第七章 软件项目管理

周 瑞

QQ群：235458708

信息与软件工程学院

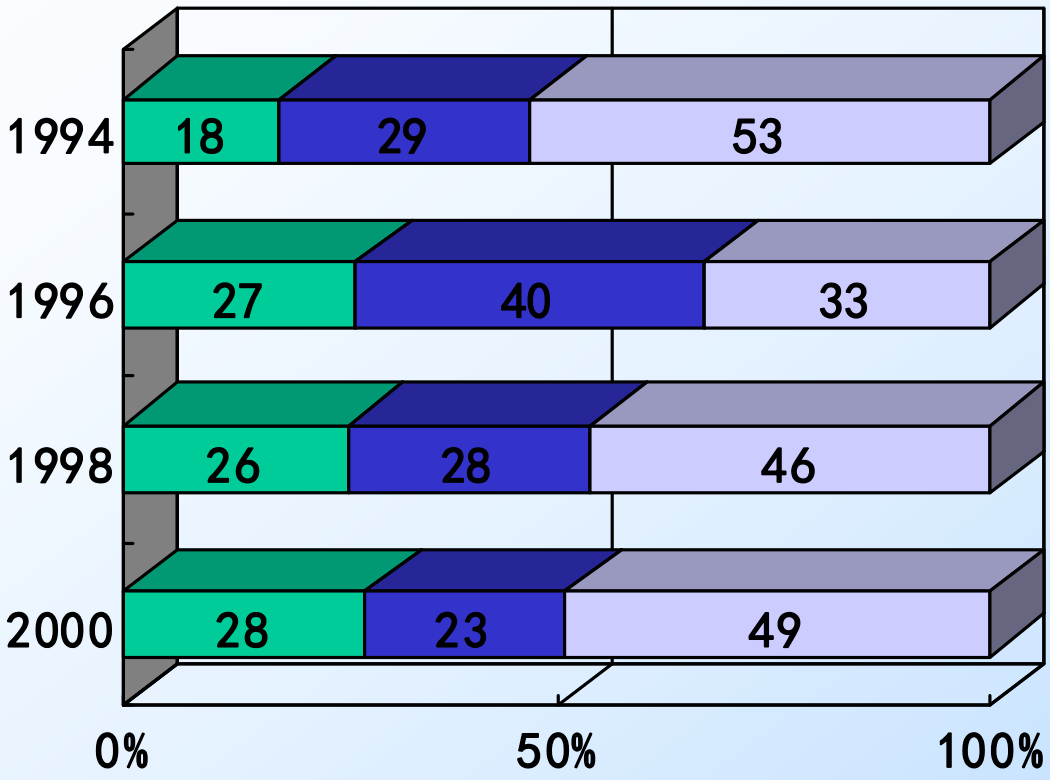
本章学习目标

1. 理解软件项目管理的四个基本要素
2. 掌握软件项目管理的主要内容和主要方法
3. 能根据具体项目进行项目估算和项目计划

本章内容

- 软件项目管理概述
- 人员组织与管理
- 软件度量
- 项目进度计划
- 软件风险管理
- 软件配置管理

软件项目现状



项目失败：

进度超出50%

费用超出50%

项目被中止

进度超期：89%

成本超额：62%

项目失败原因

- 项目目标和需求不清晰、不完整
  - 目标的错误，导致开发和交付产品的一系列活动的浪费
  - 需求过多、需求不稳定、需求模棱两可、需求不完整
  - 导致大量返工，造成资源更加紧张
  - 错误的需求往往在项目生命周期的晚期才被发现，带来高昂的代价
- 拙劣的质量管理
  - 软件产品的质量工程没有被贯彻到项目实际活动中
  - 分配于质量活动的资源经常被占用
  - 软件产品的质量工程活动本身的不完善

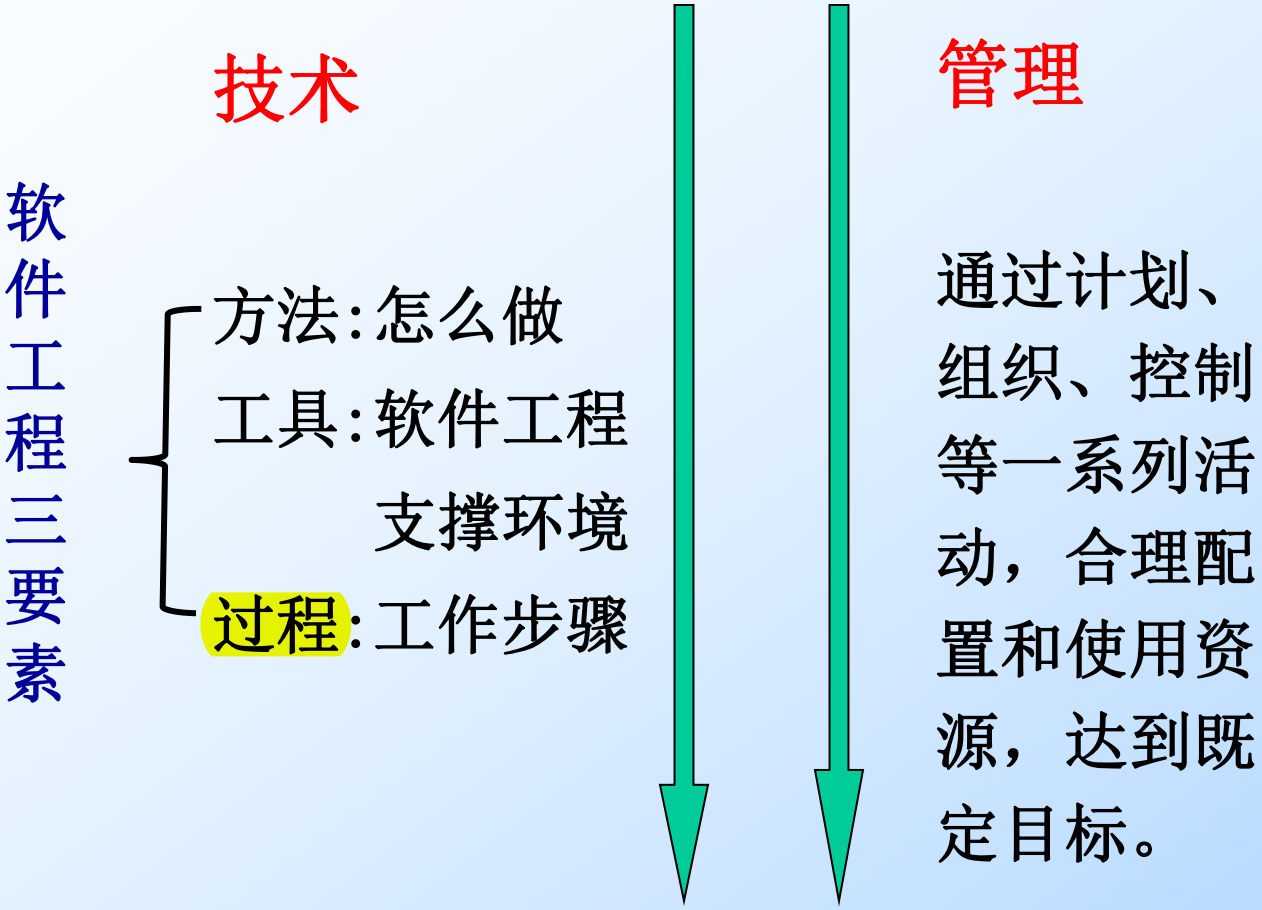
项目失败原因

- 项目进度的不切实际
  - 难以准确估算进度，或由不具备准确估算的人作出进度计划（市场人员或客户）
  - 紧张的进度使得开发更关注功能，系统性能达不到要求
- 短期利益行为
  - 在作技术方面的决策时，采用最容易想到或实施的方案，但这往往带来潜在的、严重的长期危害性
  - 只关注系统的快速实现，忽略为系统后期维护而进行的铺垫性工作
  - 系统构建过程中的验证措施被忽略，导致问题在后期暴露
  - 优先考虑功能，忽视了性能和质量

项目失败原因

- 采用了新技术，由于开发人员对新技术的掌控能力存在问题导致失败
- 风险管理不佳
- 团队成员大多是凸现个性、追求随意、无约束的创造性工作者，与其说是一个团队（team），很多情况下更像一个群体（group）
- ……

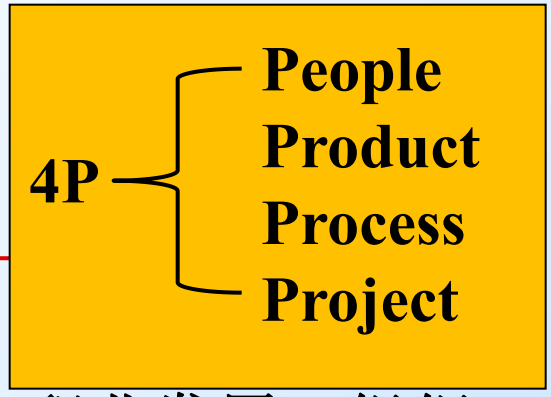
软件工程的研究内容


技术
管理
软件工程三要素
方法:怎么做
工具:软件工程支撑环境
过程:工作步骤
通过计划、组织、控制等一系列活动，合理配置和使用资源，达到既定目标。

软件项目管理

- 为了使软件项目能够按照预定的成本、进度和质量顺利完成，而对成本、人员、进度、质量、风险等进行分析和管理的活动。
- 软件项目管理定义(IEEE610.12-90)：计划、协调、度量、监控、控制及报告等管理方法在软件开发和维护中的具体应用，以保证整个过程是系统的、有原则的、可量化的。
- 目标：
  - 软件产品达到预期的功能和性能要求（质量）
  - 项目在合同期限内完成和交付（进度）
  - 项目开销控制在预算之内（成本）
- 先于任何技术活动之前，贯穿整个软件生命周期
- 对软件开发是否成功具有决定意义

软件项目管理的基本要素



- 人员管理

  招聘、选拔、绩效管理、培训、薪酬、职业发展、组织和工作设计、团队/文化的发展

- 产品管理

  产品的目标和范围、解决办法，以及技术和管理

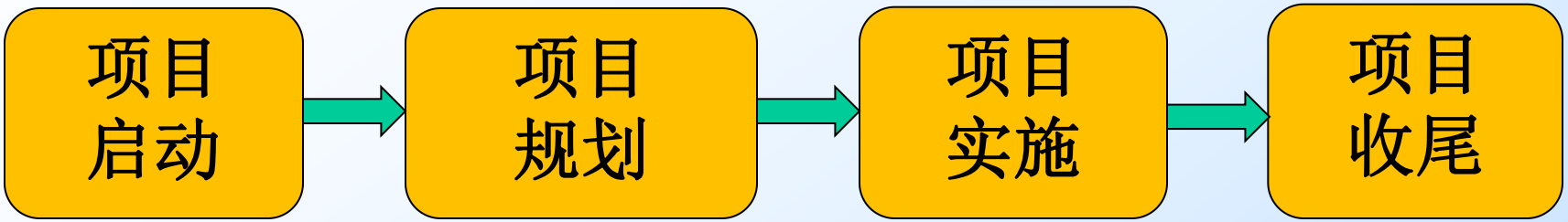
- 过程管理

  软件开发的一个全面计划

- 项目管理

  理解成功项目管理的关键因素，掌握项目计划、监控和控制的一般方法

软件项目管理活动



- 确定项目范围
- 组建项目团队
- 建立项目环境

- 确定软件过程、活动、里程碑、可交付文档以及需要的资源
- 进行项目估算
- 制定进度计划

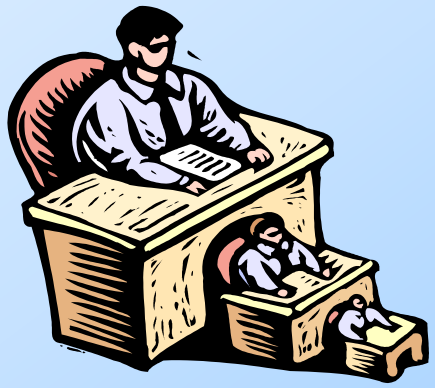
- 监控项目执行
- 管理项目风险
- 控制项目变更

- 客户验收项目
- 安装和培训
- 总结项目经验

- 软件项目管理概述
- 人员组织与管理
- 软件度量
- 项目进度计划
- 软件风险管理
- 软件配置管理

人员组织与管理

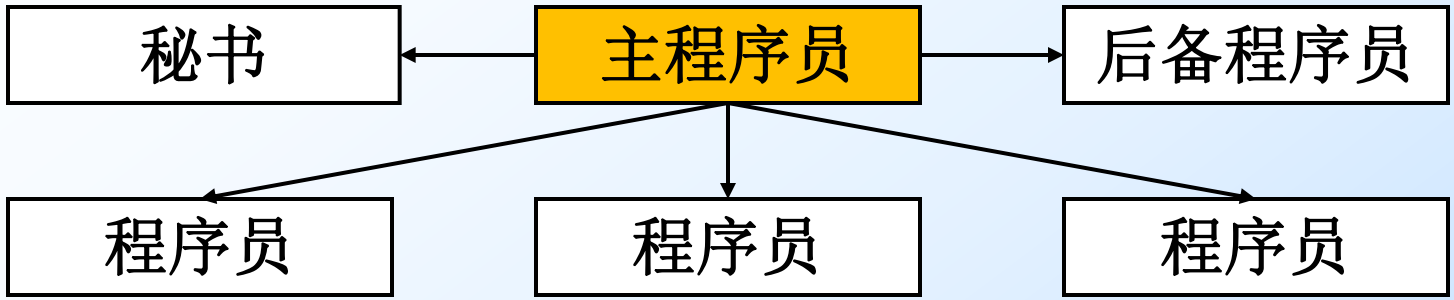
- 取决于开发组织的管理模式和软件项目的特点
- 一个项目管理的好坏，很大程度就体现在团队的建设和管理上
- 人力资源管理成熟度模型（PCMM）
- 典型的组织方式：
  - 民主式组织结构
  - 主程序员式组织结构
  - 技术管理式组织结构



人员组织与管理

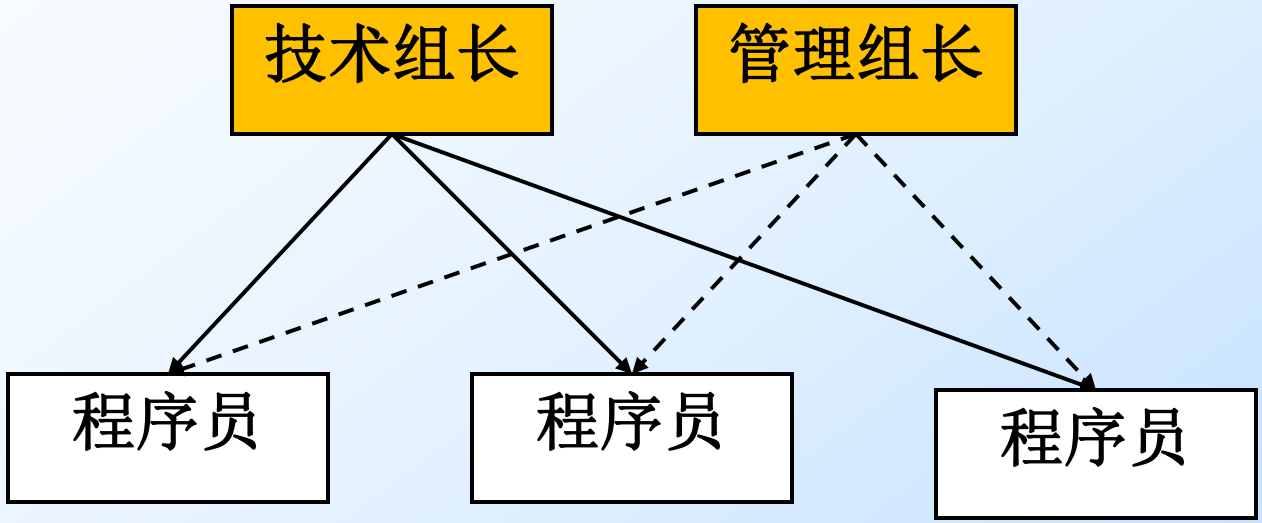
- 民主式组织结构
  - 小组成员完全平等
  - 项目工作由全体人员讨论决定
  - 适合规模小、能力强、习惯于共同工作的开发组
  - 无权威领导，难解决意见分歧，不适合大规模开发

