心工作的组，那么主要产品组生产出的用户界面将会千差万别，因为它们各自的工作重点只是制造最佳的产品。正如前任应用软件共同操作性主管克里斯·格雷姆在总结中说的，共同操作组使得各产品之间常用（或“核心”）的特性如出一辙：

> 我们创造出了应用软件的共同操作性，因为我们觉得……相配套的应用软件在市场上越来越重要。而要使我们的产品工作起来能配合良好的唯一办法是把一些出色的人员放进去，并让他们尽心尽力。相互独立，只关心本产品在同类中能出类拔萃的产品组间的信息流只会导致互相背道而驰。如果你只想成为最好的空白表格程序员或最好的文字处理员，做起决策来相对容易多了，但要想和其他组配合在某件事上达成一致意见却难上加难。所以我们创立了共同操作协调小组，把 OLE 用到应用软件里；开发出所谓的核心特性，我们不仅会把它们用于 Excel 和 Word，还会有其他应用软件。我们所谓的核心特性只是一个过分简化了的观点；它们是空白表格程序特性、文字处理、数据库，等等。它们这些特性对所有的应用软件都能通用，用户在所有应用软件中都会碰到它们，并频繁使用，因此希望它们用起来并无二致。

在共同操作协调与分享上的努力之所以能成功，一个重要原因是克里斯·格雷姆这些主要经理人员孜孜不倦的长期努力。格雷姆 1949 年生于爱尔兰，成长于加拿大的温哥华；他在英国哥伦比亚大学学习了应用科学和工程技术，然后在好几个公司做过软件开发员、顾问和经理，1989 年加入微软。最初他在一个四人小组工作，旨在找到能把微软主要的应用软件产品的设计标准化的方法。但这个组进展不大，因此不到 4 个月后格雷姆便转调到 Excel 组任程序管理员。在那里他领导一个特性小组试图重新界定 Excel 的构造，并希望和 Lotus 1-2-3 兼容。3 个月不到他便接任了 Excel 组的程序经理。然后又提任应用软件共同操作协调组主管，监管 OLE 开发、共同操作设计、想象性界面设计和 Office 产品的合作。1993 年重组之后，格雷姆曾在新的 Office 产品单位中短期担任共同操作设计组的负责人之职，向克里斯·彼得斯汇报。后来他获准转到另一工作岗位。当然，其他人员还在继续努力以推动常用特性的设计和使用。

常用特性的核心系列

1993 年，共同操作协调组鉴定出在所有产品中，大约 35 种主要特性是常用的，却还没能实现共享。于是微软新创了一种办法，该法依靠那些专门知识（比如有文本处理代码的 Word 业务单位，或有数学计算代码的 Excel 业务单位）齐全的特定的产品组担负重任，用 OLE 建构某个特性并加以简化。

其他组采纳了这些 OLE 界面，因此能把那些特性作为一个“ 标的 ”并入自己的程序。正如现在可以在 Office 成套应用软件中所看到的，这成了重新设计和共享所继续使用的方法。微软想让用户能看出来它的产品无论看上去还是用起来都差不多。克里斯·格雷姆详细做了说明：“ 我们想让微软的产品看起来像个家族,所以我们要让用户在使用一组产品时 10 分钟之内就能看出来这点。我们希望人们觉得它们都一样，所以，所有特性需要和人们首次见到的或最常用的保持一致。产品的表现应像是一个家族成员一样，它们应能配合默契。

常用特性的核心系列包括工具条，复制——剪裁——粘贴特性，字模，打印对话框和各种菜单。微软的研究表明大约 85%的用户操作需要用 35%的产品特性；常用特性组界定了这些特性，详细说明了它们的用户界面特性以及按钮、菜单、对话框和其他项目的操作表现。正如格雷姆回忆的，使数百个其余的特性一致化的边际收益是迅速下降的：“ 从这个角度出发，我们很幸运，因为在几百种特性中，客户所用的 85%只是集中在大约 35 种特性上。其他特性所有的客户都不常用，所以你对它们搞一致化收益不大。这样，通过集中处理 35 个特性——虽然那只占总特性的 10%——就抓住了 85%或 90%的用户实践。这样花出去的钱收益很大。你若去盯着那些使用频率不高的特性，收益就会急速下降。 ”

在 35 种特性的组合中,大多数用户会说在现在的产品——如 Word,Excel 和 PowerPoint 中，约有 25%至 50%的常用特性看上去、用起来都已经相当一致了。然而，共同操作协调组的成员却认为产品中至少 50%的常用特性有着显著差异。应用软件共同操作级程序经理吉姆·科勒评论道：“ 在至少 25%最多可能 50%的情形中，一致性程度非常高。我想大多数天真的用户看过这些不同的操作工具会说：‘ 哎呀，几乎一模一样。 ’但在至少 50%的情形中，是存在显著差异的。其中有一些差异是必要的，这可由不同产品间的功能差异予以证实——它们的确需要有另外的按钮，因为它们做的事不一样。但我要说……那些特性中至少 25%的视觉成份或功能成份存在差异是出于偶然的。 ”

常用特性数据的收集

微软收集了大量资料以了解哪些特性应包括在常用组里。首先他们判断并估计出主要产品——如 Excel，Word，PowerPoint，Publisher，Project 和 Windows——中超过 300 个特性是常用的。因为 Excel，Word，PowerPoint 和 Project 一共就有了 600 多个特性，所以微软最初想的只是所有现在特性的可能有的一个子集合。第二步，他们利用 Excel 和 Word 的工具版收集了特性详细的使用和频次数据。格雷姆特别提到了他们在与用户谈论及工具版数据基础上，针对命令使用频次制成了分析表：“ 我们本来认为可能有 300 种特性……我们做了大量研究，去决定最有杠杆效用的特性集合是什么。我认识到要在大量的组间取得设计的一致意见是很难的，所以我们需要开拓一种方法，借此找出最重要的东西并能确保自己做得是对的，所以我们和许多用户谈话。我们使用产品的工具版……从而我们知道了用户做不同动作的频率。

最后，小组通过对基于活动的计划的进一步洞察、补充了一些信息（参见第 4 章对这一技术的详细讨论）。研究表明，用户在产品间转移数据相当

频繁，并将常用的剪裁——复制——粘贴操作、字模的保存以及产品之间的其他格式化的重要性放在显著位置。用户还想建立混合式文件，包括文本、数据和图表。格雷姆解释道：

> 事实表明——这并不出人意外——最重要的集成操作是应用软件之间的复制和粘贴（或剪裁和粘贴）。另一个是建立混合文件……我们把基于活动的计划用作一种……决定所选特性的方法。当客户和一个“产品家族”一起工作时，什么是他们最常用的操作呢？产品间的数据转移是常用的操作。然后你必须把它分成细目来看看哪一种转移是常见的，数据转移到哪个方向，客户遇到的问题是什么，他们真正的意图而非他们今天所做的是什么？一定要先仔细分析用户的行为，再把它分解成你能处理的形式，加以区别对待。所以对每一种特性，我们都把客户普遍经历过的操作详细描写下来，而不仅仅只关注于“让这个特性一致起来”。

Office 一体化成套产品

微软推出的 Office 成套产品其实是主要应用软件的集成品，包括 Excel，Word，PowerPoint，Access 和 Mail。正如我们第 3 章里讨论的，微软对 Office 的定价很有竞争性，销售业绩空前好，现在大约有 1500 万套付诸使用了。(25)克里斯·格雷姆解释道：“他们都买 Office。我们现在售出的 Word 和 Excel 中 50%是 Office 的一部分。我们希望销售量越来越高。”最初，整装在 Office 中的应用软件产品并没有额外加以集成，相互之间也没有合作，每种产品保持本色、各自为战。Office 的唯一附加价值是一次性全部购买所带来的方便，当然，折价也很可观。

共同操作协调人员想大幅提高 Office 组合中共享和集成的程度。他们的常用特性研究使他们能够对 Office 的构建设计施加影响，于是用户界面现在就有了一致性，并形成了一种以构件为基础的构造方法。Office 4.0 在 1993 年 10 月推出，提供给用户的产品中，不同应用软件之间有关一致性外观。正如比尔·盖茨观察到的，在 Word、Excel 和 PowerPoint 间大多数高层次菜单条目是一致的：

> 我们在向各产品组推行标准时一直非常强硬。所以在所有产品的高层次设计上我们能看到，9 个高层次菜单条目中有 8 个是完全相同的。唯一不同的一个是在文本处理器中你有表格命令，在 PowerPoint 中你有绘图，在 Excel 中你有数据，因而那第 9 个菜单条目对不同软件各不相同。如果你从这个层次再往下看，就会发现在 File、 Edit 和 Window 中——在这些应用软件里任何共享的东西，甚至包括了对话框——所有的命令执行起方式都一样。那是为进一步发展提供可能的一步。那会使人们很舒服地把它们作为单一的应用软件，而不是不同的几种软件。[26]

Office 产品单位现在已有 24 名专职开发员。虽然微软也和一些外部公司签约构造一些专门构件，但这些开发员自己构造了共享型构件中的绝大部分。Office 项目还为所有应用软件（Excel，Word，PowerPoint，等等）和共享构件的构建单独制作了一份合作进度表。矩阵式的特性小组结构保证了

共享构件能满足各应用软件的需要。特定一组特性的矩阵式特性小组包括了 Office 开发员以及相关软件的代表，格雷姆对此阐述道：“ 这是我们用来管理共同操作协调过程的另一种工具……我们创建了像矩阵式特性小组这类特性组织，那里有来自每个特性领域里每种产品的代表。于是我们就有了 Excel，Word，PowerPoint，Mail，Project 等等软件，这都是特性小组中的代表。 ”

Office 的经理人员和开发员们不得不花上额外的时间、精力对这些特性与单个的应用软件加以协调。但是 Office 组的目的就是为它们提供基础结构，对此乔恩·德·沃恩也承认：“ 我们的境况是正在自力更生构建一套人人都用的共享代码。那真是苦不堪言。尤为令人头疼的问题之一是 Office 组的人员必须竭尽全力地工作，搞清楚客户的应用软件当前的各方面运作状况。我们不能不经过用客户证实自己的设计有效这么一个过程。与以前我们作为单独产品开发员时的情形相比，那可是要花一连串的管理费……Office 开发组的工作就是要为那些[应用软件]提供更多的基础结构。 ”（Office 以外的项目，如 Publisher 和 Works 等，也都在增加对 Office 组构件的使用。）

Excel 和 Word 小组起初对 Office 的观念是加以抵制的，因为它并不一定有助于 Excel 成为一种更好的空白表格程序软件，或使 Word 成为更出色的文本处理软件。但微软利用 Office 方式为用户提供了更为一体化的应用软件组合。客户通过购买 Office，而不是独立软件，用美元投了赞许票，于是 Excel 和 Word 小组妥协了。Excel 和 Word 开发员们还认识到 Office 方式的一个主要的技术优势在于比单个软件减少了内存的使用；这既能空出更多内存用于其他更多的特定目的，也节省了用户做出操作反应的时间。按照格雷姆的说法，减少使用内存显示出了另一竞争优势：“ 你要是能拿走一半的 Word，一半的 Excel，一半的 PowerPoint，那可是大大减少了 Office 用户的工作集合[内存规模]。所以比起那些来自 Borland、WordPerfect 和 Lotus 的独立软件——他们没能进行代码共享——人们更有积极性去买 Office。 ”

产品和构件中的相互依存

以构件为基础的方式导致了产品之间相互依存度的大大提高，而这种情况下的工作是会和创造出一流的独立产品这类目标的实现、按时发布的不断完善或每月构造原则的保持相抵触的。各项目组以前开发的是单个产品，他们要自己完成和控制代码编制。而现在，各项目都依赖本小组以外的各组提供自己亟需的关键构件。项目组会尽力紧密跟踪所需构件的生产过程，其方法就是尽可能多地监控构件供应者的内部生产阶段。Excel 组的程序经理约翰·法恩告诉我们：“ 工作要想干好的窍门就是确实做到跟踪并记录大量的阶段——那些必定发生的事件。如果我是一个构件使用者，我就要在短时间内跟踪我的构件供应者的许多个生产阶段。那是我检查构件供应者进展状况的一个方法。 ”构件供应者和使用者还需要频繁交流，以确保双方对构件的特性、界面和进度都有一个清楚的、现时的了解。法恩继续说：

> 基本问题是缺少……交流。在项目进展的几天、几周时间中，构件供应者会做出一些构件使用者根本不知道的假设，或者反过来。后来他们才发现这一点，于是双方的工作都不得不以放弃以前的努力而告终，要不就是没有效率。在此，任何一种最短路径的进度安排都与事无补。

真正有帮助的是双方不厌其烦地、频繁地去问问对方组——亲自验证一下，对方组是否真的在做他们该做的事情；而且确保他们不要做不该做的事情。

只对一两个使用者提供一种构件的项目组和后者一般会保持一种紧密有效的工作联系。构件供应者若仅有一两个使用者，就能和他们经常交流；而且也能依靠规格说明更准确地估计出构件出品日。比如，Visual Basic for Applications（VBA）首先向 Excel 提供了构件，然后又考虑了其他一些使用者。戴夫·穆尔强调说：“工作最有效的是只有两个客户。那样你就能按其所需提供服务，你能确切知道他们到底想要什么，你总能提供他们想要的——非常好的关系。过去，典型状况下应用软件组只有一个客户，因而很容易就能修改出品日，修改资源的数量，修改特性组，修正行为方式……在有些情况下有两个[客户]。可能你正在寻找严格意义上最终用户——家庭使用者——以及这里的公司用户，要把他们想要的具体说明出来是非常容易的事。”

系统项目总是在另一种意义上提供构件，即一个操作系统要包括大量的客户需要用以运行应用软件的构件。这样一种构件的“提供——使用”观念导致了应用软件和系统产品的主要差异。系统产品比应用软件有更多的构件使用者，而正如穆尔描述的，拥有大量构件使用者的构件提供者在预测出品日方面困难会更多：

