第 13 章　软件项目管理

在经历了若干个大型软件工程项目的失败之后，人们才逐渐认识到软件项目管理的重要性和特殊性。事实上，这些项目的失-败并不是由于从事软件开发工作的软件工程师无能，正相反，他们之中的绝大多数是当时杰出的技术专家。这些工程项目的失败主要是因为管理不善。

所谓管理就是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。

软件项目管理先于任何技术活动之前开始，并且贯穿于软件的整个生命周期之中。

软件项目管理过程从一组项目计划活动开始，而制定计划的基础是工作量估算和完成期限估算。为了估算项目的工作量和完成期限，首先需要估算软件的规模。

13.1 估算软件规模

13.1.1 代码行技术

代码行技术是比较简单的定量估算软件规模的方法。这种方法依据以往开发类似产品的经验和历史数据，估计实现一个功能所需要的源程序行数。当有以往开发类似产品的历史数据可供参考时，用这种方法估计出的数值还是比较准确的。把实现每个功能所需要的源程序行数累加起来，就可得到实现整个软件所需要的源程序行数。

为了使得对程序规模的估计值更接近实际值，可以由多名有经验的软件工程师分别做出估计。每个人都估计程序的最小规模（a）、最大规模（b）和最可能的规模（m），分别算出这 3 种规模的平均值 $\overline{a}$，$\overline{b}$ 和 $\overline{m}$ 之后，再用下式计算程序规模的估计值：

$$L=\frac{\overline{a}+4\overline{m}+\overline{b}}{6} \qquad (13.1)$$

用代码行技术估算软件规模时，当程序较小时常用的单位是代码行数(LOC)，当程序较大时常用的单位是千行代码数(KLOC)。

代码行技术的主要优点是，代码是所有软件开发项目都有的“产品”，而且很容易计算代码行数。代码行技术的缺点是：源程序仅是软件配置的一个成分，用它的规模代表整个软件的规模似乎不太合理；用不同语言实现同一个软件所需要的代码行数并不相同；这种方法不适用于非过程语言。为了克服代码行技术的缺点，人们又提出了功能点技术。

13.1.2 功能点技术

功能点技术依据对软件信息域特性和软件复杂性的评估结果，估算软件规模。这种方法用功能点(FP)为单位度量软件规模。

1.信息域特性

功能点技术定义了信息域的5个特性，分别是输入项数(Inp)、输出项数(Out)、查询数(Inq)、主文件数(Maf)和外部接口数(Inf)。下面讲述这5个特性的含义。

(1)输入项数：用户向软件输入的项数，这些输入给软件提供面向应用的数据。输入不同于查询，后者单独计数，不计入输入项数中。

(2)输出项数：软件向用户输出的项数，它们向用户提供面向应用的信息，例如，报表和出错信息等。报表内的数据项不单独计数。

(3)查询数：查询即是一次联机输入，它导致软件以联机输出方式产生某种即时响应。

(4)主文件数：逻辑主文件(即数据的一个逻辑组合，它可能是大型数据库的一部分或是一个独立的文件)的数目。

(5)外部接口数：机器可读的全部接口(例如，磁盘或磁带上的数据文件)的数量，用这些接口把信息传送给另一个系统。

2.估算功能点的步骤

用下述3个步骤，可估算出一个软件的功能点数(即软件规模)。

(1)计算未调整的功能点数UFP

首先，把产品信息域的每个特性(即Inp、Out、Inq、Mar和Inf)都分类为简单级、平均级或复杂级，并根据其等级为每个特性分配一个功能点数(例如，一个简单级的输入项分配3个功能点，一个平均级的输入项分配4个功能点，而一个复杂级的输人项分配6个功能点)。

然后，用下式计算未调整的功能点数UFP:

$$UFP = a_1 \times Inp + a_2 \times Out + a_3 \times Inq + a_4 \times Maf + a_5 \times Inf$$

其中，$a_i(1 \leqslant i \leqslant 5)$是信息域特性系数，其值由相应特性的复杂级别决定，如表13.1所示。

特性系数 \ 复杂级别	简单	平均	复杂
输入系数$_1$	3	4	6
输出系数$_2$	4	5	7
查询系数$_3$	3	4	6
文件系数$_4$	7	10	15
接口系数$_5$	5	7	10

表 13.1 信息域特性系数

(2)计算技术复杂性因子 TCF

这一步骤度量 14 种技术因素对软件规模的影响程度。这些因素包括高处理率、性能标准(例如，响应时间)、联机更新等，在表 13.2 中列出了全部技术因素，并用 F；(1≤i≤14)代表这些因素。根据软件的特点，为每个因素分配一个从 0(不存在或对软件规模无影响)到 5(有很大影响)的值。然后，用下式计算技术因素对软件规模的综合影响程度 DI：

序号		技术因素
1		数据通信
2		分布式数据处理
3		性能标准
4		高负荷的硬件
5		高处理率
6		联机数据处理
7		终端用户效率
8		联机更新
9		复杂的计算
10		可重用性
11		安装方便
12		操作方便
13		可移植性
14		可维持性

表 13.2 技术因素

(3)计算功能点数 FP

用下式计算功能点数 FP：

$$FP = UFP \times TCF$$

功能点数与所用的编程语言无关，看起来功能点技术比代码行技术更合理一些。但是，在判断信息域特性复杂级别和技术因素的影响程度时，存在着相当大的主观因素。

13.2 T作量估算

软件估算模型使用由经验导出的公式来预测软件开发工作量，工作量是软件规模(KLOC或FP)的函数，工作量的单位通常是人月(pm)。

支持大多数估算模型的经验数据，都是从有限个项目的样本集中总结出来的，因此，没有一个估算模型可以适用于所有类型的软件和开发环境。

13.2.1 静态单变量模型

这类模型的总体结构形式如下：

$$E = A + B \times (ev)^c$$

其中，A、B和C是由经验数据导出的常数，E是以人月为单位的工作量，ev是估算变量(KLOC或FP)。下面给出几个典型的静态单变量模型。

1.面向KLOC的估算模型

(1)Walston_Felix模型

$$E = 5.2 \times (KLOC)^{0.91}$$

(2)Bailey_Basili模型

$$E = 5.5 + 0.73 \times (KLOC)^{1.16}$$

(3)Boehm简单模型

$$E = 3.2 \times (KLOC)^{1.05}$$

(4)Doty模型(在KLOC>9时适用)

$$E = 5.288 \times (KLOC)^{1.047}$$

2.面向FP的估算模型

(1)Albrecht&Gaffney模型

$$E = -13.39 + 0.0545FP$$

(2)Maston，Barnett和Mellichamp模型

$$E = 585.7 + 15.12FP$$

从上面列出的模型可以看出，对于相同的KLOC或FP值，用不同模型估算将得出不同的结果。主要原因是，这些模型多数都是仅根据若干应用领域中有限个项目的经验数据推导出来的，适用范围有限。因此，必须根据当前项目的特点选择适用的估算模型，并且根据需要适当地调整(例如，修改模型常数)估算模型。

13.2.2 动态多变量模型

动态多变量模型也称为软件方程式，它是根据从4 000多个当代软件项目中收集的生产率数据推导出来的。该模型把工作量看作是软件规模和开发时间这两个变量的函数。动态多变量估算模型的形式如下：

$$E=(LOC\times B^{0.333}/P)^3\times(1/t)^4 \quad (13.2)$$

其中，

E是以人月或人年为单位的工作量；

t是以月或年为单位的项目持续时间；

B是特殊技术因子，它随着对测试、质量保证、文档及管理技术的需求的增加而缓慢增

加，对于较小的程序(KLOC=5～15)，B=0.16，对于超过 70 KLOC 的程序，B=0.39；

P 是生产率参数，它反映了下述因素对工作量的影响：

·总体过程成熟度及管理水平；

·使用良好的软件工程实践的程度；

·使用的程序设计语言的级别；

·软件环境的状态；

·软件项目组的技术及经验；

·应用系统的复杂程度。

开发实时嵌入式软件时，P 的典型值为 2 000；开发电信系统和系统软件时，P = 10 000；对于商业应用系统来说，P=28 000。可以从历史数据导出适用于当前项目的生产率参数值。

从(13.2)式可以看出，开发同一个软件(即 LOC 固定)的时候，如果把项目持续时间延长一些，则可降低完成项目所需的工作量。

13.2.3 COCOM02 模型

COCOMO 是构造性成本模型(constructive cost model)的英文缩写。1981 年 Boehm 在《软件工程经济学》中首次提出了 COCOMO 模型，本书第三版曾对此模型作了介绍。1997 年 Boehm 等人提出的 COCOMO2 模型，是原始的 COCOMO 模型的修订版，它反映了十多年来在成本估计方面所积累的经验。

COCOMO2 给出了 3 个层次的软件开发工作量估算模型，这 3 个层次的模型在估算工作量时，对软件细节考虑的详尽程度逐级增加。这些模型既可以用于不同类型的项目，也可以用于同一个项目的不同开发阶段。这 3 个层次的估算模型分别是：

(1)应用系统组成模型。这个模型主要用于估算构建原型的工作量，模型名字暗示在构建原型时大量使用已有的构件。

(2)早期设计模型。这个模型适用于体系结构设计阶段。

(3)后体系结构模型。这个模型适用于完成体系结构设计之后的软件开发阶段。

下面以后体系结构模型为例，介绍 COCOMO2 模型。该模型把软件开发工作量表示成代码行数(KLOC)的非线性函数：

$$E = a \times KLOC^{b} \times \prod_{i=1}^{17} f_i$$

其中，

E 是开发工作量(以人月为单位)，

a 是模型系数，

KLOC 是估计的源代码行数(以千行为单位)，

b 是模型指数，

f_i (I=1～17)是成本因素。

每个成本因素都根据它的重要程度和对工作量影响大小被赋予一定数值(称为工作量系数)。这些成本因素对任何一个项目的开发工作量都有影响，即使不使用 COCOMO2 模型估算工作量，也应该重视这些因素。Boehm 把成本因素划分成产品因素、平台因素、人员因素和项目因素等 4 类。

表 13.3 列出了 COCOMO2 模型使用的成本因素及与之相联系的工作量系数。与原始的 COCOMO 模型相比，COCOMO2 模型使用的成本因素有下述变化，这些变化反映了在过去十几年中软件行业取得的巨大进步：