人员组织与管理-主程序员式

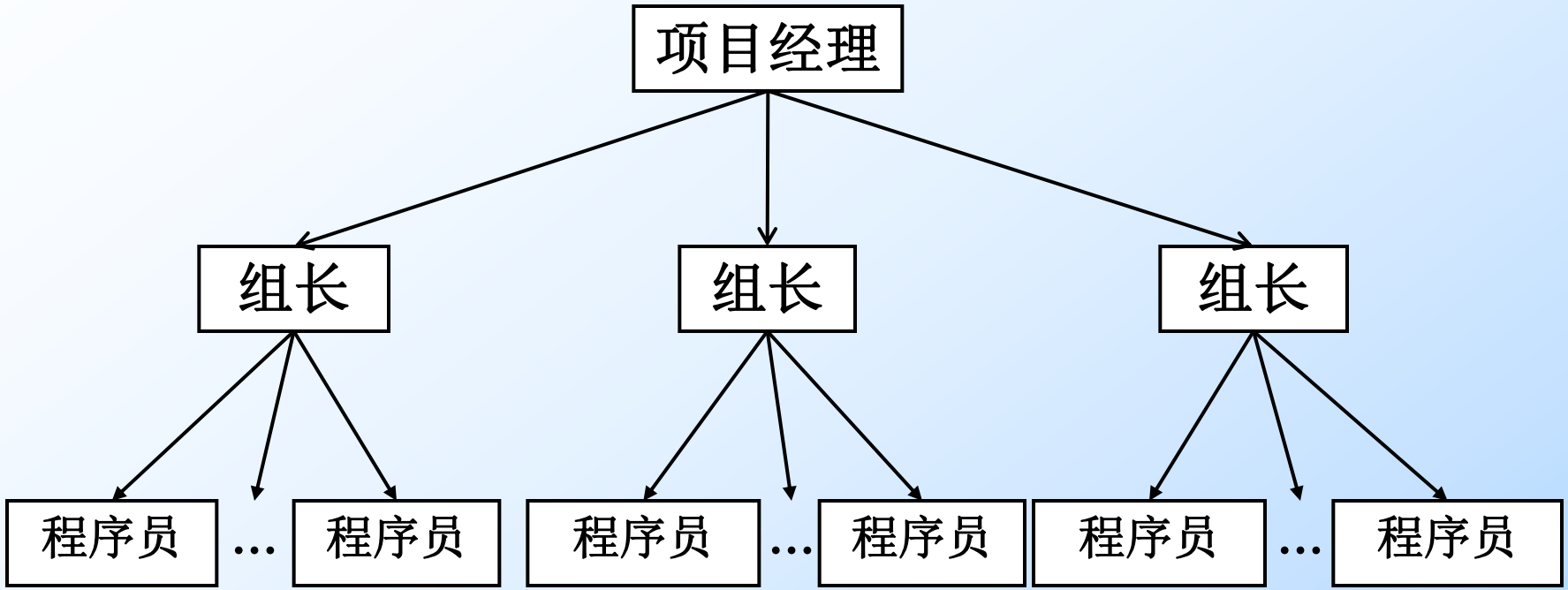


- 主程序员：体系结构设计和关键部分详细设计，管理和指导其他程序员
- 后备程序员：协助主程序员
- 程序员：详细设计和编程
- 优点：实现项目人员的专业化分工，提高了效率
- 缺点：对主程序员要求过高

人员组织与管理-技术管理式



人员组织与管理-技术管理式



大型项目的技术管理式组织结构

项目经理的技能要求

优秀的领导能力

高效的时间管理能力

快速的应变能力

非凡的沟通能力

良好的人际交往能力

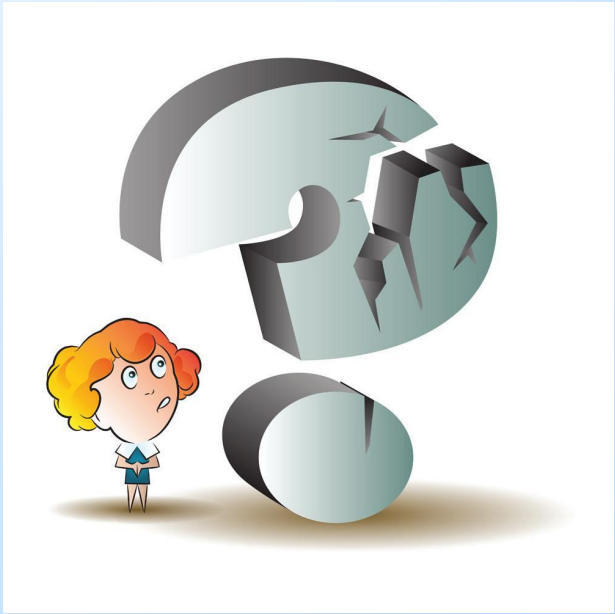
高效的激励能力

运营项目管理知识的能力

- 软件项目管理概述
- 人员组织与管理
- 软件度量
- 项目进度计划
- 软件风险管理
- 软件配置管理

思考 — 关于实践项目

- 请估计
  - 要完成该项目需要多少工作量？
  - 要完成该项目需要多长时间？
  - 要完成该项目需要多少人？
  - 项目的总成本是多少？



项目估算

- 目的：为项目分配合理的人力、时间和相关资源
  - 项目有多少代码行？
  - 项目有多少功能点？
  - 项目需要多少人月？
  - 平均生产率是多少？
- 软件度量：一种量化衡量方法，使得人们可以理解和把握软件项目的(生产)效率(或者所需要的劳动量)
  - 软件规模
  - 工作量
  - 生产率
  - 进度

软件度量

- 软件规模：软件产品的大小
  - 直接测量：以代码行表示，以千代码行为单位（KLOC）
  - 间接测量：以功能点表示，以功能点为单位（FP）
- 工作量：投入的人力
  - 是软件规模的函数，以人月为单位（PM）
- 生产率：
  - 直接测量/基于规模(KLOC)：如在一个特定时间内产生的代码行数
  - 间接测量/基于功能点(FP)：如一个给定时间内生产出的功能点和目标点
- 进度：预计的开发时间，以月为单位（M）

软件规模估算

- 直接测量/代码行技术
- 间接测量/功能点技术

软件规模估算-代码行技术

① 将软件分解成尽量小且可独立估算的子功能

② 计算每个子功能的代码行数

- 最小代码行数估算值：a
- 最可能的代码行数估算值：m
- 最大代码行数估算值：b
- 代码行数期望值：$L=(a+4m+b)/6$

③ 将所有子功能的代码行数期望值相加得到系统的代码行数

④ 由多名有经验的开发人员分别给出估算，然后得出估算的平均值

软件规模估算-代码行技术：案例

XXX银行信息系统

该系统应能增加新客户，并能从客户文件中作删除。系统支持客户的存款和取款业务。在出现透支时，系统应给出警告信息。客户可通过终端查询自己的账户余额，可以要求系统给出自己的透支报告。

软件规模估算-代码行技术：案例

- 子功能
  - 客户管理 600，750，900 → 750
  - 存款业务 500，600，800 → 620
  - 取款业务 1000
  - 查询业务 1100
  - 透支处理 800

＋ 4270行

- 代码行数估计值：4270行

软件规模估算-代码行技术

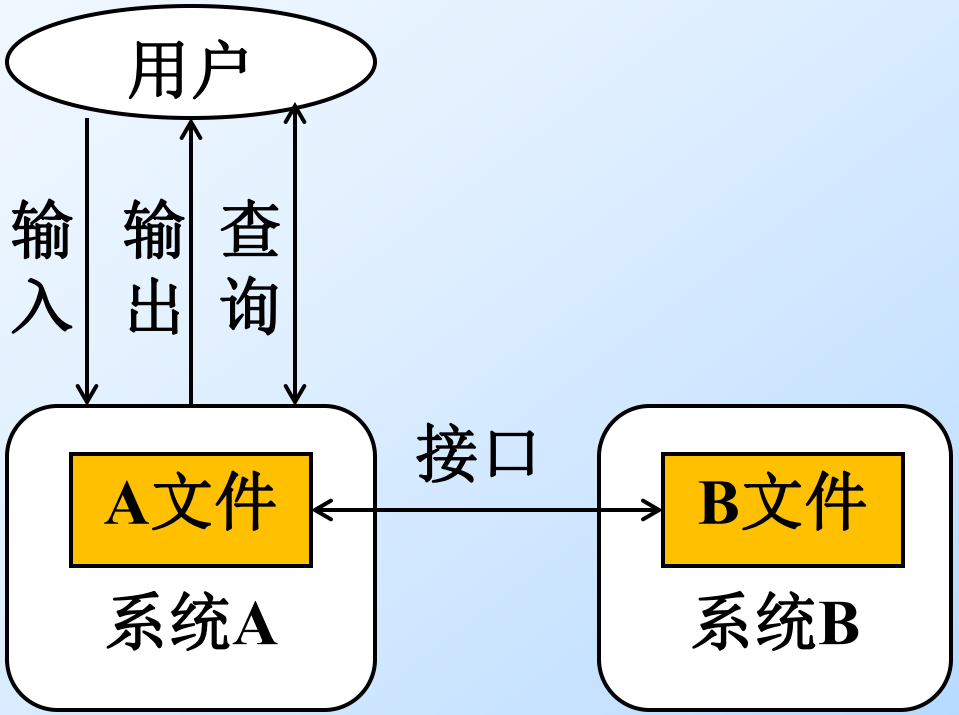
- 特点
  - 简单方便
  - 根据历史项目数据，历史数据可靠时估计精确
  - LOC、KLOC和相关度量容易计算
  - 许多现有的软件估算模型都使用LOC和KLOC作为一项重要输入
  - 有大量的关于LOC的文献和数据
- 缺点
  - 依赖于功能分解，难以在开发初期估算
  - LOC依赖于使用的语言，对短小精悍的程序不利

软件规模估算-功能点技术

- Function Point - FP
- 功能点数从直接度量软件信息域和评估软件复杂性的经验量化关系中获得
- 适合在软件开发初期进行估算
- 估算软件规模的根据
  - 需求规格说明中确认的软件功能
  - 软件功能的复杂度

软件规模估算-功能点技术

- 5类功能（*i*）
  - 外部输入
  - 外部输出
  - 外部查询
  - 外部接口
  - 内部逻辑文件



软件规模估算-功能点技术

- 5类功能的复杂性影响参数（ω_{ij}）

复杂性（*j*） 功能*(i)*	简单	中等	复杂
外部输入	3	4	6
外部输出	4	5	7
内部逻辑	7	10	15
外部接口	5	7	10
外部查询	3	4	6

软件规模估算-功能点技术

- 未调节功能点

$$UFP = \sum_{i=1}^{5}\sum_{j=1}^{3} \omega_{ij} C_{ij}$$

i=1, 2, 3, 4, 5, 代表功能类型

j=1, 2, 3，代表复杂性

ω_{ij}是第i类功能和第j级复杂性的影响参数

C_{ij}是第i类功能和第j级复杂性的功能点的个数

软件规模估算-功能点技术

- 可能影响软件规模的因素（*i*）

1. 备份与恢复
2. 数据通信
3. 分布式处理
4. 性能
5. 系统配置要求
6. 联机数据输入
7. 终端用户效率
8. 联机更新
9. 易操作性
10. 内部处理复杂
11. 可复用性
12. 易安装性
13. 多工作场所
14. 可维护性

软件规模估算-功能点技术

- 影响因素的影响级别（n_i）

  0级：无影响　　0

  1级：微小影响　1

  2级：轻度影响　2

  3级：中度影响　3

  4级：显著影响　4

  5级：重大影响　5

软件规模估算-功能点技术

- 综合影响度：$N = \sum_{i=1}^{14} n_i$

- 复杂度调节因子(Complexity adjustment factor)

$$CAF = 0.65 + 0.01N$$

- 调节后的功能点/交付功能点（Delivered function point）

$$DFP = CAF \times UFP$$

软件规模估算-功能点技术

- 交付功能点和交付代码行数的关系

开发语言	每个功能点对应的平均代码行数DLOC
汇编语言	320
Pascal	91
C	128
C++	53
Java	48
LISP	64

- 交付代码行数

$$L = DFP \times DLOC$$

软件规模估算-功能点技术：案例

XXX银行信息系统

该系统应能增加新客户，并能从客户文件中作删除。系统支持客户的存款和取款业务。在出现透支时，系统应给出警告信息。客户可通过终端查询自己的账户余额，可以要求给出透支客户报告。

软件规模估算-功能点技术：案例

功能分类：

- 外部输入：5
  - 增加新客户
  - 删除客户
  - 存款业务
  - 取款业务
  - 客户要求给出透支报告
- 外部输出：2
  - 透支的警告信息
  - 透支客户报告
- 外部查询：1
  - 客户查询存款余额
- 内部文件：1
  - 客户文件
- 外部接口：0

软件规模估算-功能点技术：案例

- 计算未调节功能点：
  - 假定所有功能的复杂性为简单
  - 外部输入：5×3=15
  - 外部输出：2×4=8
  - 外部查询：1×3=3
  - 内部文件：1×7=7
  - 未调节功能点：UFP = 33

复杂性（*j*） / 功能(*i*)	简单
外部输入	3
外部输出	4
内部逻辑	7
外部接口	5
外部查询	3

软件规模估算-功能点技术：案例

- 假定影响因素取值如下：

1. 备份与恢复 3
2. 数据通信：3
3. 分布式处理：3
4. 性能：2
5. 系统配置要求：4
6. 联机数据输入：5
7. 终端用户效率：4
8. 联机更新：4
9. 易操作性：1
10. 内部处理复杂：0
11. 可复用性：0
12. 易安装性：1
13. 多工作场所：0
14. 可维护性：0

0级：	无影响	0
1级：	微小影响	1
2级：	轻度影响	2
3级：	中度影响	3
4级：	显著影响	4
5级：	重大影响	5

软件规模估算-功能点技术：案例

- 影响因素构成的影响度为

$$N = 3+3+3+2+4+5+4+4+1+0+0+1+0+0 = 30$$

- 复杂度调节因子

$$CAF = 0.65+0.01\times N = 0.95$$

- 可交付功能点

$$DFP = 0.95\times 33 = 31.35$$

- C代码行数 $L = 128\times 31.35 =$ 4千代码行

开发语言	每个功能点对应的平均代码行数DLOC
汇编语言	320
Pascal	91
C	128

软件规模估算-功能点技术

- 优点：
  - 可在软件初期进行估算
  - 与实现语言无关
- 缺点：
  - 功能分类、功能复杂性和影响因素确定方面，主观因素难以排除
  - 适合数据处理类软件，不适用于非数据处理问题，如实时软件、科学计算软件
  - 不能借助工具完成

软件工作量估算

- 算法成本模型
- 经验模型
  - COCOMO模型
    - 基本
    - 中级
    - 高级
  - COCOMOII模型

算法成本模型

- 算法成本模型—基于经验的度量
  - 软件成本的算法成本模型

$$Effort = A + Size^B \times M$$

    - A，常量，由组织的实践和软件的类型决定。
    - B，常量，取值范围为[1,15]。
    - M，常量，反应产品、过程和人力属性。
    - Size可以是软件代码规模的估算，也可以是功能点或目标点。

在软件过程的推进过程中，可利用的信息越多，估算的精确度越高

COCOMO模型

- COnstructive COst Model
- 构造式成本模型，1981年，Boehm提出
- 经验模型：基于从大量软件项目中收集的数据
- 基于项目规模（代码行数）来估算工作量
  - 基本COCOMO模型（Basic model）
  - 中级COCOMO模型（Intermediate model）
  - 高级COCOMO模型（Advanced model）
- 好处
  - 得到广泛证明，可用于公共领域并且很多公共和商业工具都支持它
  - 它应用广泛，并得到了不同组织的评价。
  - 它有较长的历史，于1981年第一次实例化[Boehm, 1981]，为了适用于Ada进行了一次改进[Boehm and Royce, 1989]，最新版本COCOMOII发布于2000年[Boehm et al., 2000]