> 系统组正在提供一个构件。他们提供所有这些服务。一个构件提供者可能有 10 来个客户，那你会从他那里发现，他需要那 10 来个客户的要求，并且不得不做周到。他们不可能必然地服务于某一个单个客户……他们必须非常努力地工作以满足那些客户的需要，那太难了。他们在构建那类产品时与只为 3 到 9 个客户工作的构件供应者有很大的差异……系统组总会有 10 个或更多的客户。空白表格程序开发员的需要是什么？文本处理开发员的需要是什么？桌面出版系统开发员需要什么？在这一环境下工作的各式各样的应用软件组的需要又是什么？还有些要求是来自于也在空白表格程序上工作的多种类型的出售商们……在微软的一个 Excel 空白表格程序开发员会比 Lotus 的同类职员有着更为不同的要求。因此那些形形色色的要求最终会驱使他们的产品界定与应用软件领域做出来的大为不同。

分享和共同操作协调机制

微软开发员在把 Office 构件（或任何其他的共享构件）和应用软件紧密联系的过程中，使用一系列不同的共同操作协调机制，比如对象链接和嵌入（OLE）与动态链接库。因为它们通过构件和数据共享以及同步操作，使得两种或更多的应用软件能合作运行，所以开发员们称之为“共同操作”协调机制。

我们在第 3 章提到过，OLE 是一种特殊的微软产品，它使对象共享和共同操作成为可能。（这个产品单位有 15 名开发员，15 名测试员，3 名程序管理员和 3 名用户培训人员；开发员使用的是 C++。）OLE 使得一种应用软件能从另一软件上获得一项服务（比如‘绘出这个图表’或‘把这个文本格式化’），然后它能把所得服务的结果嵌入原应用软件显示器中输出。用户还能利用

OLE 建立混合型文件，包括文本、数据和表格。克里斯·格雷姆回忆起了一段有趣的小插曲,当时有人正演示用 OLE 从 Word 向 Excel 提供一段文本编辑服务，结果把比尔·盖茨也弄糊涂了（但仅仅一分钟而已）：

> OLE 是我们在集成技术上进行的大投资。一次比尔·盖茨正在 Excel 的门厅走过。为了激发开发小组的士气，他有时会在开发部里转转，和每个特性开发组的负责人谈上一会。负责人就会展示一番他们设计的一些特性，然后比尔再作评价。当他看到 OLE 时，他们转到了利用 OLE 打开 Word 进行就地编辑。在起初一两分钟里,比尔没看出来。他发现 Excel 有些不正常。突然间他意识到了是在利用 Word 做就地编辑。所以在混合文件里利用构件工作会使集成度大大提高，我们想那对用户来说将会是件美妙的事情。

OLE 影响到的不仅包括应用软件产品，而且包括系统产品和特定的应用软件。比如，两位 OLE 2.0 的设计者花了 50%的时间和系统产品开发员进行交谈。ISV 想把 Office 用作他们产品的基础构件，所以 OLE 小组成员也咨询了他们；OLE 还向他们提供了一种很灵活的机制，以把那些能压缩他们的产品并和 Office 应用软件紧密集成的特殊构件或“服务项目”记录下来。

动态链接库（DLL）提供了另一种通用的共享技术，但它们不是一种专门的产品。一个 DLL 就是软件的一个“添加”条目，当软件运行时，操作系统能自动将之装入内存。然后，不同的应用软件程序就能把它作为一种“服务”进行分享。不同的应用软件能分享的 DLL 服务有拼写检错，特殊设备驱动器、外国语言界面以及许多其他的载入内存或持续进入的各种构件。DLL 的主要优势在于，在任何既定时间里他们都能最大限度地减少一个应用软件所需占用的内存量。

重复利用

微软在某一特定产品的一系列版本中——如从 1.0 版到 2.0 版，或在 PC 和 Macintosh 平台之间——重复利用了大量的代码（超过 50%）。由于一个开发员一次就能完成一个特性的编码，而非每一版或每个平台都要重新来一次，因而重复利用代码节约了成本、减少了精力消耗。例如，Excel4.0 在 Windows 和 Macintosh 平台间共享了 69.4%的代码，另有 15.4%用的是 Macintosh 的专有代码，15.1%用了 Windows 的专有代码（见表 6.3）。

微软还运用了许多方法以进行独立产品间的代码利用,比如共享构件(以 VBA 为例）、共享一个常用的 API，或干脆采取“拿来主义”的办法。采用了构件重复利用的方法后，开发员们所提供的诸如绘画和图表功能等独立性操作就能通过 OLE 或 DLLs 供各种产品共享。微软基础类（MFC）可以提供一整套能重复使用的 C++，是采用 API 方法的一个实例。戴夫·穆尔对此解释道：

表 6.3　Excel 4.0 的产品代码重复利用矩阵

源码类型	在 Windows 和 Macintosh 间的共享代码	Macintosh 的专有代码	Windows 的专有代码	按照源码类型的合计数	共享比例	占总代码的比例
C 代码	528287	56522	59710	644519	82.0%	75.7%
汇编代码	0	47793	44579	92372	0%	10.8%
Include，token等文件	62839	27152	24586	114577	54.8%	13.5%
按平台类型的合计数	591126	131467	128875	851468	69.4%	100%
占总代码的百分比 (851468) (851468)	69.4%	15.4%	15.1%	100%		

来源：乔恩·德·沃思：《微软 Excel 4.0 的事后分析报告》，6/8/92。

“人们使用这些基础类别作为他们编码的基础。其原因是既然他们的内存使用和功能管理都来自于标准库，有着通用结构，那就可以形成一套惯例程序，使每个应用软件有更多的机会被共享。但是我们并没有命令人们使用这些面向对象的程序编写技术……基础类是一个构件库，随我们的 VisualC++工具组一起出品。”克里斯·格雷姆认为，像 MFC 这样的通用 API 可能会以 5：1 的比例大幅度地减少系统和用户界面服务。

项目组还寻找机会从其他产品中复制代码，根据需要加以利用。例如在 Windows NT 的约 400 万行源码中，开发员对以前的系统，如 Windows3.1、DOS4.0 和 OS/2 以及语言组的重复利用率达到了 35%。佩拉佐利详细解释了 WindowsNT 中采用的“拿来主义式”重复利用方法：

> Windows[3.1]有相同的排错功能。事实上，我们偷了他们大量的代码。我们认为并非一切都要从零开始。那不是 NT——只要合适，我们就从 Windows 组中拿来代码……Windows 被分解出了一种所谓的 User 和 GDI[图形设备接口]的功能，其中 User 管理直线形的窗口，而 GDI 负责绘图功能。我们把他们所有的 User 码都拿来用上了，因为它用的也是 C 语言……所有的应用软件，如文件管理器和外壳程序也都用过来了。命令解释器本来是为 DOS 4.0 产品编写的，后来该产品发展成为 OS/2。我们把它也拿了过来，装在了 NT 上……语言组的成员给了我们 C 语言的运行库。我们可不在乎它们是从哪儿来的。

总的来说，微软有大量机会重复使用现存代码，因为开发员们已经为大

批产品编写了数量惊人的代码。为什么微软没能很经常地重复使用代码呢？一种可能的解释是微软实行自给自足的小组文化，消费者软件业的飞速变化则是另一种解释，虽然这也可能是不努力实现共享的一个借口。微软各组现在正在学习更多地分享他们的一切成果。即使如此，盖茨也承认项目小组仍未能尽可能多地利用代码：

> 我们用的是小型的开发小组——他们非常偏爱自给自足，所以他们不会过多地与其他小组共享代码。目前我们却拥有比其他公司更多的代码，因而我们能比其他公司共享更多代码。然而我们在这点上仍做得不够。当然我也能给出一些很成功的案例……全公司只有一套图示代码，可为什么要那么多文本处理代码。我可以作出解释何以至此以及为什么它并不像看上去那么糟糕。但如果我们在规格、结构设计上能够做到聪明行事的话，我们本可以避免大量的重复性工作。

第七章
向未来进军

前面各章在描述微软的战略及其具体原则时，我们已经详细阐述了公司的一些出众之处。在这一章中我们把这些长处再简要回顾一下，同时重点突出其中对其他公司有借鉴作用并且能够帮助微软在未来取得良好业绩的那些方面。然后讨论在我们看来属于微软致命的弱点的方面——包括事实的和潜在的弱点以及它们对其他公司提供的教训。最后在结束此书时我们将讨论微软面临的战略挑战，根据产品、市场、科技的发展趋势微软将如何实施它最基本的战略——向未来进军。

在这最后一章，我们的思想建立在以下几个对未来的基本假定上。首先，我们相信个人计算机软件在功能方面会变得更为丰富繁多，对微软这样的软件公司来说越来越难以生产。新产品将把原先在不同的应用程序、操作系统以及网络通信产品中的许多特点都集于一身。第二，与此同时，我们相信未来的个人计算机软件，从用户的角度看，将会变得更加简单和可靠。这将使成百万（甚至上亿）的新家庭消费者更加频繁地使用电脑，处理各种各样的日常工作事务。第三，我们认为像个人计算机软件这样的高科技市场变化得很快，以至于我们无法断言十年以后微软是否仍是世界上最具实力的软件公司。同样我们无法确信，微软将成功地保持其在当前市场上所处的地位，并且将其主导权扩展到新的差异很大的市场上，例如联机服务或多媒体出版方面。我们坚信微软不会失败，它将在目前已有的市场及其他新的市场上积极进取，而且它已经拥有了一大批现存的产品、顾客、标准、技术能力以及组织机构等资源，这些将使它在今后许多年内都屹立于世界上最强大的公司之列。

关键的优势

微软是为数不多的杰出公司之一，在这里，领导才能、战略、人才、文化和机会共同作用，创造了一个极其有效的组织。不管读者喜不喜欢微软的产品和公司行为，这一点都是不容置疑的。

公司本身

我们在此书中列出的七条战略及相应的七套原则反映了微软最基本的长处，它们使微软能够高效率地运作。我们将其总结如下：

首先，微软拥有出色的总裁和高级管理队伍。比尔·盖茨是一位了不起的领导者，而且还组织了一支有效的“智囊团”来支持他。微软的经理们了解他们的技术和市场，懂得怎样利用这些知识赚取金钱。他们和任何公司的最高管理队伍一样出色或者比他们更棒。他们还巧妙地根据需要调整机构，使微软能够在快速发展和扩张的市场中保持竞争的不败。

第二，微软拥有上千个精心挑选过的才华过人的雇员，他们以其才智、技能和商业头脑闻名，拥有勇于进取、敢于开拓的精神，并且愿意为了公司长时间地辛劳工作。他们来自于广阔的并且越来越多样化的背景，这就在公司内部形成了多样性。这些人有能力承担多方面的责任，包括确定他们工作所需要的技能，招聘并培训新人以及协调公司内部不同部门。

第三，微软表现出了高度有效和一致的竞争策略和组织目标。微软的雇

员们都清楚公司领导者不是单单为了技术而重视技术，也不认为在推出产品时要固守一定的规则和程序。比尔·盖茨和其他微软经理们很重视利润：他们对能带来利润的大众市场感兴趣，而且他们相信只有把有价值的东西提供给顾客才能获得最大的利润。盖茨和其他经理们似乎还知道应该在什么时候进入哪些新兴的或正在发展的市场，以及什么时候应该转移注意力。他们赶在技术被淘汰之前就更新技术。他们清楚地了解怎样不断培育已经形成的巨大的产品市场和顾客群。他们通过建立标准来维持销售额和利润，开发新的市场，而且他们利用标准建立者的地位取得优势。他们还能看到未来真正广阔的大众市场是以新的家庭消费者为中心的。

第四，微软用一种灵活的、渐进的方式来管理产品开发和完善组织机构。微软的产品开发系统使小组和个人可以在一个项目期间内改变说明和设计规格，并且针对具体消费者活动的特点通过渐进式的发展来完成产品建设过程。另外，微软的组织机构也同样表现出我们在它的产品和项目中所见到的灵活性和渐进性。我们已经看到微软公司可以比较容易地从一个市场转移到另一个市场。当技术和市场变化时，微软不断随之调动人员，更新组织，高效率地保证产品、项目和整个组织机构的发展。

第五，微软显然在产品开发和其他工作的过程中将高效率和平行工作的能力结合起来。虽然微软对产品定价相对较低，但公司利润依然很高，其中部分原因是由于微软利用它在很多领域内的规模和范围经济优势：它在开发和测试软件产品中已经积累了大量知识，并且它现在有很多软件组成部分和工具在不同项目中都是通用的。它生产出数量庞大的新产品，并且把它们轻而易举地输送到批发和零售商业渠道。公司从收入中为不断增长的研究和开发活动提供资金。不同小组依靠同一个中心可用性实验室，而且还有为寻求顾客支持的中心培训和工具开发活动。微软每天都接到成千上万个电话，这为它提供了宝贵的来自顾客的信息反馈。其他受规模经济影响的领域包括诸如产品包装，分发和广告等活动。同时，微软的产品开发和其他工作的过程采用将很多任务平行组织起来，使责任和职能交叉重叠，并把人员分在多部门组成的小型组里工作的方式，从而促进效率和灵活性的提高。

第六，微软人有一种自我批评、学习并提高的习惯。产品组研究过去的项目，寻找哪些做对了而哪些做错了，以及下次再碰到类似情况时该怎么做。和过去不同，现在微软还非常重视顾客的问题、抱怨和建议，把它们当成值得开采的金矿，进行细致的分析，并且迅速地把这些信息反馈传送到产品开发组去。微软越来越多地利用定量措施和计量方法来了解产品、项目以及顾客的反应。各个产品小组也不断寻找新的办法来共享技术和构件以减少重复，给顾客提供更加连贯一致的产品系列，并且使公司整体结合得更紧密、效率更高。

第七，微软人表现了对未来市场的不懈追求。他们并没有因为成功而变得自满，而是像理查德·巴斯所观察到的：“史蒂夫·鲍尔莫干劲十足地提醒我们，我们过于悠闲了，而竞争形势却十分紧迫，因为后面还有其他和我们一样饥饿的人们在追赶着我们。”我们还认为“向未来进军”——我们在后面将详细解释——是微软自 1975 年以来行为的准确写照。比尔·盖茨和保罗·艾伦最初是在他们寻求当时实际上并不存在的个人计算机编程语言的大众市场时表现这种态度的。从那以后，微软赶上了其他新兴或发展的大众市场的潮流，并且还创造了一些市场。盖茨和他的人员继续展望几年后的未来

