

第13章 软件项目管理

目录

- 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

软件项目管理

- 软件项目管理是指软件生存周期中软件管理者所进行的一系列活动，其目的是在一定的时间内，有效地利用人力、资源、技术和工具，使软件系统或软件产品按原定计划和质量要求如期完成
- 软件项目管理的基本内容
 - 项目计划
 - 项目组织
 - 项目控制

项目计划

- 项目计划是项目组织根据软件项目的目标及范围，对项目实施中进行的各项活动进行周密的计划
 - 生命周期模型
 - 组织结构
 - 责任分配
 - 管理目标
 - 优先级
 - 采用的技术和工具
 - 进度安排
 - 预算和资源分配

项目计划

- 制定计划的基础是**工作量估算**和**工期估算**
- 为了估算工作量和工期，首先需要估算**软件规模**

13.1 估算软件规模

软件度量

- 软件度量是指计算机软件范围内的测量，主要是为产品开发的软件过程和产品本身定义相关的测量方法和标度
 - 对软件开发过程度量的目的是为了对过程进行改进
 - 对产品进行度量的目的是为了产品的质量
- 需要考虑：
 - 合适的度量是什么
 - 所收集的数据如何使用
 - 用于比较个人、过程或产品的度量是否合理

软件度量的分类

- 软件规模的度量
 - 代码行技术 (LOC)
 - 功能点技术 (FP)
- 软件生产率的度量
 - KLOC/pm : 每人月开发的千行代码数
 - FP/pm : 每人月开发的功能点数
- 软件质量的度量
 - Errors/KLOC : 每千行代码所含有的错误数
 - Errors/FP : 每功能点所含有的错误数

基于代码行技术的度量

- 用代码行数表示软件规模
- 测量出软件规模后可方便地度量其它软件属性，包括：

度量名	含义及表示
LOC或KLOC	代码行数或千行代码数
生产率P	$P=LOC/E$ ，E为开发的工作量(常用人月数表示)
每行代码平均成本C	$C=S/LOC$ ，S为总成本
文档代码比D	$D=Pe/KLOC$ ，其中Pe为文档页数
代码错误率EQR	$EQR=N/KLOC$ ，其中N为代码中错误数

代码行技术的优缺点

- 优点：方便、直观
- 缺点：
 - (1) 源代码只是软件的一部分；
 - (2) 代码行数目很大程度上取决于程序设计语言以及软件设计的质量

基于功能点技术的度量

- 一种针对软件的功能特性进行度量的方法
- 主要考虑软件系统的“功能性”和“实用性”
- 功能点度量：基于软件信息域的特征(可直接测量)和软件复杂性进行规模度量
- 功能点度量方法步骤：
 - 计算信息域特征的值UFP
 - 计算复杂度调整值
 - 计算功能点FP

计算信息域特征的值UFP

对5个信息域特征统计相应的特征值，然后根据信息域特征的复杂程度选择适当的加权因子进行计算，得到总计的UFP值。

测量参数	特征值	加权因子			结果 (=特征值×加权因子)
		简单	中间	复杂	
用户输入数		×3	×4	×6	
用户输出数		×4	×5	×7	
用户查询数		×3	×4	×6	
文件数		×7	×10	×15	
外部接口数		×5	×7	×10	
总计UFP					

计算复杂度调整值

- 复杂度调整值 F_i ($i=1$ 到 14) 是基于对表中问题的回答而得到的值，对每个问题回答的取值范围是0到5
 - 0：没有影响
 - 1：偶然的
 - 2：适中的
 - 3：普通的
 - 4：重要的
 - 5：极重要的

	问 题	Fi (0-5)
1	系统需要可靠的备份和恢复吗？	
2	需要数据通信吗？	
3	有分布处理功能吗？	
4	性能很关键吗？	
5	系统是否在一个现存的、重负的操作环境中运行？	
6	系统需要联机数据登录？	
7	联机数据登录是是否需要在多屏幕或多操作之间切换以完成输入？	
8	需要联机更新文件吗？	
9	输入、输出、文件或查询很复杂吗？	
10	内部处理复杂吗？	
11	代码需要被设计成可复用的吗？	
12	设计中需要包括转换及安装吗？	
13	系统的设计支持不同组织的多次安装吗？	
14	应用的设计方便用户修改和使用吗？	
总 计		

计算功能点FP

- 功能点计算公式

$$FP = UFP \times (0.65 + 0.01 \times \sum_{i=1}^{14} F_i)$$

- 一旦计算出功能点，则用类似代码行的方法来计算软件生产率、质量及其他属性

度量名	含义表示
生产率P	$P=FP/E$ ，E为开发的工作量（常用人月数表示）
每个功能点成本C	$C=S/FP$ ，S为总成本
每个功能点文档数D	$D=Pe/FP$ ，其中Pe为文档页数
功能点错误率EQR	$EQR=N/FP$ ，其中N为错误数

功能点技术的优缺点

- 优点：与所用编程语言无关
- 缺点：
 - (1) 不直观
 - (2) 估计信息域特征的复杂程度和技术复杂性因子时，存在很大主观性

软件规模的估算

- 基于代码行（LOC）的估算
 - 问题分解（功能分解）
 - 对每个模块估算代码行
 - 求和得到总的代码行数
- 基于功能点（FP）的估算
 - 对5个信息域特征进行估计
 - 对14个复杂度调整值进行估计
 - 计算FP