COCOMO模型

- COCOMO模型中的软件类型
  - 组织型项目（Organic mode）

    较为简单的项目，开发人员对项目有着较好的理解和较丰富的经验，包括各类应用软件
  - 半独立型项目（Semi-detached mode）

    主要指各类实用程序、编译程序等
  - 嵌入型项目（Embedded mode）

    主要指实时处理、控制程序、操作系统等

基本COCOMO模型

- 开发工作量（人月）：
  - 以源代码行数L（千代码行）为自变量

$$E = aL^b$$

软件类型	a	b
组织型	2.4	1.05
半独立型	3.0	1.12
嵌入型	3.6	1.2

举例：
一个具有1万行代码的嵌入式软件，其开发工作量大约是多少？

$$E = aL^b$$
$$= 3.6\times10^{1.2}$$
$$= 57.06\text{（人月）}$$

基本COCOMO模型

- 基本COCOMO模型的工作量和进度公式

总体类型	工 作 量	进度（开发时长）
组织型	$MM= 2.4（KDSI）^{1.05}$	$TDEV=2.5（MM）^{0.38}$
半独立型	$MM= 3.0（KDSI）^{1.12}$	$TDEV=2.5（MM）^{0.35}$
嵌入型	$MM=3.6（KDSI）^{1.20}$	$TDEV= 2.5（MM）^{0.32}$

基本COCOMO模型：估算示例

- Du Bridge化学品公司是一家大型化工产品公司，公司计划开发一个新的计算机程序，用以跟踪原材料的使用情况。这个程序由一个公司内部的程序员和系统分析员组成的团队负责开发，他们已有多年类似程序的开发经验。

- 这是一个组织型软件项目开发的很好实例

基本COCOMO模型：估算示例

- 最初的研究确定程序的规模大约是32000条交付的源指令（32KDSI）
- 数量FSP（Full－time－equivalent Software Personnel）代表的是相当于全职的软件人员，用以度量在某一给定时间为某项目工作的等价人员数
- 由基本公式，可估算出项目的如下特征：
  - 工作量　　$MM = 2.4(32)^{1.05} = 91$ 人月
  - 生产率　　32000DSI/91MM = 352 DSI/MM
  - 进度　　$TDEV = 2.5(91)^{0.38} = 14$ 月
  - 平均配置人员　　91 人月/14 个月 = 6.5 FSP

中级COCOMO模型

- 以基本COCOMO模型为基础
- 对影响工作量的因素（4类15种因素）进行估计，确定调节因子F，修正工作量估计E
- 开发工作量（人月）

$$E = aL^b F$$

$$F = \prod_{i=1}^{15} F_i$$

软件类型	a	b
组织型	3.2	1.05
半独立型	3.0	1.12
嵌入型	2.8	1.2

该表与基本COCOMO模型不同！

中级COCOMO模型

- 中级COCOMO模型的工作量和进度公式

总体类型	工 作 量	进　度
组织型	MM＝ 3.2（KDSI）$^{1.05}$F	TDEV＝ 2.5（MM）$^{0.38}$
半独立 型	MM＝ 3.0（KDSI）$^{1.12}$F	TDEV＝ 2.5（MM）$^{0.35}$
嵌入型	MM＝ 2.8（KDSI）$^{1.20}$F	TDEV＝ 2.5（MM）$^{0.32}$

中级COCOMO模型中影响工作量的因素

工作量因素F_i		非常低	低	正常	高	非常高	超高
产品因素	软件可靠性	0.75	0.88	1.00	1.15	1.40	
	数据库规模		0.94	1.00	1.08	1.16	
	产品复杂性	0.70	0.85	1.00	1.15	1.30	1.65
计算机因素	执行时间限制			1.00	1.11	1.30	1.66
	存储限制			1.00	1.06	1.21	1.56
	平台变动		0.87	1.00	1.15	1.30	
	环境变更		0.87	1.00	1.07	1.15	
人的因素	分析员能力		1.46	1.00	0.86		
	程序员能力	1.42	1.17	1.00	0.86	0.82	
	应用领域经验	1.29	1.13	1.00	0.91	0.71	
	平台经验	1.21	1.10	1.00	0.90	0.70	
	语言和工具经验	1.41	1.07	1.00	0.95		
项目因素	现代程序技术	1.24	1.10	1.00	0.91	0.82	
	软件工具的使用	1.24	1.10	1.00	0.91	0.83	
	开发进度限制	1.23	1.08	1.00	1.04	1.10	

中级COCOMO模型：案例

- 某一嵌入式通信软件，源代码一万行，其开发工作量大概是多少？
- 未调节的开发工作量

  $E = aL^b = 2.8 \times 10^{1.2} = 44.38$（人月）
- 调节因子（见上页表格）

  F =1.15×0.94×1.30×1.11×1.06×1.00×1.00
  ×0.86×0.86×1.13×1.00×1.00×1.10×1.00×1.00
  =1.52
- 调节后的工作量

  $E = 44.38 \times 1.52 = 67.46$（人月）

高级COCOMO模型

扩展内容

- 以中级COCOMO模型为基础
- 涉及软件工程过程中不同开发阶段的影响
- 考虑系统层、子系统层和模块层的差别

- COCOMO模型问题：开发初期难以确定代码行数

COCOMO II 模型

扩展内容

- 1997年，Boehm对COCOMO的改进
- 解决问题：开发初期难以确定代码行数
- 根据软件开发阶段估算工作量（3阶段）
  - 第1阶段：原型阶段
    - 用应用点（Application point）来估算软件规模
    - 如屏幕数、报告数、构件数等
  - 第2阶段：体系结构设计阶段
    - 用功能点来估算软件规模
  - 第3阶段：详细设计阶段
    - 功能点或者代码行数估算软件规模

- 软件项目管理概述
- 人员组织与管理
- 软件规模和工作量估算
- 项目进度计划
- 软件风险管理
- 软件配置管理

项目进度管理

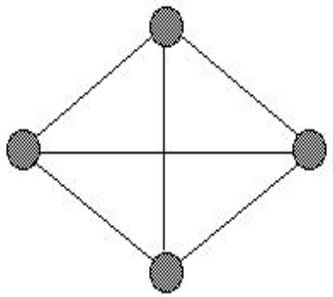
- 项目进度（Project schedule）
  - 项目中的各个阶段
  - 每个阶段中的活动
  - 每项活动的开始时间、结束时间、持续时间等
  - 里程碑
  - 可交付的产品
- 项目进度管理
  - 制定项目进度计划
  - 跟踪项目进展

项目进度计划

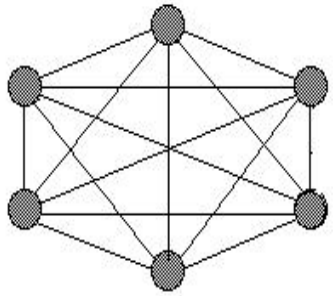
- 软件开发项目的进度计划有两种方式：
  ① 系统最终交付日期已经确定，软件开发部门必须在规定期限内完成；
  ② 系统最终交付日期只确定了大致的年限，最後交付日期由软件开发部门确定。
- 进度计划落空，会导致市场机会的丧失，使用户不满意，而且也会导致成本的增加。
- 在考虑进度计划时，要把工作量与花费时间联系起来，合理分配工作量，利用进度计划的有效分析方法严密监控软件开发的进展情况，使软件开发进度不致拖延。

开发小组人数与软件生产率的关系

- 多人共同承担软件开发项目时，人与人之间必须通过交流来解决各自承担任务之间的接口问题，即所谓通信问题
- 通信需花费时间和代价，会引起软件错误增加，降低软件生产率
- 如果一个软件开发小组有n个人，每两人之间都需要通信，则总的通信路径有 n(n-1)/2 条



（a）四人之间所有通信路径

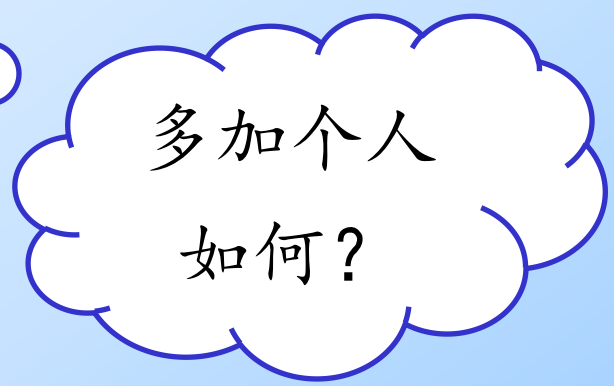


（b）六人之间所有通信路径

开发小组人数与软件生产率的关系

- 设一个人单独开发软件，生产率是5000行／人年
- 若4个人组成一个小组共同开发这个软件，则需要6条通信路径
- 若在每条通信路径上耗费的工作量是250行／人年
- 则小组中每个人的软件生产率降低为：

$$5000-6\times250/4$$
$$=5000-375$$
$$=4625\text{行／人年}。$$

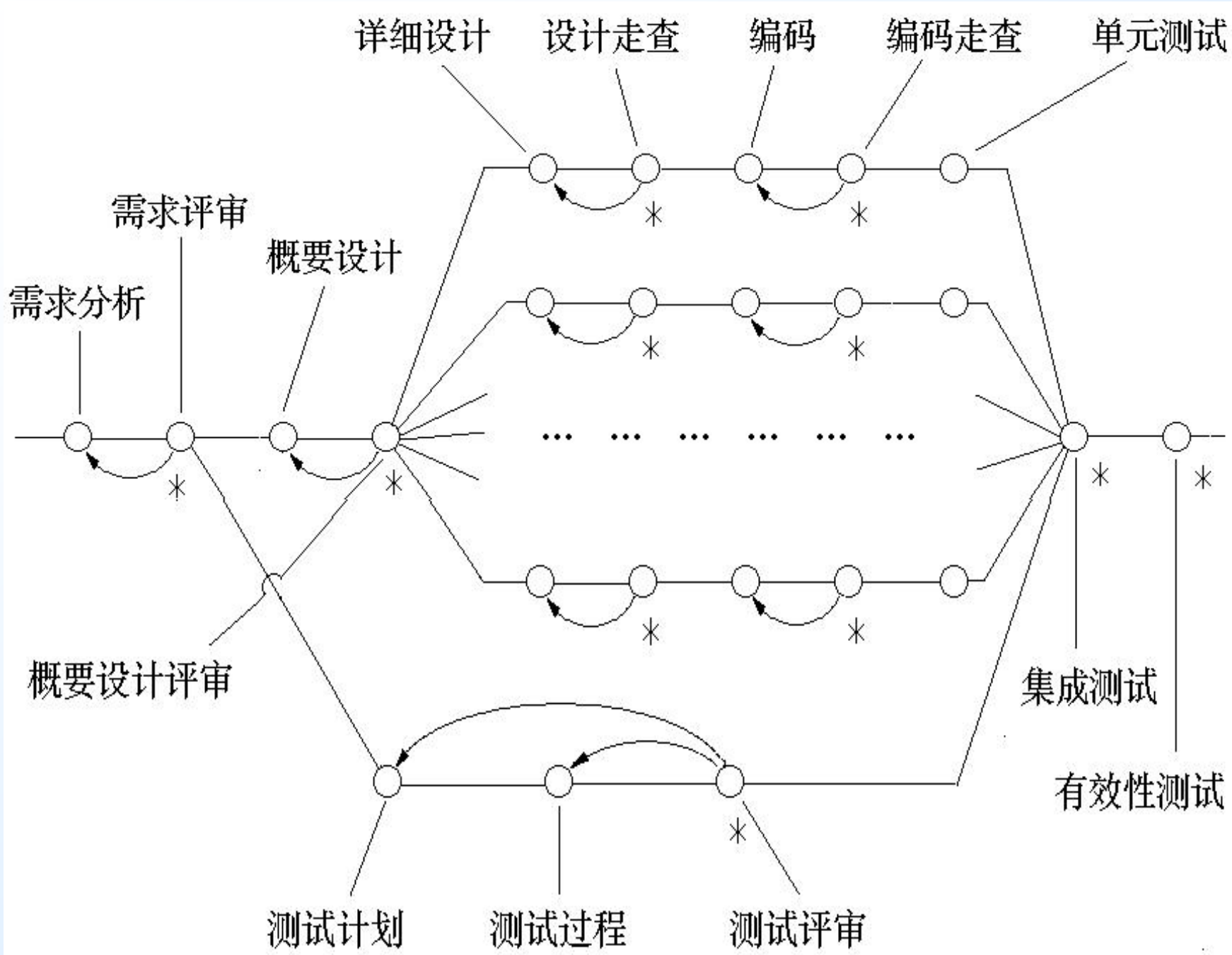


开发小组人数与软件生产率的关系

- 从上述分析可知：
  - 一个软件任务由一个人单独开发，生产率最高
  - 而对于一个稍大型的软件项目，一个人单独开发，时间太长，因此软件开发小组是必要的
  - 但是，开发小组不宜太大，成员之间避免太多的通信路径
  - 在开发进程中，切忌中途加人，避免太多的生产率损失

任务的确定与并行性

- 当参加同一软件工程项目的人数不止一人的时候，开发工作就会出现并行情形
- 软件工程项目的并行性提出了一系列的进度要求
- 软件开发进程中设置许多里程碑，为管理人员提供指示项目进度的可靠依据
- 并行任务是同时发生的，所以进度计划表必须决定任务之间的从属关系，确定各个任务的先后次序和衔接，确定各个任务完成的持续时间
- 项目负责人应注意构成关键路径的任务，即若要保证整个项目能按进度要求完成，就必须保证这些任务要按进度要求完成


详细设计
设计走查
编码
编码走查
单元测试
需求评审
概要设计
需求分析
概要设计评审
集成测试
有效性测试
测试计划
测试过程
测试评审

项目进度管理

- 项目分解
  - 阶段、活动
  - 里程碑、可交付产品
- 确定各活动之间的关系
- 为每项活动分配时间
- 识别出关键路径
- 制定进度计划
- 确认投入的人力
- 确定人员的责任

制定项目进度计划

- 跟踪项目进展

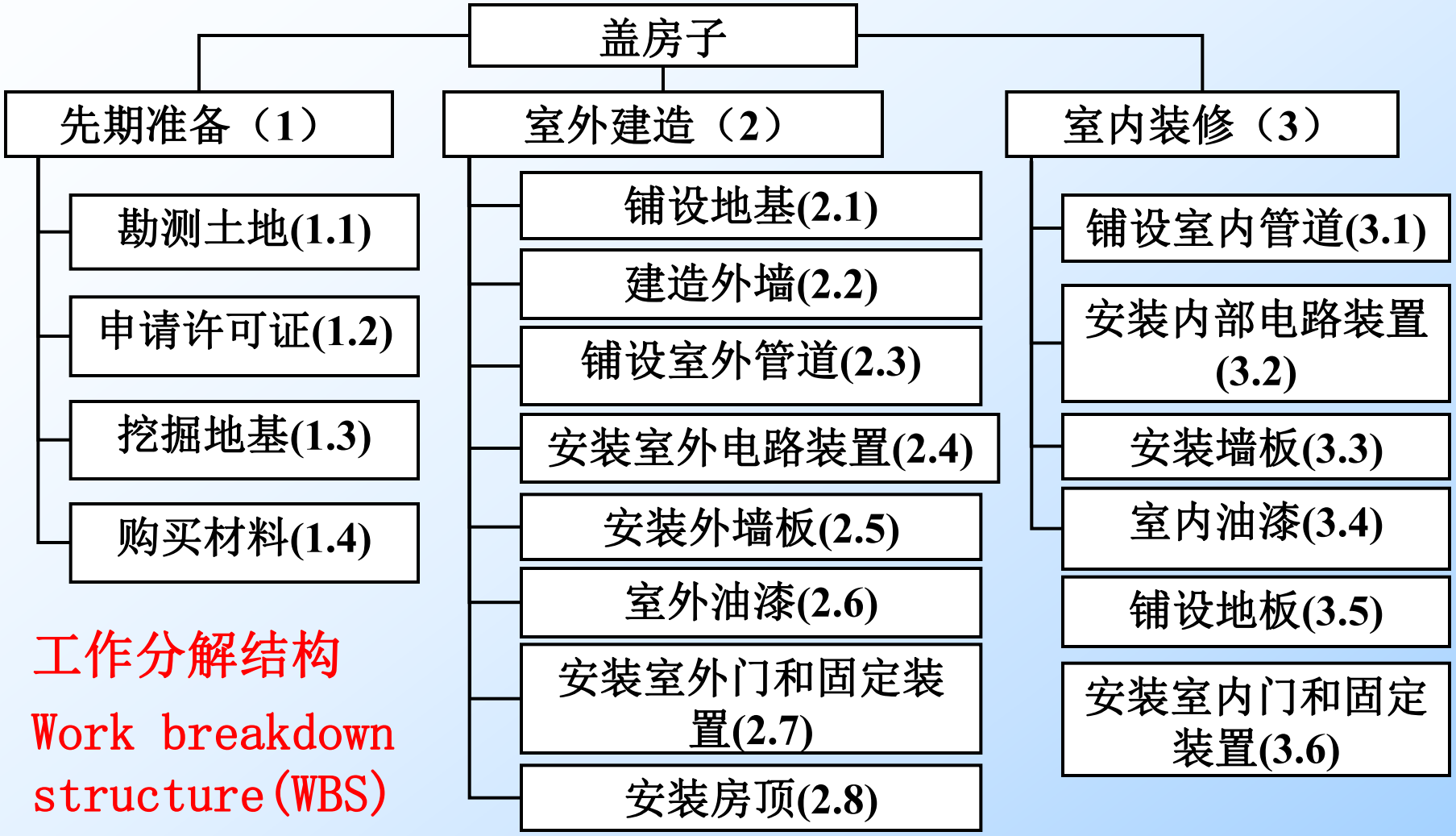
进度计划的方法

- 制定软件进度计划最常用的方法：
  - PERT技术/计划评审技术
    - Project Evaluation and Review Technique
    - PERT是利用网络分析制定计划以及对计划予以评价的技术
  - CPM方法/关键路径法
    - Critical Path Method
    - CPM方法是一种基于数学计算的项目计划管理方法