状况，他们以此来评价诸如多媒体出版，交互式电视和其他为信息高速公路和新消费者设计的产品和服务之类的新想法。

产品开发

微软像任何公司一样，实际上类似于一个动态的人体系统。它之所以能够有效运行，是因为微软人将竞争所需的各种技术能力和市场知识结合了起来，并且把他们的主意付诸行动，虽然很难说其中任何一个因素比别的更重要。然而在这本书里，我们花了相当的时间谈论微软开发新产品时所采取的战略、原则，我们认为产品开发是微软所有事业的中心，公司的存亡和盛衰关键在于新产品，从 Altair BASIC 到 MS-DOS，Word，Excel，Windows，Office，Windows NT，Windows 95，the Microsoft Network 以及更多。

例如，在 90 年代假如微软未能推出 Word 和 Excel 的新版本，并把它们结合进 Office 套装软件的话，它的收益肯定会下滑。这些产品现在占收益和利润的一半左右。仅仅为了维持它在操作系统上的销售额，微软就必须从 MS-DOS 迈进到 Windows，现在它已经成功地开发了 Windows NT 的两个版本，并有初步迹象显示新产品 Windows 95 将在商业上获得巨大成功。微软还必须源源不断地增添有用功能来说服其成百万的现有顾客购买产品的新版本，虽然旧版本对于绝大多数人已经够用。为了保持在未来持续增长，微软计划创建种类繁多的结合先进的多媒体及网络通信技术的消费者产品。显然，微软面临的一个关键问题是公司是否能够继续增进其开发能力并且建立更大更复杂的软件产品和以软件为基础的信息服务。就像我们已经指出过的那样，微软还必须极大地简化这些中间产品，从而将它们成功地推销给世界上数十亿的新兴家庭消费者。

前面我们已经论述了微软一向在产品开发上显示出非凡的能力，而且公司不断精益求精地完善和提高这些能力，因此得到回报自然在情理之中。总之，我们认为微软的产品开发组织是公司的核心和希望，而且由于它蕴含着强大的力量，会继续获得巨大成功。

第一，产品开发组织有效地支持了微软的竞争战略，即为大众市场设计产品然后通过提高原有功能或增加新功能的方式来逐步改善此产品。微软还努力建设行业标准，然后利用这些行业标准在新产品上占据优势，各产品单位通过生产出一连串兼容的新产品和新版本来支持这个目标，使成百万消费者每次都支付额外的钱来更新换代。

第二，产品开发组织运作良好是因为它的过程和目的与微软的文化和目标高度一致。程序经理和开发者有很大的自由，通过对产品设计和用户反应进行重复试验来发展产品特性。强调由各个专家单独做出决定，但在小组中共担责任、共同工作，从而将官僚式控制减少到最低程度。软件行业中的老大哥们也许不屑地称之为“黑客”电脑文化，因为它缺乏对人们和项目到底干什么的更加严格的控制。但是由于有了我们描述过的同步和稳定技术——即每日构造，里程碑集成阶段，初步测试和内部发布等，就使得个人和小组不仅能够在一起工作，而且能相对独立地展开工作。

第三，就像公司的整个组织结构一样，微软的文化也同样促成了这种将效率（即分工结构）与灵活性完美结合的产品开发过程。各部门可以毫不费力地扩大它们的产品组合。每个人也可以自由地对他们所开发的产品做多次修改，并且变更或修改产品开发的过程和工具。然而，从整个公司来看，软

件开发和其他关键活动有一个相当明确并有规可循的过程。

灵活性在软件开发活动中尤其重要，因为在许多项目的初期很难推测产品最终会变成什么样以及用户对产品功能会有何反映。在做成套软件（与专用软件相对）时，公司还面临另一个问题：即类似于如何在写出畅销书后再接着写出能取悦于那群痴心读者的续集来。一个单一的高度结构化的开发过程不会一直有效，因为实际上并不存在写作畅销书的特定套路：不管这个过程怎么好，与最终成功关系更大的却是优秀的创意、实际写作者、市场时机选择以及广告和顾客支持。然而，公司能够做到的是建立合适的产品开发过程，使它既能提供正好满足团体对设计试验所要求的结构体系，还能给他们创造的产品带来时常迸射的意想不到的火花。这就要求微妙而有效的协调与交流。

第四，微软的产品开发方式容纳了好几种把吸收信息反馈和学习（特别是从消费者那里学习）直接引入开发过程的机制。我们可以从对以下二者的分析中看到这些机制：即让用户积极参与产品规划以及利用顾客支持数据来进行特性的选择和创意。除此之外，它们的身影比比皆是：例如程序经理和开发者广泛依赖于在可用性实验室里将特性原型化，测试者如何努力复制一个产品的功能用途，在重要新产品如 Windows NT 和 Windows 95 发布前要在顾客基地做大型 β 测试，以及开发者和测试者如何在他们推出新产品后都要接听顾客电话作为顾客支持活动之一。

第五，微软的同步和稳定过程方式使公司能够将创业早期松散结构的小团体方式升级，从而相对迅速且低成本地建立起复杂的软件系统。微软的技术也有一些局限，这些我们后面会谈到。虽然现在微软的许多产品和项目都相当庞大，他们却仍然继续像灵敏的小团体一样工作。我们相信，经过一些改进，微软能够继续提高它的经营运作水平，建立起未来时代的复杂软件系统。

同步和稳定过程取得的成就

在同步和稳定思想背后的各项原则达到了一个艰巨的目标：它们给迅速变化以至于经常处于混乱中的个人计算机软件开发世界增添了一些看起来很像样的秩序。这里没有什么神秘的魔术，这 里有的是各种具体的工具和技术，一些严格的规则以及掌握了高度熟练技术并且愿意遵守这些规则的人才。微软的整个开发过程帮助各个开发者经常与同一队伍中其他成员的工作保持同步，并且在构件的发展中通过渐进办法使产品稳定。就像我们在本书中始终指出的那样，有很多因素使同步和稳定方式显著优越于早期更严格的顺序式产品开发方式（见表 7.1）。

表 7.1 同步和稳定与顺序式开发的对比

同步和稳定	顺序式开发
产品开发和测试平行进行 想象性描述和不断发展变化的规范	循序完成的独立阶段 在构造产品前产生完整的“ 冻结 ” 的说明和详细的设计
特性构件按重要性排序分组并依序在 3 或 4 个里程碑子项目中构造	力图同时构造产品的所有部分
经常保持同步（每日构造）和中期稳定（里程碑）	在项目末尾进行一次晚期大型的集成及系统测试阶段
固定发布和出品日期并且有多次发布周期	在每个项目周期中都力求特性构件和产品的 “ 完美 ”
在开发过程中不断接收顾客信息反馈	主要在开发完成后接收顾客信息反馈作为对以后项目的指导借鉴
设计产品和过程以求达到使大团组像小团组一样工作为主要目的	以由不同功能部门的人员共同在一个大型组里工作为主要方式

第一，微软并不遵循一种顺序式的 “ 瀑布 ” 过程，这个变化趋势在软件业和其他行业中也开始变得越来越普遍。微软并不把产品开发和测试当作有先有后的两个单独阶段，因为如果这样做的话，一旦事情没有确切地按照规划进行，就需要翻来倒去地反复循环修正。相反，微软的团体是把产品开发和测试同步进行。这个过程就使每个人都充分发挥自己的想象力随时对设计、编码和测试进行攻关。它还有些类似于在其他一些覆盖许多活动及阶段的产业中出现的渐进开发和并行工程行为。

第二，微软在开始构造一个产品的各构件之前并不试图 “ 冻结 ” 一个完整的说明和详细的设计。微软让说明随项目进展而不断变化发展——增添或删除特性构件，对设计细节进行试验。因此完整的说明与其说是开发过程的投入，不如说是开发过程的产出。微软对产品只有想象性描述而没有实质的详细设计或文档阶段，编码就是他们的详细设计和文件，这又是一种不正规的 “ 黑客 ” 方式，但是我们认为微软拥有特别有效的机制使不确定性变化大体处于控制之中。

第三，微软并不企图同时构造一个产品的所有部分，即把一个详细的设计分成小块，然后把所有分出的模块或特性构件配发给不同的人和小组。相反，微软是把一个设计分解为特性构件，把所有特性构件按重要性排序并分组，然后通过 3 个或 4 个里程碑构造各个特性构件群。小组们在第一个里程碑里通常致力于最重要的特性构件群，在第二个里程碑里专门干次重要的特性构件群，以此类推。对于特性构件群之间的配合关系不那么强的产品，项目小组通常会在进度实在来不及的时候舍弃最后一个里程碑里的特性构件

群。

第四，微软并不试图在项目末尾通过一次晚期的大型集成和系统测试阶段来把产品的各个部分联成一个整体。那种落后的方式会在下列情况下发生：即如果一个项目平行地构造产品的所有部分，并且在开发过程中不能使各部分同步或把它们放在一起测试。反之，微软采用频繁地进行构造的方式，每天或每星期都使许多个人和小组的工作同步进行；而且应用里程碑子项目的概念，通过 3 或 4 个渐进活动来稳定特性构件的子群。这些做法类似于其他一些公司采用的“并行工程”和“渐进构造”。然而，我们认为微软优化了这种产品开发方式并使其制度化，因此显得格外出众。

第五，微软并不要求对项目开始时所提出的每个特性构件都完成并完善。相反，尤其是在应用软件产品中，微软设置了时间及人员的限制，然后制定出除去最严重错误的目标。各小组将会把他们在前次项目中不能完成的特性构件等到下一次产品发布时再添加进去，或者消除上次他们没有发现或无法消除的不太严重的错误。这样，微软就避免了一种常见的困境，即陷于永无止境的修改，添加特性及消除错误的死循环中。别的软件公司也采用多次发布周期，包括处在每年或经常进行“式样改进”行业中的公司。但是微软却在这种开发和营销方式上更上了一层楼。它甚至想出了年度软件式样改进的绝妙主意——从而也有了 Windows 95 和 Office95 这些名称。（当然，假如微软没能精确地测定出进度表的话，这种每年出新款式的策略也会起副作用。把年份加入名称之中也产生了在某一年内完成的额外压力。）

第六，微软并不等到它已经将产品完成并推向市场后再来收集和利用顾客信息反馈。而是在开发的整个过程中，不断地把顾客信息反馈结合进来。首先是在产品规划阶段对用户进行分析，然后继续在可用性实验室里对原型进行测试，再在即将发布前把版本送到β测试基地。不仅如此，微软还做出关于顾客向公司产品支持组织提出询问的每周详细报告，并把它随时送到产品开发组。这些信息对当时正在开发的特性构件和未来的产品设计都会产生影响。

第七，微软不让开发者旁若无人地编写软件程序。它也不是通过下面这种庞大团组的方式来构造软件：小组由设计者、开发者和测试者们组成，他们在相互独立的部门中顺序式地工作，在许多严格的步骤和文档要求之下将工作逐次移交给下一阶段。而恰恰相反，微软是在多功能小组里开发软件，将力量合理组织以使大团组像小团组一样工作。微软公司有人曾对我们说：“我们花费更多时间去弄明白怎样从小处着眼去想、去做，而不是好高骛远，一味求大。”

使大团组像小团组一样工作

微软的同步和稳定方式还为怎样组织大型团组——这个在许多公司和行业中都常见的问题——提供了可贵的榜样。这个常见问题部分的困难原因在于技术上和管理上的培训不足。大学里的自然科学和工程学系以及管理学院一般都不训练学生如何在大型团组里工作或怎样控制大型团组。大学里的工程项目几乎总是小型的，人们在此学习如何独立地或在小型组中工作。而许多公司的现实情况是，为了在相对较短的时间内构造复杂的产品，有必要成立大型团组。事实上确实如此；虽然说由技能高超人员组成的小型团组也许

是设计任何类型产品的最好方式，不管是计算机软件程序，汽车或是飞机。我们觉得微软和其他“年轻”的公司（特别是那些位于像个人计算机软件这种相对来说是新兴产业中的公司），对我们生活的世界提供了许多关于如何管理团体和创新方面的课程。表 7.2 和下列的段落将回到从前讨论过的战略和原则上来，从中剖析出微软用来提高其小团组开发方式的关键因素。

项目规模和范围限制

我们已经讨论过微软如何通过限制产品规模和范围来使项目小型化。这产生于以下将讨论的几种方法中。

明确而有限的产品想象蓝图　微软力求对每个项目都设定明确的

表 7.2 使大团组像小团组一样工作

- 项目规模和范围限制（明确而有限的产品想象蓝图，人员和时间限制）
- 可分解的产品结构（根据特性构件、子系统和对象来划分模块）
- 可分解的项目结构（特性构件小组和特性构件群、里程碑子项目）
- 小团体的构成和管理（许多小型多功能组，享有高度的自主权和责任）
- 一些“强制”协调和同步的规则（每日构造，“不要破坏构造”的“故障”规则，里程碑稳定化）
- 在各功能和小组内部以及跨功能、跨小组的良好交流（共担责任，同一基地，共同语言，非官僚文化）
- 产品及过程的灵活性以容纳未知因素（不断发展变化的规范，缓冲时间，发展变化的过程）

界限。程序经理与开发者和产品经理合作，并根据顾客支持数据，做出一个简要的想象性描述，从而为项目确立一个可实现的目标。彼得斯在他 1990 年的电视讲话中说：“你必须有一个明确的目标……这样才能帮助一大群人，一群由 100 人组成的队伍，朝一个共同方向前进，并且帮助他们决定做什么而不做什么。决定你所为之工作的产品不该成为什么样与决定它该成为什么样同等重要。”(1)微软发现与全新的产品相比，在产品的第二、第三或更晚版本中这种目标的明确性更加容易实现。

通常也可以根据“能否赚大钱”来对项目进行特性构件的取舍。如果一个特性“看起来会反响冷淡”，可能只有几个消费者，但却需要 6 个人花费开发的全部日程来完成，他们就会把它删去。

人员限制　我们已经注意到产品单位的构成限制了在任何一个项目中共同工作的人员数目。微软实际上是由小的开发中心组成的集合，每个开发中心通常不超过 300 或 400 人。每个开发中心代表了一支多功能的产品开发团体，由说明编制、开发、测试、用户培训及产品规划方面的专家组成。它们的规模比起微软早期是相对较大的，因为早期微软只有几十个雇员，项目只有 4 到 5 个人。但是若与对手软件公司经常采用的上千名或更多开发者相比，就是小型的了。微软还把其所有的人力集中在推出产品上，而不把建立作为“存货”的构件和技术或者将过程和产品存档作为重点。这种对推出产品的专注有着正负两方面的作用：例如，过程及产品档案的缺乏会迫使团体重新寻求