每一项的估算

$$L = \frac{\bar{a} + 4\bar{m} + \bar{b}}{6}$$

1. 对每一项估计，每个人给出乐观估计（ a ）、悲观估计（ b ）、最可能估计（ m ）
2. 计算 a 、 b 、 m 平均值
3. 根据上述公式计算该项的估计

基于代码行的软件规模估算举例

功能	乐观值	可能值	悲观值	估计值
用户界面及控制机制（UICF）	1500	2200	3500	2300
二维几何图形分析（2DGA）	3800	5400	6400	5300
三维几何图形分析（3DGA）	4600	6900	8600	6800
数据库管理（DBM）	1850	3200	5450	3350
计算机图形显示机制（CGDF）	3100	4900	7000	4950
外部设备控制（PCF）	1400	2150	2600	2100
设计分析模块（DAM）	6200	8500	10200	8400
估算出的总代码行数				33200

基于功能点的软件规模估算举例

Information domain value	Opt.	Likely	Pess.	Est. count	Weight
Number of external inputs	20	24	30	24	4
Number of external outputs	12	15	22	16	5
Number of external inquiries	16	22	28	22	5
Number of internal logical files	4	4	5	4	10
Number of external interface files	2	2	3	2	7

$UPF = 96 + 80 + 110 + 40 + 14 = 340$

$FP = 340 * (0.65 + 0.01 * 52) = 398$

96
80
110
40
14

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5

Sum: 52

13.2 工作量估算

- 工作量的单位是人月（pm）
- 常用的估算方法：
 - 基于已经完成的类似项目进行估算，这是一种常用的也是有效的估算方法
 - 基于分解技术进行估算
 - 问题分解是将一个复杂问题分解成若干个小问题，通过对小问题的估算得到复杂问题的估算
 - 过程分解指先根据软件开发过程中的活动(分析、设计、编码、测试等)进行估算，然后得到整个项目的估算值。
 - 基于经验估算模型的估算。如：静态单变量模型、动态多变量模型、CoCoMo2模型等。
- 上述方法可以组合使用以提高估算的精度

基于分解技术的工作量估算方法

过程分解

问题分解

Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function ▼									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

静态单变量模型

$$E = A + B \times \nu^C$$

- A、B、C是由经验数据导出的常数
- E是以人月为单位的工作量
- ν ：估算变量（KLOC或FP）
- 不同的模型，即使使用相同的变量，A、B、C不同
- 根据若干应用领域中有限个项目的经验数据推导出来，适用范围有限

静态单变量模型应用举例

- 某公司曾经完成过5个软件开发项目，有关这些项目的数据记录在下表中。请根据这些历史数据计算静态单变量模型中的参数值，并且估算完成一个30KLOC的项目需要多大工作量。

项目序号	规模（KLOC）	工作量（PM）
1	50	120
2	80	192
3	40	96
4	10	24
5	20	48

解答

- 通过回归分析找出静态单变量模型中的A、B、C值

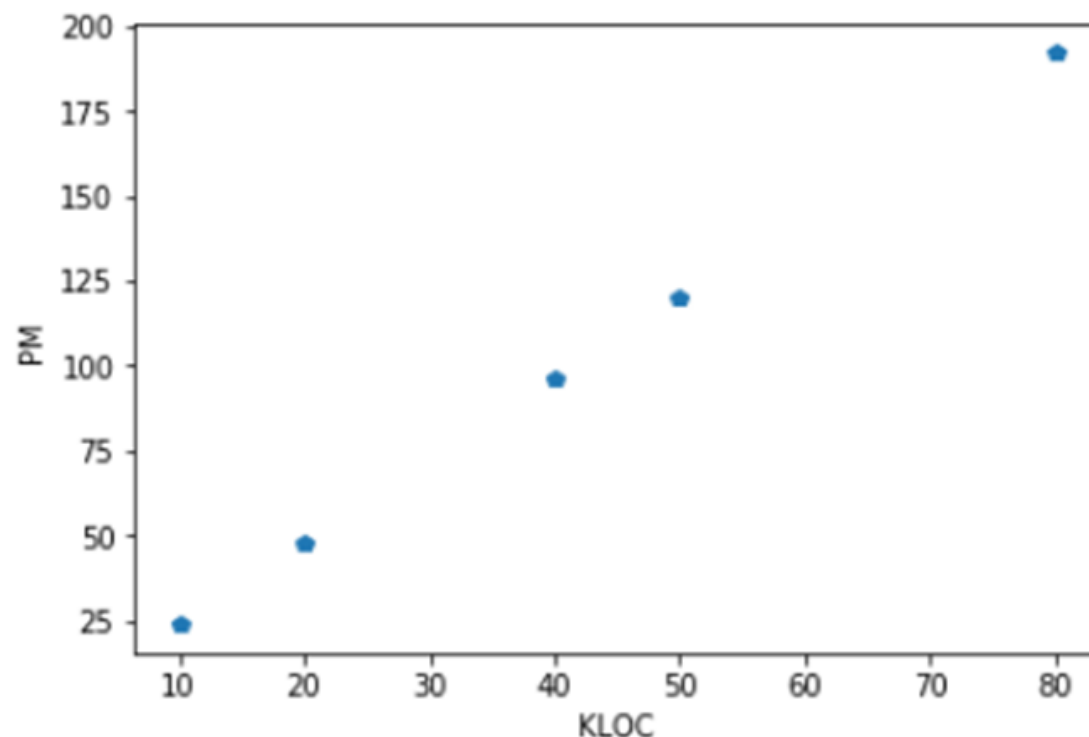
```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