成本因素	级别					
	甚低	低	正常	高	甚高	特高
产品因素						
要求的可靠性	0.75	0.88	1.00	1.15	1.39	
数据库规模		0.93	1.00	1.09	1.19	
产品复杂程度	0.75	0.88	1.00	1.15	1.30	1.66
要求的可重用性		0.91	1.00	1.14	1.29	1.49
需要的文档量	0.89	0.95	1.00	1.06	1.13	
平台因素						
执行时间约束			1.00	1.11	1.31	1.67
主存约束			1.00	1.06	1.21	1.57
平台变动		0.87	1.00	1.15	1.30	
人员因素						
分析员能力	1.50	1.22	1.00	0.83	0.67	
程序员能力	1.37	1.16	1.00	0.87	0.74	
应用领域经验	1.22	1.10	1I 00	0.89	0.81	
平台经验	1.24	1.10	1.00	0.92	0.84	
语言和工具经验	1.25	1.12	1.00	0.88	0.81	
人员连续性	1.24	1.10	1.00	0.92	0.84	
项目因素						
使用软件工具	1.24	1.12	1.00	0.86	0.72	
多地点开发	1.25	1.10	1.00	0.92	0.84	0.78
要求的开发进度	1.29	1.10	1.00	1.00	1.00	

表 13.3 成本因素及工作量系数

(1)新增加了 4 个成本因素，它们分别是要求的可重用性、需要的文档量、人员连续性(即人员稳定程度)和多地点开发。这个变化表明，这些因素对开发成本的影响日益增加。

(2)略去了原始模型中的2个成本因素(计算机切换时间和使用现代程序设计实践)。现在，开发人员普遍使用工作站开发软件，批处理的切换时间已经不再是问题。而“现代程序设计实践”已经发展成内容更广泛的“成熟的软件工程实践”的概念，并且在COCOMO2工作量方程的指数6中考虑了这个因素的影响。

(3)某些成本因素(分析员能力、平台经验、语言和工具经验)对生产率的影响(即工作量系数最大值与最小值的比率)增加了，另一些成本因素(程序员能力)的影响减小了。

为了确定工作量方程中模型指数6的值，原始的COCOMO模型把软件开发项目划分成组织式、半独立式和嵌入式这样3种类型，并指定每种项目类型所对应的6值(分别是1.05，1.12和1.20)。COCOMO2采用了更加精细得多的6分级模型，这个模型使用5个分级因素W_i(1≤i≤5)，其中每个因素都划分成从甚低(W_i=5)到特高(W_i=0)的6个级别，然后用下式计算6的数值:

$$b = 1.01 + 0.01 \times \sum_{i=1}^{5} W_i$$

因此，b的取值范围为1.01～1.26。显然，这种分级模式比原始COCOMO模型的分级模式更精细、更灵活。

COCOMO2使用的5个分级因素如下所述:

(1)项目先例性。这个分级因素指出，对于开发组织来说该项目的新奇程度。诸如开发类似系统的经验，需要创新体系结构和算法，以及需要并行开发硬件和软件等因素的影响，都体现在这个分级因素中。

(2)开发灵活性。这个分级因素反映出，为了实现预先确定的外部接口需求及为了及早开发出产品而需要增加的工作量。

(3)风险排除度。这个分级因素反映了重大风险已被消除的比例。在多数情况下，这个比例和指定了重要模块接口(即选定了体系结构)的比例密切相关。

(4)项目组凝聚力。这个分级因素表明了开发人员相互协作时可能存在的困难。这个因素反映了开发人员在目标和文化背景等方面相一致的程度，以及开发人员组成一个小组工作的经验。

(5)过程成熟度。这个分级因素反映了按照能力成熟度模型(见13.7节)度量出的项目组织的过程成熟度。

在原始的COCOMO模型中，仅粗略地考虑了前两个分级因素对指数6之值的影响。

工作量方程中模型系数口的典型值为3.0，在实际工作中应该根据历史经验数据确定一个适合本组织当前开发的项目类型的数值。

13.3 进度计划

不论从事哪种技术性项目，实际情况都是，在实现一个大目标之前往往必须完成数以百计的小任务(也称为作业)。这些任务中有一些是处于“关键路径”(见13.3.5节)之外的，其完成时间如果没有严重拖后，就不会影响整个项目的完成时间；其他任务则处于关键路径之中，如果这些“关键任务”的进度拖后，则整个项目的完成日期就会拖后，管理人员应该高度关注关键任务的进展情况。

没有一个普遍适用于所有软件项目的任务集合，因此，一个有效的软件过程应该定义一个适用于当前项目的任务集合。一个任务集合包括一组软件工程工作任务、里程碑和可交

付的产品。为一个项目所定义的任务集合，必须包括为获得高质量的软件产品而应该完成的所有任务，但是同时又不能让项目组承担不必要的工作。

项目管理者的目标是定义全部项目任务，识别出关键任务，跟踪关键任务的进展状况，以保证能及时发现拖延进度的情况。为达到上述目标，管理者必须制定一个足够详细的进度表，以便监督项目进度并控制整个项目。

软件项目的进度安排是这样一种活动，它通过把工作量分配给特定的软件工程任务并规定完成各项任务的起止日期，从而将估算出的项目工作量分布于计划好的项目持续期内。进度计划将随着时间的流逝而不断演化。在项目计划的早期，首先制定一个宏观的进度安排表，标识出主要的软件工程活动和这些活动影响到的产品功能。随着项目的进展，把宏观进度表中的每个条目都精化成一个详细进度表，从而标识出完成一个活动所必须实现的一组特定任务，并安排好了实现这些任务的进度。

13.3.1 估算开发时间

估算出完成给定项目所需的总工作量之后，接下来需要回答的问题就是：用多长时间才能完成该项目的开发工作?对于一个估计工作量为 20 人月的项目，可能想出下列几种进度表:

·1 个人用 20 个月完成该项目;

·4 个人用 5 个月完成该项目;

·20 个人用 1 个月完成该项目。

但是，这些进度表并不现实，实际上软件开发时间与从事开发工作的人数之间并不是简单的反比关系。

通常，成本估算模型也同时提供了估算开发时间 T 的方程。与工作量方程不同，各种模型估算开发时间的方程很相似，例如:

(1)Walston_Felix 模型

$$T=2.5E^{0.35}$$

(2)原始的 COCOMO 模型

$$T=2.5E^{0.38}$$

(3)COCOMO2 模型

$$T=3.0E^{0.33+0.2\times(b-1.01)}$$

(4)Putnam 模型

$$T=2.4E^{1/3}$$

其中，

E 是开发工作量(以人月为单位)，

T 是开发时间(以月为单位)。

用上列方程计算出的 T 值，代表正常情况下的开发时间。客户往往希望缩短软件开发时间，显然，为了缩短开发时间应该增加从事开发工作的人数。但是，经验告诉我们，随着开发小组规模扩大，个人生产率将下降，以致开发时间与从事开发工作的人数并不成反比关系。出现这种现象主要有下述两个原因:

·当小组变得更大时，每个人需要用更多时间与组内其他成员讨论问题、协调工作，因此增加了通信开销。

·如果在开发过程中增加小组人员，则最初一段时间内项目组总生产率不仅不会提

高反而会下降。这是因为新成员在开始时不仅不是生产力，而且在他们学习期间还需要花费小组其他成员的时间。

综合上述两个原因，存在被称为 Brooks 规律的下述现象：向一个已经延期的项目增加人力，只会使得它更加延期。

下面让我们研究项目组规模与项目组总生产率的关系。

项目组成员之间的通信路径数，由项目组人数和项目组结构决定。如果项目组共有 P 名组员，每个组员必须与所有其他组员通信以协调开发活动，则通信路径数为 P(P-1)／2。如果每个组员只需与另外一个组员通信，则通信路径数为 P-1。通信路径数少于 P-1 是不合理的，因为那将导致出现与任何人都没有联系的组员。

因此，通信路径数大约在 P～P^2／2 的范围内变化。也就是说，在一个层次结构的项目组中，通信路径数为 P^a，其中 1<a<2。

对于某一个组员来说，他与其他组员通信的路径数在 1～(P-1)的范围内变化。如果不与任何人通信时个人生产率为 L，而且每条通信路径导致生产率减少 Z，则组员个人平均生产率为

$$L_r = L - 1(P-1)^r \qquad (13.5)$$

其中，r 是对通信路径数的度量，0<r≤1(假设至少有一名组员需要与一个以上的其他组员通信，因此 r>0)。

对于一个规模为 P 的项目组，从(13.5)式导出项目组的总生产率为

$$L_{tot}=P(L-1(P-1)^r) \qquad (13.6)$$

对于给定的一组 L，1 和 r 的值，总生产率 L_{tot} 是项目组规模 P 的函数。随着 P 值增加，L_{tot} 将从 0 增大到某个最大值，然后再下降。因此，存在一个最佳的项目组规模 P_{opt} 这个规模的项目组其总生产率最高。

让我们举例说明项目组规模与生产率的关系。假设个人最高生产率为 500LOC／月(即 L=500)，每条通信路径导致生产率下降 10%(即 1=50)。如果每个组员都必须与组内所有其他组员通信(r=1)，则项目组规模与生产率的关系列在表 13.4 中，可见，在这种情况下项目组的最佳规模是 5.5 人，即 P_{opt}=5.5。

项目组规模	个人生产率	总生产率
1	500	500
2	450	900
3	400	1200
4	350	1400
5	300	1500
5.5	275	1512
6	250	1500
7	200	1400
8	150	1200

表 13.4 项目组规模对生产率的影响

事实上，做任何事情都需要时间，我们不可能用“人力换时间”的办法无限缩短一个软
件的开发时间。Boehm 根据经验指出，软件项目的开发时间最多可以减少到正常开发时间的75%。如果要求一个软件系统的开发时间过短，则开发成功的概率几乎为零。

13.3.2 Gantt 图

Gantt(甘特)图是历史悠久、应用广泛的制定进度计划的工具，下面通过一个非常简单的例子介绍这种工具。

假设有一座陈旧的矩形木板房需要重新油漆。这项工作必须分 3 步完成：首先刮掉旧漆，然后刷上新漆，最后清除溅在窗户上的油漆。假设一共分配了 15 名工人去完成这项工作，然而工具却很有限：只有 5 把刮旧漆用的刮板，5 把刷漆用的刷子，5 把清除溅在窗户上的油漆用的小刮刀。怎样安排才能使工作进行得更有效呢?

