提高软件生产率(续)

Barry W. Boehm

四、提高

按照软件生产率机会树的组织，下面对提高软件生产率的几个主要可选方面进行讨论：1)挑选精干人员，2)提高阶段效率，3)消除人工阶段，4)减少重复劳动，5)建造简单产品，6)重用软部件。

挑选精干人员 象机会树所表示的那样，挑选精干人员有三个主要可选项：

职员 表1中生产率表明人员或队伍的能力差异对生产率有一个因子为4.18的影响，而他们对应用领域、计算机系统或虚拟机以及程序设计语言的相关经验还有一个2.52的综合因子。IBM的生产率分析也取得了类似的结果。

因此，如果想提高某个项目或组织的软件生产率，最行之有效的行动是搜罗最优秀的人才为该项目或组织工作，而让那些二流人员去别处工作。但通常经理们对这些关键人员的选定表示冷漠而且常常表示反对，经常这样说：

"我们雇不起工资这么高的人。"

"我不能在这么'昂贵'的人身上冒险。"

"我不能雇用你们的超级明星，除非你们的项目得到财政保障，差一个月也不行。"

"乔先生让这些人都靠边站了，我得帮帮他们。"

"我不能等了。我希望有人能使这个项目下周马上有所进展。"

有时候两种情况需要你回答，但你一般也只能做一种暂时的而不是长久的安排。

另一个同等重要的问题是雇员的工资方面，它要求你去调整不合适的分配。无论你如何精心挑选你的软件队伍，你总会发现一些人不愿意去做，为了全队的目标应做的一份工作。甚至在你企图为他们寻找或准备好了一个队中的恰当位置他们也还是这样。遇到这种情况，你可能想拖拖再说，或者假装不知道，或者说说了事，再不就请其它队员做些额外的工作。这样做在短期内可能是个办法，但长期下去，无疑会造成不良后果。

2. SRA Software Development Environment Tools on UNIX M-Box, Software Research Associates, Inc.
3. SRA Software Development Environment Tools on UNIX D-Box, Soft ware Research Associates, Inc.
4. SRA Software Development Environment Tools on UNIX SPEX, Software Research Associates, Inc.
5. SRA Software Development Environment Tools on UNIX Testbed, software Research Associates, Inc.
6. SRA Software Development Environment Tools on UNIX CCS, Software Research Associates, Inc.
7. SRA Software Development Environment Tools on UNIX Midas, Software Research As sociates, Inc.
8. SRA Software Development Environment Tools on UNIX Pygmalion, Software Research Associates, Inc.
9. 郝克刚，软件分析与设计的支持环境，《计算机研究与发展》，1986.7.
10. 郝克刚，设计结构编辑系统（DSE）功能说明书，西北大学计算机科学系软件工程研究室，1985.11.

把某人开除出去是不容易的。但如果你花了足够的时间，考虑了这个问题，那么也许就可以通过开除对形成一个好局面起积极作用而且被开除人也会找到比以小组为单位开发软件更适当的工作。如果你不这样做，肯定会结下一个疙瘩，不要回避这个问题，要从矛盾中尽快解脱出来。

设施　当然软件开发和改进是极端劳动密集型活动，但是生产率的作用还是可以通过使软件生产向资金密集型转化而得到大幅度提高。通常，对办公室的工作人员的投资大约是2000-3000美元，而软件工作人员与此稍有差别。然而，象Xerox，TRW，IBM和Bell实验室等一些组织都付出了高于提高生产率所能收回的资金（每人一万到三万美元不等）。

为软件人员提供私人办公室，是另一种经济而有效的办法，IBM-Santa Teresa通过这种努力使生产率大约提高了11%，而TRW则有8%的提高。另外一些研究报告表明在提供良好的设备与支持能力方面投资也能取得同样的效果。

管理　低水平的管理可能比其它因素会更快地导致生产率的降低。这里有一些差劲的管理活动的主要类型。

·计划不好。它的一个例子是一个项目没有明确的测试计划。当一个20多人的测试组来到现场时，发现没有测试数据、测试目的、测试设备、测试策略和过程或者开发者准备好的代码测试标准。结果，这个项目资金超额30%，计划时间超过40%。

