


软件工程导论（第6版）



第12章 面向对象实现

第12章 面向对象实现

• 12.1.1 面向对象语言的优点

使用面向对象语言时，编译程序可以自动把面向对象概念映射到目标程序中。

使用非面向对象语言编写面向对象程序，则必须由程序员自己把面向对象概念映射到程序中。

12.1.2 面向对象语言的技术特点

- 1980年的smalltalk-80发展了Simula-67的对象和类的概念，并引入了方法、消息、元类及协议等概念，所以有人将smalltalk80称为第一个面向对象语言。
- 但是，使面向对象广泛流行的是C++。

- **选择面向对象语言时应考虑的技术特点：**

- **1. 支持类与对象概念的机制**
- **2. 实现整体－部分（聚集）结构的机制**
- **3. 实现一般－特殊（泛化）结构的机制**
- **4. 实现属性和服务的机制**
- **5. 类型检查**
- **6. 类库**
- **7. 效率**
- **8. 持久保存对象**
- **9. 参数化类**
- **10. 开发环境**

•12.1.3 选择面向对象语言

- 1. 将来能否占主导地位
- 2. 可重用性
- 3. 类库和开发环境
- 4. 其他因素
- 有否提供技术支持？提供开发人员什么开发平台？对及其性能的需求？集成已有软件的容易程度。

12.2 程序设计风格

- 12.2.1 提高可重用性**
- 12.2.2 提高可扩充性**
- 12.2.3 提高健壮性**

12.3 测试策略

• 12.3.1 面向对象的单元测试

基类：操作

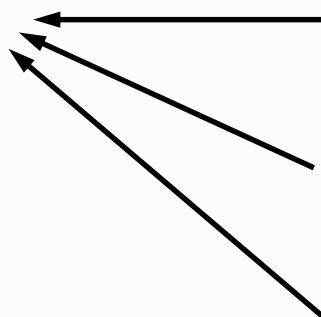
子类：操作

A: X

A1: X

A2: X

A3: X



有必要在
每个子类
中测试操
作 X

不孤立地测试单个操作（函数），而是把操作作为类的一部分进行测试。

• 12.3.2 面向对象的集成测试

- 两种不同的测试策略：
 - 1) **基于线程的测试** (thread based testing)
 - 将响应系统的一个输入或一个事件所需要的哪些类集成起来测试。
 - 2) **基于使用的测试** (use based testing)
 - 先测试独立类，再测试使用独立类的下一层次的类（依赖类），重复直至完毕。

• 12.3.3 面向对象的确认测试

主要是根据动态模型和描述系统行为的脚本来设计确认测试用例。

12.4 设计测试用例

•12.4.1 测试类的方法

•1. 随机测试

- ATM系统中account (帐户) 类的操作有: Open (打开)、Setup (建立)、Deposit (存款)、Withdraw (取款)、Balance (余额)、Summarize (清单)、CreditLimit (透支限额)、Close (关闭)。

- 可以随机地产生一系列不同的操作序列:

测试用例1: Open.Setup.Deposit.Balance.Summarize.Withdraw.Close

测试用例2: Open.Setup.Withdraw.Deposit.Balance.CreditLimit.Close

- **2. 划分测试（类似于等价类划分）**

- 1) 基于状态的划分**

- **改变Account类状态的操作：**

deposit, withdraw;

- **不改变Account类状态的操作：**

balance, summarize, creditLimit

- **测试用例：**

1.open.setup.deposit.deposit.withdraw.withdraw.close;

2.open.setup.balance.summarize.creditLimit.close;