进度计划的方法

- 都采用网络图来描述一个项目的任务网络，从一个项目的开始到结束，把应当完成的任务用图或表的形式表示出来
- 通常用两张表来定义网络图：
  - 任务分解结构（Work Breakdown Structure）：与一特定软件项目有关的所有任务
  - 限制表（RestrictionList）：应当按照什么样的次序来完成这些任务
- 有效监控软件项目的进度计划和工作的实际进展情况，表现各项任务之间进度的相互依赖关系

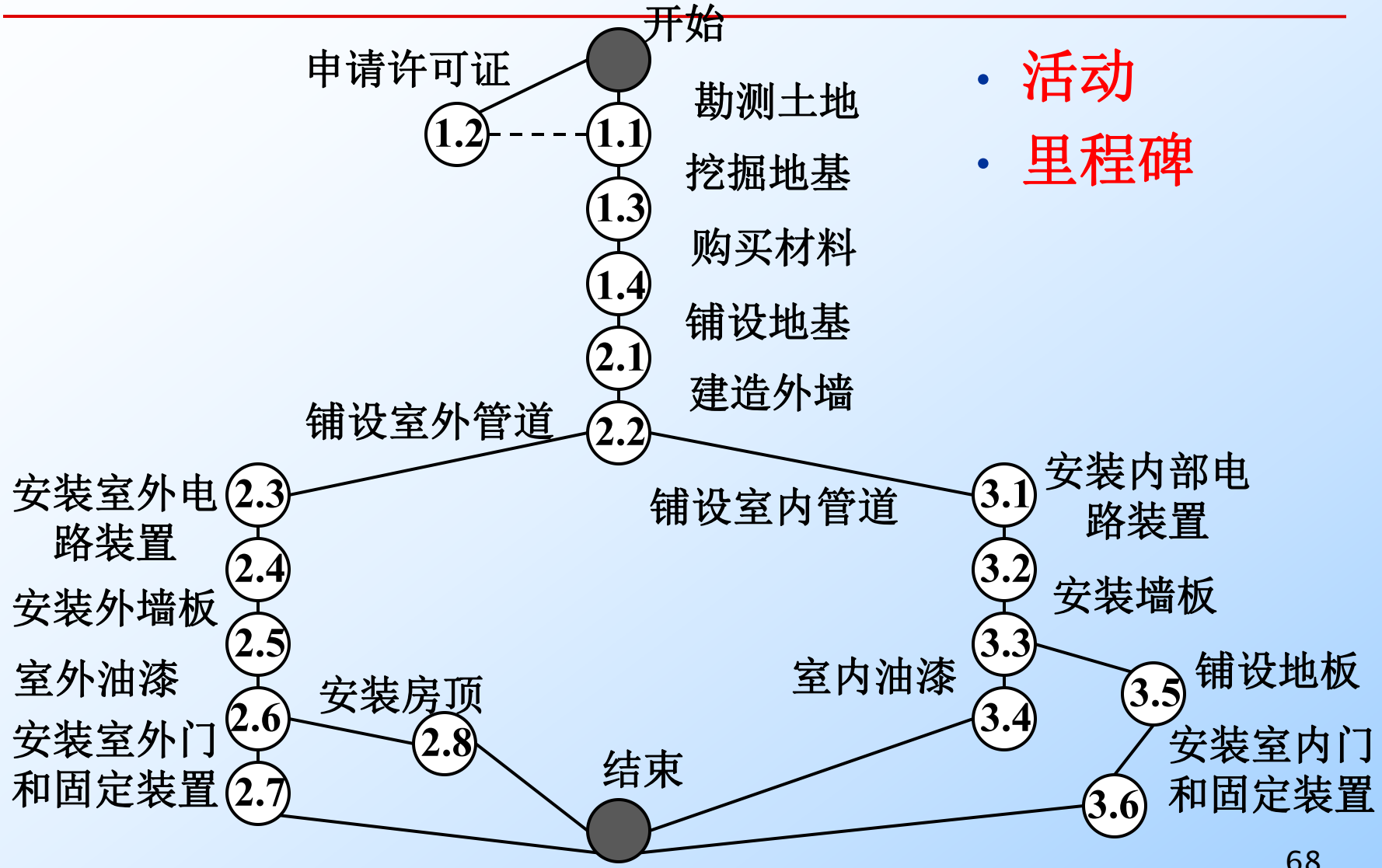
制定项目进度计划-工作分解



工作分解结构

Work breakdown structure(WBS)

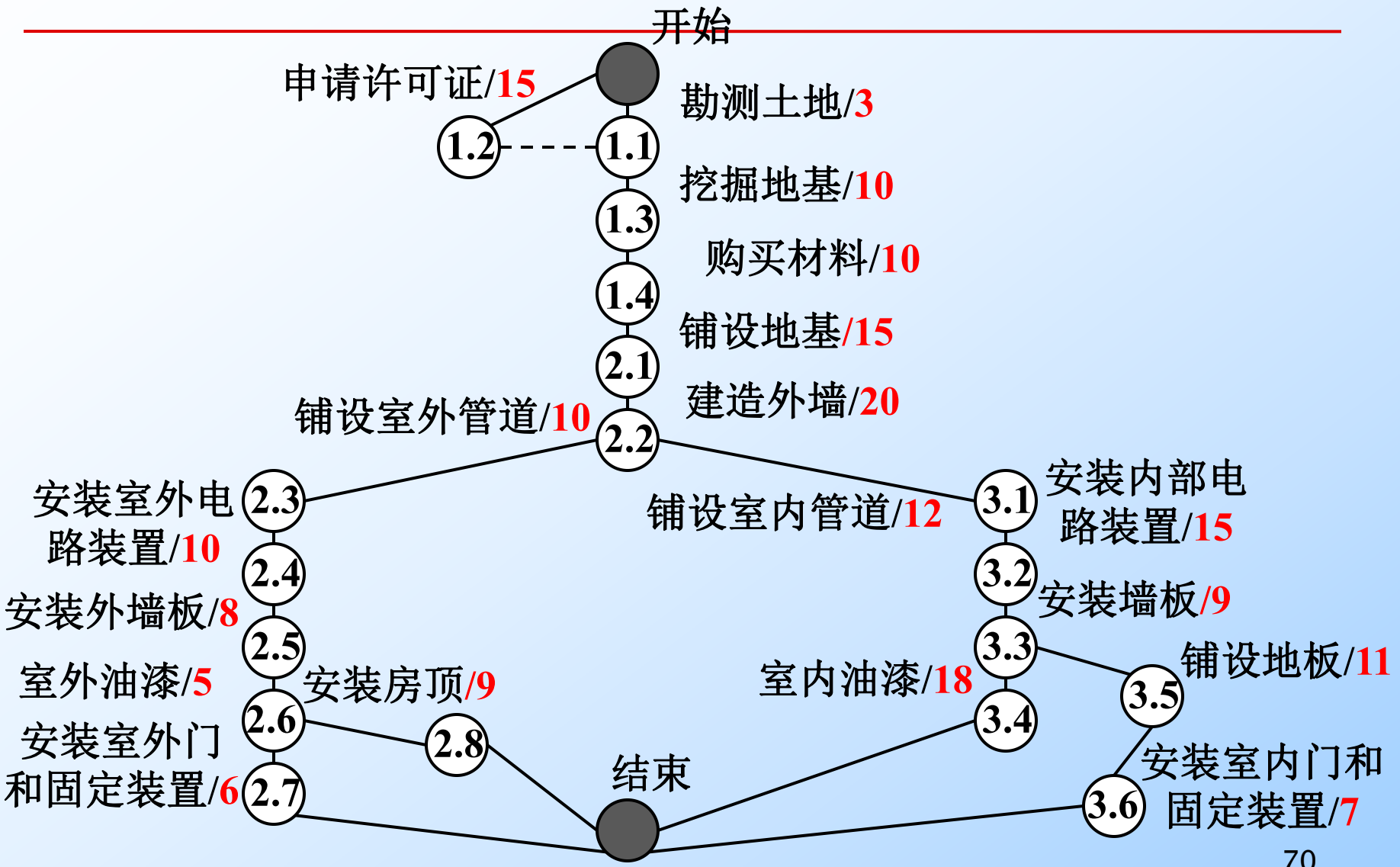
制定项目进度计划-活动图



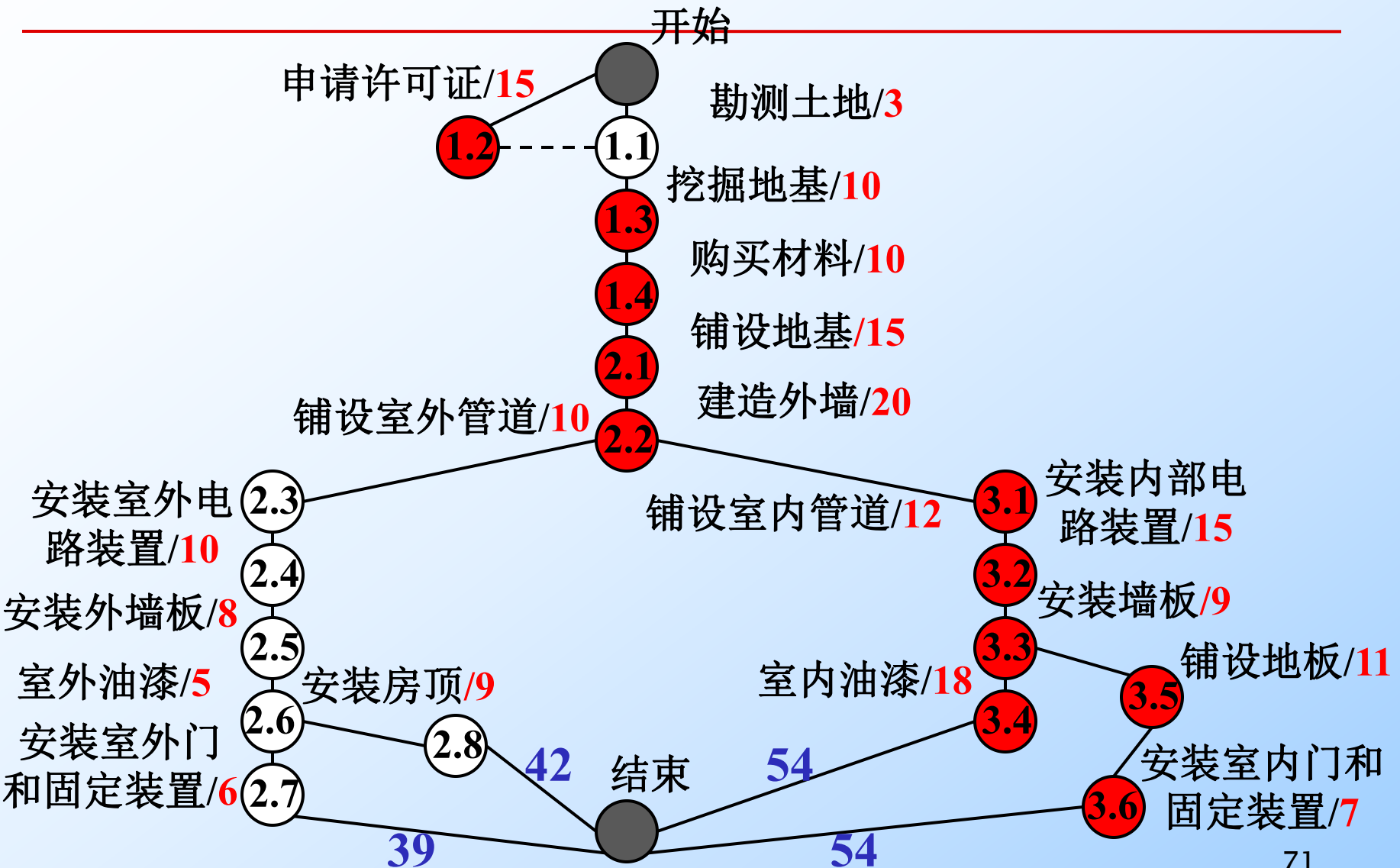
制定项目进度计划-活动图

- 应用统计模型，对每一个单独的任务确定最可能的开发持续时间的估算值
- 确定关键路径，在关键路径上的各个任务都是时间余量为零的关键任务，不能有任何时间延误
- 计算边界时间，以便为具体的任务定义时间窗口