```
def func(x, a, b, c):
    return a + b * pow(x, c)
```

$$E = A + B \times v^C$$

```
xdata = [50, 80, 40, 10, 20]
ydata = [120, 192, 96, 24, 48]
```

```
plt.plot(xdata, ydata, 'p', label="data")
plt.xlabel("KLOC")
plt.ylabel("PM")
plt.show()
```



```
popt, pcov = curve_fit(func, xdata, ydata)
print(popt)
```

```
[-2.20046203e-12  2.40000000e+00  1.00000000e+00]
```

A=0

B=2.4

C=1

动态多变量模型（软件方程式）

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

- E：人月或人年为单位的工作量
- t：以月或年为单位的项目持续时间
- B：特殊技术因子
- P：生产率参数

CoCoMo2模型

- 1981年，Boehm提出CoCoMo模型（ Constructive Cost Model ）
- 1997年，CoCoMo2模型
- CoCoMo2模型给出了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
 - 应用系统组成模型（ Application composition model ）：用于估算构建原型的工作量
 - 早期设计模型（ Early design model ）：用于体系结构设计阶段
 - 后体系结构模型（ Post-architecture model ）：用于完成体系结构设计之后的软件开发阶段

后体系结构模型

- 把软件开发工作量表示成代码行数（KLOC）的非线性函数

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

E：开发工作量（以人月为单位）

A：模型系数

KLOC：估计的源代码行数（以千行为单位）

b：模型指数

f_i (i=1~17)：成本因素（见教材P310：表13.3）

13.3 进度计划

进度计划的基本原则

- 划分：必须把项目划分成若干个可以管理的活动和任务。
- 相互依赖性：必须确定各个活动或任务之间的相互依赖性。
- 时间分配：必须给每个任务都分配一定数量的工作单位（人×天），指定开始日期和结束日期（同时考虑相互依赖性）。
- 工作量确认：必须确保在任意时段内分配给任务的人员数量，不超过项目组中的人员数量。
- 定义责任：每个任务都应该指定具体的负责人。
- 定义结果：每个任务都应该有一个定义好的输出结果。
- 定义里程碑：应该为每个任务或每组任务指定一个项目里程碑。

进度计划的步骤

1. 将项目自顶向下分解成多个任务，描述任务之间的依赖关系
2. 估算每项任务的工作量，分配人力资源，确定每项任务的持续时间
3. 利用Gantt图和工程网络安排进度

进度计划举例

- 假设有一座陈旧的矩形木板房（4面墙）需要重新油漆。这项工作必须分3道工序完成：
 - 首先刮掉油漆，
 - 然后刷上新漆，
 - 最后清除溅在窗户上的油漆。
- 每一面墙的工作量见右表。
- 一共分配了15名工人去完成这项工作。
- 可用工具：