- **2) 基于属性的划分**

根据类操作使用的属性来划分操作。

例，按属性Balance分可三类：

- **使用Balance的操作；**
- **修改Balance的操作；**
- **不使用也不修改balance的操作。**

3) 基于功能的划分

依据类操作完成的功能来划分类操作。

例，初始化操作：open, setup

计算操作：deposit, withdraw

查询操作：balance, summarize, creditLimit

3. 基于故障的测试

一般依靠经验和直觉，类似于错误推测测试法。

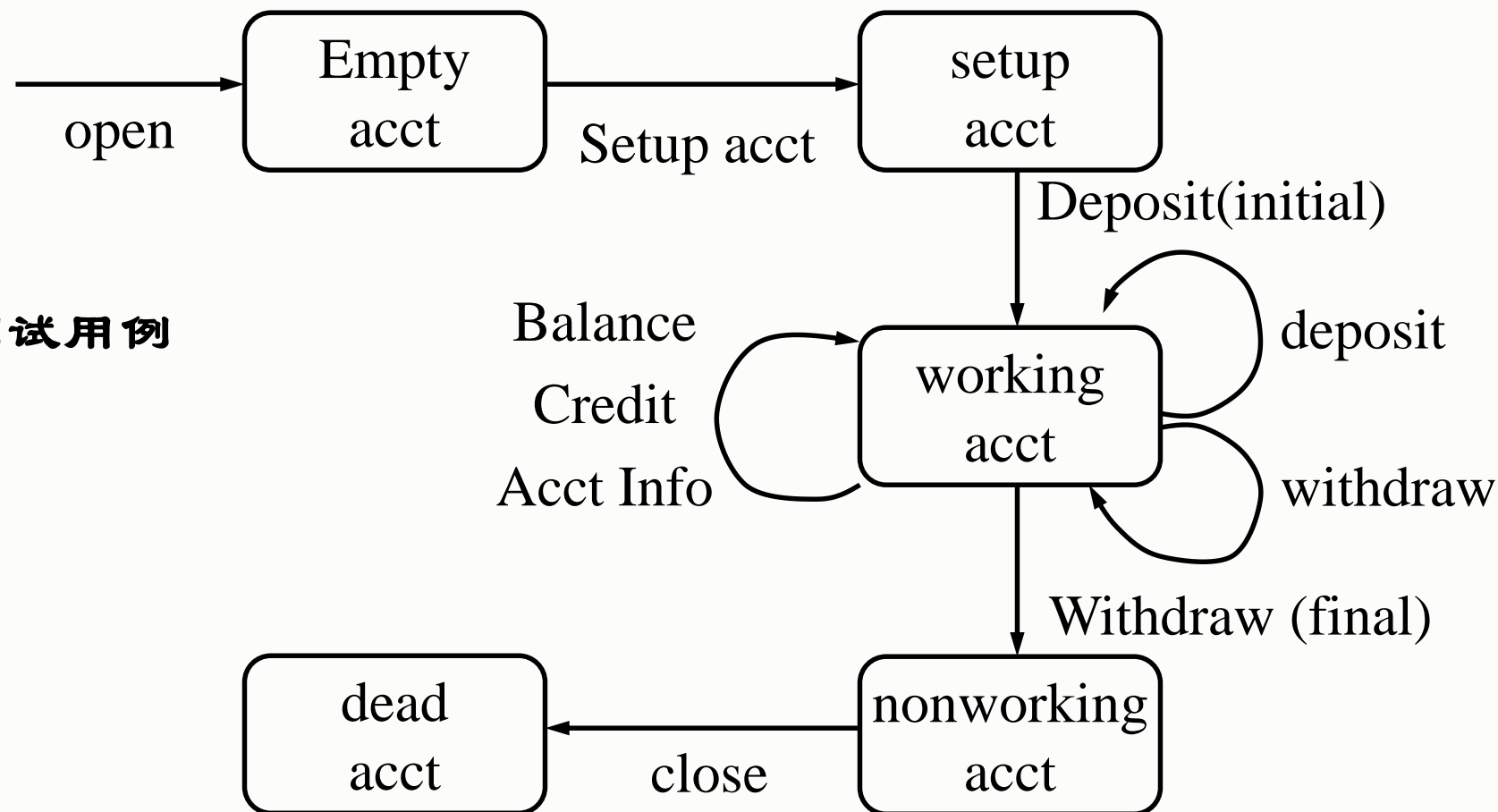
•12.4.2 集成测试方法

- 测试类协作可以使用随机测试方法、划分测试方法、行为测试等。

- 1. 多类测试

- p302

- 2. 从动态模型导出测试用例



Account类的状态转换图

- 应使Account类实例遍历所有允许的状态转换：

测试用例1：

✓ open. setupacct. deposit(initial). withdraw(final).

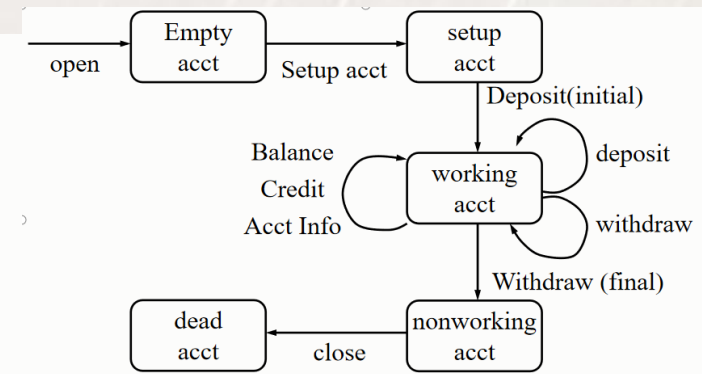
测试用例2：

✓ open. setupacct. deposit(initial). deposit. blance. credit.
withdraw(final). Close


测试用例3：

✓ open. setupacct. deposit(initial). deposit. withdraw. AcctInfo.
withdraw(final). Close

•



软件工程导论 (第6版)