对公共问题的解决办法。但它同时也给了人们更多的时间来致力于将手头的产品更好地投入市场。

时间限制　现在项目对人们能够花费多少时间在某一特定的活动上设置了更加严格的界限。通常对于已有产品的新版本来说是在 12 到 24 个月之间。但全新产品的推出时间要长得多（例如 Windows NT 3.0 或 Microsoft Exchange）。我们已经谈过为什么操作系统比应用软件更难以在一个可预测的时间框架内构造完成。（因为操作系统需要广泛的测试来覆盖所有可能出现的用户使用情景，而且在项目超期时很少有可以删除特性或功能的机会。）但是，设置时间限制——至少是在内部，能帮助人们把他们的创造性集中于完成产品的一个可运行的版本，即使它并不“完美”或者还不适于正式投放到市场上去。项目在以后还会有时间做进一步的完善和优化。

因为项目将特性构件按重要性排序，并且在项目进行中始终集中于开发一个过得去的产品，这样应用软件项目在时间不够时就可以随时停下来，并且仍然推出一个可以面向市场的产品。过去微软的一些项目经常拖延上几年之久，而现在则显著不同了，许多应用软件产品的第二或更晚的版本在原先估计的出品日之后一两个月就推向市场了。然而对于全新产品或操作系统的主要改进来说，微软的成绩就差得多了。如果与其他项目中的构件（例如在 Office 最近几个版本的一些构件）有相互依赖关系，困难也同样会增加。

可分解的产品结构

产品结构在将大团组分解为小团组方面扮演了决定性角色，这也许比它在对项目所占用资源的总体进行限制中所起的作用更为重要。模块化在软件业和其他许多工程领域是常见而必要的，绝大部分公司也都通过特性构件，功能，或子系统来设计产品。微软在过去并没有充分重视建立一个高等级的与其产品中的源代码相分离的结构。然而，情况在发生变化，微软的开发小组（包括开发 OLE，Office 和 Cairo 及其他）越来越以可分解产品结构和共享构件的概念进行思考。微软的各小组还通过每日及每周构造过程以及里程碑稳定化来有效地协调并同步各个构件的开发。

根据特性和功能划分模块　我们已经说过微软怎样把应用软件产品分解为特性构件；操作系统不仅包括特性构件，还包括叫做功能构件或子系统的部分。由于这些构件中许多会发生相互作用，项目必须在某一时候把它们放在一起测试；小组可能独立地设计许多特性构件或功能构件，这样就可以把一个相对较大的项目拆散为一些小项目。

根据子系统和对象划分模块　微软逐步把特性及功能编排成动态链接库（DLL）子系统或对象链接及嵌入（OLE）构件，使多个产品可以共用。这些子系统和对象需要在初始规划和设计阶段进行广泛地协调来规定它们的技术细节和界面。采用共享的子系统和对象来设计产品，并且设计出可以共享的编码，需要在开发过程中进行许多次调整。一般来说，这些构件又进一步使项目可以细分为相对独立工作的小组，至少在开发阶段是如此。

可分解的项目结构

我们已经论述过微软如何依照产品结构来划分项目结构。这种做法不仅帮助各小组创造出设计方面符合逻辑且高效率的产品，而且造就了人员分组符合逻辑且高效率的项目组织结构。

特性和构件小组　项目包含几个特性小组或构件小组。每个小组集中构造组成产品设计的一个或几个特性或功能。因此人员分组的结构体现了产品的结构，而且保持小组的总数和各小组规模都较小，以形成一个更加紧密组合的产品。

特性（和构件）群　微软将特性按照它们的重要程度和技术上相互依赖程度进行优先排序，然后分为各组群。项目在一个时期只致力于一个组群，由各小组平行地建立最优先组群中的各个特性和功能构件。

里程碑子项目　特性和构件小组在行进到下一个里程碑之前对每个构件群都走过了开发（设计和编码），测试和稳定化这样一个全部的周期过程。结果是经理们不必再去控制那种庞大项目：例如在 18 或 24 个月内要平行地构造一大堆特性构件。他们只需从每日项目管理的角度出发，目标是在三个月的时间内建立一些特性构件。

里程碑可能看上去比那种同时开发所有构件然后只在项目末尾进行一次性集成并测试完成的方式要花费更多的日历时间。然而，即使在常规的公司中，软件构件也经常容易在项目进行中产生很大的变化而难以事先进行精确的规范。与原始规范的不同则使产品在后来难以再进行集成和测试，除非严格地控制各人和小组的构造。里程碑方式使任何规模的团组都能像小团组一样行动，能够改变设计或省略掉构件的细节。他们享有这样的自由是因为他们可以在一个项目中多次将产品的主要部分稳定化——而不是等到太多的变化已潜入产品中而使集成和稳定变得几乎不可能了。

小团组的构造和管理

根据将产品划分为特性（甚至将特性再划分为子特性）的原则，项目可以将产品任务分为各个部分，每个部分由一些人完全负责，而且通常能在几星期或几个月内完成。

小型多职能组　我们已经说过每个小组的核心组是如何由一个程序经理和 3 到 8 个开发者，再包括一个特性小组领队组成的。在核心组基础上扩展形成的小组还包括一个平行的与开发者数量相当的特性测试者队伍。

自主权和责任　克里斯·彼得斯曾说过“ 当你使组织退化，或是说将组织中每个人责任压得很低的时候，大团组是会奏效的。他们会工作得很好而且非常具有竞争力。(2)另一种办法是给经理们（例如小组长）更多的责任和权威来使他们严密指挥小组成员的工作。而与前两者恰恰相反，微软给予每个小组，以及每个人，相当大的自主权和责任。这就遵循了要雇佣能独立工作和学习的杰出人才的一条原则。自主权和责任使每个小组能够相对独立地工作，例如，个人和特性小组负排安排和维持他们自己的进度表。此外，他们做修改变化时很少受到规则的限制——个人和特性小组“ 拥有 ”他们的特性

群，项目还可适当根据要求裁剪过程和工具。

一些强制实行协调与同步的严格规则性

虽然微软的小组享有可观的自主权，项目在一些关键时候还要依赖严格的纪律来保证各小组能够协调他们的工作。

每日构造　几乎所有的项目都做其产品的每日构造。开发者可以随心所欲地爱什么时候工作就什么时候工作，以及愿意参与加入多少次产品构造就参与多少次。然而，当开发者登录他们对产品的工作时，他们必须在某一个特定时间前登录，通常在下午（2：00 或 5：00）。在此之后项目就产生了一个新的构造。受前任智囊团主管戴夫·马里茨之类人的激励，产品组像以色列军队一样严格地遵守构造纪律。每日构造的过程迫使各个单独的开发者及小组尽可能经常地使他们的工作同步。开发者把写入他们的代码拖延得越久，他们就越有可能与其他特性发生冲突而使构造失败。

不要破坏构造　小组发现代码中的问题后就得通过不断的测试来创造一个新的构造，这个过程中既有自动的也有手动的，并且给每个开发者都配备了“伙伴”测试者。一旦他们决定登录其工作，必须遵守的一个关键规则就是他们不能破坏在各特性构件及功能子系统之间的结构界面或相互依赖关系，或犯任何其他使构造失败的错误。导致破坏构造的开发者必须立即消除障碍，这些‘有罪’之人可能还必须负责总领第二天的构造或者支付巨额罚款。戴夫·汤普森说明了这条规则的严厉程度：“我们对此要求十分严格：你最好不要破坏它，你必须十分小心地进行编码。”

里程碑稳定化　在某些结构约束和经验型项目限制下，个人和小组可以自由地对新产品的构件设计进行变更或者修改其特性群。各组在结尾时几乎总是改变了他们的最后出品日。然而，在项目进展期间，经理们认为人们超过了中期里程碑的截止日期是不能被接受的。

在各技术专长和小组内部以及相互之间的交流

有好几个因素促进了在项目中的各技术专长和小组内部以及它们相互之间的进行良好的交流。

共担责任和任务　我们已经讨论过虽说有一些职能专门化和人员分工，微软是如何通过模糊这些区分来使人们共担责任和任务的。程序经理和产品经理共同起草产品的想象性描述；程序经理和开发者一起确定产品的特性构件；开发者和测试者共同测试代码；程序经理，开发者和测试者在新产品发布之后都要接听顾客支持电话，为顾客解答问题。

同一基地开发　主要的开发工作都在微软总部进行，一些收购活动除外。这就便于小组成员在面对面的会议中交流从而迅速解决问题（比尔·盖茨坚持这种会议交流，虽然他对电子邮件情有独钟）。

一个共同的语言　项目都依赖一个共同的开发语言（主要是 C 语言）。应用软件项目还采用微软内部的“匈牙利”命名法惯例。许多开发者认为匈牙利

命名法使他们更容易理解对方的代码，即使在没有单独设计文件的情况下。这一做法促进了代码共享和问题的解决。

开放的文化　微软的文化离松散联合起来的业余高手程序员的世界不是太远。这里的人们憎恨政治上的“争权夺利”以及官僚主义的规章和手续，不必要的公文档案，或过分正式的交流方法。结果是，个人和小组具有在他们认为重要的事情上的快速作战能力。

借助产品及过程的灵活性适应变化

对于一个公司来说，往往很有必要使产品和过程充分灵活以适应未预见的变化或容纳个人或小组可能采取的创造性行为。在经理们给予个人和小组独立行动的担负重大责任的工作环境中，以及身处科技和市场需求都迅速变化的产业中，尤其需要如此。

发展变化的规范　我们已经描述了产品经理和程序经理如何在项目开头做出一个想象性描述和特性优先排序表。一些组（例如 Excel）写出所有或绝大部分特性构件的长篇详细的功能描述。然而，各小组并不觉得被这些描述锁住了手脚，特性构件的规范和细节都可以随着项目的进程而发展变化。最终的特性一览表可能会有变化或增加了 20%—30%，这得依据开发过程中工作的进展状况，竞争对手在干什么，以及项目成员得到了何种信息反馈而定。

缓冲时间　微软的项目进度表中留出了一部分作为缓冲时间：在应用软件中大约是 20%，在系统项目和全新项目中达到了 50%。缓冲对于适应未预见的变化十分有用；这些变化可能来自于没想到的特性构件变化，难以排解的错误，或其他没想到的问题（它们似乎总会发生）。经理们把这些缓冲时间安排在介于每个里程碑之间及最后发布之前。麦克·梅普尔斯解释说项目需要弄明白是哪些因素总爱发生变化而扰乱进度。许多耽搁是来自于像他所称的“你所不知道的事情发生了。结果很大程度上小组对各个不同的项目都同样地盲目自信和缺乏了解。一旦你能弄明白是什么因素总在捣乱，则几乎只需例行加上一个常数。因为他们总是把需花费的时间低估了 x%。”

发展变化的过程　如果有更好的选择存在，微软的经理们从不要求人们拘泥于特定的做法或工具。若某人有怎样做某事的更好主意，并且能够论证其优越性，小组就可能在项目中或项目后更改他们的过程。克里斯·彼得斯对这种发展变化能力评论道：“我认为我们知道现行的最好做法是什么；而且当它们发生变化时，我们也随之变化。”

致命的弱点

就像任何公司一样，微软有很多事实上和潜在的弱点，了解和纠正这些弱点对微软和其他公司的经理都是十分有益的。潜在的弱点是相当多的，因为如果走入极端，我们在此书中描述过的每个策略和原则都可能随时间推移而由资产变成负债。

组织和管理公司

当我们对微软人如何组织和管理公司背后的各项原则做一回顾时，有四个忧虑便在脑海中产生了：

- 即使是一个无比杰出的总裁，他的注意力、知识和任期也是有限的。
- 主要提升技术人员可能造成优秀中层经理的缺乏。
- 重视“赚大钱”可能会挫伤真正富有创造性的人才的积极性。
- 在产品单位之间不断增加的相互依赖也许会削弱它们的市场定位聚焦度。

微软可能要为此付出代价的第一点可能是微软对比尔·盖茨作为公司领导的依赖。说这种依赖在微软的发家史上只是“弱点”是不充分的。盖茨不仅一直有敏锐的直觉，他还给微软带来了令人艳羡的大批技术专家和经理。他没有重蹈许多公司创始人的覆辙，即在公司已经扩张到组织庞大且多样化而无法由一个人掌握时，还力图对其保持太多控制。但是，盖茨仍然在战略性决定以及产品战略的关键要素上保持控制，而且担任在各产品组之间协调资源的角色。对于现在拥有如此广泛的科技、涉及多种行业并且继续膨胀的微软公司来说，盖茨（或其他任何人）是否能够继续对它保持透彻深入的了解？我们将拭目以待。

盖茨最初成功地创建了微软，是因为他比其他同代人更早地看到了未来的方向，或者说他比别人更加自信而且先发制人。他的公司在大众市场的火爆浪潮中平步青云，从较简单的编程语言到操作系统到桌面应用软件等等。然而，在微软所新从事的事业中，盖茨需要考虑许多更加复杂的情况：与 70 年代和 80 年代初期微软刚刚建立地位时相比，现在有了更多的更大的竞争者。盖茨认为消费者软件和信息高速公路将是未来增长的巨大领域，这无疑是正确的。但却不知盖茨和他的高级管理队伍是否懂得如何比其他任何人都更好地利用这些新科技能力和商业机会。这并不意味着我们会打赌说盖茨和微软在这些新兴领域中会输掉。尤其是微软在这些新兴领域中已有了出色表现后，例如 Microsoft Network，我们不会做出这种预言，但是，微软在除桌面软件之外所占据的优势强弱，仍是一个值得探讨的问题。

一个相关的忧虑是盖茨自身——他的注意力范围以及他的任期和以后的接班人。他仍然是微软无可争议的最高领导，常常是需要通过他的整个人格力量来使不同的微软团组顺利合作。盖茨曾表示他打算至少在以后十年内继续领导微软，然后在公司担任一个稍微次要的角色。[3] 但当他离开总裁的位置后，他很可能不会走得太远，肯定还留在董事会，就像保罗·艾伦一样。然而，在最高层就将出现一个难以填补的真空。最近像内森·梅尔沃尔德，皮特·赫金斯，保罗·马罗茨，克里斯·彼得斯这些年轻高级经理的成才之路表明微软有能力培养出有才干的高级经理，但是至今没有谁看起来能够继承盖茨的职位。