 - 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

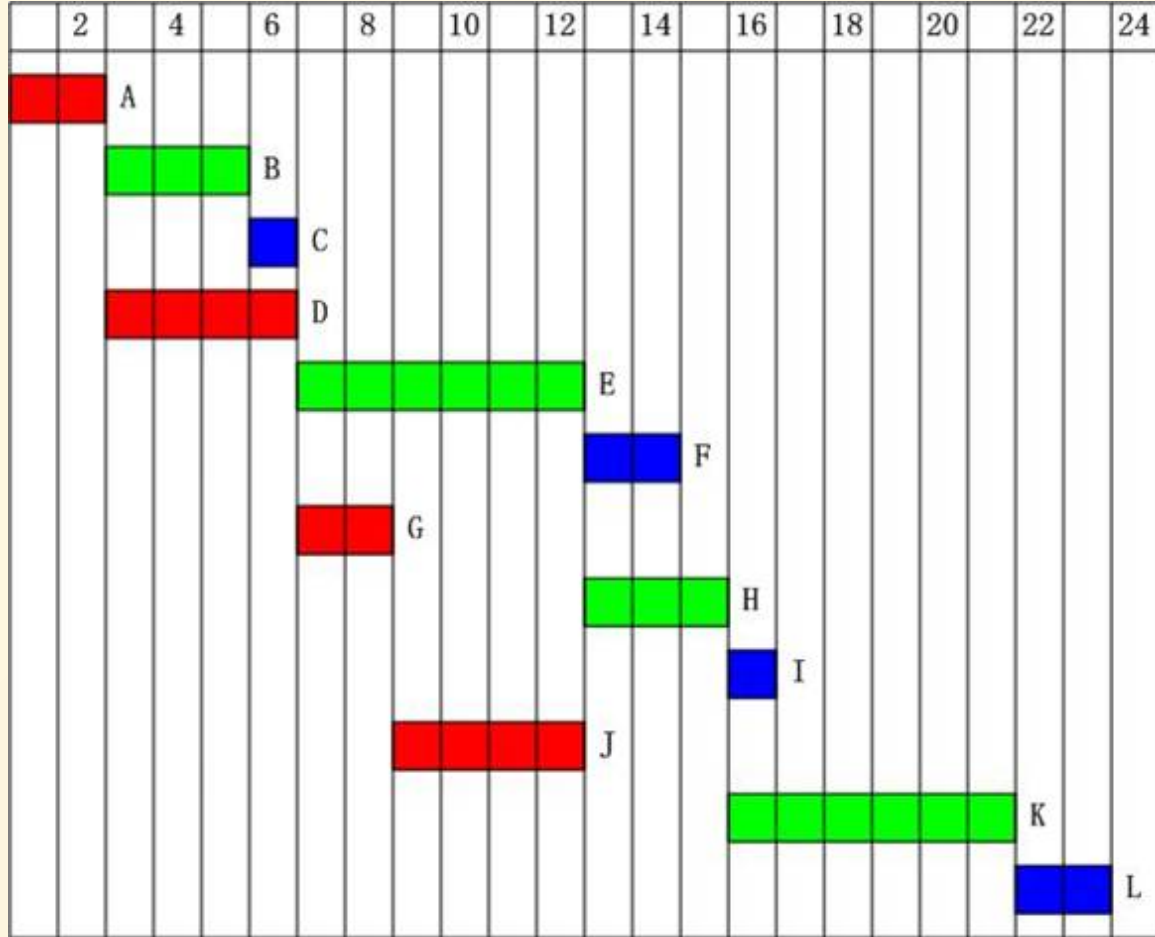
墙面	刮旧漆	刷新漆	清理
东	10	15	5
南	20	30	10
西	10	15	5
北	20	30	10

每道工序的工作量（人×时）

子任务列表

任务	工时	依赖于...
A. 第1面墙刮旧漆	2	
B. 第1面墙刷新漆	3	A
C. 第1面墙清理	1	B
D. 第2面墙刮旧漆	4	A
E. 第2面墙刷新漆	6	B, D
F. 第2面墙清理	2	C, E
G. 第3面墙刮旧漆	2	D
H. 第3面墙刷新漆	3	E, G
I. 第3面墙清理	1	F, H
J. 第4面墙刮旧漆	4	G
K. 第4面墙刷新漆	6	H, J
L. 第4面墙清理	2	I, K

甘特图（Gantt图）

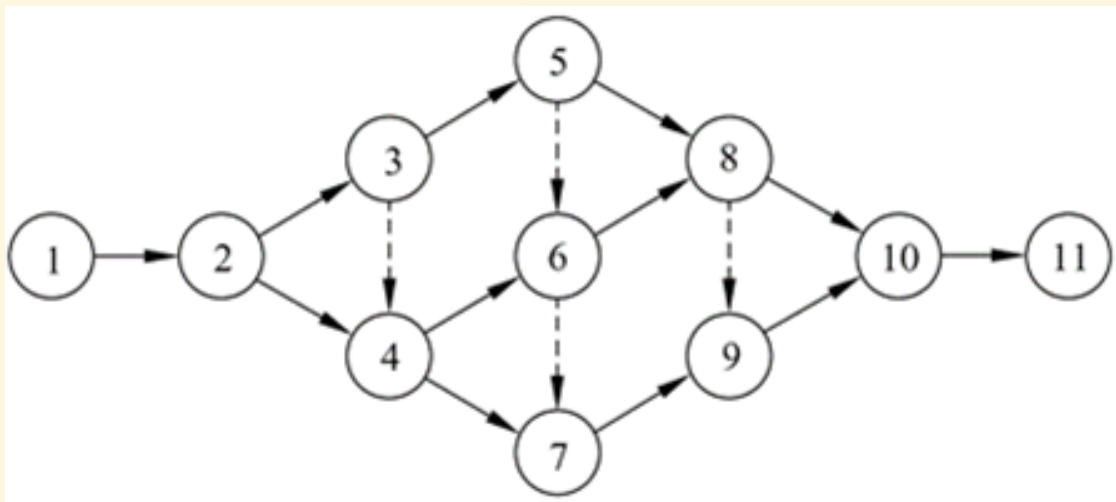


- Gantt图是一种应用广泛的进度计划工具
- 缺点
 - 不能显式地描绘各项作业彼此间的依赖关系
 - 进度计划的关键部分不明确，难于判定哪些部分应当是主攻对象
 - 计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费

工程网络

- 工程网络是制订进度计划的另一种常用的图形工具
- 描绘任务分解情况以及每项作业的开始时间和结束时间
- 显式地描绘各个作业彼此间的依赖关系

工程网络举例



旧木板房刷漆工程的工程网络（尚待完善）：

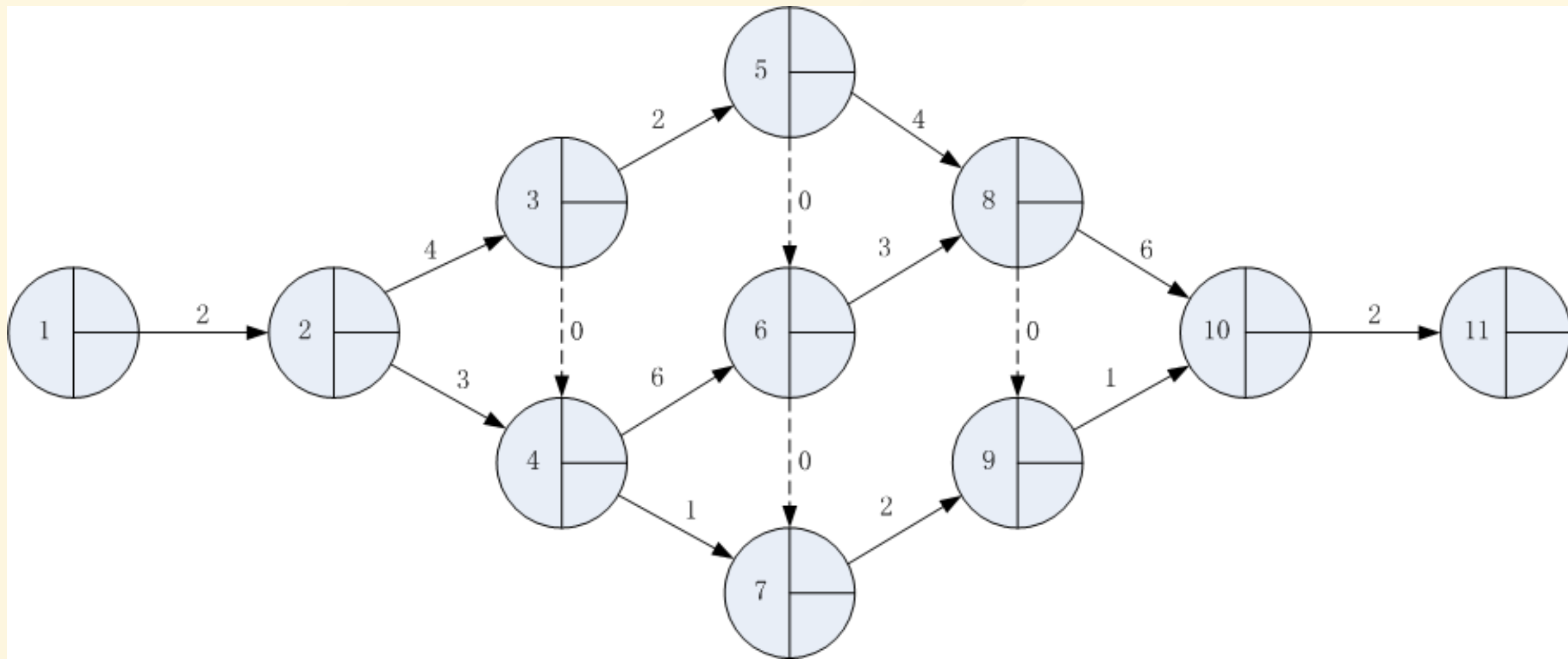
- 箭头表示作业
- 圆圈表示事件
- 虚线箭头表示虚拟作业，用于表示依赖关系。虚拟作业不消耗资源和时间。