·不合理技术搭配。不合理技术搭配经常遵循"彼得原则"，即：在一个层次中，每个雇员都趋向于把自己抬高到没有竞争的级别上。彼得原则在软件工程方面最经常的实践就是把好的程序员抬高到管理的层次上。有时这么做是成功的，但总的来说，它在软件工程方面导致了比其它方面更多的不匹配，挫折和事业上的碌碌无为。许多组织已认识到这一点，并已实行双重或多重职业道路，最后达到象"超级经理"那样的"超级程序员"或"超级分析员"的职务。

·不成熟的雇员。这方面的一个例子是引用一个小项目的负责人的一段话："在设计的最初阶段，我担任项目负责人并接受了三个实习生当助手。我最大的错误就是花了自己和另一些高级程序员一半的时间去使实习生们有活干。结果，我们的设计中留下了很大的漏洞，并最终导致了我们的失败"。一个与生产率降低相关的原因就是企图为了加快速度而增加更多的人，它违背了布鲁克定律："拖延了的项目越增加人越要延期"。

·不成熟的编码。这方面的一个例子是WISCA，（Why isn't Sam coding anything？）意思是"为什么萨姆什么也不编？"，一个相对的俗语是："我们得赶紧开始编程，因为一大堆调试等着做呢！"一个有许多人参加的开发软件的重要的管理特征是，事前制订一整套完整的、有效的、稳定的模块接口规格，以使得开发者的各自并行工作不至被互相间程序的通讯问题所困扰。早在1961年，软件管理者就已认识到"精确接口定义的每一个细节，毋庸置疑，比黄金还贵重"。

·不合理的报酬结构。这方面一个例子是，一个组织给上层人员提薪6%，而给中层人员提5%，这最终导致了优秀人员的离去。这方面问题可以通过设立其它奖励来解决，比如特殊贡献奖、分级别的鼓励奖、旅行和特别活动，以及了解上层人员的计划等。

提高阶段效率　图1中的价值链提供了在软件过程中各个阶段减少或提高有效性对有关生产率产生的作用的基本见解。例如，因为执行代码过程和单元测试功能的耗费仅占软件生存周期成本的8%，所以工具对减少代码或单元测试或使它更有效所产生的生产率影响不会超过8%（除非这方面的工具同时减轻了其它类型的工作，例如后一阶段讨论的重复劳动）。

使现在软件过程的各阶段更有效的基本方法就是使用软件工具使现在各阶段中重复次数多的和劳动密集的部分自动化。

目前经验表明，如果软件工具作为集成化项目支持环境（IPSE）的一部分就会更为有效。IPSE和一个即席的软件工具集的主要区别在于：

· 它有一套关于软件工具支持软件过程的公认的假设（或者，要求更高些，一个特定的软件工具支持软件开发的方法。）

· 一个集成化项目主数据库或一个一致性的对象库用作软件开发过程中建立技术的或管理的实体时的资料库，这是依照它们各自不同的版本、属性和关系建立的。

· 它不仅对程序员开发代码的支持，还对软件项目中所有范围的用户和活动的支持。

· 一致的用户接口为使用IPSE中软件工具的各类人员提供方便自然的方法（各种人员包括程序设计专家、一般图书馆员、秘书、管理人员以及计划和控制人员等）。

· 真正是大量软件工具的合成，包括软件项目活动的各个重要部分。

· 计算机通讯结构使用户可以方便地存取IPSE中的数据和资源。

消除人工阶段 一个良好的自动化辅助目标，不是简单地使各阶段更有效，而是要达到完全消除以前的人工开发阶段。如果我们把今天的软件开发和五十年代做一些比较，汇编程序和编译程序就是消除人工阶段而大幅度提高生产率的卓越的例子。过程构造系统、软件标准检验程序和其它质量保证功能，以及需求和设计一致性检查程序都是最新的一些努力。