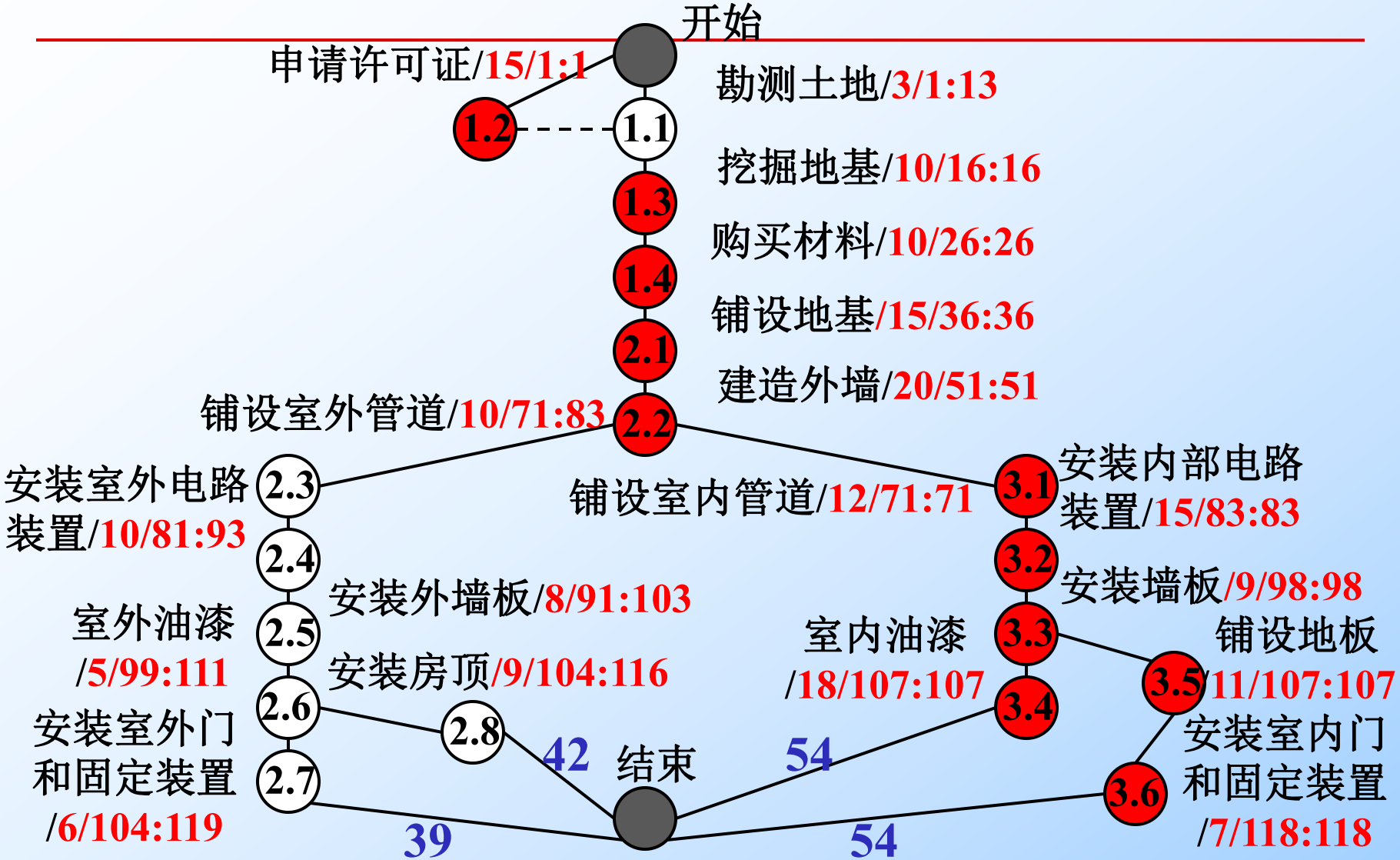
制定项目进度计划-估算活动时间



制定项目进度计划-关键路径

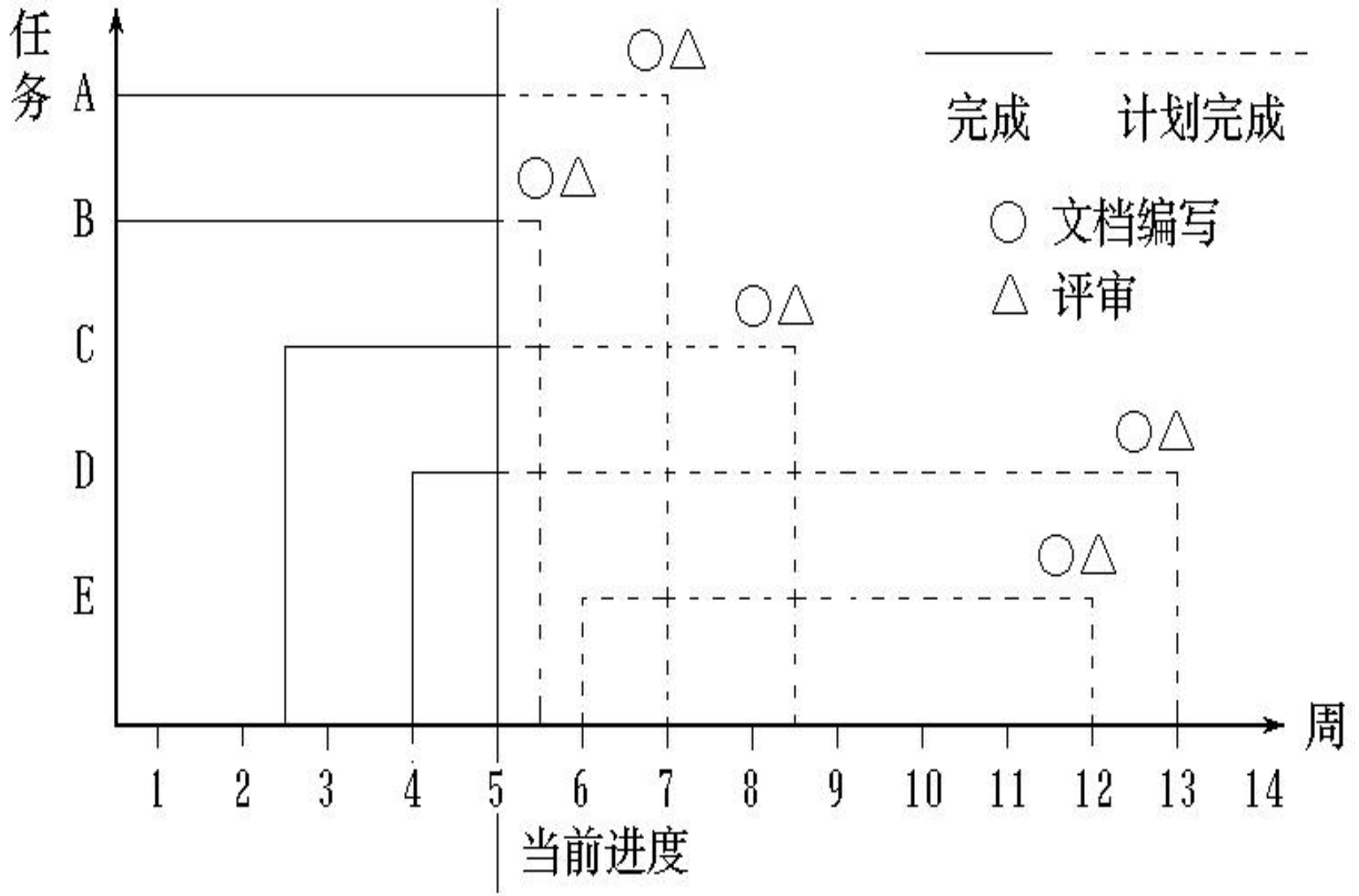


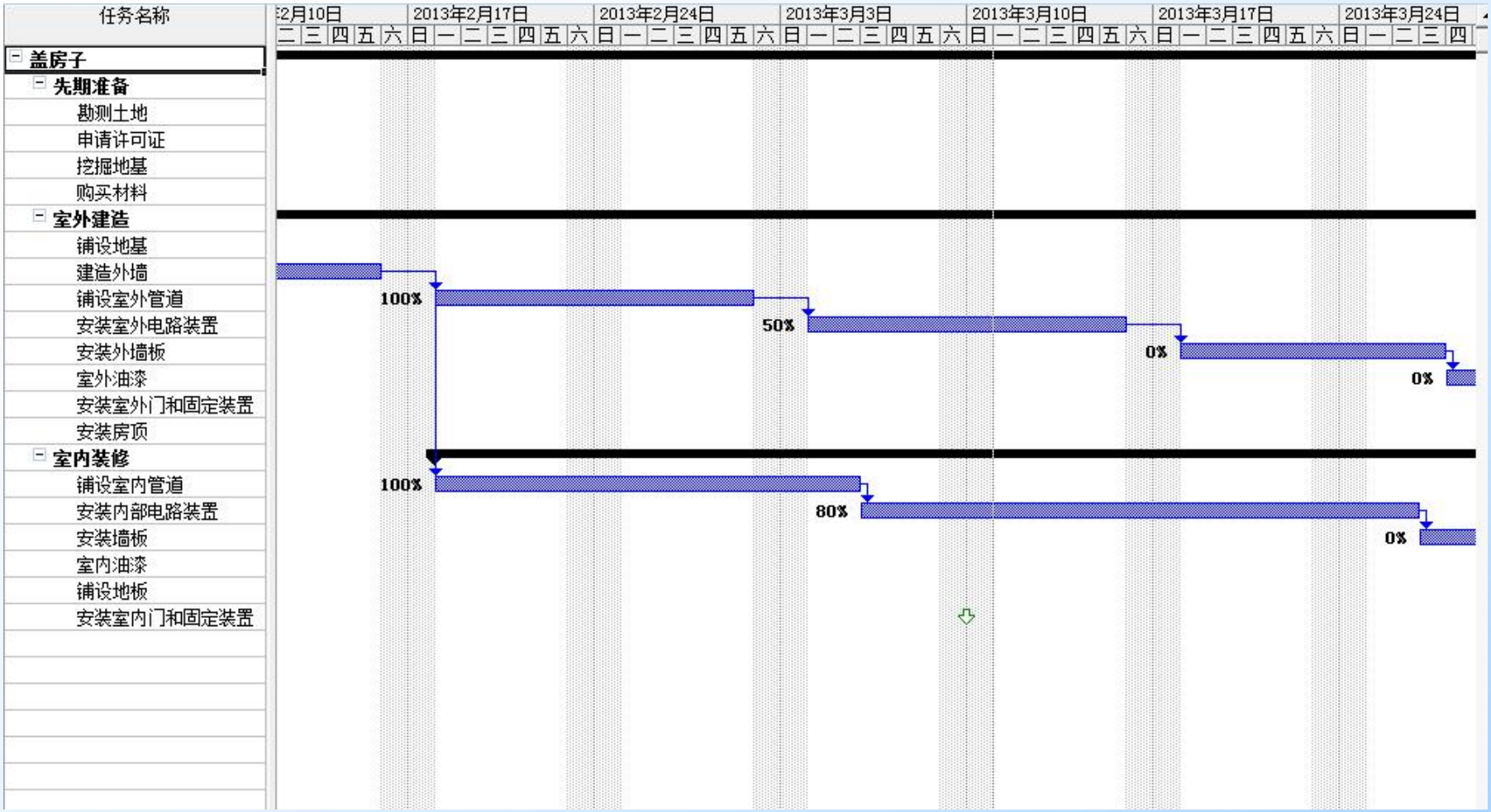
制定项目进度计划-活动开始时间



跟踪项目进度计划

- 图示方法，明确标明：
  - 各个任务的计划开始时间，完成时间；
  - 各个任务完成标志（即○文档编制和△评审）；
  - 文档编制与评审是软件开发进度的里程碑
  - 每一任务完成的标准，不是以能否继续下一阶段任务为标准，而是以必须交付应交付的文档与通过评审为标准
  - 各个任务与参与工作的人数，各个任务与工作量之间的衔接情况；
  - 完成各个任务所需的物理资源和数据资源
- 甘特图（Gantt Chart）


任务
A
B
C
D
E
完成
计划完成
○ 文档编写
△ 评审
周
1
2
3
4
5
6
7
8
9
10
11
12
13
14
当前进度


任务名称
盖房子
先期准备
勘测土地
申请许可证
挖掘地基
购买材料
室外建造
铺设地基
建造外墙
铺设室外管道
安装室外电路装置
安装外墙板
室外油漆
安装室外门和固定装置
安装房顶
室内装修
铺设室内管道
安装内部电路装置
安装墙板
室内油漆
铺设地板
安装室内门和固定装置
2月10日
2013年2月17日
2013年2月24日
2013年3月3日
2013年3月10日
2013年3月17日
2013年3月24日
100%
50%
0%
0%
100%
80%
0%

项目的追踪和控制

- 软件项目管理一项重要工作就是在项目实施过程中进行追踪和控制：
  - 定期举行项目状态会议，每位项目成员报告其进展和遇到的问题
  - 评价在软件工程过程中所产生的所有评审的结果
  - 确定由项目的计划进度所安排的可能选择的正式的里程碑
  - 比较在项目资源表中所列出的每一个项目任务的实际开始时间和计划开始时间
  - 非正式地与开发人员交谈，以得到他们对开发进展和刚冒头的问题的客观评价
  - 当问题出现的时候，项目管理人员必须实行控制以尽快地排解问题

- 软件项目管理概述
- 人员组织与管理
- 软件规模和工作量估算
- 项目进度计划
- 软件风险管理
- 软件配置管理

风险管理

- 什么是风险（risk）？
  - 一种具有负面后果的、人们不希望发生的事件
  - 该事件发生后会产生损失（风险影响）
  - 该事件有可能发生（风险概率）
  - 能够改变结果的程度（风险控制）
- 风险事件例子
  - 人员流失
  - 不现实的进度和预算
  - 持续的需求变化
  - 外部提供的构件达不到要求

风险管理


风险识别
• 确定项目有哪些风险
• 确定风险来源
风险分析
• 确定风险影响
• 确定风险概率
• 对风险排序
风险规划
• 制定风险应对措施
风险监控
• 监控风险发生情况
• 监控风险应对措施的落实情况

风险识别方法

- 检查表法
- 德尔菲方法
- 头脑风暴法
- 情景分析法

检查表法

- 软件项目常常存在一些共同的风险
  - 如：人员缺乏、不现实的人员和成本估计、后期需求变化、外购构件缺陷等
- 风险检查表中列出项目中常见的风险
- 项目相关人员通过核对风险检查表，判断哪些风险会出现在项目中
- 可根据项目经验对风险检查表进行修订和补充
- 该方法可以使管理者集中识别常见类型的风险
- 优点：快速而简单，可以用来对照项目的实际情况，逐项排查，从而帮助识别风险
- 缺点：由于每个项目都有其特殊性，很难做到全面周到

风险识别

- 风险检查表
  - 软件规模风险
  - 商业影响风险
  - 客户相关风险
  - 软件过程风险
  - 开发技术风险
  - 开发环境风险
  - 开发人员风险
- 其中每一项都包含很多风险条目

风险识别：开发人员风险

- 是否有最优秀的人员可用；
- 人员在技术上是否配套；
- 是否有足够的人员可用；
- 开发人员是否能够自始至终地参加整个项目的工作；
- 项目中是否有一些人员只能部分时间工作；
- 开发人员对自己的工作是否有正确的期望；
- 开发人员是否接受过必要的培训；
- 开发人员的流动是否仍能保证工作的连续性；
- …

德尔菲（Delphi）方法

- 德尔菲方法又称专家调查法，本质上是一种匿名反馈的函询法
- 起源于20世纪40年代末，最初由美国兰德公司应用于技术预测
- 基本过程：
  - 把需要做风险识别的软件项目的情况分别匿名征求若干专家的意见
  - 然后把这些意见进行综合、归纳和统计，再反馈给各位专家，再次征求意见
  - 这样反复经过四至五轮，逐步使专家意见趋向一致，作为最后预测和识别风险的依据

头脑风暴法

- 头脑风暴（Brain Storm）法简单来说就是团队的全体成员自由地提出自己的主张和想法，它是解决问题时常用的一种方法。
- 基本过程：
  - 将项目主要参与人员代表召集到一起
  - 然后他们利用自己对项目不同部分的认识，识别项目可能出现的问题
  - 一个有益的做法是询问不同人员所担心的内容
- 优点是可对项目风险进行全面的识别。

情景分析法

- 情景分析法是根据项目发展趋势的多样性，通过对系统内外相关问题的系统分析，设计出多种可能的未来前景，然后用类似于撰写电影剧本的手法，对系统发展态势做出自始至终的情景和画面的描述
- 情景分析法是一种适用于对可变因素较多的项目进行风险预测和识别的技术
- 它在假定关键影响因素有可能发生的基础上，构造多重情景，提出多种未来的可能结果，以便采取适当措施防患于未然

风险分析

- 对识别的风险进行评估：确定风险发生的概率和影响
- 风险概率级别：
  - 低概率（0-0.3）
  - 中概率（0.3-0.7）
  - 高概率（0.7-1.0）
- 风险影响级别：
  - 可忽略的（0-3）
  - 轻微的（3-7）
  - 严重的（7-9）
  - 灾难性的（9-10）

风险分析

- 灾难性的风险
  - 软件无法满足需求、软件无法达到性能要求
  - 资金严重短缺、超出预算、无法在交付日期内完成
- 严重的风险
  - 性能下降，使得任务成功受到质疑
  - 资金不足、可能超出预算、可能延期
- 轻微的风险
  - 性能有些下降、成本上有些问题、进度上有些问题
- 可忽略的风险
  - 使用不方便、成本可能低于预算、交付日期可能提前

风险分析

- 风险评估表和风险排序

风险	类型	概率	影响
项目资金不到位	客户相关	中等	灾难性的
规模估计可能非常低	软件规模	高	严重的
人员流动频繁	开发人员	高	严重的
技术达不到预期的效果	开发技术	低	灾难性的
交付期限将被紧缩	商业影响	中等	严重的
人员缺乏经验	开发人员	低	严重的
用户数量大大超出计划	软件规模	低	轻微的