一种做法是首先刮掉四面墙壁上的旧漆，然后给每面墙壁都刷上新漆，最后清除溅在每个窗户上的油漆。显然这是效率最低的做法，因为总共有 15 名工人，然而每种工具却只有 5 件，这样安排工作在任何时候都有 10 名工人闲着没活干。

读者可能已经想到，应该采用“流水作业法”，也就是说，首先由 5 名工人用刮板刮掉第 1 面墙上的旧漆(这时其余 10 名工人休息)，当第 1 面墙刮净后，另外 5 名工人立即用刷子给这面墙刷新漆(与此同时拿刮板的 5 名工人转去刮第 2 面墙上的旧漆)，一旦刮旧漆的工人转到第 3 面墙而且刷新漆的工人转到第 2 面墙以后，余下的 5 名工人立即拿起刮刀去清除溅在第 1 面墙窗户上的油漆，……。这样安排每个工人都有活干，因此能够在较短的时间内完成任务。

假设木板房的第 2、4 两面墙的长度比第 1、3 两面墙的长度长一倍，此外，不同工作需要用的时间长短也不同，刷新漆最费时间，其次是刮旧漆，清理(即清除溅在窗户上的油漆)需要的时间最少。表 13.5 列出了估计每道工序需要用的时间。可以使用图 13.1 中的 Gantt 图描绘上述流水作业过程：在时间为零时开始刮第 1 面墙上的旧漆，两小时后刮旧漆的工人转去刮第 2 面墙，同时另 5 名工人开始给第 1 面墙刷新漆，每当给一面墙刷完新漆之后，第 3 组的 5 名工人立即清除溅在这面墙窗户上的漆。从图 13.1 可以看出 12 小时后刮完所有旧漆，20 小时后完成所有墙壁的刷漆工作，再过 2 小时后清理工作结
束。因此全部工程在 22 小时后结束，如果用前述的第一种做法，则需要 36 小时。

工作 / 墙壁	刮旧漆	刷新漆	清理
1 或 3	2	3	1
2 或 4	4	6	2

表 13.5 各道工序估计需用的时间 (小时)

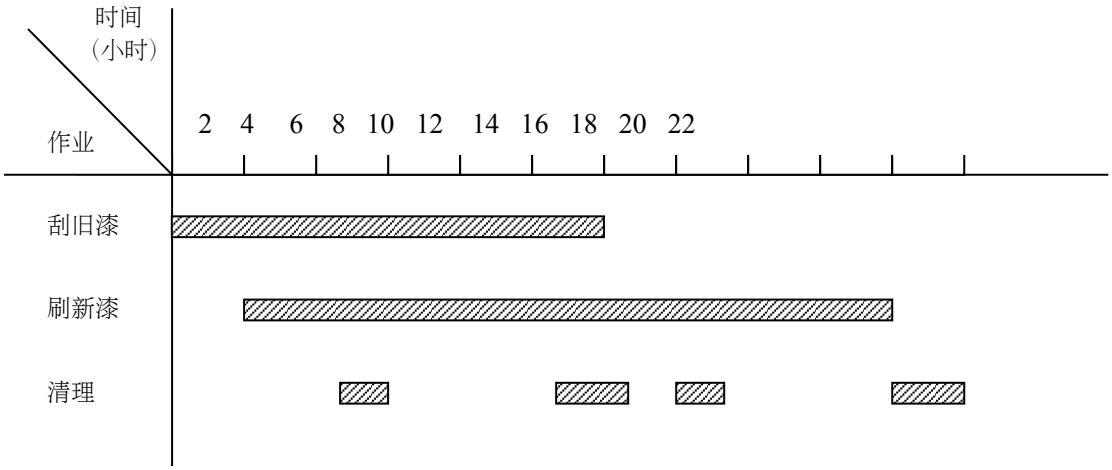


图 13.1 旧木板房刷漆工程的 Gantt 图

13.3.3 工程网络

上一小节介绍的 Gantt 图能很形象地描绘任务分解情况，以及每个子任务(作业)的开始时间和结束时间，因此是进度计划和进度管理的有力工具。它具有直观简明和容易掌握、容易绘制的优点，但是 Gantt 图也有 3 个主要缺点：

(1)不能显式地描绘各项作业彼此间的依赖关系；

(2)进度计划的关键部分不明确，难于判定哪些部分应当是主攻和主控的对象；

(3)计划中不

图 13.1 旧木版房刷漆工程的 Gantt 图

当把一个工程项目分解成许多子任务，并且它们彼此间的依赖关系又比较复杂时，仅仅用 Gantt 图作为安排进度的工具是不够的，不仅难于做出既节省资源又保证进度的计划，而且还容易发生差错。

工程网络是制定进度计划时另一种常用的图形工具，它同样能描绘任务分解情况以及每项作业的开始时间和结束时间，此外，它还显式地描绘各个作业彼此间的依赖关系。因此，工程网络是系统分析和系统设计的强有力的工具。

在工程网络中用箭头表示作业(例如，刮旧漆，刷新漆，清理等)，用圆圈表示事件(一项作业开始或结束)。注意，事件仅仅是可以明确定义的时间点，它并不消耗时间和资源。作业通常既消耗资源又需要持续一定时间。图 13.2 是旧木板房刷漆工程的工程网络。图中表示刮第 1 面墙上旧漆的作业开始于事件 1，结束于事件 2。用开始事件和结束事件的编号标识一个作业，因此“刮第 1 面墙上旧漆”是作业 1—2。

在工程网络中的一个事件，如果既有箭头进入又有箭头离开，则它既是某些作业的结

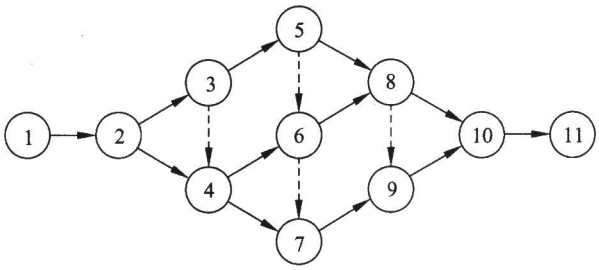


图 13.2　旧木板房刷漆工程的工程网络

图中：1 一 z 刮第 1 面墙上的旧漆；2—3 刮第 2 面墙上的旧漆；2—4 给第 1 面墙刷新漆；3—5 刮第 3 面墙上旧漆；4—6 给第 2 面墙刷新漆；4—7 清理第 1 面墙窗户；5—8 刮第 4 面墙上旧漆；6—8 给第 3 面墙刷新漆；7—9 清理第 2 面墙窗户；8—10 给第 4 面墙刷新漆；9—10 清理第 3 面墙窗户；10—11 清理第 4 面

墙窗户；虚拟作业：3—4；5—6；6—7；8—9。

束又是另一些作业的开始。例如，图 13.2 中事件 2 既是作业 1—2(刮第 1 面墙上的旧漆)的结束，又是作业 2—3(刮第 2 面墙上旧漆)和作业 2—4(给第 1 面墙刷新漆)的开始。也就是说，只有第 1 面墙上的旧漆刮完之后，才能开始刮第 2 面墙上旧漆和给第 1 面墙刷新漆这两个作业。因此，工程网络显式地表示了作业之间的依赖关系。

在图 13.2 中还有一些虚线箭头，它们表示虚拟作业，也就是事实上并不存在的作业。引入虚拟作业是为了显式地表示作业之间的依赖关系。例如，事件 4 既是给第 1 面墙刷新漆结束，又是给第 2 面墙刷新漆开始(作业 4—6)。但是，在开始给第 2 面墙刷新漆之前，不仅必须已经给第 1 面墙刷完了新漆，而且第 2 面墙上的旧漆也必须已经刮净(事件 3)。也就是说，在事件 3 和事件 4 之间有依赖关系，或者说在作业 2—3(刮第 2 面墙上旧漆)和作业 4—6(给第 2 面墙刷新漆)之间有依赖关系，虚拟作业 3—4 明确地表示了这种依赖关系。注意，虚拟作业既不消耗资源也不需要时间。请读者研究图 13.2，参考图下面对各项作业的描述，解释引入其他虚拟作业的原因。

13.3.4 估算工程进度

画出类似图 13.2 那样的工程网络之后，系统分析员就可以借助它的帮助估算工程进度了。为此需要在工程网络上增加一些必要的信息。

首先，把每个作业估计需要使用的时间写在表示该项作业的箭头上方。注意，箭头长度和它代表的作业持续时间没有关系，箭头仅表示依赖关系，它上方的数字才表示作业的持续时间。

其次，为每个事件计算下述两个统计数字：最早时刻 EET 和最迟时刻 LET。这两个数字将分别写在表示事件的圆圈的右上角和右下角，如图 13.3 左下角的符号所示。

事件的最早时刻是该事件可以发生的最早时间。通常工程网络中第一个事件的最早时刻定义为零，其他事件的最早时刻在工程网络上从左至右按事件发生顺序计算。计算最早时刻 EET 使用下述 3 条简单规则：

(1)考虑进入该事件的所有作业；

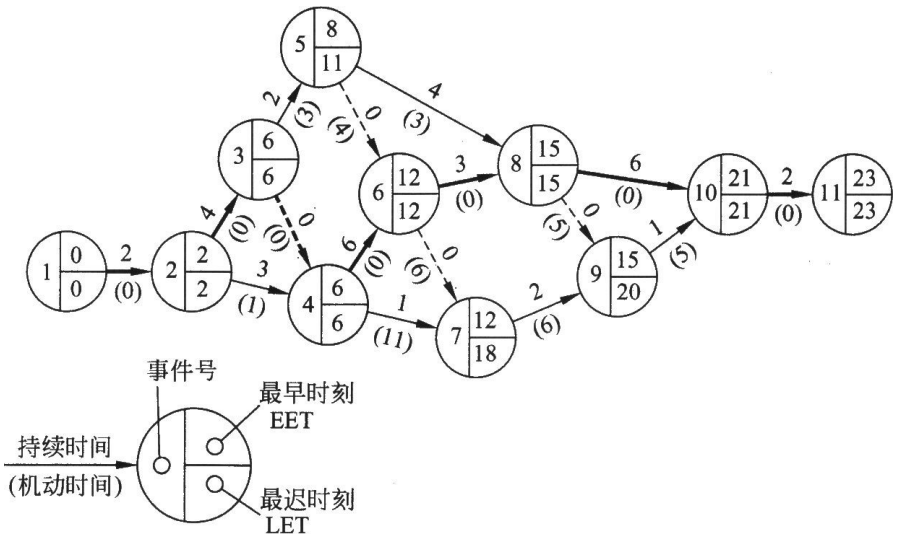


图 13.3　旧木板房刷漆工程的完整的工程网络
(粗线箭头是关键路径)

(2)对于每个作业都计算它的持续时间与起始事件的 EET 之和;

(3)选取上述和数中的最大值作为该事件的最早时刻 EET。

例如，从图 13.2 可以看出事件 2 只有一个作业(作业 1-2)进入，就是说，仅当作业 1-2 完成时事件 2 才能发生，因此事件 2 的最早时刻就是作业 1-2 最早可能完成的时刻。定义事件 1 的最早时刻为零，据估计，作业 1-2 的持续时间为 2 小时，也就是说，作业 1-2 最早可能完成的时刻为 2，因此，事件 2 的最早时刻为 2。同样，只有一个作业(作业 2-3)进入事件 3，这个作业的持续时间为 4 小时，所以事件 3 的最早时刻为 2+4=6。

事件 4 有两个作业(2-4 和 3-4)进入，只有这两个作业都完成之后，事件 4 才能出现(事件 4 代表上述两个作业的结束)。已知事件 2 的最早时刻为 2，作业 2-4 的持续时间为 3 小时；事件 3 的最早时刻为 6，作业 3-4(这是一个虚拟作业)的持续时间为 0，按照上述三条规则，可以算出事件 4 的最早时刻为

$$EET = \max\{2+3,\ 6+0\} = 6$$

按照这种方法，不难沿着工程网络从左至右顺序算出每个事件的最早时刻，计算结果标在图 13.3 的工程网络中(每个圆圈内右上角的数字)。

事件的最迟时刻是在不影响工程竣工时间的前提下，该事件最晚可以发生的时刻。按惯例，最后一个事件(212 程结束)的最迟时刻就是它的最早时刻。其他事件的最迟时刻在工程网络上从右至左按逆作业流的方向计算。计算最迟时刻 LET 使用下述 3 条规则:

(1)考虑离开该事件的所有作业;

(2)从每个作业的结束事件的最迟时刻中减去该作业的持续时间;

(3)选取上述差数中的最小值作为该事件的最迟时刻 LET。

例如，按惯例图 13.3 中事件 11 的最迟时刻和最早时刻相同，都是 23。逆作业流方向接下来应该计算事件 10 的最迟时刻，离开这个事件的只有作业 10—11，该作业的持续时间为 2 小时，它的结束事件(事件 11)的 LET 为 23，因此，事件 10 的最迟时刻为

$$LET=23-2=21$$

类似地，事件 9 的最迟时刻为

$$LET=21-1=20$$

事件 8 的最迟时刻为

LET=min{21-6，20-0}=15

图 13.3 中每个圆圈内右下角的数字就是该事件的最迟时刻。

13.3.5 关键路径

图 13.3 中有几个事件的最早时刻和最迟时刻相同，这些事件定义了关键路径，在图中关键路径用粗线箭头表示。关键路径上的事件(关键事件)必须准时发生，组成关键路径的作业(关键作业)的实际持续时间不能超过估计的持续时间，否则工程就不能准时结束。

工程项目的管理人员应该密切注视关键作业的进展情况，如果关键事件出现的时间比预计的时间晚，则会使最终完成项目的时间拖后；如果希望缩短工期，只有往关键作业中增加资源才会有效果。

13.3.6 机动时间

不在关键路径上的作业有一定程度的机动余地——实际开始时间可以比预定时间晚一些，或者实际持续时间可以比预定的持续时间长一些，而并不影响工程的结束时间。一个作业可以有的全部机动时间等于它的结束事件的最迟时刻减去它的开始事件的最早时刻，再减去这个作业的持续时间：

$$\text{机动时间}=(\text{LET})_{\text{结束}}-(\text{EET})_{\text{开始}}-\text{持续时间}$$

对于前述油漆旧木板房的例子，计算得到的非关键作业的机动时间列在表 13.6 中。

作业	LET(结束)	EET(开始)	持续时间	机动时间
2—4	6	2	3	1
3—5	11	6	2	3
4—7	18	6	1	11
5—6	12	8	0	4
5—8	15	8	4	3
6—7	18	12	0	6
7—9	20	12	2	6
8—9	20	15	0	5
9—10	21	15	1	5

表 13.6 旧木板房刷漆网络中的机动时间

在工程网络中每个作业的机动时间写在代表该项作业的箭头下面的括弧里(参看图 13.3)。

在制定进度计划时仔细考虑和利用工程网络中的机动时间，往往能够安排出既节省资源又不影响最终竣工时间的进度表。例如，研究图 13.3(或表 13.6)可以看出，清理前三面墙窗户的作业都有相当多机动时间，也就是说，这些作业可以晚些开始或者持续时间长一些(少用一些资源)，并不影响竣工时间。此外，刮第 3、第 4 面墙上旧漆和给第 1 面墙刷新漆的作业也都有机动时间，而且这后三项作业的机动时间之和大于清理前三面墙窗户需要用的工作时间。因此，有可能仅用 10 名工人在同样时间内(23 小时)完成旧木板房刷漆工程。进一步研究图 13.3 中的工程网络可以看出，确实能够只用 10 名工人在同样时间内完成这项任务，而且可以安排出几套不同的进度计划，都可以既减少 5 名工人又不影响竣工时间。在图 13.4 中的 Gantt 图描绘了其中的一种方案。

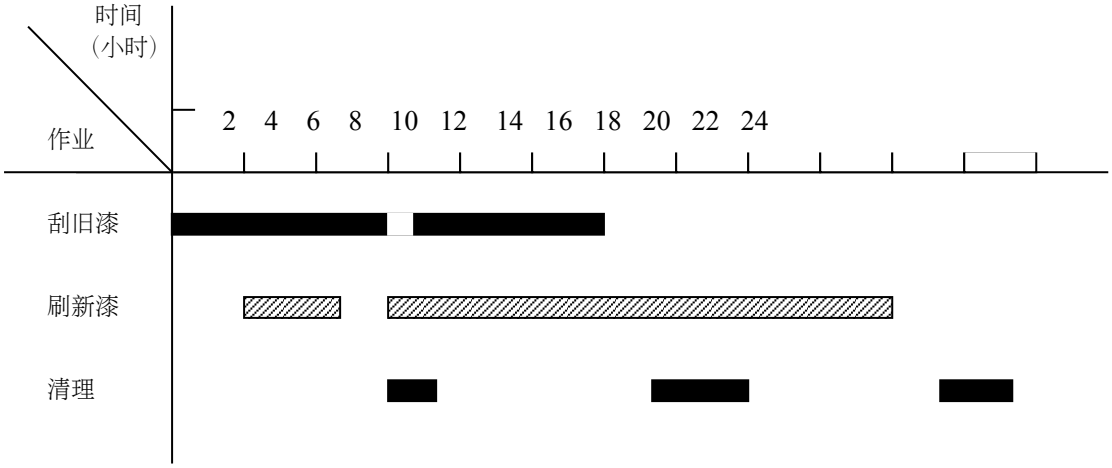


图 13.1 旧木板房刷漆工程改进的 Gantt 图之一

图中：粗实线代表由甲组工人完成的作业；斜划线代表由乙组工人完成的作业

图 13.4 的方案不仅比图 13.1 的方案明显节省人力，而且改正了图 13.1 中的一个错误：因为给第 2 面墙刷新漆的作业 4-6 不仅必须在给第 1 面墙刷完新漆之后(作业 2-4 结束)，而且还必须在把第 2 面墙上的旧漆刮净之后(作业 2-3 和虚拟作业 3-4 结束)才能开始，所以给第 1 面墙刷完新漆之后不能立即开始给第 2 面墙刷新漆的作业，需等到把第 2 面墙上旧漆刮净之后才能开始，也就是说，全部工程需要 23 个小时而不是 22 个小时。

这个简单例子明显说明了工程网络比 Gantt 图优越的地方：它显式地定义事件及作业之间的依赖关系，Gantt 图只能隐含地表示这种关系。但是 Gantt 图的形式比工程网络更简单更直观，为更多的人所熟悉，因此，应该同时使用这两种工具制订和管理进度计划，使它们互相补充取长补短。

以上通过旧木板房刷新漆工程的简单例子，介绍了制订进度计划的两个重要工具和方法。软件工程项目虽然比这个简单例子复杂得多，但是计划和管理的基本方法仍然是自顶向下分解，也就是把项目分解为若干个阶段，每个阶段再分解成许多更小的任务，每个任务又可进一步分解为若干个步骤等等。这些阶段、任务和步骤之间有复杂的依赖关系，因此，工程网络和 Gantt 图同样是安排进度和管理工程进展情况的强有力的工具。第 13.2 节中介绍的工作量估计技术可以帮助我们估计每项任务的工作量，根据人力分配情况，可以进一步确定每项任务的持续时间。从这些基本数据出发，根据作业之间的依赖关系，利用工程网络和 Gantt 图可以制定出合理的进度计划，并且能够科学地管理软件开发工程的进展情况。

13.4 人员组织

软件项目成功的关键是有高素质的软件开发人员。然而大多数软件的规模都很大，单个软件开发人员无法在给定期限内完成开发工作，因此，必须把多名软件开发人员合理地组织起来，使他们有效地分工协作共同完成开发工作。

为了成功地完成软件开发工作，项目组成员必须以一种有意义且有效的方式彼此交互和通信。如何组织项目组是一个重要的管理问题，管理者应该合理地组织项目组，使项目组有较高生产率，能够按预定的进度计划完成所承担的工作。经验表明，项目组组织得越好，其生产率越高，而且产品质量也越好。

除了追求更好的组织方式之外，每个管理者的目标都是建立有凝聚力的项目组。一个有高度凝聚力的小组，由一批团结得非常紧密的人组成，他们的整体力量大于个体力量的总和。一旦项目组具有了凝聚力，成功的可能性就大大增加了。

现有的软件项目组的组织方式很多，通常，组织软件开发人员的方法，取决于所承担的项目的特点、以往的组织经验以及管理者的看法和喜好。下面介绍3种典型的组织方式。

13.4.1 民主制程序员组

民主制程序员组的一个重要特点是，小组成员完全平等，享有充分民主，通过协商做出技术决策。因此，小组成员之间的通信是平行的，如果小组内有咒个成员，则可能的通信信道共有n(n一1)／2条。

程序设计小组的人数不能太多，否则组员间彼此通信的时间将多于程序设计时间。此外，通常不能把一个软件系统划分成大量独立的单元，因此，如果程序设计小组人数太多，则每个组员所负责开发的程序单元与系统其他部分的界面将是复杂的，不仅出现接口错误的可能性增加，而且软件测试将既困难又费时间。

一般说来，程序设计小组的规模应该比较小，以2～8名成员为宜。如果项目规模很大，用一个小组不能在预定时间内完成开发任务，则应该使用多个程序设计小组，每个小组承担工程项目的一部分任务，在一定程度上独立自主地完成各自的任务。系统的总体设计应该能够保证由各个小组负责开发的各部分之间的接口是良好定义的，并且是尽可能简单的。

小组规模小，不仅可以减少通信问题，而且还有其他好处。例如，容易确定小组的质量标准，而且用民主方式确定的标准更容易被大家遵守；组员间关系密切，能够互相学习等等。

民主制程序员组通常采用非正式的组织方式，也就是说，虽然名义上有一个组长，但是他和组内其他成员完成同样的任务。在这样的小组中，由全体讨论协商决定应该完成的工作，并且根据每个人的能力和经验分配适当的任务。

民主制程序员组的主要优点是，组员们对发现程序错误持积极的态度，这种态度有助于更快速地发现错误，从而导致高质量的代码。

民主制程序员组的另一个优点是，组员们享有充分民主，小组有高度凝聚力，组内学术空气浓厚，有利于攻克技术难关。因此，当有难题需要解决时，也就是说，当所要开发的软件的技术难度较高时，采用民主制程序员组是适宜的。

如果组内多数成员是经验丰富技术熟练的程序员，那么上述非正式的组织方式可能会

非常成功。在这样的小组内组员享有充分民主，通过协商，在自愿的基础上作出决定，因此能够增强团结、提高工作效率。但是，如果组内多数成员技术水平不高，或是缺乏经验的新手，那么这种非正式的组织方式也有严重缺点：由于没有明确的权威指导开发工程的进行，组员间将缺乏必要的协调，最终可能导致工程失败。