具有更大雄心的消除人工阶段的努力，包括通过提供直接操作于一套软件规格说明以自动产生计算机程序的能力实现整个程序设计过程的自动化。这种方法的两个主要分支是：领域有关的和领域无关的自动化程序设计。

领域有关方法是利用领域知识把规格说明变换成程序而且把整个程序设计范围限制在相对较窄的范围。在这方面，像Visicalc这样的第四代语言就可以作为说明，这是一个在狭窄的范围内的出色的自动化程序设计系统，但超出了特定的领域就相对地无效了。

领域无关方法，也就是通用的程序设计自动化，长远来看，它提供了更为广泛的效益，但它在有效地实现大规模程序方面仍然有着更多的困难。

减少重复劳动 图1中的价值链分析表明，提高软件开发生产率的最大机会就是减少占30%的对生产率作用的重复劳动。实际上，在整个生存期中这部分所占比例可能超过50%，因为节省重复工作的大部来源（例如，先进的程序设计技术和快速原型）都将对现行的开发后软件修改（例如，为确定残留的错误或为最后地确认需求的正确）以及使这些修改更为有效产生影响。

图2中的机会树表明，减少重复劳动的机会主要在于：前期辅助工具，基于知识的软件辅助，信息隐藏和先进的编程技巧，增量开发，改进的加工模型，以及快速原型。（此外，软部件的重用也能有效地减少重复劳动。）

前期辅助工具 软件的计算机辅助设计和需求分析工具，能够通过对软件需求和设计说明书的更好的表达、更加形式化的和无岐义性的规格说明、自动化的一致性和完全性检验以及对设计所要求的自动示踪等手段减少大量的重复劳动。也许这类系统中影响最大的要算分布计算机系统，它包括系统规格说明语言，分布系统设计语言以及模块描述语言。许多可行的商品性前期辅助工具已经出现，比如 ISDOS/PSL—PSA，SADT，CASE，Excelerator，IDE，Cadre 和 Ada Graph等。此外，像快速仿真辅助系统如RSA，和可执行的规格说明辅助系统如Paisley也都是有关的前期辅助手段。

基于知识的软件辅助 在许多应用领域，人工智能界已经发现把知识密集型功能完全自动化是当前最难的一类工作，但是传统技术和AI技术的结合却能为专家们在执行复杂任务时提供非常有用的自动化帮助。这就是基于知识的软件辅助（KBSA）的概念。

KBSA的主要好处在于它能够减少由于后期发现的前期程序设计或项目决策不合理而导致的软件项目的重复劳动。目前在获取管理、配置管理、问题报告的示踪、算法选

择、数据结构、可重用部件的选择，以及项目计划与控制等方面正在开发着KBSA的原型。

信息隐藏和其它先进的程序设计技术　一般来说，先进程序设计技术（MPPS）象早期的验证与确认，模块化设计，自顶向下开发，结构化程序设计，走查或审查会，以及软件质量保证等通过避免重复劳动而达到提高生产率的效果。表1表明开发过程中，对一个大型的软件产品的生存期来说，MPPS的生产率指数从1.51开始，最大到1.92。

一种减少重复劳动的强有力的技术是由Parnas研制并应用于美国海军A—7项目中的信息隐藏方法。这种方法通过把实现决策隐藏在模块中而最大限度地减小重复劳动；从而使软件实现决策需要修改时常遇到的波动影响最小。这种信息隐藏技术在减小软件改进期间的重复劳动时特别有效，它是通过确认软件的哪些部分最可能修改（工作站的特点、输入数据格式等），然后，把这些不断变化的资源隐藏在模块中。

例如，现在要求使用一个特定的用户工作站或终端。通过确认经常需要改变的终端特性（如行宽、字符集、存取协议等），设计者就可以把这些终端特征的细节隐藏于终端处理模块，这样就把软件的其余部分与经常随着终端特性的改变而产生的波动效应相隔离。

这种方法改革了需求的规格说明的概念。这时需求的规格说明不再只是某一单一时刻系统软件需求的快照（影象），它同样能给出系统将要经历的大部分的需求变化的路径。这意味着设计的确认工作不仅应该阐明当前需求影象的跟踪结果，而且应该表明这一设计与所预期的修改方面适应能力如何。