一个更加紧迫的危险是盖茨已经使自己和微软都过分超负荷运转，同时进行太多领域的工作和太多的个人投资，以致难以对所有事情都集中其精力。有一个资深的微软观察家甚至怀疑对 Intuit 公司的兼并失败是否显示出微软开始变得软弱了。[4] 盖茨和他的微软曾经决心与个人计算机软件业的每位成员进行较量，做任何必要的事以成为赢家。他们用 DOS 对 CP/M 的制造者 Digital Research 公司形成挑战，他们用 Multiplan 然后又用 Excel 超过了

Lotus1-2-3；他们用 Word 压倒了 WordPerfect；他们通过建立 Windows 打败了 Apple 计算机公司和 Macintosh；他们用 WindowsNT 迎击了 Novell 公司的 NetWare，Unit 各种版本以及 IBM 的 OS/2；他们积极推销 Windows 95 和 Microsoft Network，从而对各个联机网络提供者构成直接冲击。而与前面的一往无前相反，微软开发出 Money 来挑战 Intuit 公司的 Quicken，然后又试图花 20 亿美元来收购该公司——这个决定使竞争者和反托拉斯管制者大为失色。在遇到美国司法部的反对之后，微软的经理决定不再花大量钱财和精力来完成此收购。但是我们感到奇怪的是盖茨为什么不坚持让他的人员更加努力地使 Money 成为一个成功的产品，而不是觊觎别的公司的产品。

第二，在盖茨和他的高级经理队伍之下，微软的中层管理队伍比较薄弱。我们已经说过微软喜欢将开发者提升到经理的位置，主要依据他们的技术能力及成就而不是他们的管理才能。而且微软在 80 年代后期和 90 年代早期增长如此飞速以至于非常年轻的开发者也承担了极其重大的责任，例如经营几百万美元甚至几十亿美元的事业。有一些开发者令人惊叹的成功适应了这种经理角色转换的挑战，但他们或许只是特例。这种短缺是一个严重的问题，因为几乎在任何公司里，中层经理都是使整个组织运作的中坚力量。

微软处理这个问题的主要战略包括三方面：（1）增长得略微缓慢一点，给人们更多的时间在他们现行的责任水平上积累经验。（2）如果有机会，可以继续在需要的时候从其他公司雇佣经验丰富的经理。（3）更注意培养中层经理。微软现在开展更多的会议，“休假会”，专题讨论工作以及培训课来使经理们讨论共同问题，交换经验和信息并接受更正式的管理教育。微软还任命职能主管来促进跨产品组的相互学习。因此我们感觉微软的高层总裁们对这个严重的弱点是有足够认识的。而且看起来这些措施再加上平稳的增长，能够在缓解中层经理的缺乏方面起很大作用。

第三个潜在的弱点是微软一直以来都雇佣“能赚大钱”类型的人物，聘用盖茨眼中的杰出人才——即懂技术，会学习，又有敏锐的商业头脑的聪明年轻人，这显然对公司帮助很大。然而微软的雇员们历来一般把他们的创造性都集中在开发有广阔市场销路的产品特性上。绝大部分特性由渐进式逐步创新组成，很多产品都是步其他公司的后尘。增设的消费者部门和研究部门使微软的能力扩展得更为全面，人才库也得以多元化，但微软很可能在基本业务领域还需要更有发明创造天赋的雇员来使这些领域充满新鲜活力并不断向前推进。

第四个弱点是由于微软大力促进各产品组之间进行共享而导致相互依赖程度加大。从战略上和技术上看，进行更多的共享和集成是很有必要的。问题是微软这种基本上根据产品而划分的组织结构已深深地铭刻在了公司文化之中，而且更重要的是，这种组织结构有自己的一套优点。在过去，它使得不同的业务单位能够集中精力于某一竞争者和某一产品，而且他们拥有构造和推出自己产品所需的一切资源。而现在情况已经不再如此了。再加上同时管理 2 或 3 个相互依赖的项目远远比每次管理一个项目要复杂得多，对这个微软还需进一步学习掌握。更大更多的相互依赖可能导致更多的产品拖延和产品妥协，至少在微软使它所有关键队组协调步伐之前是如此。许多产品的良好性能和及时性都被他们最薄弱的构件拖了后腿。

管理人员和技能

微软对人员和技能进行管理的原则暴露了四个方面的隐患：

- 太多的责任重叠可能导致混乱和重复劳动。
- 边干边学，边干边教对许多复杂产品来说可能是不够的。
- 对“官僚主义”的憎恶可能产生太多的重复创造和不断试错。
- 因为绝大部分知识是贮藏在头脑之中而未被诉诸文字的，保留关键雇员就具有决定性的重要程度。

第一个弱点在于微软所裁定的各技术专长之间的重叠程度。分出这些不同的技术专长对于在软件开发和测试这种高度专业化领域中培养专家是必要的。然而，有些技术专长已经变得更加难以定义，例如程序管理一类的队组就与产品管理和软件开发队组都有相当部分的重叠。各专长过于正规化的危险在于它可能助长对人员和技能不健康的分隔，从而无助于微软克服其在人才方面的缺乏。然而，太多的任务和责任重叠同样也有危险，在于它可能导致混乱和重复劳动或目的相反的劳动。当微软的程序经理和开发者在特性优先排序上不能取得一致，而且程序管理领导和开发领导都没有做决定的权威时，这种混乱就发生了。结果是，人们在对产品于市场上取得成功无补的特性上也继续工作，或者人们曾致力于的特性后来竟被项目所抛弃。盖茨和其他微软经理相比而言似乎宁愿承受后者：即使有一些由任务和责任重叠所导致的混乱和多余，也不要一个过于分隔化的组织结构。而且，盖茨和其他高级技术人员在必要时候可以出面解决冲突，虽然说这种出于万不得已。由他们给思想独立的雇员和经理下达解决办法的事例还很罕见。

第二个弱点在于微软指导和培训新雇员的方式。公司避免了将此过程过分正式化，并停办了以前曾为新来开发者设置的长期“大学”。理论上讲，边干边教，由经验更丰富人员担任导师进行指导，是训练新人的极佳方法。它使经验丰富的雇员把只可意会，不可言传的知识表达出来。然而在实践中，这种方式既费时，又需要有足够的既有经验又是好导师的人员。所以，这种系统在微软最快速的增长时期和期限紧迫的项目中都曾失效过。假如微软的增长减慢，尤其是在雇佣开发者方面，这个问题的严重性将会缓解。此外，微软和其他采取这种方式的公司需要为导师提供一些准则，并为指导人员提供适时的进度表。

第三个弱点是在微软里大概有不少多余的重新创造和不断试错。公司人员讨厌正式或书面的规则和手续。这种态度是好的，因为它使官僚主义达到最小，并且帮助微软随着发展进程改变其组织结构或采纳更好的新做法。但是这也需要有一个限度。例如，事后分析档案就表明各项目经常地重复犯共同错误而没有从其他项目中充分学到应该接受的东西。而且各产品单位也始终没有一贯的接受建议的适当过程。同样，产品现在变得庞大而复杂，因此功能分工——编写说明和手册，开发和测试代码——也相应地变得复杂。然而，人们大部分的学习还是通过口头交流进行的，从一个人到另一个人，或从一个组到另一个组。

第四，由于如此多的知识是在头脑中贮存，是默契的，通过口头交流而未被记录下来，保留雇员就具有决定性作用。看上去人们在微软呆过几年后，保留率是较高的。然而，测试者对测试感到腻烦，开发者和程序经理也厌倦了在自己被迫设置的极端严格期限下改变说明和代码。事业的轨道，提升的机会，以及有趣的技术挑战都帮助人们保持冲劲，但是许多人后来精疲力竭

了，从而离开公司或长期度假。微软的测试和每日构造过程尤其严重地依赖极聪明的人来迅速找到并消除棘手的缺陷。开发者和测试者已经有一些相当不错的自动测试和检查清单，能够自动获得一些过程和产品的知识，然而，若是能更加始终一贯地记录并存档产品的设计和执行结构，它们就可以用来获得更多的知识。当然，这又表现了在使档案不断更新所需花费的时间与当时改变产品的难易程度之间的矛盾与权衡。不管怎么样，随着产品变得更加功能多样而难以构造，公司在这种默契知识和雇员流失不利的影响面前将会显得更加软弱无力。

用产品和标准开展竞争

在下一段中我们讨论几个对微软形成挑战的战略性问题。这里我们想突出微软的产品组合和竞争策略方面的四个潜在弱点：

·从渐进式创新难以前进到根本的激进式创新或发明。

·在科技导向与消费者及内容导向的矛盾之间难以寻找均衡，但却很有必要。

·内容导向的产品国际化过于麻烦。

·一个公司若多样化程度高，在进入很多市场后，即使这些市场联系紧密，也会损失集中度。

我们发现的第一个问题是微软一直主要依赖渐进式创新而开展竞争。偶尔它把这些创新集合起来，与新科技相结合，然后推出新产品，使老产品过时。 Windows 3.1、Windows NT 及 Windows95 与 MS－DOS 的任一版本相比都显得是根本的创新。然而作为其基础的科学技术以及产品开发和导入的方式，仍然主要表现了渐进式创新。甚至 Windows 95，虽然作为一个产品获得了巨大成功，却与有 10 年历史的 Macintosh 操作系统相类似。Windows 95 还太多建筑在 Windows 3.1 的基石之上，系统中仍有 MS－DOS 代码存在，并且从 Windows NT 中借用了一些特性构件。在过去的几年里，微软增设了高级消费者产品、联机网络和研究这三个部分，并且建立了几十个合伙企业，也收购了不少企业。我们相信所有这些积极动作都极大地扩展了微软的能力。然而，公司仍然可能在与更专业化公司的竞争中失利，因为这些公司在发明新科技或引入适应新市场的创新产品方面做得更好。这些公司包括 Intuit 公司，它拥有个人金融管理软件 Quicken；Novell 公司，它拥有办公室网络的 NetWare；和 Lotus 公司，它拥有办公室组合软件 NetWare。

第二个弱点是微软或许应该以消费者或内容为导向。微软在高阶层职位中有许多非程序员，成千的市场和销售人员，上百的程序经理和产品规划者，他们都努力担当顾客的代言人。除此之外，高级消费者系统和消费者系统这两个部门也在帮助微软将更多的资源和注意力转移到最大的大众市场——新家庭使用者上去。但是，虽然比尔·盖茨和其他高级人员拥有关于顾客的极其大量的数据，他们仍然根据微软所卖的软件来认识这个世界的大部分。应用软件开发集团则喜欢用他们所构造的特性和产品的眼光来看待世界。

这并不总是坏事。创造软件科技、具体特性及产品的能力使微软得以进入各种各样的新市场中，并至少提供了某些实在的东西。然而与此同时，驱动未来家庭消费者产品和信息高速公路服务销售的很可能是其他因素：低价位、易于使用和易于获得。微软以及盖茨本人持续不断地添加娱乐、教育和

信息方面的内容，公司正在学习如何使产品更加容易使用，而且它经常降价，例如在 Office 上已见到的。这些从战略上讲都是应该做的事。然而，微软在内容和新消费者产品方面缺乏实力背景。微软还面临更大、更强竞争者猛烈的潜在挑战，从电信公司到电子游戏软件生产者到书籍出版商和电影制造商。（电子游戏硬件和软件生产者是另一股竞争来源，虽然实际上他们也可能为微软的软件起到扩大市场作用。）鉴于现在流入微软的顾客信息是如此大量而稠密，并且目前在公司工作的人如此多种多样，我们认为微软的主流会成功地进行这次转变。然而转变将是坎坷不平的，其他公司在这些新市场里可能会比微软更为成功。

第三，我们认为当产品变得更加内容导向而非特性导向时，建立产品的外国语言版本所需的边际花费将会大大增加。因为微软零售收益的 70%左右来自海外，这就成为一个至关重要的问题。电影，教育材料，金融数据，新闻以及新产品所提供的其他类型内容，一般都倾向与特定的国家相联系，因此可能需要对每个新的国际产品版本进行相应改变。在目前的微软产品中，开发者可以仅仅通过把带有英语标签（在对话框按钮，菜单选项及其他之中）的电子文件替换成相应的外国语言就可以生产出一个新的国际版本。然而在多媒体产品或信息数据库中，这种技术就难以驾驭了。为了有效地解决这个问题，微软很可能需要给海外子公司更多的财务和技术上的责任，通过放权让它们能够为当地市场开发产品并使之适应当地需求。这又将进一步使决策和产品开发管理复杂化。

第四是过于多样化而产生市场集中度下降的问题。微软已经有 200 个左右的产品，还有几十个正在开发之中，从为儿童设计的游戏一直到为电话和有线电视公司制作的视频服务器系统。这包罗了如此广泛的产品和市场，显然微软无法对所有这些完全不同的消费者给予同等的注意力。有一种真实的危险存在着，就是微软将失去它的市场集中度和构造它最核心产品——桌面操作系统和应用软件——的能力。“赚大钱”的哲学帮助人们集中精力于最大、最有利可图的市场上。然而，这不是以顾客为核心，也不是以科技为核心的集中，这是围绕钱的集中。我们已经看到微软在维持其最强大市场之一的 Macintosh 上遇到了难题。Windows 产品现在占据了主导地位，并且微软将 Macintosh 各版本建立在主要为 Windows 设计的核心代码基础上。如果有远见有规划，这种方式会工作得很好，就像在 Excel 中一样，然而如果做得不好，反而会使顾客走远，就像在 Macintosh Word 6.0 中一样。我们希望微软即使仅仅是从自尊心出发，也应该改进 Word 和其他 Macintosh 应用软件，这个事例可能指出了微软在对小（但很重要）市场的注意方面所面临的问题。