为了使少数经验丰富、技术高超的程序员在软件开发过程中能够发挥更大作用，程序设计小组也可以采用下一小节中介绍的另外一种组织形式。

13.4.2 主程序员组

美国 IBM 公司在 20 世纪 70 年代初期开始采用主程序员组的组织方式。采用这种组织方式主要出于下述几点考虑：

(1)软件开发人员多数比较缺乏经验；

(2)程序设计过程中有许多事务性的工作，例如，大量信息的存储和更新；

(3)多渠道通信很费时间，将降低程序员的生产率。

主程序员组用经验多、技术好、能力强的程序员作为主程序员，同时，利用人和计算机在事务性工作方面给主程序员提供充分支持，而且所有通信都通过一两个人进行。这种组织方式类似于外科手术小组的组织：主刀大夫对手术全面负责，并且完成制订手术方案、开刀等关键工作，同时又有麻醉师、护士长等技术熟练的专门人员协助和配合他的工作。此外，必要时手术组还要请其他领域的专家(例如，心脏科医生或妇产科医生)协助。

上述比喻突出了主程序员组的两个重要特性：

(1)专业化。该组每名成员仅完成他们受过专业训练的那些工作。

(2)层次性。主刀大夫指挥每名组员工作，并对手术全面负责。

当时，典型的主程序员组的组织形式如图 13.5 所示。该组由主程序员、后备程序员、编程秘书以及 1～3 名程序员组成。在必要的时候，该组还有其他领域的专家协助。

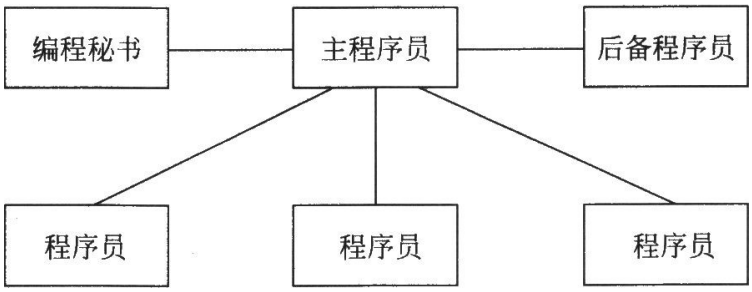


图 13.5 主程序员组的结构

主程序员组核心人员的分工如下所述：

(1)主程序员既是成功的管理人员又是经验丰富、技术好、能力强的高级程序员，负责体系结构设计和关键部分(或复杂部分)的详细设计，并且负责指导其他程序员完成详细设计和编码工作。如图 13.5 所示，程序员之间没有通信渠道，所有接口问题都由主程序员处理。主程序员对每行代码的质量负责，因此，他还要对组内其他成员的工作成果进行复查。

(2)后备程序员也应该技术熟练而且富于经验，他协助主程序员工作并且在必要时(例如，主程序员生病、出差或“跳槽”)接替主程序员的工作。因此，后备程序员必须在各方面都和主程序员一样优秀，并且对本项目的了解也应该和主程序员一样深入。平时，后备程序员的工作主要是，设计测试方案、分析测试结果及独立于设计过程的其他工作。

(3)编程秘书负责完成与项目有关的全部事务性工作，例如，维护项目资料库和项目文

档，编译、链接、执行源程序和测试用例。

注意，上面介绍的是20世纪70年代初期的主程序员组组织结构，现在的情况已经和当时大不相同了，程序员已经有了自己的终端或工作站，他们自己完成代码的输入、编辑、编译、链接和测试等工作，无须由编程秘书统一做这些工作。典型的主程序员组的现代形式将在下一小节介绍。

虽然图13.5所示的主程序员组的组织方式说起来有不少优点，但是，它在许多方面却是不切实际的。

首先，如前所述，主程序员应该是高级程序员和优秀管理者的结合体。承担主程序员工作需要同时具备这两方面的才能，但是，在现实社会中这样的人才并不多见。通常，既缺乏成功的管理者也缺乏技术熟练的程序员。

其次，后备程序员更难找。人们期望后备程序员像主程序员一样优秀，但是，他们必须坐在“替补席”上，拿着较低的工资等待随时接替主程序员的工作。几乎没有一个高级程序员或高级管理人员愿意接受这样的工作。

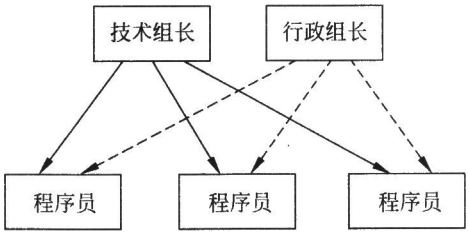
第三，编程秘书也很难找到。专业的软件技术人员一般都厌烦日常的事务性工作，但是，人们却期望编程秘书整天只干这类工作。

我们需要一种更合理、更现实的组织程序员组的方法，这种方法应该能充分结合民主制程序员组和主程序员组的优点，并且能用于实现更大规模的软件产品。

13.4.3 现代程序员组

民主制程序员组的一个主要优点，是小组成员都对发现程序错误持积极、主动的态度。但是，使用主程序员组的组织方式时，主程序员对每行代码的质量负责，因此，他必须参与所有代码审查工作。由于主程序员同时又是负责对小组成员进行评价的管理员，他参与代码审查工作就会把所发现的程序错误与小组成员的工作业绩联系起来，从而造成小组成员出现不愿意发现错误的心理。

解决上述问题的方法是，取消主程序员的大部分行政管理工作。前面已经指出，很难找到既是高度熟练的程序员又是成功的管理员的人，取消主程序员的行政管理工作，不仅解决了小组成员不愿意发现程序错误的心理问题，也使得寻找主程序员的人选不再那么困难。于是，实际的“主程序员”应该由两个人共同担任：一个技术负责人，负责小组的技术活动；一个行政负责人，负责所有非技术性事务的管理决策。这样的组织结构如图13.6所示。技术组长自然要参与全部代码审查工作，因为他要对代码的各方面质量负责；相反，行政组长不可以参与代码审查工作，因为他的职责是对程序员的业绩进行评价。行政组长应该在常规调度会议上了解每名组员的技术能力和工作业绩。



图例：

图 13.6 现代程序员组的结构

在开始工作之前明确划分技术组长和行政组长的管理权限是很重要的。但是，即使已经做了明确分工，有时也会出现职责不清的矛盾。例如，考虑年度休假问题，行政组长有权批准某个程序员休年假的申请，因为这是一个非技术性问题，但是技术组长可能马上否决了这个申请，因为已经接近预定的项目结束日期，目前人手非常紧张。解决这类问题的办法是求助于更高层的管理人员，对行政组长和技术组长都认为是属于自己职责范围内的事务，制定一个处理方案。

由于程序员组成员人数不宜过多，当软件项目规模较大时，应该把程序员分成若干个小组，采用图 13.7 所示的组织结构。该图描绘的是技术管理组织结构，非技术管理组织结构与此类似。由图可以看出，产品开发作为一个整体是在项目经理的指导下进行的，程序员向他们的组长汇报工作，而组长则向项目经理汇报工作。当产品规模更大时，可以适当增加中间管理层次。

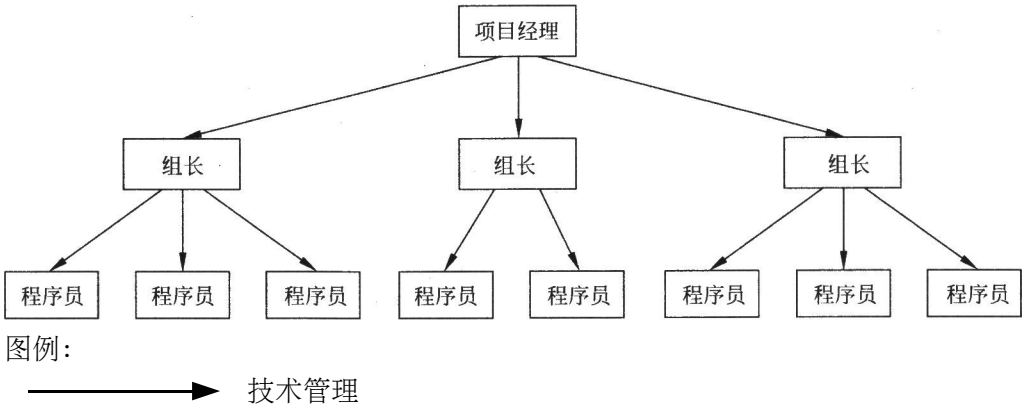


图 13.7 大型项目的技术管理组织结构

把民主制程序员组和主程序员组的优点结合起来的另一种方法，是在合适的地方采用分散做决定的方法，如图 13.8 所示。这样做有利于形成畅通的通信渠道，以便充分发挥每个程序员的积极性和主动性，集思广益攻克技术难关。这种组织方式对于适合采用民主方法的那类问题(例如，研究性项目或遇到技术难题需要用集体智慧攻关)非常有效。尽管这种组织方式适当地发扬了民主，但是上下级之间的箭头(即管理关系)仍然是向下的，也就是说，是在集中指导下发扬民主。显然，如果程序员可以指挥项目经理，则只会引起混乱。

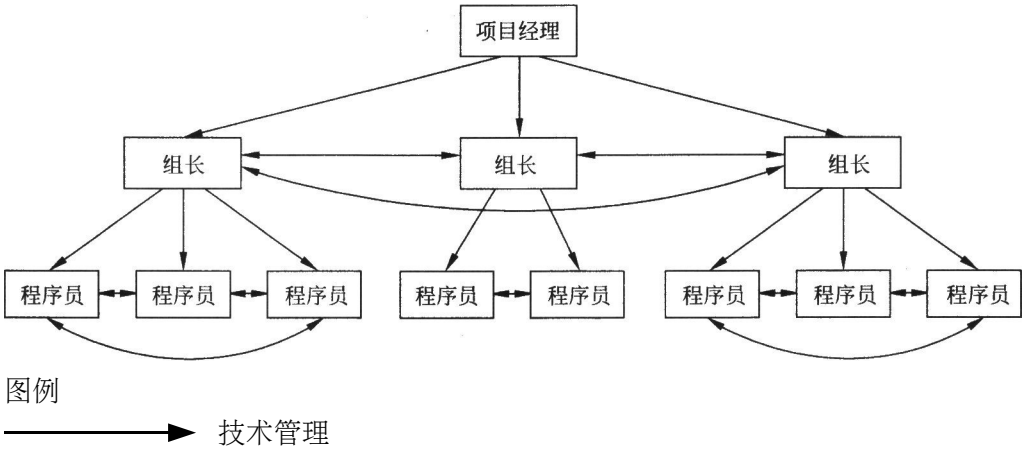


图 13.8 包含分散决策的组织方式

13.5 质量保证

质量是产品的生命，不论生产何种产品，质量都是极端重要的。软件产品开发周期长，耗费巨大的人力和物力，更必须特别注意保证质量。那么，什么是软件的质量呢?怎样在开发过程中保证软件的质量呢?本节着重讨论这两个问题。