先进的程序设计技巧和Ada语言　一个把先进程序设计技术和信息隐藏概念嵌入标准化程序设计的主要体现就是美国国防部的程序设计语言Ada的开发。Ada有以下一些结构：支持模块化的软件包，信息隐藏和重用；强类型化功能（避免了由于一般程序设计的错误导致的重做），结构化程序设计结构，以及一系列其它先进的特点，如并行，例外处理和类属程序。把所有这些特性集中在一起加以实现，当代的编译器几乎难以应付，但经过努力现在一些有效的Ada编译器已经可以使用。

评价Ada对生产率的冲击是困难的，因为Ada理论上想包括所有的程序设计和工程项目环境，而且大多数Ada功能还没有成熟。然而一些研究还是把Ada项目的生存期效率和传统的高级语言项目的生存期效率作为时间的函数作了一个比较，其中用开销来源变量和类似Cocomo的模型的生产率指数作为参数。这些研究的典型结果表明，初期使用Ada时带来的初期附加费用为12%到30%，而1988-89年期间将是一个突破点，完全成熟的Ada支撑环境和开发人员将导致长期的40%到50%节省。

改进的加工模型　当前处于领导地位的软件加工模型是"瀑布模型"，它往往是重视软件项目中所产生的一系列的文件（系统规格说明书、软件需求规格说明书、总体设计文件、详细设计文件）。当和整个前期确认活动一起使用时，瀑布方法对于减少重复工作是非常有效的。但瀑布模型的文档驱动解释，常常使得整个项目趋于更快地产生文档而不是仔细考虑关键问题。例如，最近提议的政府软件进展汇报计划只重视软件需求和设计规格方面未解决问题的数量。如果一个工程项目经理想显示他的迅速进展，就会被吸引到那些解决较容易的问题而不是较困难的问题的工作上，或者随意地完成文档报告而不是完全有根据的规格说明书。

这方面一个要点是重复工作的出现往往遵循Pareto分布：80%的重复工作的开销经常是来源于20%的问题。这种分布的主要含义是验证和确认活动应重视发现和消除软件项目可能遇到高风险的特定问题，而不应把

精力消耗在不分问题严重程度的初期减少错误方面。特别强调的是，这里预示着类似螺旋线模型的软件生存期的风险驱动方法比类似瀑布模型的文件驱动方法要好。螺旋线模型把软件开发过程组成为一个逐步细化的定义周期序列。每个周期中文档、模拟、原型或其它定义活动的重要程度是由它们尚未解决的关键性定义问题的相对风险而决定的。因此，螺旋线模型把重点放在识别和尽早解决那些将导致80%重复工作开销的20%的问题上。

快速原型 图1中通过数据表示出来的一个重复工作的主要来源是基于对任务或用户界面的需求的不求甚解而编制的软件规格说明书。这有一个简单的例子，用户说"我不能准确地告诉你我想干什么，当我看到它时我就知道了。"一些快速原型辅助手段已能用来有效地改善这种情况。大多数都基于类似Interlisp的先进的人工智能环境的解释执行能力。另一部分依靠使用传统高级语言的两阶段交互式图形组合和执行能力。还有一些快速原型系统为快速估价实时执行问题和分布式数据处理问题提供了减少风险的能力。

建造简单产品 象表1所示的，对软件开发者来说，最大的生产率指数来源于他所选择开发的指令条数。这有两个基本选择：一是建造简单产品，另一个是重用软部件。

上述两种方法即快速原型和改进的软件加工模型不仅可以减少重复工作，还可以有效地提高基准生产率，它们通过制造简化样品减少对软件"镀金"：指那些不仅耗费额外劳动而且减少产品概念的完整性的额外软件。

例如，图3描述了七个项目试验以比较用面向规格说明的方法和面向原型的方法开发小型用户密集的应用软件产品。主要发现有：

- 两种方法的平均值(图3中$\overline{P}$和$\overline{S}$)按每人时交付源指令数(DSI/MH)计算的生产率大致相等。
- 原型项目开发性能大致相同的产品可以减少近40%的DSI并节省40%的人时。
- 规格说明项目在排错和完整性方面困难较少是因为它们的界面的开发依据了良好的规格说明。