第13章 软件项目管理



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。

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**。

主要内容

13.1 估算软件规模

13.2 工作量估算

13.3 进度计划

13.4 人员组织

13.5 质量保证

13.6 软件配置管理

13.7 能力成熟模型

主要内容



13.1 估算软件规模

13.2 工作量估算

13.3 进度计划

13.4 人员组织

13.5 质量保证

13.6 软件配置管理

13.7 能力成熟模型

13.1 估算软件规模

13.1.1 代码行技术

- 估算软件规模的单位是代码行数 (LOC)和千行代码数 (KLOC) 。
- 优点：容易计算
- 缺点：
 - ① 不太合理；
 - ② 用不同语言实现同一个软件所需要的代码行数并不相同；
 - ③ 不适用于非过程语言。

13.1 估算软件规模

13.1.2 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。

信息域特性

功能点技术定义了信息域的5个特性：

- (1) 输入项数
- (2) 输出项数
- (3) 查询数
- (4) 主文件数
- (5) 外部接口数

(1) 计算未调整的功能点数UFP

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

特性系数	复杂级别		
	简单	平均	复杂
输入系数 a_1	3	4	6
输出系数 a_2	4	5	7
查询系数 a_3	3	4	6
文件系数 a_4	7	10	15
接口系数 a_5	5	7	10

(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

- 表中列出了全部技术因素，并用 $F_i(1 \leq i \leq 14)$ 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：

$$DI = \sum_{i=1}^{14} F_i$$

- 技术复杂性因子TCF由下式计算：

$$TCF = 0.65 + 0.01 \times DI$$

因为DI的值在0~70之间，所以TCF的值在0.65~1.35之间。

序号	F_i	技术因素
1	F_1	数据通信
2	F_2	分布式数据处理
3	F_3	性能标准
4	F_4	高负荷的硬件
5	F_5	高处理率
6	F_6	联机数据输入
7	F_7	终端用户效率
8	F_8	联机更新
9	F_9	复杂的计算
10	F_{10}	可重用性
11	F_{11}	安装方便
12	F_{12}	操作方便
13	F_{13}	可移植性
14	F_{14}	可维护性

(3) 计算功能点数FP

$$FP = UFP \times TCF$$

主要内容

13.1 估算软件规模



13.2 工作量估算

13.3 进度计划

13.4 人员组织

13.5 质量保证

13.6 软件配置管理

13.7 能力成熟模型

13.2 工作量估算

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）。

13.2.1 静态单变量模型

13.2.2 动态多变量模型

13.2.3 COCOMO2 模型

13.2 工作量估算

13.2.1 静态单变量模型

总体结构形式如： $E_{\text{工作量}} = A + B \times (ev_{\text{估算变量}})^C$

E 是以人月为单位的工作量， ev 是估算变量（KLOC或FP）。

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 $E = 585.7 + 15.12FP$

13.2 工作量估算

13.2.2 动态多变量模型

- 根据从4000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软件规模和开发时间这两个变量的函数。

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

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，它反映了管理水平、技术及经验等因素对工作量的影响

13.2 工作量估算

13.2.3 COCOMO2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。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10多年来在成本估计方面所积累的经验。

COCOMO2估算模型： $E=a \times KLOC^b \times \prod_{i=1}^{17} f_i$

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；

$f_i(i=1\sim 17)$ 是成本因素。

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	1.00	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	

主要内容

13.1 估算软件规模

13.2 工作量估算

 **13.3 进度计划**

13.4 人员组织

13.5 质量保证

13.6 软件配置管理

13.7 能力成熟模型

13.3 进度计划

13.3.1 估算开发时间

13.3.2 Gantt图

13.3.3 工程网络

13.3.4 估算工程进度

13.3.5 关键路径

13.3.6 机动时间

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。

13.3 进度计划

13.3.1 估算开发时间

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

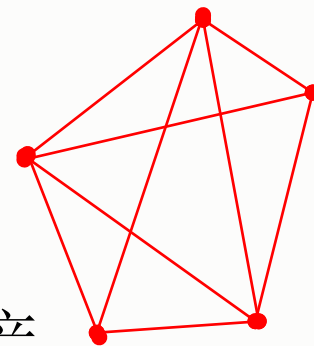
(4) Putnam模型 $T=2.4E^{1/3}$

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）。

13.3 进度计划



- 为了缩短开发时间应该增加从事开发工作的人数。
- 但是，经验告诉人们，随着开发小组规模的扩大，个人生产率将下降，以致开发与从事开发工作的人数并不成反比关系。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。

13.3 进度计划

举例说明项目组规模与项目组总生产率的关系

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 $I=50$ ）。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参见下表，可见，在这种情况下**项目组的最佳规模是5.5人**，即 $P_{opt}=5.5$ 。

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。

项目组规模	个人生产率	总生产率
1	500	500
2	450	900
3	400	1 200
4	350	1 400
5	300	1 500
5.5	275	1 512
6	250	1 500
7	200	1 400
8	150	1 200