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面墙上的旧漆;
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面墙上的旧漆;
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

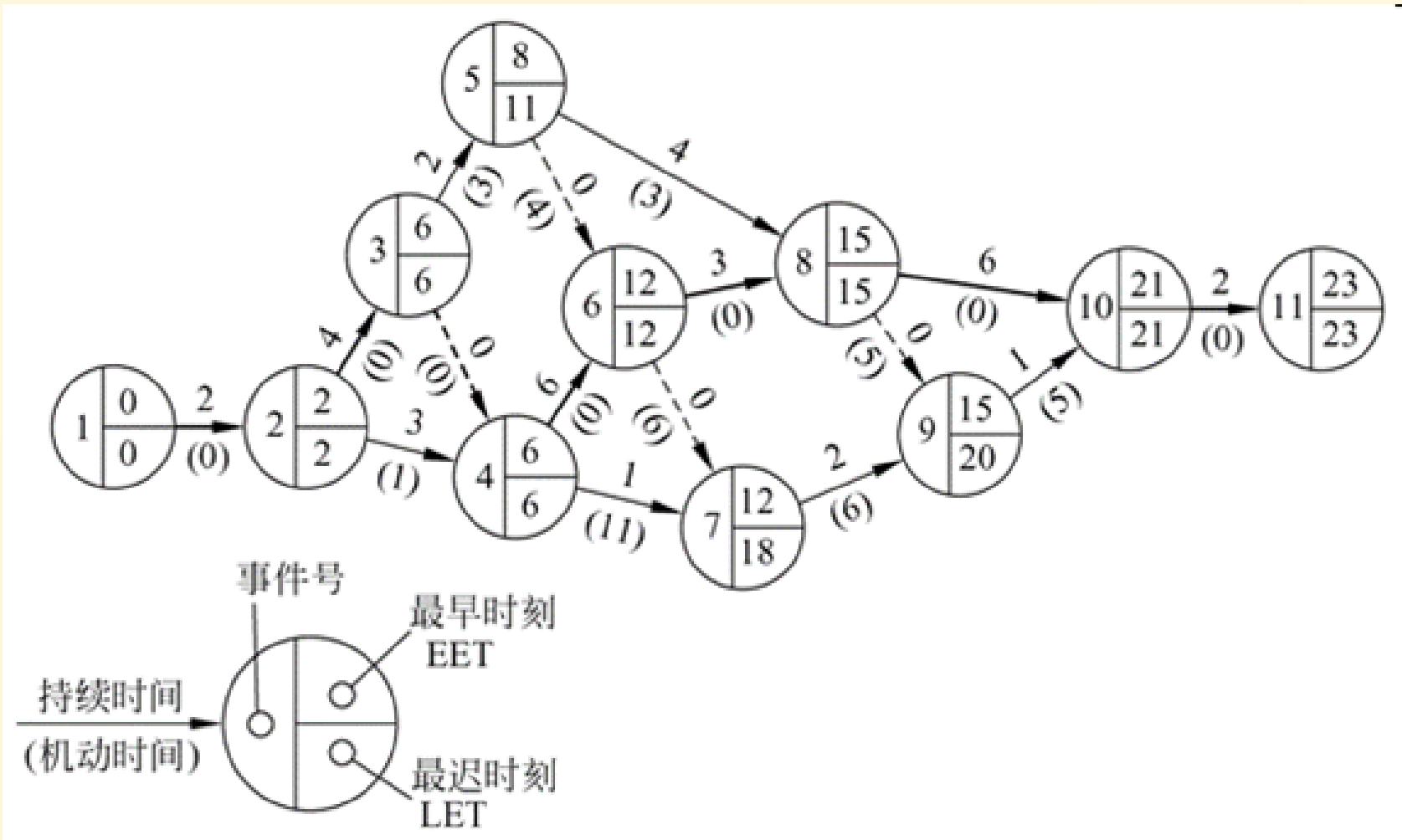
利用工程网络估算进度

- 事件的最早时刻 (EET)
 - 计算方法：按事件发生顺序计算
 - 计算步骤：
 - 考虑进入该事件的所有作业
 - 对于每个作业都计算它的持续时间与起始事件的EET之和
 - 选取上述和数中的最大值作为该事件的最早时刻EET
- 事件的最迟时刻 (LET)
 - 计算方法：逆作业流的方向计算
 - 计算步骤：
 - 最后一个事件(工程结束)的最迟时刻就是它的最早时刻
 - 考虑离开该事件的所有作业
 - 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
 - 选取上述差数中的最小值作为该事件的最迟时刻LET

工程网络举例：计算EET和LET



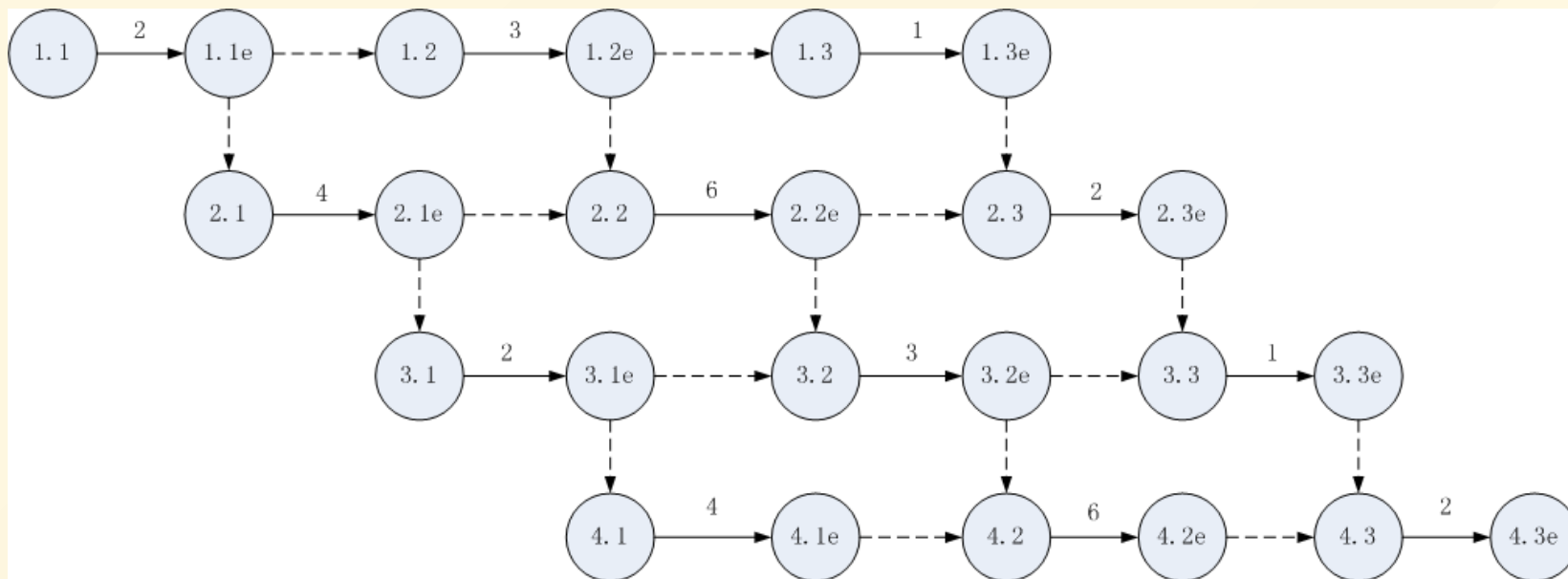
工程网络举例：完整的工程网络



关键路径和机动时间

- 关键事件： $EET=LET$
 - 关键路径：工程网络中由关键事件组成的路径
 - 关键路径的长度（该路径上的作业持续时间的总和）即为工程的总时间
 - 关键路径上的事件必须准时发生，即关键路径上的作业必须准时完成，否则工程就会延误
 - 作业的机动时间：不在关键路径上的作业有一定程度的机动余地