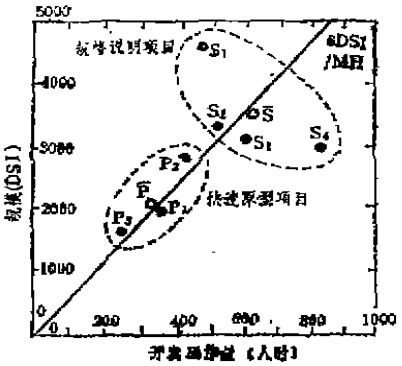

图3 原型方法与规格说明方法的实验结果，规模和工作量关系

最后一点表明原型方法不是万能的，规格说明仍然非常重要。然而，规格方法的一位参加者的评论却是这个实验的真谛—"说是容易的"。在制定规格说明阶段，正象P. Heckel所写的：

"大多数程序设计者……为了使用一种软件特性就说，'如果你不想用就别用，这能有什么害处呢？'但它确实害处极大。用户可能要花很多时间去理解一种特性，只是为了解决是否需要它，或者用户偶然用到这个特性，却不知道发生了什么事，也不知道如何排除错误。如果一种特性和用户接口的其它部分不一致，用户可能对其它命令得出错误的结论。这种特性必须被记载，于是用户的操作手册变厚了。这些特性积累起来的效果，就是使用户摸不着头脑，同时与你的程序的通讯也变得含糊不清了…"

一些较新的软件加工模型促进了简化产品的开发。传统的瀑布模型的一个困难就是规格说明书驱动的方法经常把人沿着"说是容易的"道路上引到"镀金"产品的方向。制定大系统规格说明时的经验是，许多用户很少关心计算机系统，而是常常重复提出许多要求以确信他们所需要的功能和性能将包括在系统中。

进化式开发模型强调使用原型所具有的把对用户任务必需的或很重要的软件产品特性集中起来的能力。相关的变换式模型简化了从规格说明到执行代码的直接转换的问题，因此既可支持基于规格说明的方法又可支持进化式开发方法。前面讨论过螺旋线模型由于过分重视持续地判断用户的任务目标以及不断从成本效益分析候选软件产品性质对任务目标的贡献，导致了镀金问题。

重用软部件 通过写更少的代码来提高生产率的关键是重用已经存在的软件功能部件。这方面最简单的方法包括开发和使用软部件库。这类工作已经取得了很大进展，特别是数学和统计的子程序以及操作系统的实用程序。在应用领域内，通过同样的方法也会有进一步进展。例如，Ragtheon的可重用商务应用部件库系统已经为新的应用提供了60%的可重用代码。而在成本的节约方面，通常设计阶段大约为10%，编码和测试阶段大约50%，维护阶段60%。东芝的工业过程控制可重用软部件系统已经在高质量的工业软件产品方面达到了超过每人月2000条源指令的速度。

这类系统所达到的灵巧程度，如果称作应用程序生成器，要比部件库更合适，因为它们有一些与系统部件有关的兼容性问题，如软部件接口约定，数据结构，以及程序控制和错误处理约定等。类似的特征使得Unix系统成为开发应用程序生成器的坚强基础。

人们可在这方面更进一步，以建立甚高级语言或第四代语言(4GL)，即增加一种描述所希望的应用的语言和一组解释用户规格说明的功能，形成一组恰当的软部件以及执行结果程序。

一些第四代语言的倡议者声称使用第四代语言生产率可以有10到100倍的提高。这能够实现吗？

第四代语言对生产率作用的最好实验事实是在一个6个项目的实验比较中得到的，任务是使用第三代语言（Cobol）和第四代语言（Focus）开发一个复合的小型事务应用系统，人员包括专家和新手，应用项目有简单也有复杂的。这一实验的主要发现，表示图4中，总结如下：