13.3 进度计划

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。

13.3 进度计划

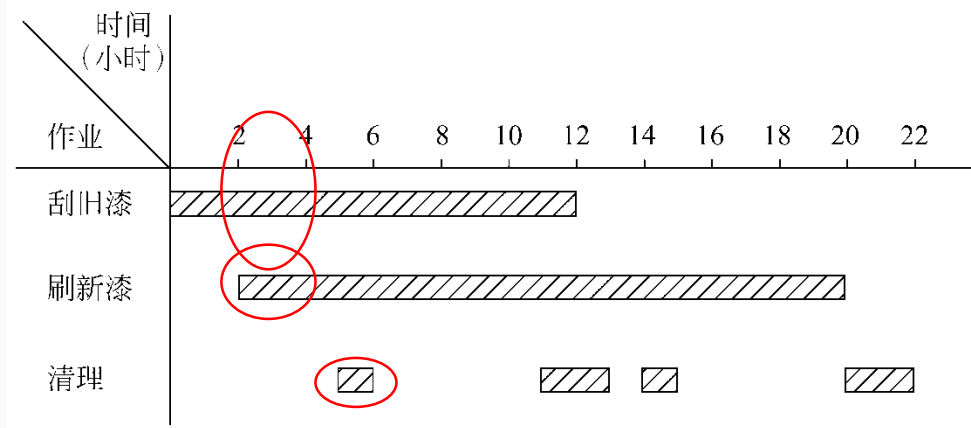
大家可能已经想到，应该采用“**流水作业法**”，也就是说，首先由5名工人用刮板刮掉第1面墙上的旧漆(这时其余10名工人休息)，当第1面墙刮净后，另外5名工人立即用刷子给这面墙刷新漆(与此同时拿刮板的5名工人转去刮第2面墙上的旧漆)，一旦刮旧漆的工人转到第3面墙而且刷新漆的工人转到第2面墙以后，余下的5名工人立即拿起刮刀去清除溅在第1面墙窗户上的油漆，……。这样安排使每个工人都有活干，因此能够在较短的时间内完成任务。

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。下表列出了估计每道工序需要用的时间。

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

13.3 进度计划

可以使用下图所示的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。



优点?
缺点?

从上图可以看出，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。

13.3 进度计划

13.3.3 工程网络

- 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。
- 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) 计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费。

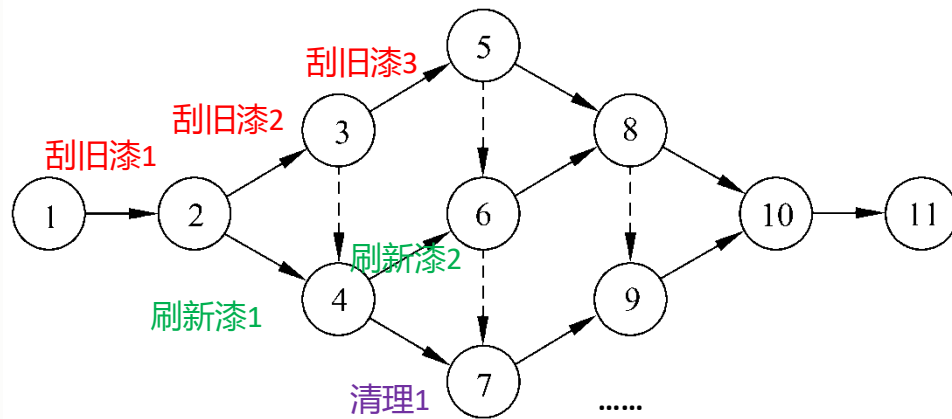
13.3 进度计划

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。

13.3 进度计划

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。下图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。

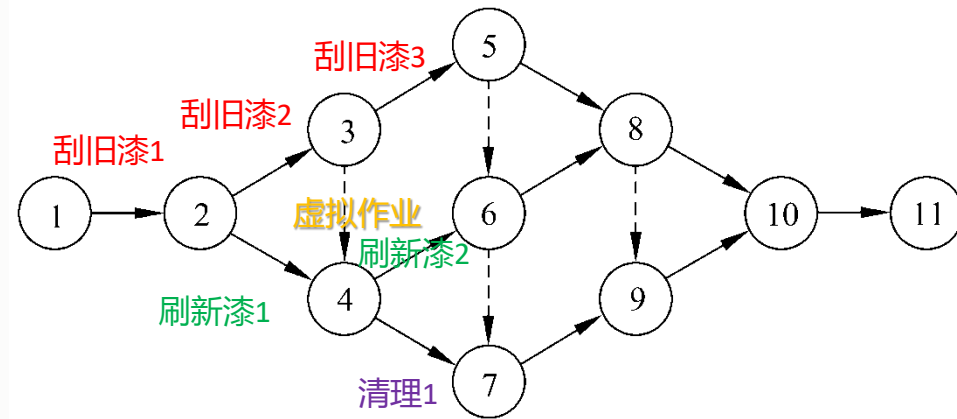


图中:

1—2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。

13.3 进度计划

在右图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。

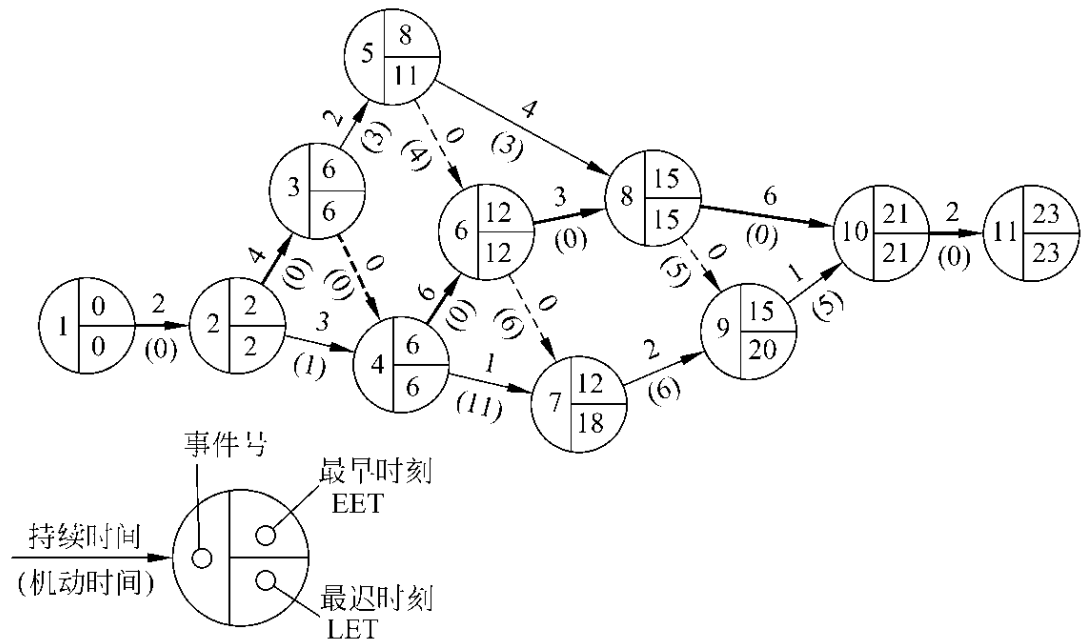
注意，虚拟作业既不消耗资源也不需要时间。可以研究图下面对各项作业的描述，解释引入其他虚拟作业的原因。

13.3 进度计划

13.3.4 估算工程进度

系统分析员就可以借助它的帮助估算工程进度：

- 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
- 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，如有图左下角的符号所示。
- 事件的最早时刻是该事件可以发生的最早时间。
- 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。