13.5.1 软件质量

概括地说，软件质量就是“软件与明确地和隐含地定义的需求相一致的程度”。更具体地说，软件质量是软件与明确地叙述的功能和性能需求、文档中明确描述的开发标准以及任何专业开发的软件产品都应该具有的隐含特征相一致的程度。上述定义强调了下述的3个要点：

(1)软件需求是度量软件质量的基础，与需求不一致就是质量不高。

(2)指定的开发标准定义了一组指导软件开发的准则，如果没有遵守这些准则，几乎肯定会导致软件质量不高。

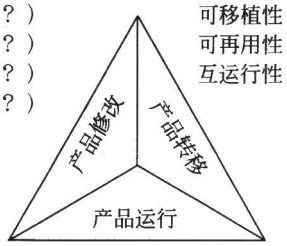
(3)通常，有一组没有显式描述的隐含需求(例如，软件应该是容易维护的)。如果软件满足明确描述的需求，但却不满足隐含的需求，那么软件的质量仍然是值得怀疑的。

虽然软件质量是难于定量度量的软件属性，但是仍然能够提出许多重要的软件质量指标(其中绝大多数目前还处于定性度量阶段)。

本节介绍影响软件质量的主要因素，这些因素是从管理角度对软件质量的度量。可以把这些质量因素分成3组，分别反映用户在使用软件产品时的3种不同倾向或观点。这3种倾向是：产品运行、产品修改和产品转移。图13.9描绘了软件质量因素和上述3种倾向(或产品活动)之间的关系，表13.7列出了软件质量因素的简明定义。

可理解性
可维修性
灵活性
可测试性
(我能理解它吗
(我能修复它吗
(我能改变它吗
(我能测试它吗

？)
？)
？)
？)



(我能在另一台机器上使用它吗?)
(我能再用它的某些部分吗?)
(我能把它和另一个系统结合吗?)

正确性(它按我的需要工作吗?)
健壮性(对意外环境它能适当地响应吗?)
效率 (完成预定功能时它需要的计算机资源多吗?)
完整性(它是安全的吗?)
可用性(我能使用它吗?)
风险 (能按预定计划完成它吗?)
图 13.9 软件质量因素与产品活动的关系

表 13.7 软件质量因素的定义

质量因素	定 义
正确性	系统满足规格说明和用户目标的程度，即，在预定环境下能正确地完成预期功能的程度
健壮性	在硬件发生故障、输入的数据无效或操作错误等意外环境下，系统能做出适当响应的程度
效率	为了完成预定的功能，系统需要的计算资源的多少
完整性(安全性)	对未经授权的人使用软件或数据的企图，系统能够控制(禁止)的程度
可用性	系统在完成预定应该完成的功能时令人满意的程度
风险	按预定的成本和进度把系统开发出来，并且为用户所满意的概率
可理解性	理解和使用该系统的容易程度
可维修性	诊断和改正在运行现场发现的错误所需要的工作量的大小
灵活性(适应性)	修改或改进正在运行的系统需要的工作量的多少
可测试性	软件容易测试的程度
可移植性	把程序从一种硬件配置和(或)软件系统环境转移到另一种配置和环境时，需要的工作量多少。有一种定量度量的方法是：用原来程序设计和调试的成本除移植时需用的费用
可再用性	在其他应用中该程序可以被再次使用的程度(或范围)
互运行性	把该系统和另一个系统结合起来需要的工作量的多少

13.5.2 软件质量保证措施

软件质量保证(software quality assurance，SQA)的措施主要有：基于非执行的测试(也称为复审或评审)，基于执行的测试(即以前讲过的软件测试)和程序正确性证明。复审主要用来保证在编码之前各阶段产生的文档的质量；基于执行的测试需要在程序编写出来之后进行，它是保证软件质量的最后一道防线；程序正确性证明使用数学方法严格验证程序是否与对它的说明完全一致。

参加软件质量保证工作的人员，可以划分成下述两类：

·软件工程师通过采用先进的技术方法和度量，进行正式的技术复审以及完成计划周密的软件测试来保证软件质量。

·SQA 小组的职责，是辅助软件工程师以获得高质量的软件产品。其从事的软件质量保证活动主要是：计划，监督，记录，分析和报告。简而言之，SQA 小组的作用是，通过确保软件过程的质量来保证软件产品的质量。

1.技术复审的必要性

正式技术复审的显著优点是，能够较早发现软件错误，从而可防止错误被传播到软件过程的后续阶段。

统计数字表明，在大型软件产品中检测出的错误，60%～70%属于规格说明错误或设计错误，而正式技术复审在发现规格说明错误和设计错误方面的有效性高达 75%。由于能够检测出并排除掉绝大部分这类错误，复审可大大降低后续开发和维护阶段的成本。

实际上，正式技术复审是软件质量保证措施的一种，包括走查(walkthrough)和审查(inspection)等具体方法。走查的步骤比审查少，而且没有审查正规。

2.走查

走查组由 4～6 名成员组成。以走查规格说明的小组为例，成员至少包括一名负责起草规格说明的人，一名负责该规格说明的管理员，一位客户代表，以及下阶段开发组(在本例中是设计组)的一名代表和 SQA 小组的一名代表。其中 SQA 小组的代表应该作为走查组的组长。

为了能发现重大错误，走查组成员最好是经验丰富的高级技术人员 ◦必须把被走查的材料预先分发给走查组每位成员 ◦走查组成员应该仔细研究材料并列出两张表：一张表是他不理解的术语，另一张是他认为不正确的术语。

走查组组长引导该组成员走查文档，力求发现尽可能多的错误 ◦走查组的任务仅仅是标记出错误而不是改正错误，改正错误的工作应该由该文档的编写组完成 ◦走查的时间最长不要超过 2 小时，这段时间应该用来发现和标记错误，而不是改正错误。

走查主要有下述两种方式：

(1)参与者驱动法。参与者按照事先准备好的列表，提出他们不理解的术语和认为不正确的术语 ◦文档编写组的代表必须回答每个质疑，要么承认确实有错误，要么对质疑做出解释。

(2)文档驱动法 ◦文档编写者向走查组成员仔细解释文档 ◦走查组成员在此过程中不时针对事先准备好的问题或解释过程中发现的问题提出质疑 ◦这种方法可能比第一种方法更有效，往往能检测出更多错误 ◦经验表明，使用文档驱动法时许多错误是由文档讲解者自己发现的。

3.审查

审查的范围比走查广泛得多，它的步骤也比较多 ◦通常，审查过程包括下述 5 个基本步骤：

(1)综述 ◦由负责编写文档的一名成员向审查组综述该文档 ◦在综述会结束时把文档分发给每位与会者。

(2)准备 ◦评审员仔细阅读文档 ◦最好列出在审查中发现的错误的类型，并按发生频率把错误类型分级，以辅助审查工作 ◦这些列表有助于评审员们把注意力集中到最常发生错误的区域。

(3)审查 ◦评审组仔细走查整个文档 ◦和走查一样，这一步的目的也是发现文档中的错误，而不是改正它们 ◦通常每次审查会不超过 90 分钟 ◦审查组组长应该在一天之内写出一份关于审查的报告。

(4)返工。文档的作者负责解决在审查报告中列出的所有错误及问题。

(5)跟踪 ◦组长必须确保所提出的每个问题都得到了圆满的解决(要么修正了文档，要么澄清了被误认为是错误的条目)。必须仔细检查对文档所做的每个修正，以确保没有引入新的错误。如果在审查过程中返工量超过 5%，则应该由审查组再对文档全面地审查一遍。

通常，审查组由4人组成。组长既是审查组的管理人员又是技术负责人。审查组必须包括负责当前阶段开发工作的项目组代表和负责下一阶段开发工作的项目组代表，此外，还应该包括一名SQA小组的代表。

审查过程不仅步数比走查多，而且每个步骤都是正规的。审查的正规性体现在：仔细划分错误类型，并把这些信息运用在后续阶段的文档审查中以及未来产品的审查中。

审查是检测软件错误的一种好方法，利用审查可以在软件过程的早期阶段发现并改正错误，也就是说，能在修正错误的代价变得很昂贵之前就发现并改正错误。因此，审查是一种经济有效的错误检测方法。

4.程序正确性证明

测试可以暴露程序中的错误，因此是保证软件可靠性的重要手段；但是，测试只能证明程序中有错误，并不能证明程序中没有错误。因此，对于保证软件可靠性来说，测试是一种不完善的技术，人们自然希望研究出完善的正确性证明技术。一旦研究出实用的正确性证明程序(即，能自动证明其他程序的正确性的程序)，软件可靠性将更有保证，测试工作量将大大减少。但是，即使有了正确性证明程序，软件测试也仍然是需要的，因为程序正确性证明只证明程序功能是正确的，并不能证明程序的动态特性是符合要求的，此外，正确性证明过程本身也可能发生错误。

正确性证明的基本思想是证明程序能完成预定的功能。因此，应该提供对程序功能的严格数学说明，然后根据程序代码证明程序确实能实现它的功能说明。

在20世纪60年代初期，人们已经开始研究程序正确性证明的技术，提出了许多不同的技术方法。虽然这些技术方法本身很复杂，但是它们的基本原理却是相当简单的。

如果在程序的若干个点上，设计者可以提出关于程序变量及它们的关系的断言，那么在每一点上的断言都应该永远是真的。假设在程序的P_1，P_2，…，P_n只等点上的断言分别是a(1)，a(2)，…，a(n)，其中a(1)必须是关于程序输入的断言，a(n)必须是关于程序输出的断言。

为了证明在点P_i和P_{i+1}，之间的程序语句是正确的，必须证明执行这些语句之后将使断言a(i)变成a(i+1)。如果对程序内所有相邻点都能完成上述证明过程，则证明了输入断言加上程序可以导出输出断言。如果输入断言和输出断言是正确的，而且程序确实是可以终止的(不包含死循环)，则上述过程就证明了程序的正确性。

人工证明程序正确性，对于评价小程序可能有些价值，但是在证明大型软件的正确性时，不仅工作量太大，更主要的是在证明的过程中很容易包含错误，因此是不实用的。为了实用的目的，必须研究能证明程序正确性的自动系统。

目前已经研究出证明PASCAL和LISP程序正确性的程序系统，正在对这些系统进行评价和改进。现在这些系统还只能对较小的程序进行评价，毫无疑问还需要做许多工作，这样的系统才能实际用于大型程序的正确性证明。

13.6 软件配置管理

任何软件开发都是迭代过程，也就是说，在设计过程会发现需求说明书中的问题，在实现过程又会暴露出设计中的错误，……。此外，随着时间推移客户的需求也会或多或少发生变化。因此，在开发软件的过程中，变化(或称为变动)既是必要的，又是不可避免的。但是，变化也很容易失去控制，如果不能适当地控制和管理变化，势必造成混乱并产生许多严重的错误。