- 机动时间 = $(LET)_{\text{结束}} - (EET)_{\text{开始}} - \text{作业持续时间}$

工程网络举例：更直观的画法



- 圆圈表示事件
- 箭头表示作业
- 虚线箭头表示虚拟作业，用于表示依赖关系。虚拟作业不消耗资源和时间。

13.4 人员组织

软件开发组织形式

多名软件开发人员合理组织起来，分工协作完成开发工作。

- 民主制程序员组
- 主程序员组
- 现代程序员组

民主制程序员组

- 特点
 - 小组成员完全平等，享有充分民主，通过协商做出技术决策。
 - 为减少通信，小组规模小，人员少而精。
- 优点
 - 组员对发现程序错误持积极态度，有助于提高软件质量。
 - 小组有高度凝聚力，有利于攻克技术难题。
- 缺点
 - 缺少必要的权威作用。
 - 通信开销较大。
- 适用领域：适合于研制时间长、开发难度大的项目。

民主制程序员组通信开销举例

假设一个人单独开发软件5000行/人月，4人一组共同开发且采用民主制小组形式，每条路径耗费工作量250行/人月，分析生产率变化情况。

答：4人共6条通信路径，每人生产率降低： $250 \times 6/4 = 375$ 行/人月

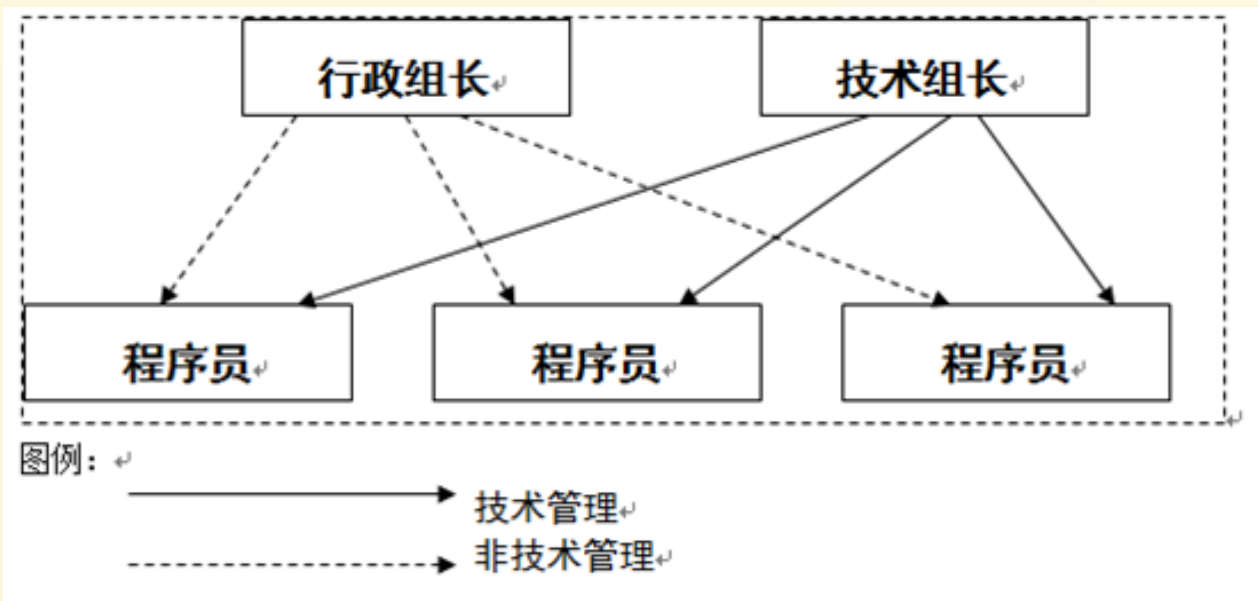
主程序员组

- 分工：
 - 主程序员：负责体系结构设计和关键部分的详细设计，负责指导其他程序员完成详细设计和编码工作。
 - 后备主程序员：协助主程序员工作并且在必要时接替主程序员的工作。
 - 编程秘书：负责完成与项目有关的全部事务性工作。
- 特点：
 - 专业化：每名成员完成受过专业训练的工作。
 - 层次化：主程序员有绝对权威。

现代程序员组