确定产品和开发过程

我们看到四个值得担心的方面——这四个方面都是许多公司共同的问题，不仅仅在软件领域内——即微软定义产品和组织其开发的各项原则中的问题：

- 以特性为导向的开发导致特性扩散，并且忽视了产品结构的重要性。
- 产品概念和特性不仅应考虑到用户活动，还要考虑到用户行为表现。
- 让说明和特性发生太多的变化妨碍了有效的开发和项目管理。
- 产品单位之间日益增长的相互依赖使项目管理更加复杂化。

第一个问题是微软的特性导向方式容易导致产品包含用户实际并不需要

的特性，（或在产品升级的情况下，用户并不需要那么新的特性）。这问题在系统产品中要好一点，虽然，甚至 Windows95 也有一大堆的新功能，其开发、测试和调试花费了大大超出原先计划的时间。[5] 同时，对特性的全神贯注促使项目低估其根本的产品结构的重要性。没有一个优良的结构来确定构件之间如何相互作用，则小组所添加或改变的所有不同特性在进行开发、重整、测试和调试时就会变得异常困难和耗时。就像特性一样，结构在工作中，也需要定期更新而使小组在扩展产品功能时更加顺手。

有一些微软的小组已经认识到他们必须腾出一些时间来优化产品结构，并且建立了事实上的“里程碑 0”。在工作中，由一个小组致力于主要特性和结构的变化，同时有一个平行小组在较短的项目中致力于略次要的特性。即使这样，添加新特性和迅速完成产品的巨大压力还是把时间从完善并发展产品的结构那里夺走了。而且，微软主要是依靠开发者在编写代码时把功能分解为各个构件，而没有一个明显的高层次设计阶段来专门做这件事。特性的扩散还使产品对内存和磁盘空间的要求不必要地加大，而项目在一个产品版本到下一个版本时常常要改变一半的编码，并带来缺陷和更多的测试要求。如果在每个项目早期就能更加重视完善现存的结构和特性设计，则他们要改变或重整的代码很可能会变少。对特性的测试和调试也会更简单，而且长期的产品结构从开发者和用户的角度来看都将有一个更加流畅一致的构成。

第二个弱点在于微软全神贯注于用户活动，而不是注意用户行为。以实际活动为基础的规划提供了一种有效的技术，来分析人们如何书写文件、阐释预算，如何从事其他“创造性定向”的活动，把上述种种绘制成图形特性。比起那些让工程师们闭门造车般勾勒并不符合人们真实行动的产品定义和特性的作法，这种定义产品的方法更具优势。但由于这种聚焦于活动的作法没有发现计算机支持下的合作工作的需要，微软没能推出像 Lotus Notes 一样的组合件产品，直到它在市场上看到了该产品。而且，为了更加深入地进驻家庭消费者市场，微软和其他公司需要在深层次上研究消费者定向的行为模式——例如，了解在诸如娱乐、教育、通信和信息或者新闻检索这些领域中，人们究竟想要什么类型的产品和服务。然后他们需要介绍完整的符合所描绘的行为模式的产品（也要兼具个性特性），而这要比测度书写文件之类的具体活动难得多了。微软偶尔使用过的脉络相关调查技术在分析用户行为时多少会有些帮助，尽管它可能太笼统、太费时，以至于无法经常性地使用。

第三个问题是具体化分工当中过多的演进和变化。这种具体化分工机制对于决定人事需要及人事安排是十分关键的。允许这种机制继续发展并使计划仍旧受到控制的能力，是微软发展过程中的一个重要力量。然而，在实际应用时却有局限，有些项目超出了这些限制范围。正如人们在事后检讨报告中所抱怨的，具体化分工可能会在项目执行过程中变得过分杂乱无章，或是改变得太厉害。在这些情形下，项目管理者连同独立开发员、检测人员，在估计构造、检测和传递产品所需要的时间长短及人员多少时，常常遇到麻烦。（类似地，如果产品结构诠释得太糟糕或者变得陈旧过时，那么项目将会低估潜在的特性互动、臭虫的最终数目以及可能需要的测试和返工次数。）我们并不想说这种平衡易于达到。因为管理者想要具体化分工有所演变发展，项目成员或许只是不得不忍受一种日程安排的高度的不确定性，忍受返工和测试。他们也可以尝试多做些先期设计工作、典型化工作以及可用性实验室

测试，以此作为规划阶段的一部分，而非作为开发的组成部分，或者他们也可以为真正的新产品或大幅度变动的产品安排更多的缓冲时间。

第四个存在的问题是项目之间的相互依赖使项目管理大为复杂化。在一定时间管理一个软件项目已经十分棘手，而当具体化分工和特性在开发过程中频繁变动时，管理起来尤其困难。当一个项目依靠其他项目传递必不可少的构件时，人力估算和安排的复杂程度会急剧上升。同时，想要通过经常性构造的同步化和里程碑稳固化来找准问题也会变得更困难，除非每个互相依赖的小组都能按部就班，为经常性构造向其他小组传递构件。微软已在限期上有了麻烦，特别是在系统产品方面。通过设置更多的缓冲时间来控制不确定性是可能的，然而，这造成了更长的开发期。还有这样一种可能，即许多人将无所事事地闲坐一旁，等候某个落后的小组完成它的构件。作为兼并和海外本土化的结果，更多的小组在微软总部之外工作，这样相互依存也许会进一步复杂化。

不过，上述问题都有解决的办法。产品需要在构件当中具有非常清楚的结构和界面。相互依赖的小组需要遵循一种井然有序的开发过程（换句话说，他们不能自行改变具体化分工的关键部分）并且在说明文件中充分详细地记录基础构件上的全部特性依赖关系。这样他们必须小心谨慎地合作，并不断回顾整个相互依赖的项目的进展情况。至少为了某些构件和项目，微软已经采用了这些实践形式。

开发和推出产品

我们能够识别出三个主要缺陷（在其他标题下，我们尚未触及过），这些缺陷和微软开发以及推出产品的原则有关：

- 运用像设计回顾这样的技术，依靠“构造性”质量防范误差，而不是靠“测试性”质量来避免误差。
- 测试应当强调典型用户情景以及错误发现。
- 有效质量和项目控制需要数量矩阵以及可靠的历史数据。

首先，目前在管理和工程技术圈子里，再来谈论所谓“构造性质量要比测试性质量优越”已属老生常谈。在软件方面，这句话好说不好做。无论一家公司如何谨慎地构思和开发一种软件产品，永远存在着似乎只有测试才可能发现潜在错误的问题。在错误尚未有机会根植于代码中并和其他代码纠缠在一起之前就应当避免之。然而，微软主要依赖于它的测试过程（自动式快速检测、测试员进行的私人版本测试、日常构造之上的自动测试、里程碑构造检测，不一而足）来发现错误。项目同样允许特性发生改变或演化，一项导致了大量错误的政策可能是由于对特性的相互影响几乎缺乏了解，即使正确理解了每个编码本身的个性特性。

微软也从事相对较少的关于具体化分工、设计或者编码的正式或集中的回顾检查。新成员通常要由指导人员或小组负责人来检查他们的编码，开发员也常常要求另外某个人来核查他们的编码。关键的特性或共用型构件通常要经过详细的复查。但是微软很有可能避免一些返工和测试，不止通过更多关注特性间的相互作用，而且还要靠更加集中深入地回顾具体化分工及编码。当然，如果具体化分工和编码在一个项目执行期间变动得太过显著——在微软就是如此——那么用在回顾工作上的某些时间就会被白白浪费掉。

这个问题没有简单的解决办法。不过，数目众多的企业组织（如 IBM、

TRW 以及喷射推进实验室）已经出版其研究发现，支持了设计和编码回顾的有效性，这里包括了被称为“检视”的集中性编码回顾。他们已经发现，在开发员将问题一并装入产品之前，这些回顾有利于查出和明确问题。回顾同样有助于在一个连续的基础上优化产品结构和特性设计，并且对下层开发员还有传授有关设计、实施技术以及产品细节方面的知识技能的好处。

第二是微软（和其他软件公司）仍旧需要做的“用户测试范围”的问题。对 Windows 95 而言，就像前文讨论过的，微软发起了一次前所未有的β检测，该检测涉及到足有 40 万个消费者使用现场，以检查产品的测试版。这类信息来得太迟，以至于无法促成产品上的重大变革。然而，在查出那些典型用户可能遇到的以及硬件及应用程序的不同组合交织在一起的问题上，它还是有用的。微软的室内测试员可能从不希望在一段合理的时间内、花费可以接受的成本来复制这些不同的“用户情景”；此次声势浩大的 Windows95β测试努力是公司（以及行业）里的一个例外。微软已经碰到过在 MS-DOS 6.0 和 Macintosh Word 6.0 此类产品当中出现过的问题，而这些问题如果通过对消费者以及在不同的用户情景中进行更早的和更集中的测试，也许早已经被查明了。部分问题来自微软内部，微软在进行商业式产品投放之前，有时过于依赖它自己的非典型雇员来提供关于产品的信息反馈。想要进入真正意义上的大规模的非专业消费者市场的公司，有必要以一种尽可能接近或类似普通人将会如何使用那些产品的方式来检测他们的全部产品。

第三个弱点是微软拥有的、可用于便利日程安排、人事估计、质量分析以及产品推出决定的有限的数量矩阵和历史数据。日本的日立公司具有可追溯到本世纪 60 年代末的项目数据；IBM、NEC、东芝和富士通则拥有可回溯到本世纪 70 年代初的数据库。[6]微软自 1975 年起出现在商界，它的许多项目都收集了大量数据，但是它仍有待于建立一套丰富的、由所有项目共同促成的通用矩阵和中心数据库。随着时间的推移，我们预计矩阵和历史数据会变得对微软更加重要。当更多的项目要依靠别的项目来提供构件时，当消费者要求更高水平的可靠性以及传递日期的更可预见性时，这种情况就出现了。

作为一个企业整体来学习

微软在努力学习过去的产品开发经验，以及在不同的产品组织间分享更多的过程知识和构件方面，已取得了显著进步。虽然如此，微软人感到公司还能做得更多。我们选择以下三方面作为关注的重点：

- 在事后检讨中记录下问题，但是不进行深入分析或实施具体解决办法。
- 公司的松散的组织结构可能会在小组间分享知识技能方面产生麻烦。
- 不是太少就是太多的矩阵和控制带来了问题。

首先，项目的事后检讨表明微软人在自我批评和指出问题上做得很好，并且在提议解决办法方面做得相当出色。但是他们却不擅长进行基础问题的深入分析或贯彻解决办法，以确保小组不再轻易地重复错误。问题的一部分在于微软对待官僚作风的传统的轻视态度；微软只有相对很少的正规培训计划或书面的过程文件来指导人们的行动，也缺少中心化的质量保证小组。微软正在加强它的培训努力和参与产品开发和测试的职能指导人员的力量，但是最佳解决方法是使得上层项目人员更严肃地对待这类基础问题的解决。我们已经看到了变化，但仍然没有大规模的文化变迁。

其次，微软只有有限的，在小组间分享过程知识的正规机制。和第一个

问题一样，这也是非官僚主义的公司文化的一部分，它同时具有积极和消极两种结果。在类似测试这样的小范围的技术和专业当中，分享共用似乎进行得最好，此处贸易工具和技能很明显是有用的。开发员和程序管理人员已经比以前更不情愿修改过程以及分享他们的知识技术。这里的某些问题就一般来说，是设计工作的本质，具体来看，则是软件开发的本性：它是部分的艺术，部分的科学，和部分的工程技术。职能指导员试图鼓励学习，并且经常地，他们发现自己并不比那些组织几顿午餐或“休假会”的报酬不错的组织人员干得更多。他们正在编写着指导准则和最新过程文件，例如如何进行项目管理、日常构造或者测试。然而，这些也许会被证实要比那些午饭缺少价值，甚至是反生产的，如果人们对其作出消极反应。我们推断微软仍旧正在努力发现一种在促进知识分享方面的不正规性和正规性之间的平衡。

再次是一个有关矩阵和控制的平衡问题。微软项目常常伴随着数目很少的数量产品和过程测度。它们可以被使用得多些，但是公司也可能走极端。例如，可能存在这样一种提议，建议用职能辅导员负责的 250 项调查问卷和听证来代替事后检讨。这样做可能会带来一定程度的，要由少数小组承受的官僚主义，同时或许还将破坏微软已经成功地创立的严肃认真地——甚至幽默地——在事后检讨中评价过去项目的传统。我们得出结论，微软仍然正在努力地寻找什么需要测度和控制，什么不需要这二者间的平衡。同时，提高项目管理和决策制定必然需要比微软当前使用的更为全面广泛的矩阵和控制。

战略挑战

除了公司内部的弱点之外，微软还面临着若干战略挑战：

- 同一家公司统治多个技术时代，这是非常罕见的。
- 新的大规模市场可能需要新技能以及不同的竞争环境。
- 合伙公司也许要求分得财务馅饼上的更大一块。
- 如果一家公司变得太强大，反垄断的考虑或许要限制其成功的实践活动。
- 竞争对手们能够形成同盟，来同占统治地位的公司的实力相抗衡。
- 其他公司也能创造和共享构件。

首先，历史于微软不利。占据 70—80%市场份额的公司（微软之所以拥有如此高的市场占有率,是由于 PC 操作系统和 Office 应用成套设备这两组明星产品）通常只有一条路可走，即下坡路。出于各种原因，很少有统治了一代产品的公司能够继续统治下一代产品，这要看产品更新换代的程度如何。[7] 占据优势的公司常常会变得如此自负、傲慢，或是紧紧依附于其现有投资，以至于它们会在消费者需求或技术所发生的重大变革面前，在来自于更灵活的竞争对手的威胁压力面前变得非常脆弱，不堪一击。特别地，建立在实际技术标准之上的市场地位看上去似乎不太可能在一个飞速发展的行业中保持多年。

计算机行业中有众多的公司——IBM，DEC，Wang—先后丧失了它们的市场优势地位。我们同样能够在许多知名大公司的历史上看到这种现象，例如福特、通用汽车、施乐和柯达，它们都曾一度接近行业垄断地位。认识到这一点可以帮助我们解释为什么微软会在 20 世纪 80 年代如此强有力地支持图