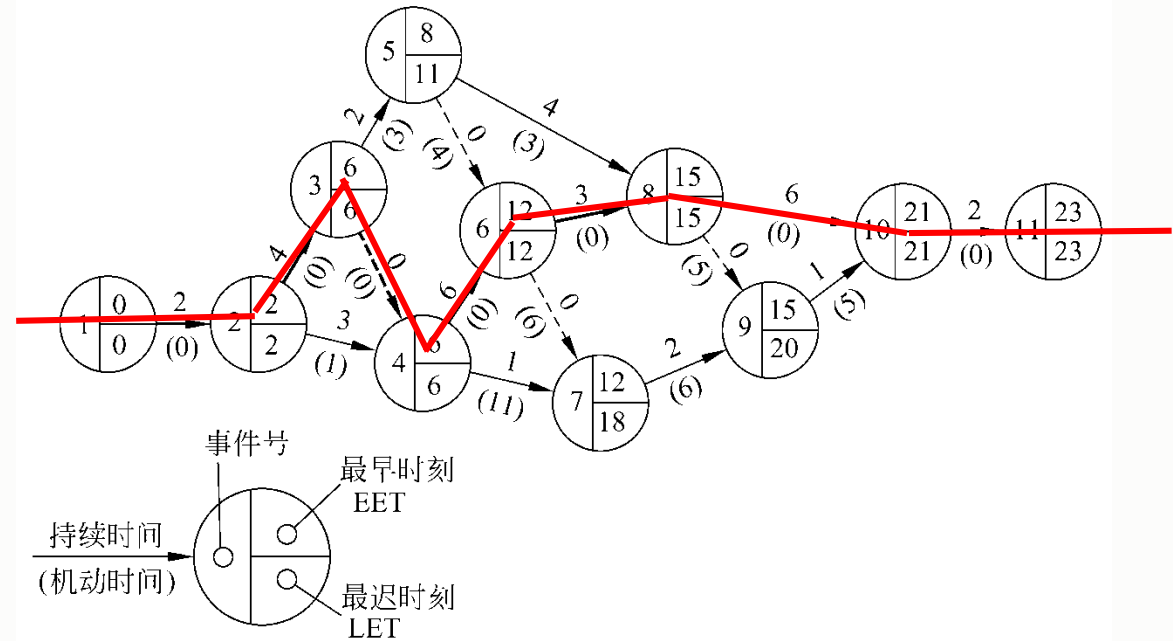
13.3 进度计划

13.3.5 关键路径

思考：
哪一条？

右图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**。

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 进度计划

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见下表：

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

13.3 进度计划

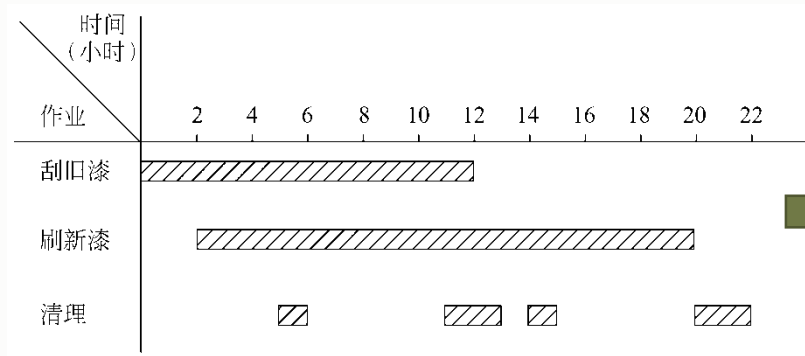


图13.1旧木板房刷漆工程的Gantt图

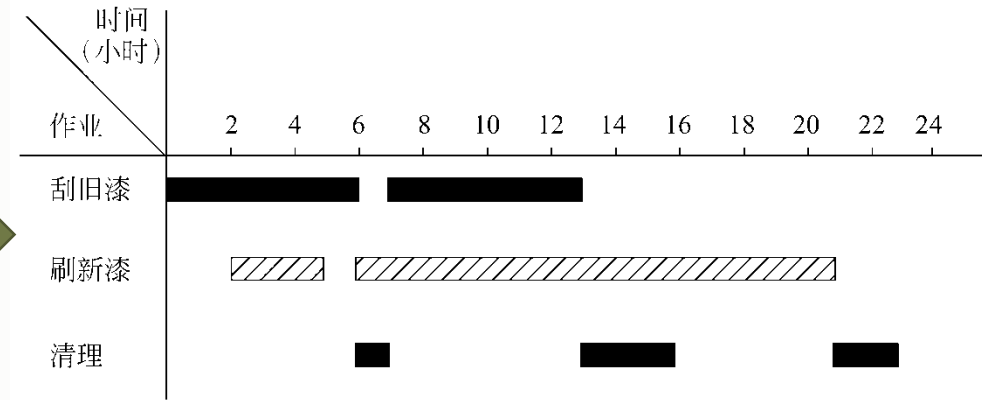


图13.4旧木板房刷漆工程改进的Gantt图之一

- 图13.4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**。
- 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。

主要内容

13.1 估算软件规模

13.2 工作量估算

13.3 进度计划



13.4 人员组织

13.5 质量保证

13.6 软件配置管理

13.7 能力成熟模型

13.4 人员组织

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。

- 小组成员之间的通信是平行的，如果小组内有 n 个成员，则可能的通信信道共有 $n(n-1)/2$ 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。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**。
- 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。

13.4 人员组织

- 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。
- 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。
- 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，当有技术难题需要解决时，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。

13.4 人员组织

13.4.2 主程序员组

IBM在20世纪70年代初期开始采用主程序员组的组织方式

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。

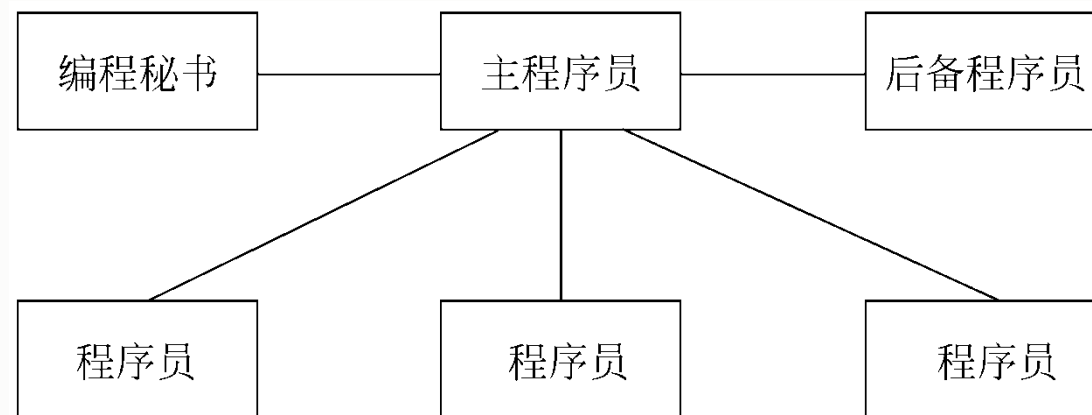
13.4 人员组织

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。

13.4 人员组织

当时，典型的主程序员组的组织形式如下图所示。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。



接下来介绍主程序员组的结构主程序员组核心人员的分工：

13.4 人员组织

(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。

13.4 人员组织

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，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。
- 再次，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。