- 从总的平均来看(图4中的$\overline{C}$和$\overline{F}$)，对于生产相同的产品，第四代语言比第三代语言少60%的DSI，节省60%的人时(按DSI/MH表示的生产率又大致相等)。
- 对于不同的项目，第三代语言和第四代语言的比例有显著的变化，DSI (0.9:1到27:1)，人时(1.5:1到8:1)，DSI/MH (0.5:1到5:1)。

尽管平均来说，对同一项目Cobol比Focus多花费2.5倍的劳动，但结果还是随着应用范围有很大变化的。因此，很难预言对任一特定项目，第四代语言都能有较高的生产率。

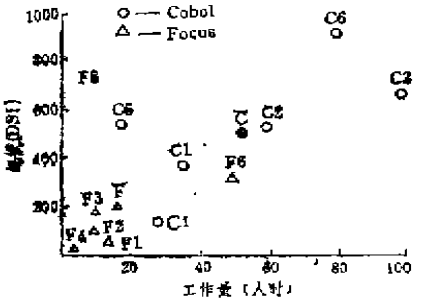

图4 COBOL和FOCUS的比较

Guimaras从对43个组织的调查提出了进一步的证实，第四代语言可以减少人员开支，减少用户损失，以及更快地满足用户对本应用领域的信息需求。另一方面，调查发现第四代语言对计算机资源使用效率极低而且与传统的应用程序连接困难。当企图用纯第四代语言去解决大型的，高性能的应用(如新泽西州陆上交通工具登记系统)时带来了巨大的灾难。

总之，尽管第四代语言为显著提高软件生产率提供了非常吸引人的选择，但人们还在希望建立具有其它领域功能的第四代语言。在缺乏第四代语言的功能时，另外一些有较多限制的可重用性方法，如功能部件库

和应用生成器等可以产生短期的开销节省，并可作为长期的更宏伟的第四代语言功能的基础。

五、 趋势

要总结提高软件生产率中的各种纷繁复杂的问题是困难的。图5至少提供了一些主要作用方面的部分小结。它根据每人月的相同的机器指令数，表示出了1960-65年期间和1980-85年期间，在提高软件生产率方面总体上的成就，也预示了我们在1995—2000年间可能的进步。

图5中的横坐标是一个定性标度，它表示在一指定软件生产率能力下应用领域的广度。它反映了这样一个事实，即迄今为止我们取得的给人印象最深的软件生产率方面的成就是通过发掘我们应用领域专门知识得到的。

因此，甚至在六十年代初期，那时最大的通用系统是用汇编语言以每人月一百条交付机器指令数（DMI/MM）开发的，谁也想不到存在着生产率为三千条DMI/MM或更高的应用程序生成器。火箭轨道计算方面就有这样的一些例子，例如Rocket系统提供可重用软部件库(在空气动力学，动力装置，导航和控制，地球模型等方面）和扩充的Fortran语言以便把软部件按要求连接，从而模拟出所希望的多级火箭运载工具和飞行计划。

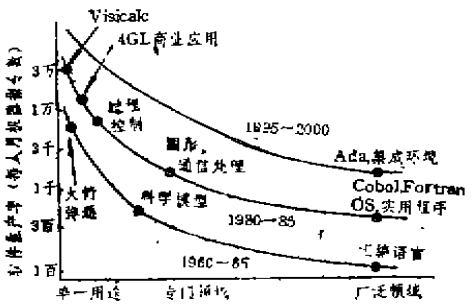

图5 软件技术和生产率趋势

到了八十年代初期，我们已经达到了拥有专门领域应用程序生成器的较高水平，在从电子报表计算（30.000DMI/MM或更高）、工业过程控制到事务处理第四代语言的相当广泛的领域内达到了更高的生产率。同时，也从汇编语言和基本的批处理操作系统扩大到了在大型和广泛的应用领域内提供集成式工具达到约600 DMI/MM的高级语言，从而使通用处理能力也得到了扩充。按照Brooks的见解，这些功能解决了开发软件中的"偶发性"困难。图5中左边所示的有关领域功能旨在进一步减小软件费用的"实质性"部分。