形用户界面，它同样可以解释为什么微软会在投资于基础研究的同时，大规模投资于新产品版本的开发领域，并且已经扩展渗透至其他产业部门，从而和不同类型的企业结成了合伙关系。微软早已不同凡响；它已从以字符为基础的计算发展到图形计算，保持着在操作系统方面的市场领先地位，同时提高了其在应用产品方面的市场位置，然而任何一家公司在激进的变革来临之时都是脆弱的。下一年，可能就会有某个竞争对手推出某种假想中的操作系统。这种操作系统不仅和过去曾出现过的每一种程序都兼容，而且又只需要很小的存储器，并能对日常语言命令作出反应。上述情况一旦出现，可能将使 Windows 95、 Windows NT 以及微软的其他一些应用产品遭到淘汰。

我们不知道何时或者是否 PC 软件行业将经历这样一次技术上的根本变革。我们并不认为这种革新会很快到来；大多数技术革新是渐进的，而 PC 软件的未来将很可能会由技术和产品使用上的一系列渐进变化逐步积聚而成——这是微软擅长的一种竞争方式。出于这个考虑，我们相信微软的市场顶尖地位很有可能继续保持至少 10 年。仅仅让现有的一代 PC 使用者更换他们的机器和软件，差不多就需要这么长的时间。

其次是一个更长期性的问题，也就是微软是否能够获得或者创造出在全新的竞技场上较量所需的技术或其他技能。对微软来说，目前仍有一些正在发展中的，可能给其带来丰厚收益的巨大市场亟待开发：网络操作系统，多媒体出版物和应用产品，家庭消费者产品，工作场所产品，交互式电视以及信息高速公路服务。上述种种大约会用十年或者更久的时间来发挥出它们的充分潜力。微软正以迅捷之势快速投资于或进入所有这些新市场。公司看起来在一些领域拥有技术优势，例如 Windows NT 或者微软网络，然而公司不太可能在如此多的新领域中都占据优势。Novell 统治了公司网络，Intuit 占领了个人财务系统，Lotus 则在办公组合软件方面一枝独秀。众多公司，如 CompuServe（H&R Block 所有），Prodigy（隶属于 IBM 和 Sears Roebuck），美国联机，和 MCI 通讯，在联机服务方面拥有更丰富的经验。其他公司，如 Oracle 和惠普，已经改善了视频服务器技术的兼容性能。全球许多公司则在通讯上掌握更多技能。而像任天堂和 Sega 这样的电视游戏公司具备为那些真正意义上的新用户——孩子们生产有着令人难以置信的高能量、低价格的家庭电脑以及娱乐软件的能力。所有这些和其他许多公司无不想从比尔·盖茨的未来世界中争夺一席之地。

当然，为了取得投资成功，微软并不必须获得这些巨大的新市场上 80%或 90%的份额，它甚至无需达到 40%或者 50%的份额，尽管批评家们或许要把盖茨抬举到这样的高标准。因此，为平衡起见，微软可通过开拓新的，迅速成长的市场来赢得一切。这是正确的，即使在多数市场上微软不见得比第二名、第三名或第四名来得好；即使微软始终主要是 PC 操作系统和一些基础应用系统的供应者，同时将一种联机网络引入作为副产品。相反地，没有哪一家公司能在为顾客提供将操作系统、应用系统、网络通信程序等多种功能集为一身的组合产品和服务方面比微软做得更出色。伴随着这种功能集合可能性的出现，微软作为 PC 软件基础设施提供者的地位或许会随着时间的推移变得越来越有价值，而非渐趋无益，并且在这些不同的竞技场上轻松地左右逢源。

第三种挑战也许来自于那些想从财务馅饼上分得更大一块的微软的现有合伙者。举例来说，家庭银行事务和其他联机用户服务很明显是微软将提供

的基础设施软件包中的一部分。即使并不拥有其太多内容，微软仍将能够从每笔交易中收取费用，而不是仅仅售出个人产品。但是康柏和另外一些高容量 PC 硬件的制造商们可能会抵制向微软支付那些主要替微软带来额外收益的集合软件产品的使用费用。硬件公司可能要求部分版税，或者实际上已开始向某些特定类型的组合程序收取费用,比如那些从连接 Internet 或从联机服务中间产生的费用。对于许可软件而言，这将是一个长期实践的逆转。信息高速公路方面的其他合伙公司（如有线电视公司、电话公司、银行、内容提供者，诸如此类）都会想得到更多的收益，如果电子网络市场在规模和价值上保持持续增长。按理说似乎应该有足够的资金用于分配，但是像微软这样的软件生产商或许并不像一些人目前认为的那么能赚钱。

第四个问题产生于人们对反垄断的关注以及微软在政府反垄断官员和法官们严密监视的情形下是否会不断改变其行为方式。微软的竞争对手们同样热衷于向政府权威投诉。Novell 就曾在欧洲这样做过，像前文论述过的那样促成了 1994 赞成协议的出台。有几家公司同样地资助一份报告，借以帮助美国司法部反对微软兼并 Iutuit 的企图。[8] 其他公司则正在联合力量以确保微软并未开发它自己的联机网络。[9] 许多大大小小的公司，包括 Apple 公司，频繁起诉，宣称微软盗用或非法复制了它们的技术。（例如，最近一例 Apple 诉讼案，阻止了微软在 Windows 开发设备上装配 Video 中的某些零构件。起诉书中断言微软将 Apple 的 Quicktime 中的组成构件非法使用到 Windows 多媒体技术当中。[10]

我们认为，这些威胁和持续的警惕，将会使微软更加难以维系对这个它奋斗于其中的行业的未来变迁的把握。同样地，也会使微软更加难于组合产品，获取更多的公司。而将来直面美国司法部和联邦法庭的可能或许将迫使微软在如何运作方面发生更多的改变。甚至 1994 年 7 月的赞成协议（微软同意遵守，尽管一位美国联邦法官最初并不曾证实这一点）将会保证硬件销售者能在财务上更易于提供 Windows 的替代品,如 OS/2 或者 Macintosh 上使用的 Apple 的操作系统。[11] 微软显然并非无懈可击：它在推销一些脱离了其传统专业技术领域的产品（就像 LAN Manager 和 Money）时打了败仗。

然而，1994 协议作用有限。微软仍保持着多项自由：和用户结成同盟，提供数量折扣，组合产品和提供产品折扣，未来产品发布竞争中的优先权，以及可获得更多公司的自由。要想制止微软为自身利益设计应用产品，提供新型工具和语言来支持其产品，或是使用其编程人员在 Windows 方面的知识技能，将是非常困难的。何况，微软仍旧能够利用 Windows 的兼容性这面旗帜来推动其应用产品、操作系统甚至硬件装置（如新的“ 自然的 ” 键盘，1994 年推出）的销售。如果将来出台一项新的赞成协议，它一定会对微软设置更多的限制，但究竟有多少限制，会导致何种结果都不得而知。任何一种解决办法都应当满足扶持竞争均势的需要，这将普遍有利于消费者（除了出现评判标准的市场混乱的时候，正像在录像机问题上 VHS 对抗 Beta 一案），同时伴随着在那些或许是美国最具国际竞争力的公司身上施加太多限制的危险。

需关注的第五个问题在于竞争对手们可能继续联合对抗微软，并最终成功地减弱其影响。这方面的一个例子是老对头 Apple 公司和 IBM 共同签署协议来开发一种普通 PowerPC，此外 Apple 决定一改它有着 10 年历史的老政策，特许其他公司使用 Macintosh 软件，以便创造自己的无差别市场。其他众多竞争者急切地倾注大量资源来接管本行业中那些最强有力的公司。IBM

已兼并了 Lotus,接收了热门的 Notes 产品。Novell、Word Perfect 和 Borland 的电子表格分部合并来对抗 Microsoft Office。Lotus 和 AT&T 形成了合作关系，力图把 Notes 和 AT&T 网络软件合二为一。Aldus 和 Adobe 在打印字符方面合作。Apple、IBM、WordPerfect、Novell 以及 Lotus 已提议一项 OLE 的替代产品。Apple、AT&T、IBM 和西门子正在设计一种能允许电话和计算机彼此连接的标准，以便能复制 Microsoft At Work 和 Windows 95 的某些兼容性。其他联盟则包括各种企图使工作站,电视游戏能更好地和 PC 桌面系统竞争的努力。在信息高速公路的竞技场上，存在着许多强有力的竞争者以及不包含微软在内的合伙企业。

第六个问题是微软在组成构件一级的竞争中的软弱性。用多个项目可共享的构件来构成产品在技术上是有效率的，同时使得像微软这样的公司能够以低价向用户提供多种功能。然而，如果这种共用分享机制在市场上是可获得的，其他公司就可能生产足以同微软的构件相抗衡的构件。在 Windows 结构范围内，这在技术上是可行的，从而使不同的产品和构件得以一道工作。OLE 的不断普及推广使这个目标更容易实现。微软的战略是把产品集中成为软件包或组合件，然后在售价上打折。这种战略会阻止消费者购买不同的产品并进行类似此种构件组合，例如同时购买 Excel 和 WordPerfect，同时使用，而不是仅仅支付一半的价格来购买 Microsoft Office（或是 Novell 的 Perfect Office，或者 Lotus 的 Smartsuite）。然而，很有可能，竞争者们创造出高度创新型的产品（如组合件功能或网络功能），把这些产品联结到微软产品之上，再进行销售，而非只销售微软的替代品。当微软没能提供在价格和功能上都足以和其竞争者的产品相媲美的构件时，这种机会就出现了。

向未来进军

微软总是面临技术的以及市场营销方面的障碍，但其奉行或许是它最基本的战略：向未来进军！我们已经简要描述过公司在研究和新产品开发上正在做些什么，但是除了已经公布的之外，微软还可能会探索何种产品，查询何种用户情景？它需要怎样的技术能力？我们现在就来谈谈这些问题——依靠我们自己的观察和推断，而非依赖于来自微软的专有或机密情报。

鉴于公司在其系统、应用、消费者，以及研究组织方面正在从事的种种活动，我们相信微软正为向消费者提供前所未有的集中化程度和访问其全部产品及服务上打下坚实的基础。这种集成化和产品便利能够建立在 MS-DOS/Windows 用户基础以及

Windows 家庭技术之上。它同时能够扩大那些现有的或开发中的作为 Microsoft Office 和 Microsoft Home 产品线组成部分的产品供应。同样地，也能扩大类似组合式 Bob 应用系统,新型 MicrosoftNetwork 这些产品的市场供应。更具体些，我们相信微软有能力做到以下几方面：

- 把其目标市场从那些从事创造性工作的人转向纯粹消费型的人。
- 开发包含先期支付的购买价格和每次使用费或每笔交易费在内的产品。
- 尽量把几乎全部产品装入差不多每年都推出新版本的简化联合包装中。

·模糊由早期区分应用、操作系统和网络产品而造成的功能界限。
·消除基础计算机硬件平台、电视、有线电视系统三者间的多种差异。

消费者市场重点

微软似乎正在将市场重点转向作为从事创造性工作者对立面而存在的消费型用户身上。像 Office 这样的产品主要支持了那些从事创造性活动的人们：撰写信件、报告、备忘录和书籍者；编制并分析不同形式的预算及数据者；制作演说稿、幻灯片、图表以及诸如此类者。目前的 Office 产品极好地支持了这些以创造为基础的行动，这正是为什么该产品能占据应用成套设备市场 70%份额的原因之一。然而，世界上的大多数人，比起创造事物来，要花费更多的时间在消费物品上面。人们消费什么？在其他众多事物当中，他们消费娱乐、信息、教育资料，通信和财务数据。

微软的人们看来也相信“家庭”软件的消费市场在未来年份内会急剧增长。与此相应，公司将很可能推出广泛的，能满足这些产生于消费者的需求的集合产品。Microsoft Bob，一种把基础程序合成一体供家庭使用的产品，正是朝上述方向所迈出的一步。这方面的一个更先进的版本或许被称为“Microsoft Home”，与“MicrosoftOffice”相对应。（微软为 Bob 产品命名时的考虑事实上就包括了“Home Foundation”和“Essential Home”双重含义，因此我们的建议并不见得漫无边际。[12] 而一种更为先进的产品将在各种范畴内容纳更复杂成熟的个人、家庭特征，正如下面所列示的：

通信
·电子邮件和电子公告板
·用于交换信息、分享意见、保持家庭信息沟通和维系家庭关系纽带的“组合件”

娱乐
·流行娱乐程序，例如电影或游戏
·运动得分和游戏分析

财务
·财政、税收，以及预算规划软件
·证券市场报价、电子银行事务和帐单支付

教育
·畅销教育用书书目，包括那些给孩子们以及给成年人看的书
·访问公共图书馆、数据库和政府出版物

信息
·Internet 访问软件（如通过 Microsoft Network 联网）
·取得报纸、杂志和书籍的电子版本

旅行
·航班和列车时刻表以及预订服务

·旅馆信息和预订服务

在下个 10 年当中,家庭电脑很可能将成为必不可少的家居用品——一个人每天要使用几次的设施或用具。家庭电脑将向每个使用者传递一个知识的世界和许多感性的经验。当然，微软将继续开拓对于公司的成功功不可没的个人桌面系统和办公产品市场。然而，大多新的市场容量将可能来自于那些尚未被触及的家庭消费者。构成这种 Microsoft Home 或先进的 Bob 产品基础的潜力包括了发达世界中的所有家庭。

这些消费者定向的软件产品体现了许多内容方面的问题。软件所增加的价值依赖于它所传递的内容，诸如艺术品、影碟、新闻、或者证券市场数据。微软或者不得不自己产生出更多这样的内容（在许多情况下都非常困难），或者继续从其他源泉获得这些内容，或是取得使用许可。一个基本的技术性问题是许多内容内在反映了特定国家的文化或兴趣。我们已经注意到微软和其他公司无法通过仅仅改动目录盒中或用户界面屏幕上的母语来把这些产品供应到外国市场上。（举例来说，法国消费者可能不会需要美国农民年鉴的电子译本；他们想要和法国有关的资料的电子译本。）此外，许多这种产品把图像和文字、声音结合起来。这种多媒体产品比那些传统软件产品更难设计。微软已从开发 Encarta 百科全书和其他多媒体产品中取得了一定经验，但是它需要在全公司向不同的产品组织推广这种经验基础。