人们需要一种更合理、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。

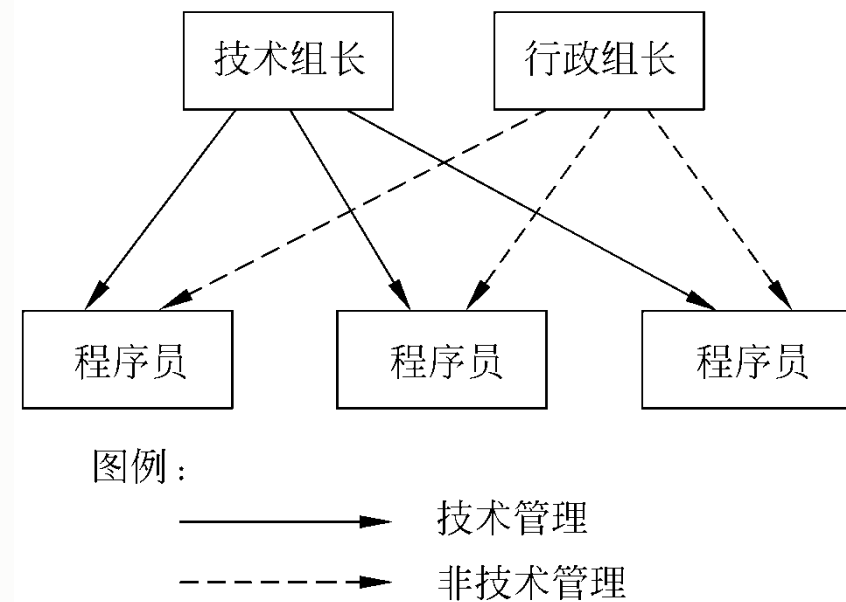
13.4 人员组织

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。

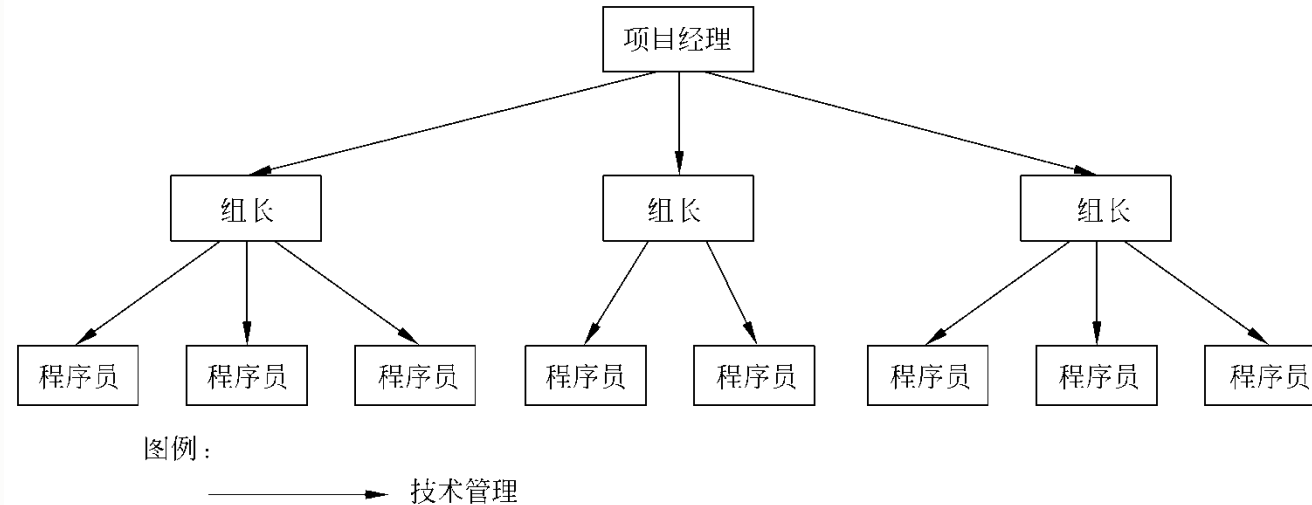
解决上述问题的方法是，取消主程序员的大部分行政管理工作。

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。



13.4 人员组织

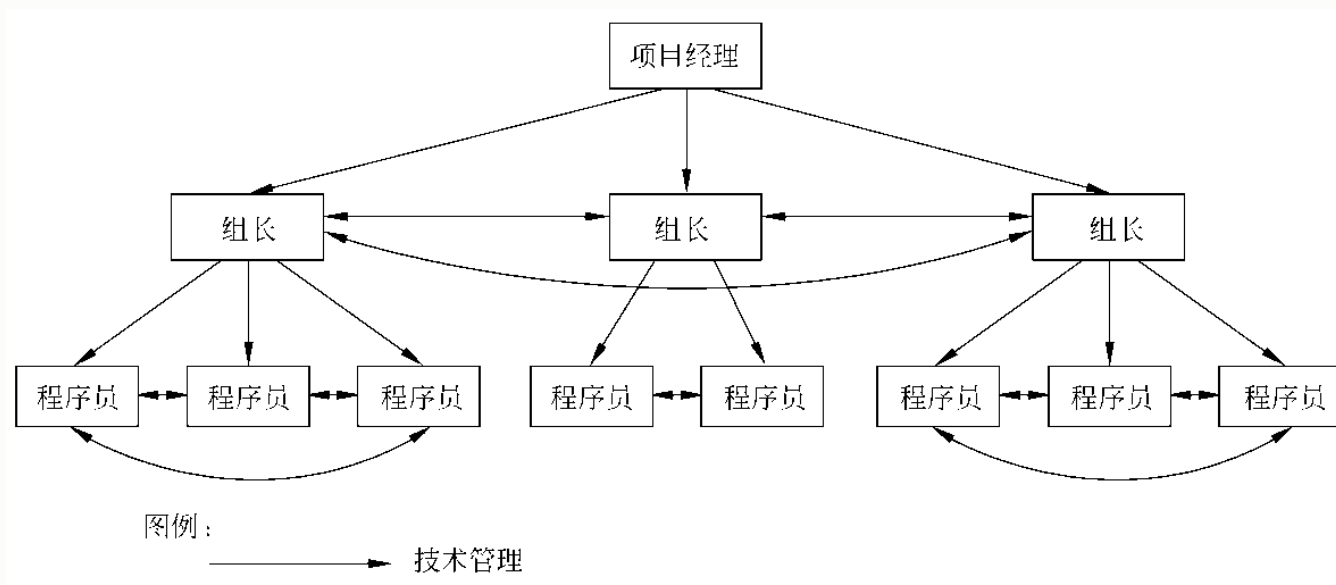
由于程序员组成员人数不宜过多，当软件项目规模较大时，应该把程序员分成若干小组，采用下图所示的组织结构。



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。

13.4 人员组织

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，如下图所示。



主要内容

13.1 估算软件规模

13.2 工作量估算

13.3 进度计划

13.4 人员组织



13.5 质量保证

13.6 软件配置管理

13.7 能力成熟模型

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。

1. 软件质量
2. 软件质量保证措施

13.5 质量保证

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。

反映用户在使用软件产品时的3种倾向是：产品运行、产品修改和产品转移。右图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，下页表列出了软件质量因素的简明定义。



13.5 质量保证

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 质量保证

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**，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。

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。

13.5 质量保证

1. 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。

走查主要有下述两种方式。

- （1）**参与者驱动法**。参与者提问，文档编写组解释。
 - （2）**文档驱动法**。文档编写者解释文档，走查组成员提出质疑
- 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。

13.5 质量保证

2. 审查

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。

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。

13.5 质量保证

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。

- 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。
- 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。
- 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。

主要内容

13.1 估算软件规模

13.2 工作量估算