因此，在1995—2000期间，我们可以看到存在着提高软件生产率的两类机会：一是为更广泛的应用领域提供更好的支持系统，包括完全集成的方法、环境和先进的程序设计语言如Ada；另一种是增加我们能用第四代语言和应用程序生成器的领域数量和范围。可以预言的将来的应用领域（诸如通讯处理，事务处理等），更广泛的过程控制领域(例如航空电子设备、劳务商店生产控制)，和更广泛的面向DBMS领域（如库存控制和生产管理等）将会进一步提高软件生产率。

我们已经看到了软件费用的比重和持续增长对提高软件生产率的强烈要求。这预示着需要谨慎地定义软件生产率。我们现有的量度不能令人完全满意，希望有一个更好的的发展。上述分析还预示了不仅要提高软件生产率而且也要提高软件质量。

对软件生产率指数和软件价值链的分析导出了软件生产率的机会树的定义。按照这种定义，提高生产率的主要机会：

- 通过更好的管理、挑选人员、奖励和工作环境等多方面搜罗精于人员。
- 开发和使用集成化项目支撑环境，使得开发过程部分自动化，且更有效。
- 通过更好的前期辅助手段，风险管理，快速原型，增量型开发，和先进的程序设计技术，特别是信息隐藏技术以减少重复劳动。

办公信息系统中的计算机科学技术问题

王能斌　黄大海　冷嘉玲

（东　南　大　学）

摘要

计算机应用是计算机科学技术发展的动力和目的。在当前诸多计算机应用领域中，最有代表性的是基于知识的系统(KBS)、计算机集成制造系统(CIMS)、办公信息系统(OIS)和实时控制、指挥、通信系统(C^3IS)。它们向计算机科学技术提出一系列富有挑战性的课题，成为当前计算机科学技术研究的热点。本文仅就办公信息系统所提出的计算机科学技术问题，做一概述。

办公信息系统不仅有其技术上的特点，还有其社会的属性；它不仅涉及计算机科学技术，而且还与系统工程、知识工程，人机工程、管理科学、经济学、社会学、人类学、心理学等多种学科有关[1]。本文不拟对此有所论及。办公信息系统与其他计算机应用系统一样，是建立在计算机硬、软件和通信、办公设备发展的基础上。对于这些物质基础，办公信息系统虽有其自身的要求，但与其他系统基本上是共同的。事实上，办公信息系统一般是用这些设备的商品化产品构成的，故本文也不拟对此进行讨论。办公信息系统是一种综合的计算机应用系统，要讨论其发展中的所有问题是困难的，下面择其主要问题，谈谈作者的看法。

1. 联邦式的系统结构 (Federated Architecture)

办公室一般分为多个部门。这些部门往往在地理上是分散的，在组织上既相对独立，又互相联系、彼此合作地工作，而且还可能有派出人员，分布在世界各地。在这样的系统中，数据的来源是分散的，既有供部门自用的私有数据，也有允许其他部门使用的共享数据，甚至允许不同部门拥有重复且不一致的数据。目前已开发的分布式数据库绝大部分在物理上是分布的，在逻辑上是集中的，即把地理上分散的数据综合成一个全局数据模式，由DBA集中管理。这种形式的数

· 通过重用软部件，开发和使用应用程序生成器以及第四代语言，避免软件镀金现象等来减少代码的编写。

作为最后的结论，有一点值得特别强调。在回顾和追踪软件生产率提高的过程中，我们必须注意不要把方法与目的相混淆。提高软件生产率本身并不是目的，它是帮助人们更好地扩大处理信息和做出决策的能力的一种方法。通常帮助别人做些事情时将使我们牵连到各种活动中（如花两个星期帮某人找到一个问题的非软件的有效解答），而这些活动并不增加我们在生产率计分牌上的分数。我们必须时时记住软件生产率这块计分牌只是我们用计算机更好地为人民服务方面的进步的许多尺度中的一个。

参考文献略

〔杨德元、杨　军译自《Computer》1987，№9，王鼎鸣校〕