- 风险显露（risk exposure）：$Re(R)=Prob(R)\times Loss(R)$

风险规划和风险监控

- 风险规划：制定风险应对策略
  - 风险规避：对可能发生的风险尽可能地规避，采取主动放弃或者拒绝使用导致风险的方案。如：改变需求、放弃采用新技术
  - 风险缓解：在风险发生之前采取一些措施降低风险发生的可能性或减少风险可能造成的损失。如：为了防止人员流失，提高人员待遇，改善工作环境；
  - 风险转移：将风险转移给第三方，如采购、分包、免责合同、保险
  - 风险接受：采取应急方案应对风险的发生
- 风险监控
  - 监控指示风险变化的影响因素
  - 监视风险应对措施的执行

风险及化解措施举例

- 过多的需求变更
  - 让客户在最初的需求规格说明上签字
  - 让客户理解需求的变更将会影响项目的进度
  - 制定需求变更规程
  - 按实际开发工作量计算开发费
- 人员流失
  - 项目的关键部分有多人参与
  - 开展团队建设工作专题研讨
  - 保持项目人员备份
  - 保存员工个人工作的专门文件

风险案例分析

- 案例描述
  - 江西某行业业务运营支撑网络管理工程是全国性重点工程，受到该公司领导层的高度重视，委派业务支撑部部门经理为项目总监，张工为项目经理。
  - 在编制早期计划书时，市场部李工不断提出新的需求，而张工“来者不拒”，不停地更改项目计划。
  - 在工程的机房设备平面设计中，张工组织人员自行设计，将大部分机架式的小型机集中摆放在一片较小区域内。
  - 本期工程正式完全割接上线前，旧系统仍然需保持运行。保证系统稳定运行是项目团队的第一要务，在系统割接期间，确保7天×24小时的业务连续平稳运行。

风险案例分析

问题：

该工程中有哪些风险？

应采取怎样的应对策略？

案例分析

- 分析
  - 频繁的需求变更必然会影响信息工程项目的三大目标（进度、成本、质量）。因此引导客户需求对项目经理来说就非常关键，引导得好，项目的开发就会比较顺利，相反，就会给项目带来很多负面影响。
  - 在该项目中，项目经理张工对市场部李工不断提出的新需求采取了“来者不拒”的态度，这是不恰当的，这会使项目计划不断变动，导致项目范围无法确定，工期和成本不可控制，团队成员工作目标也不明确，因此出现了非常严重的需求风险。
  - 为了应对这一风险，张工应该与李工积极地沟通和谈判，使他明白工程的重要意义，并承诺工程不是交钥匙项目，可为系统升级和扩容留有扩展接口，将来新的需求能够通过后续工程逐步实现，从而使需求趋于稳定。

案例分析

- 在工程的机房设备平面设计中，将大部分机架式的小型机集中摆放在一片较小区域内，从表面上看，提高了机房平面空间的使用率，但是由于未充分考虑到设备散热因素，容易造成该区域机器过热而宕机。因此团队的机房设计技术经验不足给项目带来了系统运行不稳定的风险。
- 可采取风险转移策略来应对这一风险。张工可聘请具有通信设计资质的专家来负责机房设备平面设计，从机房空调、电源、布线、承重、消防等各个方面进行详细的勘察和设计，从而保证设备运行的可靠性，实现工程设计风险的良性转移。
- 在系统割接期间，新旧系统要顺利交接，这给系统业务的7天×24小时连续平稳运行带来了风险，因此项目组必须制定详尽可行的系统割接方案、新旧系统并运行方案和故障应急处理方案。

- 软件项目管理概述
- 人员组织与管理
- 软件规模和工作量估算
- 项目进度计划
- 软件风险管理
- 软件配置管理

软件配置管理

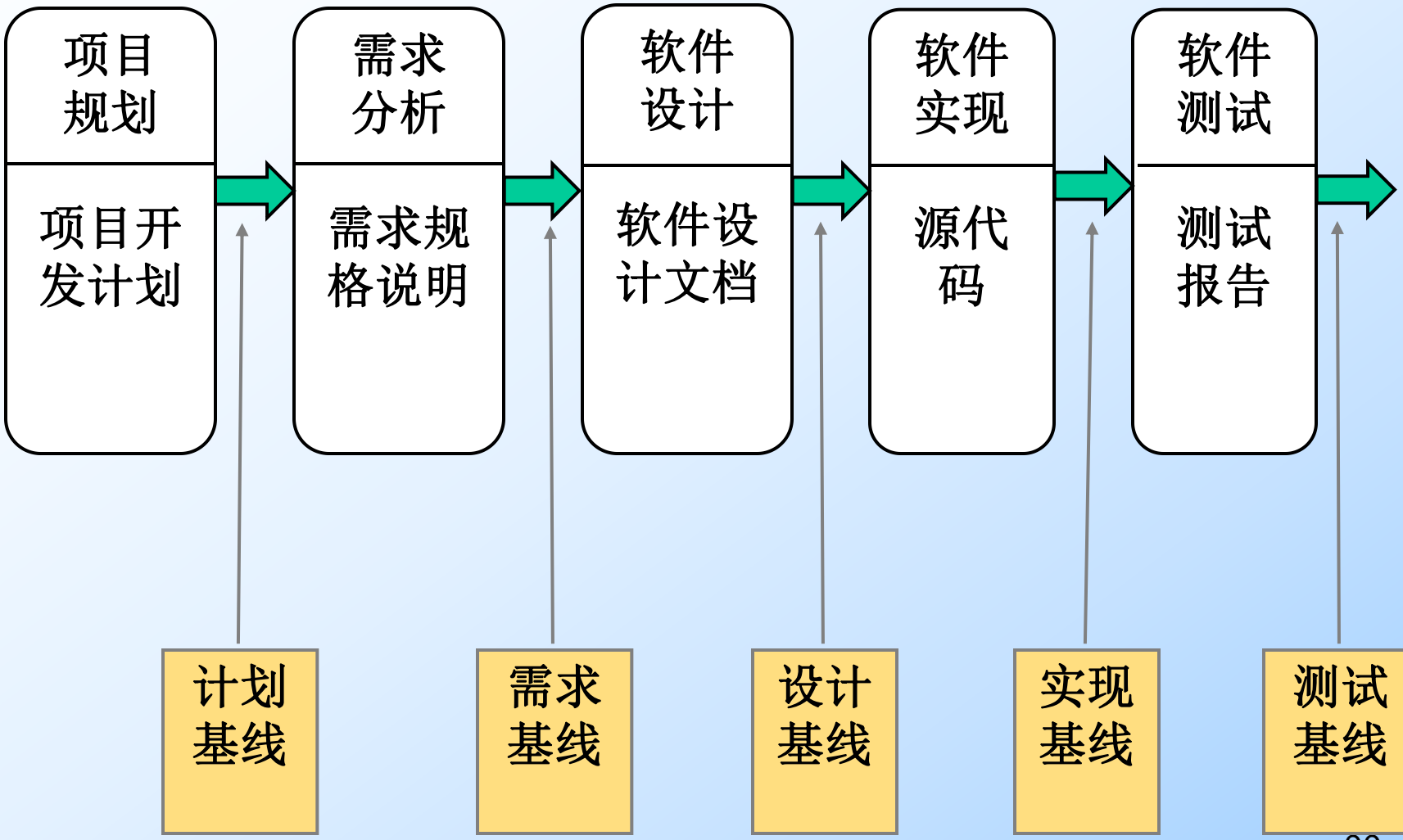
- Software Configuration Management - SCM
- 简单地说，软件配置管理就是对软件变更的管理
- 软件开发中变更是不可避免的
- 变更容易造成混乱和产生错误
- 软件配置管理也称为软件变更管理
- 目标：使错误量减少到最小、使生产率最高

如果你不控制变更，变更就会控制你

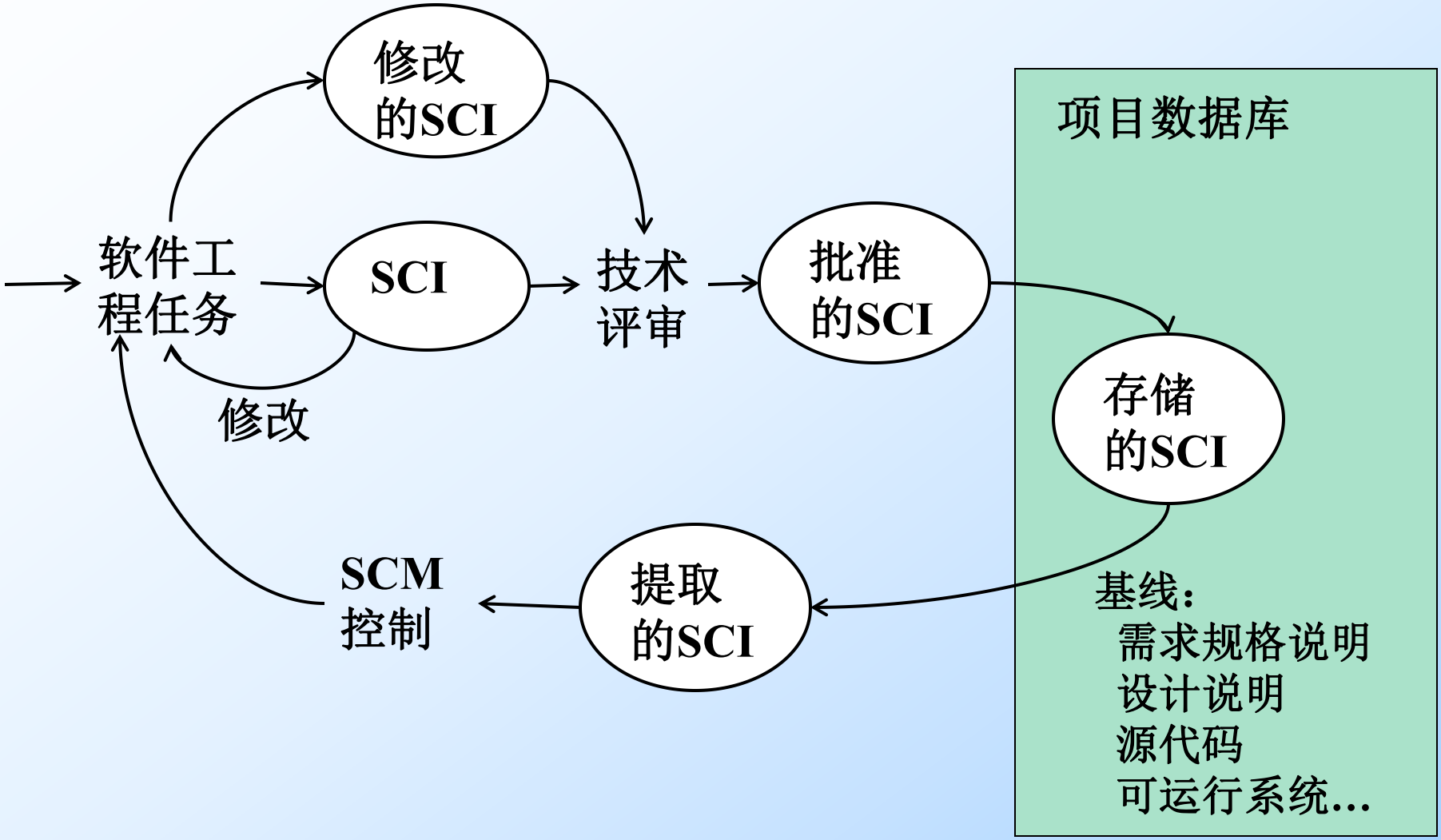
软件配置管理

- 软件配置项
  - Software Configuration Item - SCI
  - 与合同、过程、计划和产品有关的文档和数据
  - 源代码、目标代码和可执行代码
  - 软件工具、可复用软件、外购软件及用户提供的软件
- 基线（Baseline）
  - 通过正式评审和批准的规格说明或者产品
  - 软件配置项在成为基线之前可以迅速非正式修改
  - 软件配置项成为基线之后，只有通过正式的变更控制过程才能修改
  - 标志软件开发的里程碑