同时具有购买价格和每次使用费的产品

在将来，我们相信，许多家庭消费者软件产品的定价将包含先期购买价格和每次使用费或每笔交易费两个组成部分。今天，当人们购买一种软件产品时，他们支付一笔购买价格（例如 100 美元），然后就拥有了使用该软件的许可。在多数情况下，顾客可根据需要无限次，无限期地使用这个软件。然而，我们预计，大部分微软家用型产品当中的软件，需要使用者在每次使用作为联机网络之一部分的特定性能或服务时支付小额费用（例如 25 美分）。这些费用之所以出现，是因为许多消费者软件产品需要访问或使用一些并不属于原有产品组成部分的外部服务或产品；这类例子有使用好莱坞制片厂所有的影片，向一家本地银行检索支票帐户，或者从纽约的一个出版商处参阅一份电子报纸。这些外部依赖明显有别于今天那些提供具显著独立性功能的产品，例如设立一个文件或改变一张电子表格。

收取每笔交易费或每次使用费的可能的财务回报是巨大的。举一个简单的例子，我们假设在今后几年当中的某个时刻，世界范围内有一亿人正在使用 Windows 或 Windows NT 的一种新版本。(这个数字并非虚构。目前 Windows 已拥有大约 7 000 万使用者，MS-DOS 有大约 1.4 亿，而 Windows 95 最终每年应该有大约 3 000 万被安装入电脑中。）如果这些人当中恰好有 1/10 登记进入微软网络系统，那么微软将会拥有 1000 万的网络用户。每个人可能每天平均选择两次服务——例如支付帐单和查询证券市场价格，或者查询运动得分情况及观赏电影。如果每笔交易向微软支付 25 美分的费用，则公司每天将会收入 500 万美元，一年就是 18 亿，大约相当于 Microsoft Office 在 1994 年全年的收入。如果有两倍的用户进行登记，且每天平均使用四次服务，那么日收入就是 4 倍，而每年将会累计超过 70 亿美元，或者超出微软 1994 年全年销售水平的 50%还多！如果每笔交易费增至 50 美分，我们就会得到 140

亿的收益，假如使用状况未发生变化。这个市场有吸引力的地方就在于收入可能成倍增长的潜力上（使用者数目×以收费为基础的交易次数×价格）。

当然，微软也会产生和这些收入相关的成本。微软必须购买内容或向不同的制片厂、银行、出版商、艺术家以及其他一些提供这些产品和服务的组织支付版税。微软还不得不向诸如电话公司或有线电视供应商这类信息输送者支付费用。它也许还不得不向那些把这种类型的软件装入新型 PC 的硬件公司付款。同时微软还必须为那些围绕这些服务而展开的软件开发提供资金，并备有专门人员随时回答顾客的提问。然而，这种获取巨大利润的潜在可能性将数倍于微软从购买某种特定软件产品的顾客处一次收取的价格当中得到的赢利。

考虑到技术上的障碍，许多这些每笔交易产品都需要实时软件功能，即意味着计算机软件和硬件必须在一个固定的时间间隔内连续完成一种特定功能。举个例子，如果消费者想观赏电影、支付帐单、进行采购，或得到证券价格的报价，他们期待在数秒内取得最新结果。开发实时软件对于电子通信公司或防御工业生产者来说是很平常的，但对于一家 PC 软件公司来讲却非同寻常。尽管已联网，但一家 PC 软件公司仍很难让类似 Windows NT 和 Windows 95 这样有着一些实时功能或相似能力的操作系统进行多任务运行，就像一些复杂的应用系统所做的那样（例如多媒体或相关数据库管理产品）。

成功地开发先进的实时软件，需要对用户提出的需求类型有深入的数学理解。它同时需要把那些功能加以数学模型化的能力，然后在产品开发循环过程的初期在电脑上模拟操作，作为需求分析和特定化过程的一部分。公司还必须在产品结构和构件界面上做大量工作，以确保系统无偏差地运行。在应用项目上，微软通常不做数学分析、计算机模拟，和广泛的结构设计工作。研究分部已雇用了一些掌握这些技能的员工，但是如果微软和其他 PC 软件制造商想创造大量的实时交易功能的话，他们还需要在组织的各个部门更多地雇用此类人材。

成套包装，联合供应的产品

微软和其他 PC 软件公司的倾向是把多种产品组合成为少数一些统一的或成套的产品供应。我们在 Office、Bob、Windows 和 MS-DOS 身上都能看到这一点，这些产品如今已涵盖屏幕保存器、通信功能、工作组织者软件，数据压缩软件和其他类似软件。我们期待微软继续把众多分散的产品融入 Windows、Office，以及 Bob 的未来版本，或者我们假想的 Microsoft Home 当中，并且具有不同的组合程度。这意味着多数消费者将不会再单独购买 Excel、Word 或 Complete Baseball；他们很可能会购买 Windows、Office 或 Bob（Home）。这些主要的联合产品中的每一项也可能会类似于每年推出新型汽车一样，发生年度性的“式样改进”，伴随着新特性的出现或使用上的更简便化。如果多数人每年购买最新的产品版本（或至少每隔一年就要购买），收入将是巨大的，此外为使这种情况真正出现，价格可能会被适当降低。普通消费者并不一定想成为软件专家，以便决定购买何种产品。微软希望购买决策过程真正简单化：如果是在 1997 年需要办公使用的软件，则顾客应当购买 Office 97（以及 Windows 97，如果他们尚未拥有）。如果是在 1997 年需要家庭使用的软件，则顾客应当购买 Home 97（或 Bob 97）。

每年，联合产品版本都将在微软内部编织一张产品间互相依存的大网，

结果造成“产品片断”的发展或使用建筑结构中的骨架形式来集合构件。小组（不超过 10 或 15 个开发员）应当能在 6 个月左右的短期开发循环过程中创造出这些产品片断中的大多数。而更大、更繁杂的特性和结构性改进将需要多年的开发循环过程。如果情况确实如此，那么软件当中主要的新特性将只有每隔一年才能出现，对于 Windows 而言，甚至会更迟缓。我们估计许多平行的、长期的项目将被安排间隔公布其成果，以便每年的年度产品供应中都能包含足够的新特性来吸引购买者。在这种情景下，日常构造将变得更难开展，正像微软已从 Office 和 Windows 95 上所发现的。微软很可能还需要一个敬业勤奋的构造小组，甚至对应用项目而言，来协调产品片断和多年项目。我们期待出现这样一种计划来联合运用为单独的或密切配合的构件使用的日常构造以及为多重构件和整个体系使用的星期构造。

模糊应用产品，操作系统和网络

我们已经提到微软存在一种趋势，即模糊由于软件产品的不同层次造成的功能之间的界限。这种“集成”背后的逻辑是非常简单和顺理成章的。就像我们在第 3 章中讨论过的（参阅表 3.1），软件产品分为三个基本层次：在操作系统之上运行的应用产品；置于硬件之上的操作系统；以及把个人使用者同其他使用者或联机网络相连接的网络通信程序。

使用者不应该在乎一次运作或一种特定功能所需要的软件类型或软件程序的数目。微软应该也会希望这些区别不被消费者所察觉。而且，它还应该希望使用像操作系统一样的基础产品或 Office、Bob 此类大规模市场上的应用产品来向网络以及信息高速公路服务这样的新领域不断地拓宽市场。微软在操作系统和独立应用产品方面颇具实力，但在网络市场上却是个相对意义上的后来者。然而，软件的新前沿，由网络产品和服务组成，这当中包括可以联系同一办公室或公司内部用户的软件以及使用户足以接通环球国际互联网络和其他外部联机网络的软件。如果微软继续模糊传统软件各层次之间的功能分界，则它的用户将能直接通过执行操作系统或某个应用程序中的一两个步骤，轻而易举地联络其他用户和享有联机产品及服务。当然，Novell、AT&T、DEL 以及其他网络软件的供应商可能愿意保持传统的软件层次区别，以使他们能够把网络程序作为完全独立的产品来出售。CompuServe、Prodigy、美国联机和其他公司可能也愿意保留这种差异，以便他们能继续出售那些同操作系统和应用产品供应者相分离的联机网络服务。

Office、Bob 和 Windows 的统一产品包装战略的逻辑或许会导致一种单一产品的出现：“Microsoft 2000”。这种产品可能会包容多数用户可能希望购买的全部软件产品。这种水平上的一体化或集成看起来似乎遥不可及，但在技术上却是可行的。它同样会提供一个简化的包装和市场营销计划。假设，微软的某个成员试图将一个单独的软件“Mega-glob”连接到世界上生产出的每一台 PC 上，对反垄断的关注很可能将会阻止微软公然采用这一战略。无论怎样，这种未来方向是没有问题的：微软（和其他公司）将继续把那些已在单独的应用、操作系统及网络产品上发现的功能和特性组合在一起。我们同样认为微软正处在一个十分有利的位置，使其能够向它的操作系统和应用产品中安插各种各样的“钩子”。这些“钩子”恰能提高消费者未来所购不同产品的性能，从而让消费者根据需要创造自己的软件组合。只要分享机制始终是开放的（例如 DLL 或 OLE 标准），消费者同样能够从其他公司那里

组合构件。

为了使这项战略服务于微软的利益，公司需要在网络和电子通信技术方面更具专业优势。以消费者为定向的产品和服务要求具备能够连续，可靠地传递和收集大量数据的能力。未来的网络产品必须克服现有网络的容量和可靠性在技术和具体执行方面的局限。这种现有网络主要在电话线、电缆和光纤的混合基础上运行。微软已着手，力争实现和有线电视公司的合作，因为后者具有比目前的电话线容量更大的网络基础设施。另外一项更长期的抉择是联系那些有办法使用卫星网络的蜂窝式电话公司。这方面的一个潜在实例是比尔·盖茨和克雷格·麦考，麦考蜂窝式通信公司的创立者，在 Teledesic 事业中的合作关系。该计划是把环绕地球的数以百计的卫星连接成一个巨大的数字通信网络，同时使用 PC 或其他手动电脑作为基础通信装置。

模糊电脑、电视和有线系统

除了减弱操作系统、应用和网络之间的区别之外，我们预期微软会尝试消除在基础电脑硬件平台、电视和有线电视通道之间存留的多项差异。在未来，当用户需要娱乐或新闻时，他们应该能够向自己的电脑寻求帮助，而不需有一台电视或有线电视通道。广播电视和有线电视仍将存在，但消费者将在电脑上收看电视节目。（通过和一台价值数百美元的 PC 连接，已可以做到这一点。）电视网络，电影制片厂以及有线公司将继续制作节目内容，但是他们不会再提供或控制以电脑为基础的传送或收视机制。

今后数年中，至少有一些家庭电脑将具有大型的、高精度的彩色监视仪，用以准确逼真地再现图画、影碟、图表、音乐、声音、数据和文字。这些监视仪将是平面仪表显示器（类似于目前笔记本电脑上的显示器），但它们可能仅有 2 英尺×3 英尺的面积，且仅有数英寸的厚度。许多供应商将生产出能够驱动显示器监视仪的基础电脑芯片。在消费者看来，各种芯片明显的区别仅在于它们的价格和性能，而非结构上的不兼容性以及如今存在于像 Windows PC 和 Macintosh 这样的机器中间的视觉感觉上的微妙差别。电脑可能会把 Microsoft Bob 的未来版本作为应用系统，把 Windows 当作操作系统和网络来执行。

用户们不会知道或者关心，在网络上的其他地方，在他们的家庭之外，“支持终端”或服务电脑可能正在运行 Windows NT 的未来的不同版本。服务器将为可获得的大量联机产品和服务提供存储、检索和连接功能。为了在这种组合式的，电脑、电视、有线网络三者合一的装置上传送信息，微软必须深入钻研多媒体、实时系统、高容量数据存储设备以及和显示器有关的多方面，多角度的问题，所有这些都需要在电脑网络上可靠运行。这些产品不能仅仅显示刻板的文字或数据；它们必须提供实际经验，模拟日常生活中的图像和声音。

微软已经用原始形式或经过改良的形态展示了这些产品类型和技术能力。微软也已把种种产品装配在一起，并使其达到如此的可靠、廉价和便利，以至于任何一个电脑新手都能使用并愿意出钱购买。然而我们还尚未论及目前存在的任何一方面的不可实施性。

一个重要的问题是微软是否能够制造为这些新产品、新服务打基础的复杂的软件成分，而不是仅仅造出雏型。使微软网络成为数以百万计人们的新型用具所需的软件，必须兼有多媒体内容、实时交易和网络通信功能。它必

须没有明显的缺点；它必须使用起来简易而且价格低廉。微软不可避免地要同那些拥有自己的顾客群、有经验的网络及联机服务供应的大型电子通信制造商竞争，同时还要和其他在网络及实时软件方面具有技术专业优势的公司竞争。软件技术上无捷径可寻，微软必须继续获取不同类型的软件内容，必须明了那些家庭电脑初学者的行为特征，并把这种理解认知信息反馈到市场营销、产品规划及开发之中。

我们认为微软在今后 10 年中会有出色表现,因为公司具有可用于向未来“进军”的强大力量。微软将继续开发其密集的新产品和联机服务，其中多数会很有竞争实力并随着时间的推移不断进步。公司正在迅速获得新的人材、技能和合作者。然而，今后想取得和过去一样的骄人成绩是非常不易的。微软将不得不在创造下一代 PC 软件技术方面占领先地位,否则它将成为变革和岁月流逝的牺牲品（占统治地位的公司常常如此）。

我们的确不清楚二三十年后微软是否仍将存在。微软还要奋斗多年才能达到像福特、通电汽车或 IBM（这些公司均成立于本世纪初）那样悠久的历史。其他环球巨头，例如 AT＆T、NEC、东芝和西门子，其发家史可追溯到一个世纪以前。不过，微软在仅仅存在两个 10 年之后已在历史上留下了深深的烙印。竞争对手们、政府调控者以及不在少数的消费者都担心微软会滥用它相当大的影响力。我们认为，反垄断规则和市场强制力量——竞争和用户的警惕性——的联合作用很可能会使微软循规蹈矩，且能使其不断创造出更好更便宜的产品。鉴于家庭中 PC 的使用已变得像办公室里一样的无所不在，世界上最有实力的软件公司很可能在未来的某一天成为世界上唯一的、在任何一个行业当中都是最为强有力的公司。这种设想对多数竞争者来说更多是可望不可即，但就比尔·盖茨，他的辅助管理班子以及使公司完善运转的数千微软人而言，这种可能性确实存在。