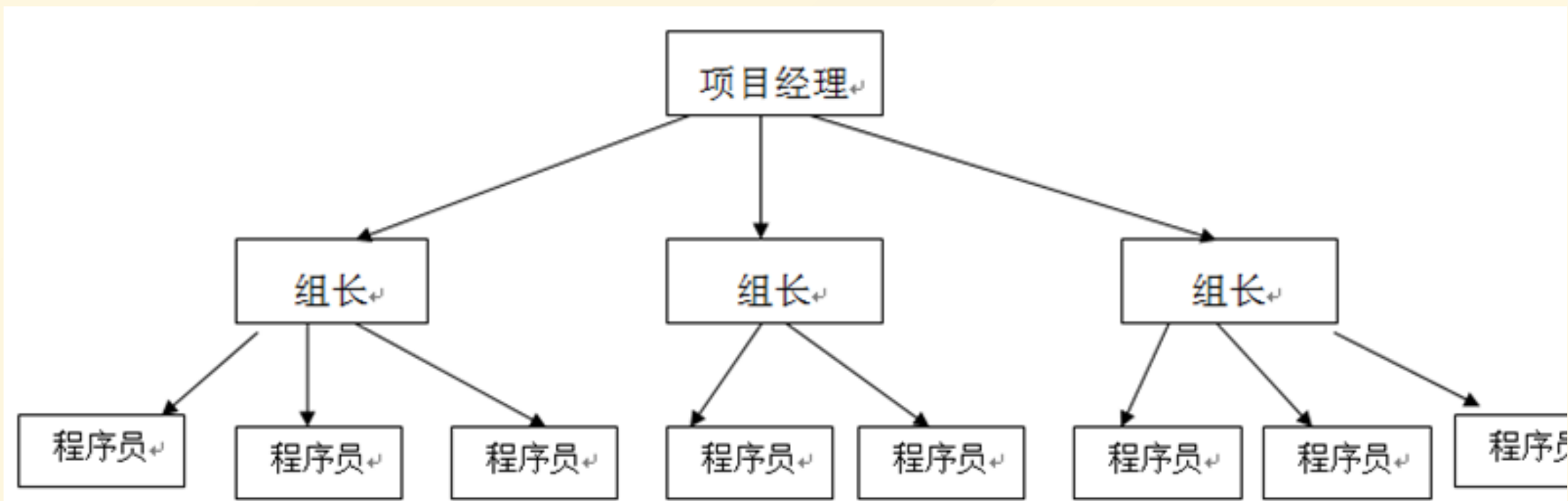
主程序员的职责由两人担任：

- 技术负责人：负责小组的技术活动。
- 行政负责人：负责所有非技术性事务的管理决策。



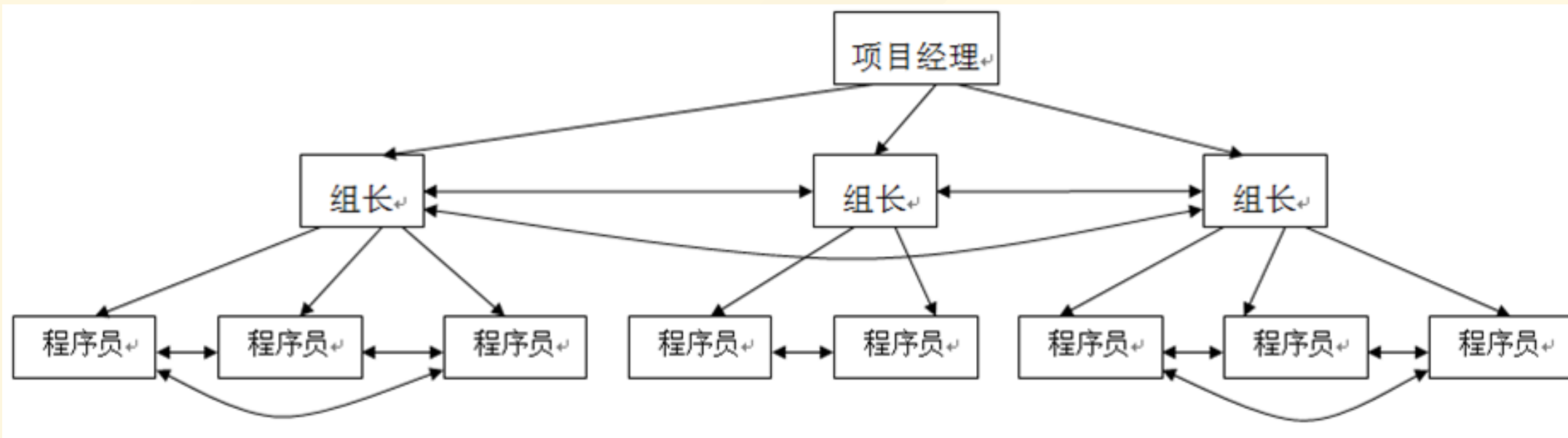
大型项目的技术管理组织结构

软件项目规模较大，程序员组分成若干小组。



包含分散决策的组织形式

将民主式程序员组与主程序员组的优点结合进来，形成包含分散决策组织形式。



13.5 质量保证

软件质量的定义

软件质量：与软件产品满足规定的和隐含的需求能力有关的特征或特性全体。

- 软件需求是度量软件质量的基础。
- 如果不遵守规范化开发准则，质量不能保证。
- 不能忽略隐含需求。

影响软件质量的因素

- 产品运行
 - 正确性、健壮性、效率、完整性、可用性、风险
- 产品修改
 - 可理解性、可维修性、灵活性、可测试性
- 产品转移
 - 可移植性、可再用性、互运行性

软件质量保证措施

- 基于非执行的测试：复审或评审
- 基于执行的测试：软件测试
- 程序正确性证明

13.6 软件配置管理

软件配置项

- (1) 计算机程序（源程序及目标程序）；
- (2) 文档（包括技术文档和用户文档）；
- (3) 数据。

基线（Baseline）

IEEE定义：已经通过正式复审的规格说明或中间产品，可作为进一步开发基础，并且只有通过正式的变化控制才能改变它。

软件配置管理过程

1. 标识软件配置中的对象
2. 版本控制
3. 变化控制
4. 配置审计
5. 状态报告

本章重点

- 软件规模的度量和估算，基于代码行的方法和基于功能点的方法。
- 工作量的估算。
- 进度计划的制订，Gantt图和工程网络图。