软件配置管理是在软件的整个生命期内管理变化的一组活动。具体地说，这组活动用来：①标识变化；②控制变化；③确保适当地实现了变化；④向需要知道这类信息的人报告变化。

软件配置管理不同于软件维护。维护是在软件交付给用户使用后才发生的，而配置管理是在软件项目启动时就开始，并且一直持续到软件退役后才终止的一组跟踪和控制活动。

软件配置管理的目标是，使变化更正确且更容易被适应，在必须变化时减少所需花费的工作量。

13.6.1 软件配置

1.软件配置项

软件过程的输出信息可以分为3类：①计算机程序(源代码和可执行程序)；②描述计算机程序的文档(供技术人员或用户使用)；③数据(程序内包含的或在程序外的)。上述这些项组成了在软件过程中产生的全部信息，我们把它们统称为软件配置，而这些项就是软件配置项。

随着软件开发过程的进展，软件配置项的数量迅速增加。不幸的是，由于前述的种种原因，软件配置项的内容随时都可能发生变化。为了开发出高质量的软件产品，软件开发人员不仅要努力保证每个软件配置项正确，而且必须保证一个软件的所有配置项是完全一致的。

可以把软件配置管理看作是应用于整个软件过程的软件质量保证活动，是专门用于管理变化的软件质量保证活动。

2.基线

基线是一个软件配置管理概念，它有助于我们在不严重妨碍合理变化的前提下来控制变化。IEEE把基线定义为：已经通过了正式复审的规格说明或中间产品，它可以作为进一步开发的基础，并且只有通过正式的变化控制过程才能改变它。

简而言之，基线就是通过了正式复审的软件配置项。在软件配置项变成基线之前，可以迅速而非正式地修改它。一旦建立了基线之后，虽然仍然可以实现变化，但是，必须应用特定的、正式的过程(称为规程)来评估、实现和验证每个变化。

除了软件配置项之外，许多软件工程组织也把软件工具置于配置管理之下，也就是说，把特定版本的编辑器、编译器和其他CASE工具，作为软件配置的一部分“固定”下来。因为当修改软件配置项时必然要用到这些工具，为防止不同版本的工具产生的结果不同，应该把软件工具也基线化，并且列入到综合的配置管理过程之中。

13.6.2 软件配置管理过程

软件配置管理是软件质量保证的重要一环，它的主要任务是控制变化，同时也负责各个软件配置项和软件各种版本的标识、软件配置审计以及对软件配置发生的任何变化的报告。

具体来说，软件配置管理主要有5项任务：标识、版本控制、变化控制、配置审计和报告。

1.标识软件配置中的对象

为了控制和管理软件配置项，必须单独命名每个配置项，然后用面向对象方法组织它们。可以标识出两类对象：基本对象和聚集对象(可以把聚集对象作为代表软件配置完整版本的一种机制)。基本对象是软件工程师在分析、设计、编码或测试过程中创建出来的“文本

单元”，例如，需求规格说明的一个段落、一个模块的源程序清单或一组测试用例。聚集对象是基本对象和其他聚集对象的集合。

每个对象都有一组能惟一地标识它的特征：名字、描述、资源表和“实现”。其中，对象名是无二义性地标识该对象的一个字符串。

在设计标识软件对象的模式时，必须认识到对象在整个生命周期中一直都在演化，因此，所设计的标识模式必须能无歧义地标识每个对象的不同版本。

2.版本控制

版本控制联合使用规程和工具，以管理在软件工程过程中所创建的配置对象的不同版本。借助于版本控制技术，用户能够通过选择适当的版本来指定软件系统的配置。实现这个目标的方法是，把属性和软件的每个版本关联起来，然后通过描述一组所期望的属性来指定和构造所需要的配置。

上面提到的“属性”，既可以简单到仅是赋给每个配置对象的具体版本号，也可以复杂到是一个布尔变量串，其指明了施加到系统上的功能变化的具体类型。

3.变化控制

对于大型软件开发项目来说，无控制的变化将迅速导致混乱。变化控制把人的规程和自动工具结合起来，以提供一个控制变化的机制。典型的变化控制过程如下：接到变化请求之后，首先评估该变化在技术方面的得失、可能产生的副作用、对其他配置对象和系统功能的整体影响以及估算出的修改成本。评估的结果形成“变化报告”，该报告供“变化控制审批者”审阅。所谓变化控制审批者既可以是一个人也可以由一组人组成，其对变化的状态和优先级做最终决策。为每个被批准的变化都生成一个“工程变化命令”，其描述将要实现的变化，必须遵守的约束以及复审和审计的标准。把要修改的对象从项目数据库中“提取(check out)”出来，进行修改并应用适当的SQA活动。最后，把修改后的对象“提交(check in)”进数据库，并用适当的版本控制机制创建该软件的下一个版本。

“提交”和“提取”过程实现了变化控制的两个主要功能——访问控制和同步控制。访问控制决定哪个软件工程师有权访问和修改一个特定的配置对象，同步控制有助于保证由两名不同的软件工程师完成的并行修改不会相互覆盖。

在一个软件配置项变成基线之前，仅需应用非正式的变化控制。该配置对象的开发者可以对它进行任何合理的修改(只要修改不会影响到开发者工作范围之外的系统需求)。一旦该对象经过了正式技术复审并获得批准，就创建了一个基线。而一旦一个软件配置项变成了基线，就开始实施项目级的变化控制。现在，为了进行修改开发者必须获得项目管理者的批准(如果变化是“局部的”)，如果变化影响到其他软件配置项，还必须得到变化控制审批者的批准。在某些情况下，可以省略正式的变化请求、变化报告和工程变化命令，但是，必须评估每个变化并且跟踪和复审所有变化。

4.配置审计

为了确保适当地实现了所需要的变化，通常从下述两方面采取措施：①正式的技术复审；②软件配置审计。

正式的技术复审(见13.5.2节)关注被修改后的配置对象的技术正确性。复审者审查该对象以确定它与其他软件配置项的一致性，并检查是否有遗漏或副作用。

软件配置审计通过评估配置对象的那些通常不在复审过程中考虑的特征(例如，修改时是否遵循了软件工程标准，是否在该配置项中显著地标明了所做的修改，是否注明了修改日期和修改者，是否适当地更新了所有相关的软件配置项，是否遵循了标注变化、记录变化和报告变化的规程)，而成为对正式技术复审的补充。

5.状态报告

书写配置状态报告是软件配置管理的一项任务，它回答下述问题：①发生了什么事?

②谁做的这件事?③ 这件事是什么时候发生的?④ 它将影响哪些其他事物?

配置状态报告对大型软件开发项目的成功有重大贡献。当大量人员在一起工作时，可能一个人并不知道另一个人在做什么。两名开发人员可能试图按照相互冲突的想法去修改同一个软件配置项；软件工程队伍可能耗费几个人月的工作量根据过时的硬件规格说明开发软件；察觉到所建议的修改有严重副作用的人可能还不知道该项修改正在进行。配置状态报告通过改善所有相关人员之间的通信，帮助消除这些问题。

13.7 能力成熟度模型

美国卡内基梅隆大学软件工程研究所在美国国防部资助下于 20 世纪 80 年代末建立的能力成熟度模型(capability maturity model，CMM)，是用于评价软件机构的软件过程能力成熟度的模型。最初，建立此模型的目的主要是，为大型软件项目的招投标活动提供一种全面而客观的评审依据，发展到后来，此模型又同时被应用于许多软件机构内部的过程改进活动中。

多年来，软件危机一直困扰着许多软件开发机构。不少人试图通过采用新的软件开发技术来解决在软件生产率和软件质量等方面存在的问题，但效果并不令人十分满意。上述事实促使人们进一步考察软件过程，从而发现关键问题在于对软件过程的管理不尽人意。事实证明，在无规则和混乱的管理之下，先进的技术和工具并不能发挥出应有的作用。人们逐渐认识到，改进对软件过程的管理是消除软件危机的突破口，再也不能忽视在软件过程中管理的关键作用了。

能力成熟度模型的基本思想是，由于问题是由我们管理软件过程的方法不当引起的，所以新软件技术的运用并不会自动提高软件的生产率和质量。能力成熟度模型有助于软件开发机构建立一个有规律的、成熟的软件过程。改进后的软件过程将开发出质量更好的软件，使更多的软件项目免受时间和费用超支之苦。

软件过程包括各种活动、技术和工具，因此，它实际上既包括了软件开发的技术方面又包括了管理方面。CMM 的策略是，力图改进对软件过程的管理，而在技术方面的改进是其必然的结果。

CMM 在改进软件过程中所起的作用主要是，指导软件机构通过确定当前的过程成熟度并识别出对过程改进起关键作用的问题，从而明确过程改进的方向和策略。通过集中开展与过程改进的方向和策略相一致的一组过程改进活动，软件机构便能稳步而有效地改进其软件过程，使其软件过程能力得到循序渐进的提高。

对软件过程的改进，是在完成一个又一个小的改进步骤基础上不断进行的渐进过程，而不是一蹴而就的彻底革命。CMM 把软件过程从无序到有序的进化过程分成 5 个阶段，并把这些阶段排序，形成 5 个逐层提高的等级。这 5 个成熟度等级定义了一个有序的尺度，用以测量软件机构的软件过程成熟度和评价其软件过程能力，这些等级还能帮助软件机构把应做的改进工作排出优先次序。成熟度等级是妥善定义的向成熟软件机构前进途中的平台，每个成熟度等级都为软件过程的继续改进提供了一个台阶。

CMM 对 5 个成熟度级别特性的描述，说明了不同级别之间软件过程的主要变化。从“1 级”到“5 级”，反映出一个软件机构为了达到从一个无序的、混乱的软件过程进化到一种有序的、有纪律的且成熟的软件过程的目的，必须经历的过程改进活动的途径。每一个成熟度级别都是该软件机构沿着改进其过程的途径前进途中的一个台阶，后一个成熟度级别是前一个级别的软件过程的进化目标。CMM 的每个成熟度级别中都包含一组过程改进的目标，

满足这些目标后一个机构的软件过程就从当前级别进化到下一个成熟度级别；每达到成熟度级别框架的下一个级别，该机构的软件过程都得到一定程度的完善和优化，也使得过程能力得到提高；随着成熟度级别的不断提高，该机构的过程改进活动取得了更加显著的成效，从而使软件过程得到进一步的完善和优化。CMM 就是以上述方式支持软件机构改进其软件过程的活动。

CMM 通过定义能力成熟度的 5 个等级，引导软件开发机构不断识别出其软件过程的缺陷，并指出应该做哪些改进，但是，它并不提供做这些改进的具体措施。

能力成熟度的 5 个等级从低到高依次是：初始级（又称为 1 级），可重复级（又称为 2 级），已定义级（又称为 3 级），已管理级（又称为 4 级）和优化级（又称为 5 级）。下面介绍这 5 个级别的特点。