13.3 进度计划

13.4 人员组织

13.5 质量保证

 **13.6 软件配置管理**

13.7 能力成熟模型

13.6 软件配置管理

13.6.1 软件配置

- 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，人们把它们统称为软件配置，而这些项就是软件配置项。
- 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。
- 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。

13.6 软件配置管理

13.6.2 软件配置管理过程

- 标识:标识软件配置中的对象，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；
- 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。
- 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。
- 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。
- 报告：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？

主要内容

13.1 估算软件规模

13.2 工作量估算

13.3 进度计划

13.4 人员组织

13.5 质量保证

13.6 软件配置管理



13.7 能力成熟模型

13.7 能力成熟模型

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。

能力成熟度模型的基本思想是，由于问题是由人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。

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。

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。

11.7 能力成熟模型

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。

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。

11.7 能力成熟模型

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。

11.7 能力成熟模型

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。

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。

11.7 能力成熟模型

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。

11.7 能力成熟模型

5. 优化级

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，将是多么困难。

本章小结

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。有效的管理是大型软件工程项目成功的关键。

1. 软件项目管理始于项目计划。为了估算项目工作量和完成期限，首先需要预测软件规模。
2. 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。
3. 软件质量保证是在软件过程中的每一步都进行的活动。
4. 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。
5. 能力成熟度模型（CMM）是改进软件过程的有效策略。