软件配置管理：软件开发中的基线



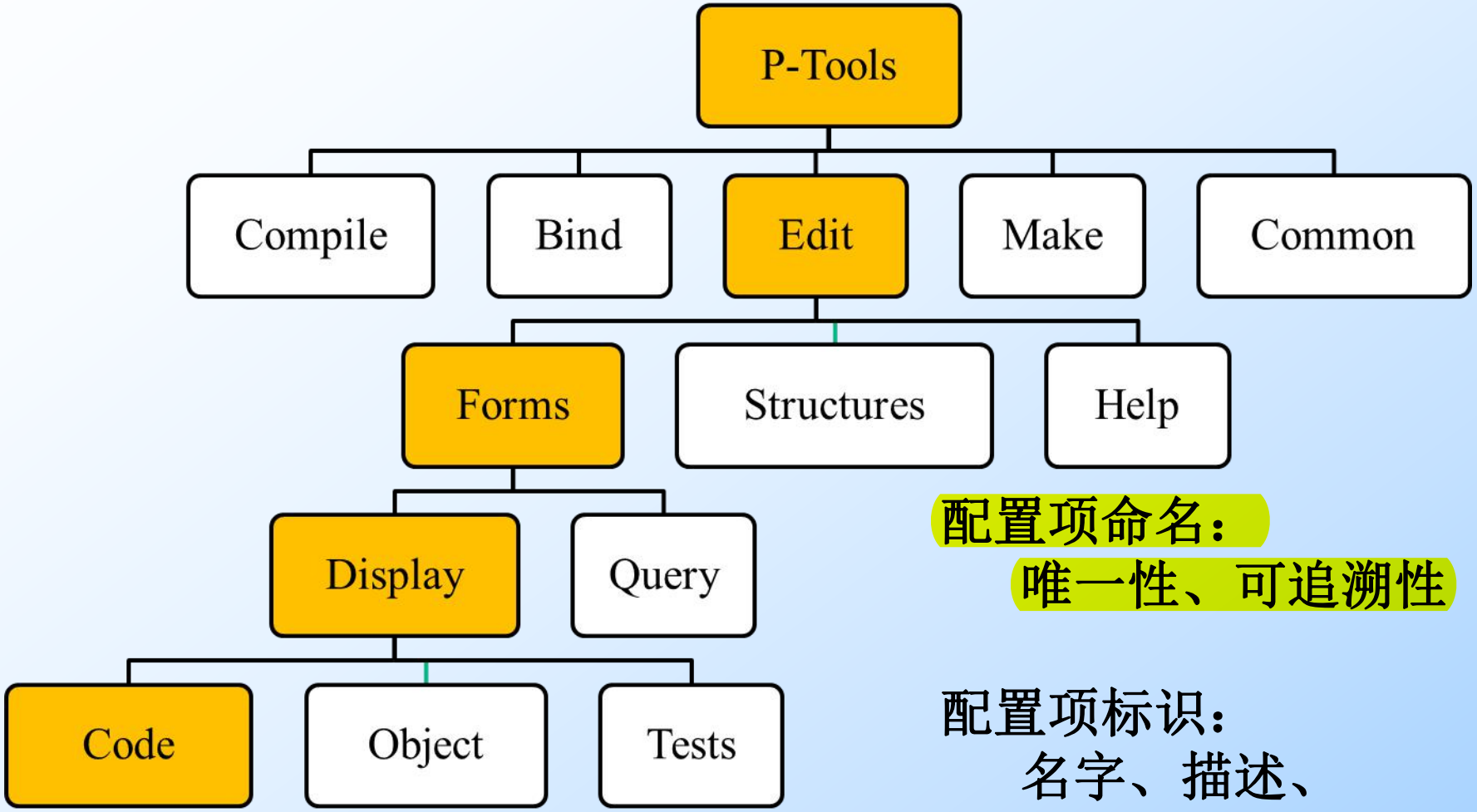
软件配置管理：基线化的SCI



软件配置管理的任务

- 软件配置项标识
- 变更控制
- 版本控制
- 配置审核
- 状态报告
- 系统构建

软件配置管理：层次化的配置项标识



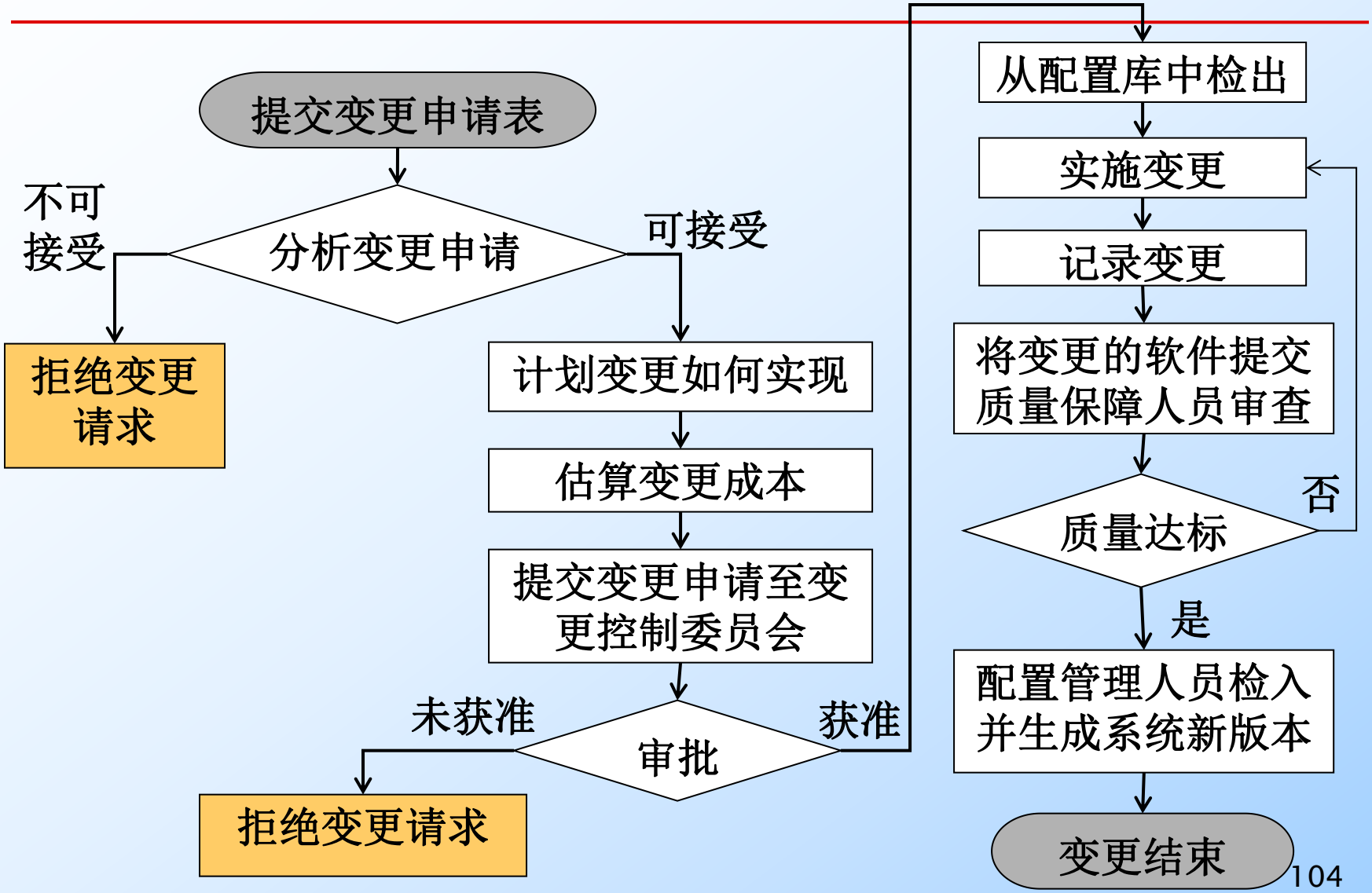
配置项命名：
唯一性、可追溯性

配置项标识：
名字、描述、
资源、实现

软件配置管理：变更控制

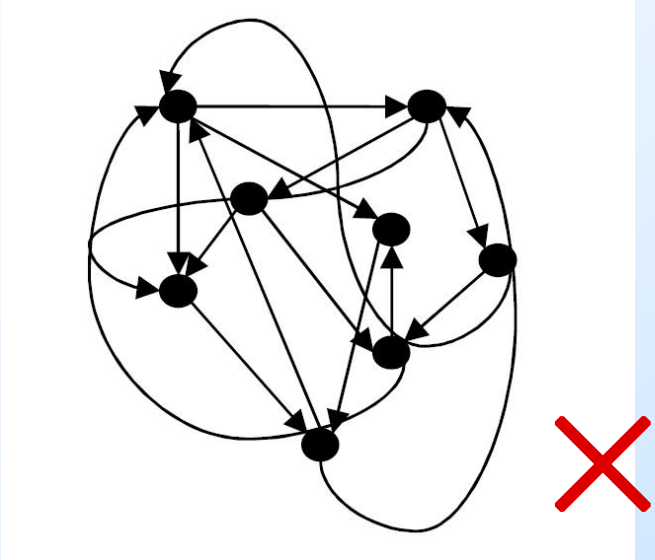
- 跟踪来自客户和开发者的变更请求，决定是否变更、何时变更
- 跨越里程碑进行修改/对基线的修改
- 开发库：开发人员使用，可频繁修改
- 受控库
  - 存放在生存期某一阶段工作结束时释放的阶段产品/基线
  - 软件配置管理的对象，也称为软件配置管理库
- 产品库:完成系统测试，等待交付

软件配置管理：变更控制过程

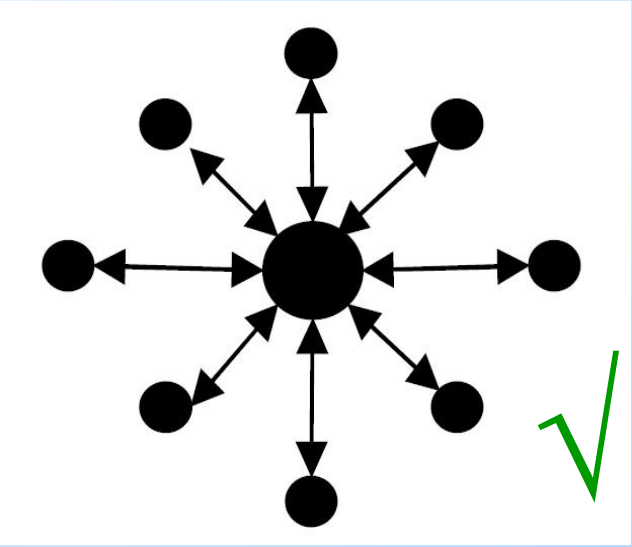


软件配置管理：版本控制

- 一个程序员
  - 及时保存软件，备份软件
- 多个程序员



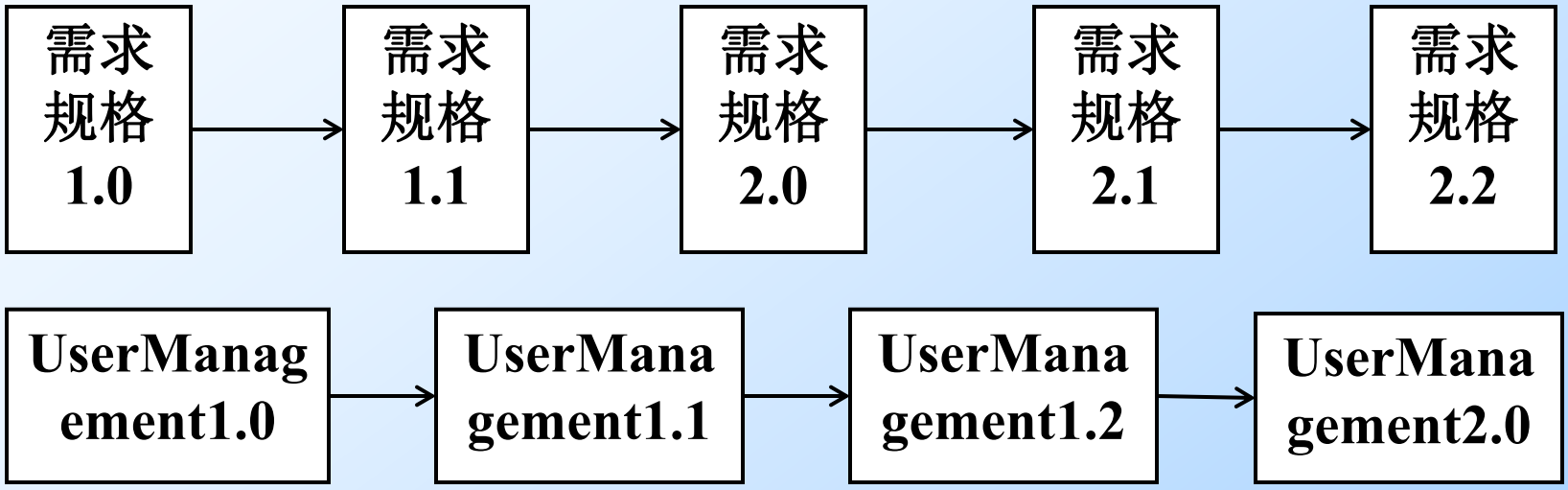
程序员之间随意传递代码



公共存储区

软件配置管理：版本控制

- 版本（Version）
  - 软件配置项的一个实例，在明确定义的时间点上某个配置项的状态
  - 通常由配置项名称加上版本号组成



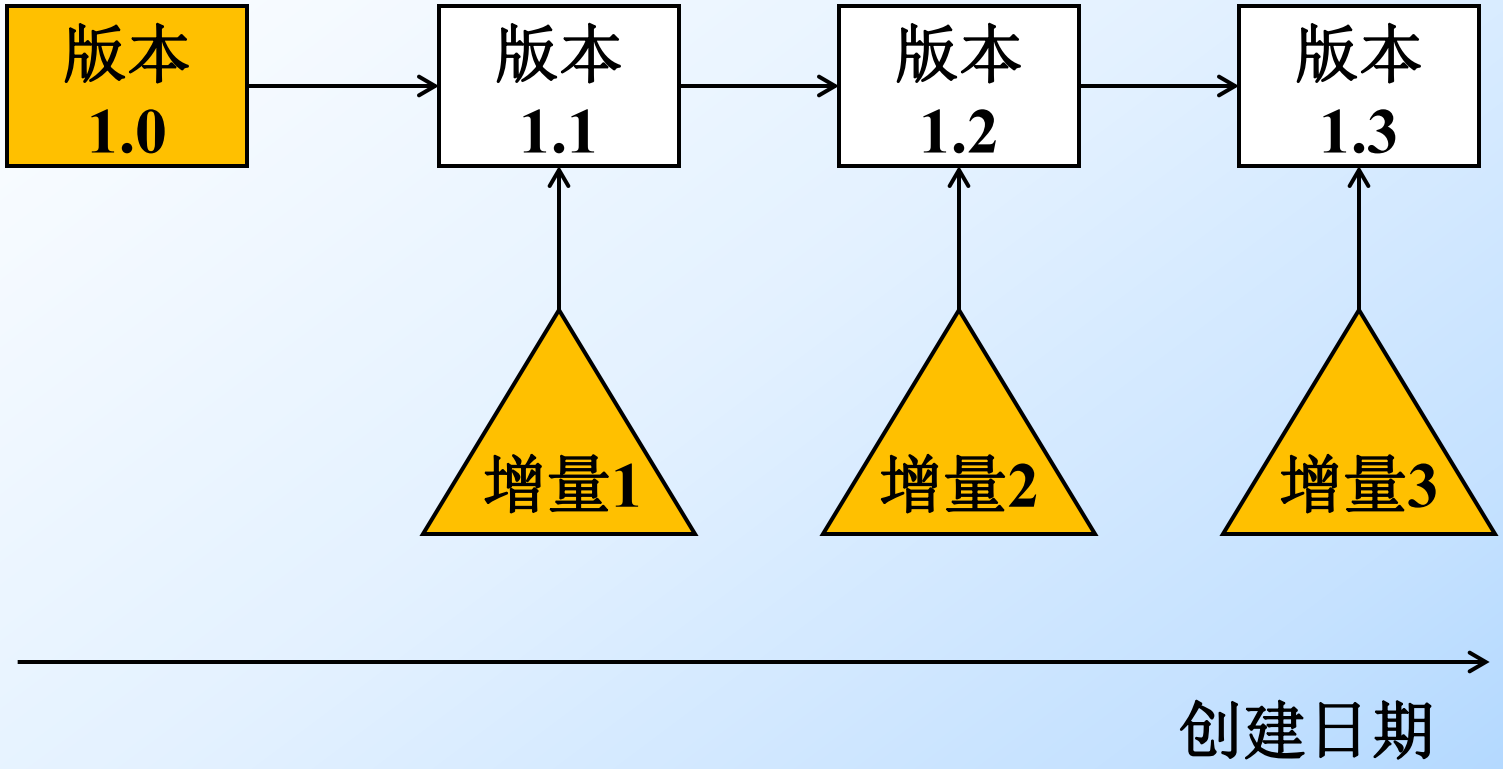
软件配置管理：版本控制

- 版本控制
  - 对软件的不同版本进行标识和跟踪、对版本的各种操作进行控制，如检入检出、分支与合并、版本发布等。
- 主要功能
  - 项目数据库：存储所有软件配置项
  - 版本管理：存储软件配置项的所有版本并进行管理
  - 系统构建：收集所有相关的配置项，构建软件特定版本
  - 错误跟踪

版本控制系统

- 软件：SVN、CVS、git ...
- 版本的标识
  - 通常由配置项名称加上版本号组成
- 变更的历史记录
  - 记录所有对软件做出的变更
- 存储管理
  - 建立中心数据库（Repository）
  - 增量存储（Delta storage）：只存储每个版本之间的差异

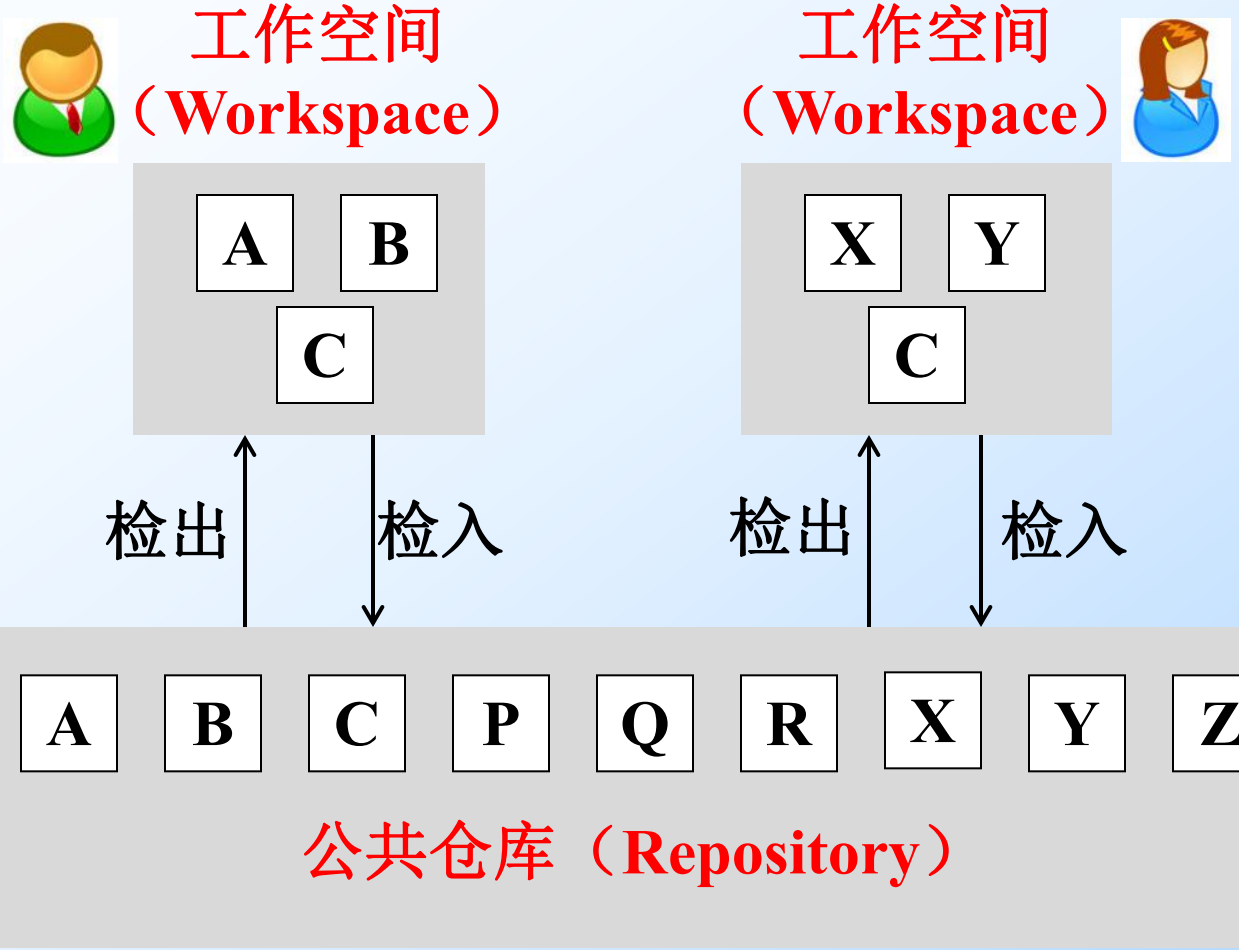
版本控制系统：增量存储



版本控制系统

- 并行开发
  - 串行开发既不有效也不实用
  - 开发人员不应等待其他开发人员的工作完成
  - 进行有效并行开发的关键是隔离软件配置项
  - 为每位开发者建立不同的工作目录
  - 避免同时修改同一个文件
  - 公共仓库和私有工作空间