1.初始级

软件过程的特征是无序的，有时甚至是混乱的。几乎没有什么过程是经过定义的（即没有一个定型的过程模型），项目能否成功完全取决于开发人员的个人能力。

处于这个最低成熟度等级的软件机构，基本上没有健全的软件工程管理制度，其软件过程完全取决于项目组的人员配备，所以具有不可预测性，人员变了过程也随之改变。如果一个项目碰巧由一个杰出的管理者和一支有经验、有能力的开发队伍承担，则这个项目可能是成功的。但是，更常见的情况是，由于缺乏健全的管理和周密的计划，延期交付和费用超支的情况经常发生，结果，大多数行动只是应付危机，而不是完成事先计划好的任务。

总之，处于 1 级成熟度的软件机构，其过程能力是不可预测的，其软件过程是不稳定的，产品质量只能根据相关人员的个人工作能力而不是软件机构的过程能力来预测。

2.可重复级

软件机构建立了基本的项目管理过程（过程模型），可跟踪成本、进度、功能和质量。已经建立起必要的过程规范，对新项目的策划和管理过程是基于以前类似项目的实践经验，使得有类似应用经验的软件项目能够再次取得成功。达到 2 级的一个目标是使项目管理过程稳定，从而使得软件机构能重复以前在成功项目中所进行过的软件项目工程实践。

处于 2 级成熟度的软件机构，针对所承担的软件项目已建立了基本的软件管理控制制度。通过对以前项目的观察和分析，可以提出针对现行项目的约束条件。项目负责人跟踪软件产品开发的成本和进度以及产品的功能和质量，并且识别出为满足约束条件所应解决的问题。已经做到软件需求条理化，而且其完整性是受控制的。已经制定了项目标准，并且软件机构能确保严格执行这些标准。项目组与客户及承包商已经建立起一个稳定的、可管理的工作环境。

处于 2 级成熟度的软件机构的过程能力可以概括为，软件项目的策划和跟踪是稳定的，已经为一个有纪律的管理过程提供了可重复以前成功实践的项目环境。软件项目工程活动处于项目管理体系的有效控制之下，执行着基于以前项目的准则且合乎现实的计划。

3.已定义级

软件机构已经定义了完整的软件过程（过程模型），软件过程已经文档化和标准化。所有项目组都使用文档化的、经过批准的过程来开发和维护软件。这一级包含了第 2 级的全部特征。

在第 3 级成熟度的软件机构中，有一个固定的过程小组从事软件过程工程活动。当需要时，过程小组可以利用过程模型进行过程例化活动，从而获得一个针对某个特定的软件项目的过程实例，并投入过程运作而开展有效的软件项目工程实践。同时，过程小组还可以推进软件机构的过程改进活动。在该软件机构内实施了培训计划，能够保证全体项目负责人和项目开发人员具有完成承担的任务所要求的知识和技能。

处于 3 级成熟度的软件机构的过程能力可以概括为，无论是管理活动还是工程活动都

是稳定的。软件开发的成本和进度以及产品的功能和质量都受到控制，而且软件产品的质量具有可追溯性。这种能力是基于在软件机构中对已定义的过程模型的活动、人员和职责都有共同的理解。

4.已管理级

软件机构对软件过程(过程模型和过程实例)和软件产品都建立了定量的质量目标，所有项目的重要的过程活动都是可度量的。该软件机构收集了过程度量和产品度量的方法并加以运用，可以定量地了解和控制软件过程和软件产品，并为评定项目的过程质量和产品质量奠定了基础。这一级包含了第3级的全部特征。

处于4级成熟度的软件机构的过程能力可以概括为，软件过程是可度量的，软件过程在可度量的范围内运行。这一级的过程能力允许软件机构在定量的范围内预测过程和产品质量趋势，在发生偏离时可以及时采取措施予以纠正，并且可以预期软件产品是高质量的。

5.优化级

软件机构集中精力持续不断地改进软件过程。这一级的软件机构是一个以防止出现缺陷为目标的机构，它有能力识别软件过程要素的薄弱环节，并有足够的手段改进它们。在这样的机构中，可以获得关于软件过程有效性的统计数据，利用这些数据可以对新技术进行成本／效益分析，并可以优化出在软件工程实践中能够采用的最佳新技术。这一级包含了第4级的全部特征。

这一级的软件机构可以通过对过程实例性能的分析和确定产生某一缺陷的原因，来防止再次出现这种类型的缺陷；通过对任何一个过程实例的分析所获得的经验教训都可以成为该软件机构优化其过程模型的有效依据，从而使其他项目的过程实例得到优化。这样的软件机构可以通过从过程实施中获得的定量的反馈信息，在采用新思想和新技术的同时测试它们，以不断地改进和优化软件过程。

处于5级成熟度的软件机构的过程能力可以概括为，软件过程是可优化的。这一级的软件机构能够持续不断地改进其过程能力，既对现行的过程实例不断地改进和优化，又借助于所采用的新技术和新方法来实现未来的过程改进。

一些统计数字表明，提高一个完整的成熟度等级大约需要花18个月到3年的时间，但是从第1级上升到第2级有时要花3年甚至5年时间。这说明要向一个迄今仍处于混乱的和被动的行动方式的软件机构灌输系统化的方式，将多么困难。

13.8 小　结

软件工程包括技术和管理两方面的内容，是技术与管理紧密结合的产物。只有在科学而严格的管理之下，先进的技术方法和优秀的软件工具才能真正发挥出威力。因此，有效的管理是大型软件工程项目成功的关键。

软件项目管理始于项目计划，而第一项计划活动就是估算。为了估算项目工作量和完成期限，首先需要预测软件规模。

度量软件规模的常用技术主要有代码行技术和功能点技术。这两种技术各有优缺点，应该根据项目特点及从事计划工作的人对这两种技术的熟悉程度，选用适用的技术。

根据软件规模可以估算出完成该项目所需的工作量，常用的估算模型为静态单变量模型、动态多变量模型和COCOMO2模型。为了使估算结果更接近实际值，通常至少同时使用上述3种模型中的两种。通过比较和协调使用不同模型得出的估算值，有可能得到比较准确的估算结果。成本估算模型通常也同时提供了估算软件开发时间的方程式，这样估算出的开发

时间是正常开发时间，经验表明，用增加开发人员的方法最多可以把开发时间减少到正常开发时间的75%。

管理者必须制定出一个足够详细的进度表，以便监督项目进度并控制整个项目。常用的制定进度计划的工具有Gantt图和工程网络，这两种工具各有优缺点，通常，联合使用Gantt图和工程网络来制定进度计划并监督项目进展状况。

高素质的开发人员和合理的项目组组织结构，是软件项目取得成功的关键。比较典型的组织结构有民主制程序员组、主程序员组和现代程序员组等3种，这3种组织方式的适用场合并不相同。

软件质量保证是在软件过程中的每一步都进行的活动。软件质量保证措施主要有基于非执行的测试(也称为复审)、基于执行的测试(即通常所说的测试)和程序正确性证明。软件复审是最重要的软件质量保证活动之一，它的优点是在改正错误的成本相对比较低时就能及时发现并排除软件错误。

软件配置管理是应用于整个软件过程中的保护性活动，是在软件整个生命期内管理变化的一组活动。软件配置管理的目标是，使变化能够更正确且更容易被适应，在需要修改软件时减少为此而花费的工作量。

能力成熟度模型(CMM)是改进软件过程的有效策略。它的基本思想是，因为问题是管理软件过程的方法不恰当造成的，所以采用新技术并不会自动提高软件生产率和软件质量，应该下大力气改进对软件过程的管理。事实上对软件过程的改进不可能一蹴而就，因此，CMM以增量方式逐步引入变化，它明确地定义了5个成熟度等级，一个软件开发组织可以用一系列小的改良性步骤迈人更高的成熟度等级。

习题 13

1.研究本书2.4.2小节所述的定货系统，要求：

(1)用代码行技术估算本系统的规模；

(2)用功能点技术估算本系统的规模；

(3)用静态单变量模型估算开发本系统所需的工作量；

(4)假设由一个人开发本系统，请制定进度计划；

(5)假设由两个人开发本系统，请制定进度计划。

2.研究本书习题2第2题中描述的储蓄系统，要求：

(1)用代码行技术估算本系统的规模；

(2)用功能点技术估算本系统的规模；

(3)用静态单变量模型估算开发本系统所需的工作量；

(4)假设由一个人开发本系统，请制定进度计划；

(5)假设由两个人开发本系统，请制定进度计划。

3.下面叙述对一个计算机辅助设计(CAD)软件的需求：

该CAD软件接受由工程师提供的二维或三维几何图形数据。工程师通过用户界面与CAD系统交互并控制它，该用户界面应该表现出良好的人机界面特征。几何图形数据及其他支持信息都保存在一个CAD数据库中。开发必要的分析、设计模块，以产生所需要的输出，这些输出将显示在各种不同的图形设备上。应该适当地设计软件，以便与外部设备交互并控制它们。所用的外部设备包括鼠标、数字化扫描仪和激光打印机。

要求：

(1)进一步精化上述要求，把CAD软件的功能分解成若干个子功能；

(2)用代码行技术估算每个子功能的规模；

(3)用功能点技术估算每个子功能的规模；

(4)从历史数据得知，开发这类系统的平均生产率是620 LOC／pm，如果软件工程师的平均月薪是8 000元，请估算开发本系统的工作量和成本；

(5)如果从历史数据得知，开发这类系统的平均生产率是6.5 FP／pm，请估算开发本系统的工作量和成本。

4.假设你被指定为项目负责人，你的任务是开发一个应用系统，该系统类似于你的小组以前做过的那些系统，但是规模更大且更复杂一些。客户已经写出了完整的需求文档。你将选用哪种项目组结构?为什么?你打算采用哪种(些)软件过程模型?为什么?

5.假设你被指派为一个软件公司的项目负责人，你的任务是开发一个技术上具有创新性的产品，该产品把虚拟现实硬件和最先进的软件结合在一起。由于家庭娱乐市场的竞争非常激烈，这项工作的压力很大。你将选择哪种项目组结构?为什么?你打算采用哪种(些)软件过程模型?为什么?

6.你被指派作为一个大型软件产品公司的项目负责人，你的工作是管理该公司已被广泛应用的字处理软件的新版本开发。由于市场竞争激烈，公司规定了严格的完成期限并且对外公布了。你将选择哪种项目组结构?为什么?你打算采用哪种(些)软件过程模型?为什么?

7.什么是软件质量?请叙述它与软件可靠性的关系。

8.一个程序能既正确又不可靠吗?请解释你的答案。

9.仅当每个与会者都在事先作了准备时，正式的技术复审才能取得预期的效果。如果你是复审小组的组长，你怎样发现事先没做准备的与会者?你打算采取什么措施来促使大家事先做准备?

10.什么是基线?为什么要建立基线?

11.配置审计和技术复审有何不同?可否把它们的功能放在一次复审中完成?

12.CMM的基本思想是什么?为什么要把能力成熟度划分成5个等级?