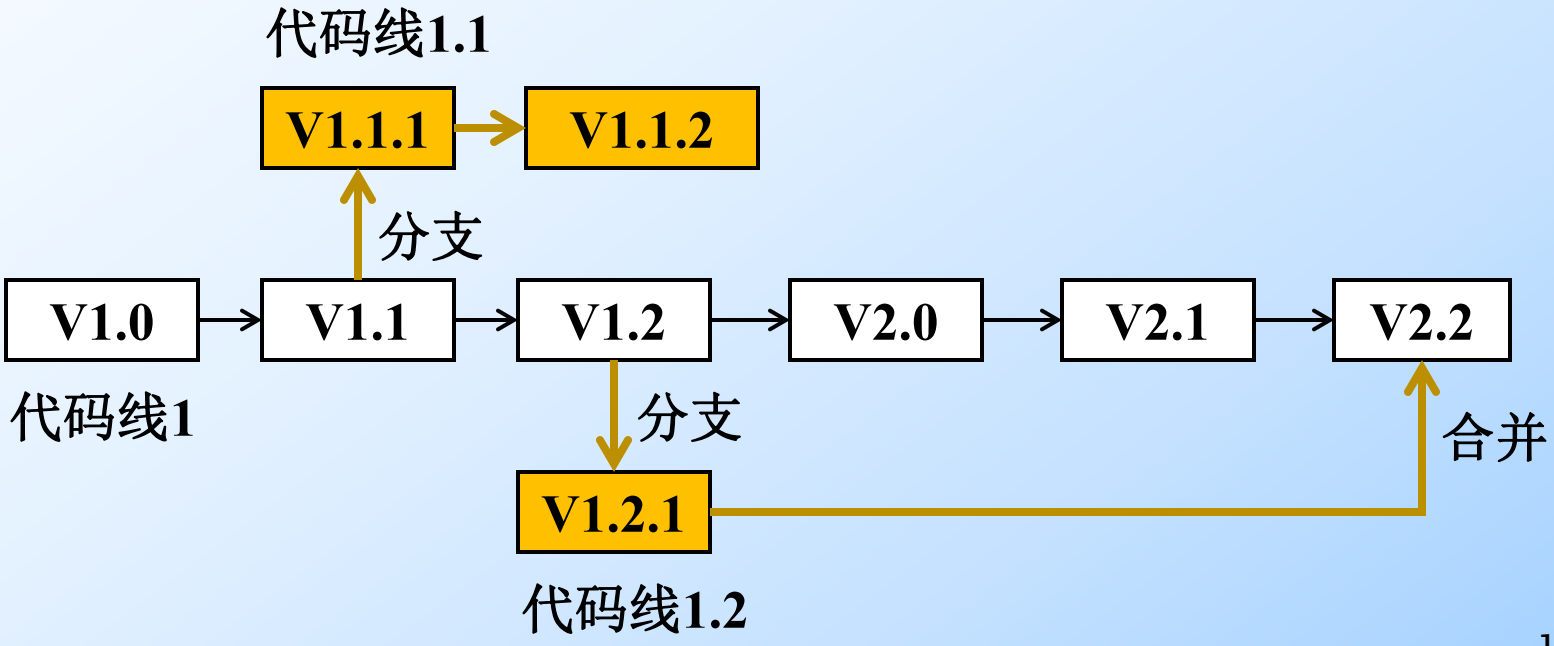
版本控制系统


工作空间
（Workspace）
工作空间
（Workspace）
A
B
C
X
Y
C
检出
检入
检出
检入
A
B
C
P
Q
R
X
Y
Z
公共仓库（Repository）

并行开发

版本控制系统

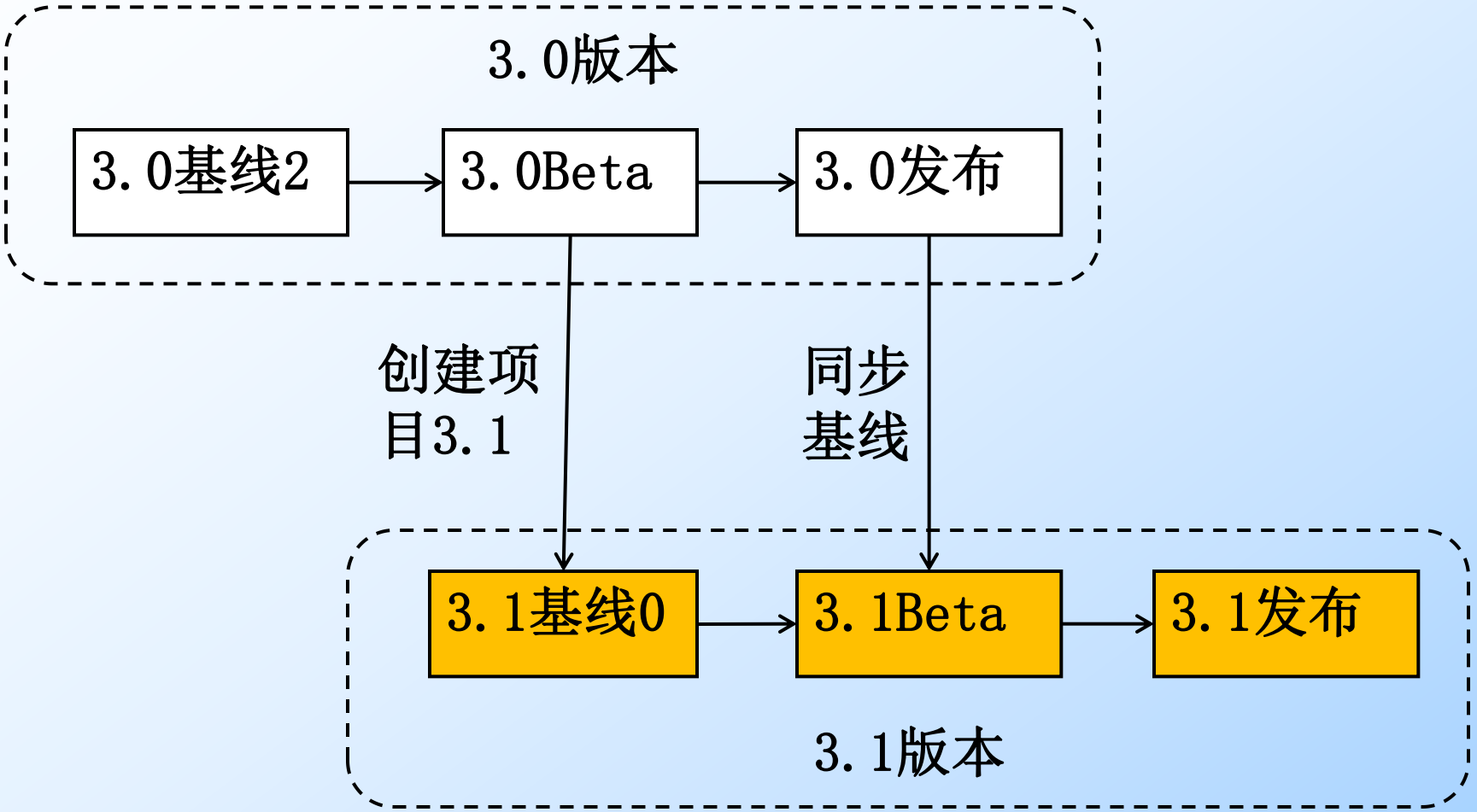
- 代码线分支（Branch）
  - 多个开发人员并行开发
  - 不同的软件开发路线、可能的实验性路线
  - 不同平台的版本、同一内容的不同界面开发



版本控制：案例

- 一个项目现处于版本3.0的Beta测试阶段，客户提出了很多功能增强请求。
- 为了保证产品按时发布，大部分功能增强请求将在下一个版本3.1中发布。
- 为了充分利用开发资源，项目组决定现在就开始3.1版本的开发。
- 如何管理版本3.0和3.1的并行开发？

版本控制：案例



软件配置管理：配置审核

- 确保变更的正确性
  - 正式的技术评审
    - 审查被修改的软件配置项的技术正确性
    - 与其它软件配置项的一致性
    - 是否有遗漏、是否引起副作用
  - 软件配置审核
    - 对正式的技术评审的补充
    - 评估正式技术评审没有考虑的特征
    - 修改时是否遵循软件工程标准
    - 是否在该配置项中显著地表明了修改
    - 是否注明修改日期和修改者
    - 是否适当更新了所有相关的软件配置项

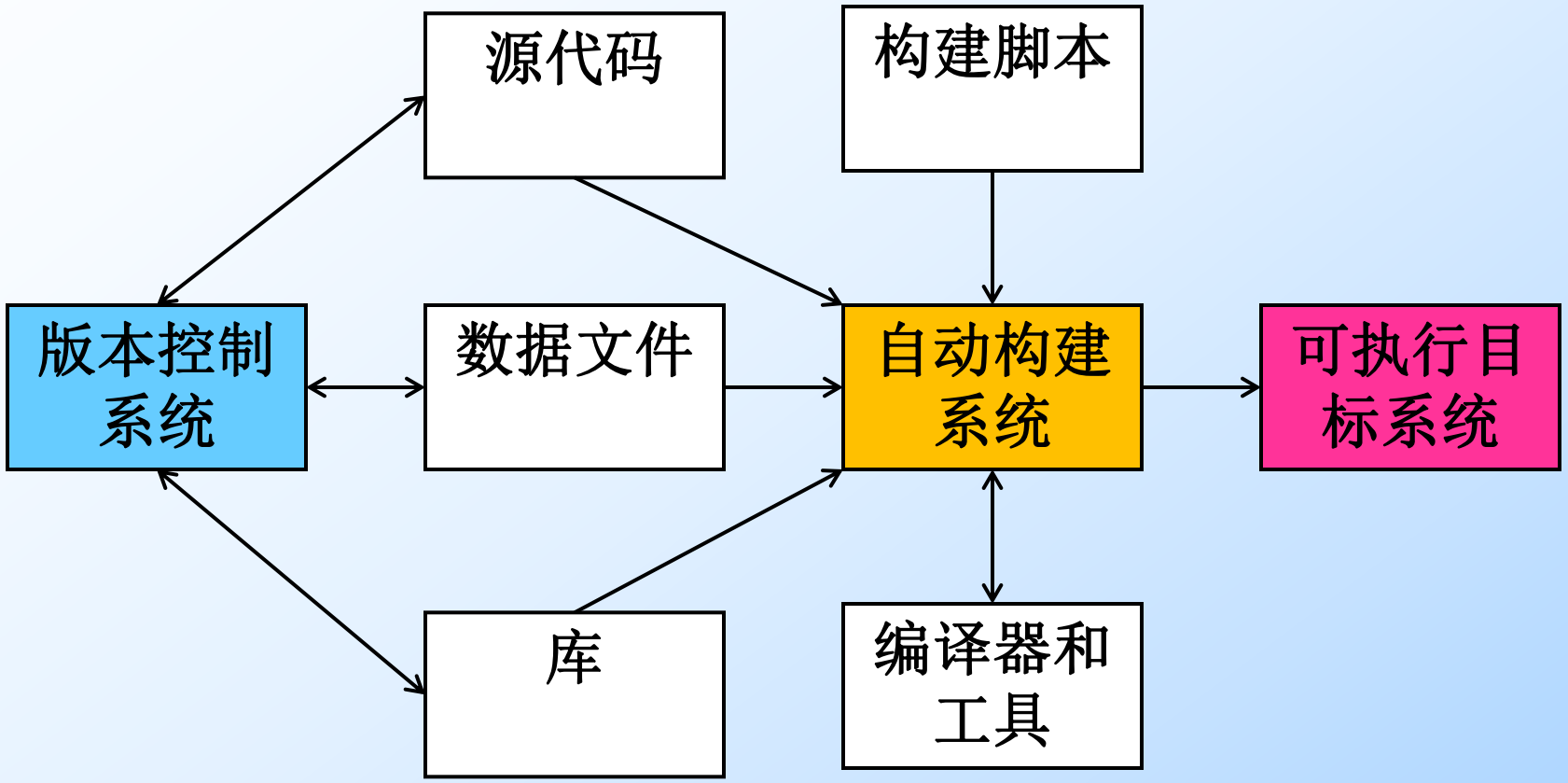
软件配置管理：状态报告

- 配置状态报告：记载软件配置的变化
  - 发生什么事？
  - 谁做了这件事？
  - 这件事是什么时候发生的？
  - 它将影响哪些其它事物？
- 状态报告时机
  - 增加新的软件配置项
  - 更改软件配置项的标识（名字、描述、资源、实现）
  - 变更软件配置项
  - 审核软件配置项

软件配置管理：系统构建

- 系统构建是把软件组件、外部库、配置文件等编译和链接成一个完整的能在特定目标配置上运行的程序的过程
  - 构成系统的所有部分是否都已包含在构建指令中
  - 是否包含每个部分的合适版本
  - 是否包含所有必需的数据文件
  - 编译程序和其它所需工具的版本是否合适
- 自动构建工具
- 构建脚本
  - 不同组件之间的依赖关系
  - 编译和链接工具

软件配置管理：系统构建过程



软件项目计划书

- Software Project Management Plan
- IEEE标准1058-1998

1 概述

1.1 项目概述

1.2 项目管理计划的演化

2 参考资料

3 定义和缩写

4 项目组织

4.1 外部接口

4.2 内部组织结构

4.3 角色与职责划分

软件项目计划书

5 管理过程
- 5.1 项目启动计划
- 5.2 工作计划
- 5.3 控制计划
- 5.4 风险管理计划
- 5.5 项目收尾计划

6 计划过程
- 6.1 过程模型
- 6.2 方法、工具和技术
- 6.3 基础设施
- 6.4 产品验收

7 支持过程
- 7.1 配置管理计划
- 7.2 验证和确认计划
- 7.3 文档计划
- 7.4 质量保证计划
- 7.5 评审计划
- 7.6 问题解决
- 7.7 分包管理计划
- 7.8 过程改进计划

8 其他计划

附录

索引

本章复习要点

- 软件项目管理：概念、基本要素4P
- 人员组织与管理
- 软件规模和工作量估算
  - 软件规模估算：代码行技术、功能点技术
  - 工作量估算：算法成本模型、COCOMO模型
- 项目进度计划：甘特图
- 软件风险管理
- 软件配置管理：软件配置项、基线、版本控制、配置管理的任务
- 软件项目计划书