北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

高级软件工程

第6节课：软件项目管理

主讲：刘 驰

主要内容

- 软件项目管理概述
- 项目估算
- 进度管理
- 配置管理

1 软件项目管理概述

管理目标

- 通常认为，项目成功的标志，也是项目管理人员争取的目标，应该包括以下几个方面。
  - 达到项目预期的软件产品功能和性能要求。也就是软件产品达到了用户已认可的需求规格说明的要求。
  - 时限要求。项目应在合同规定的期限内完成。
  - 项目开销限制在预算之内。

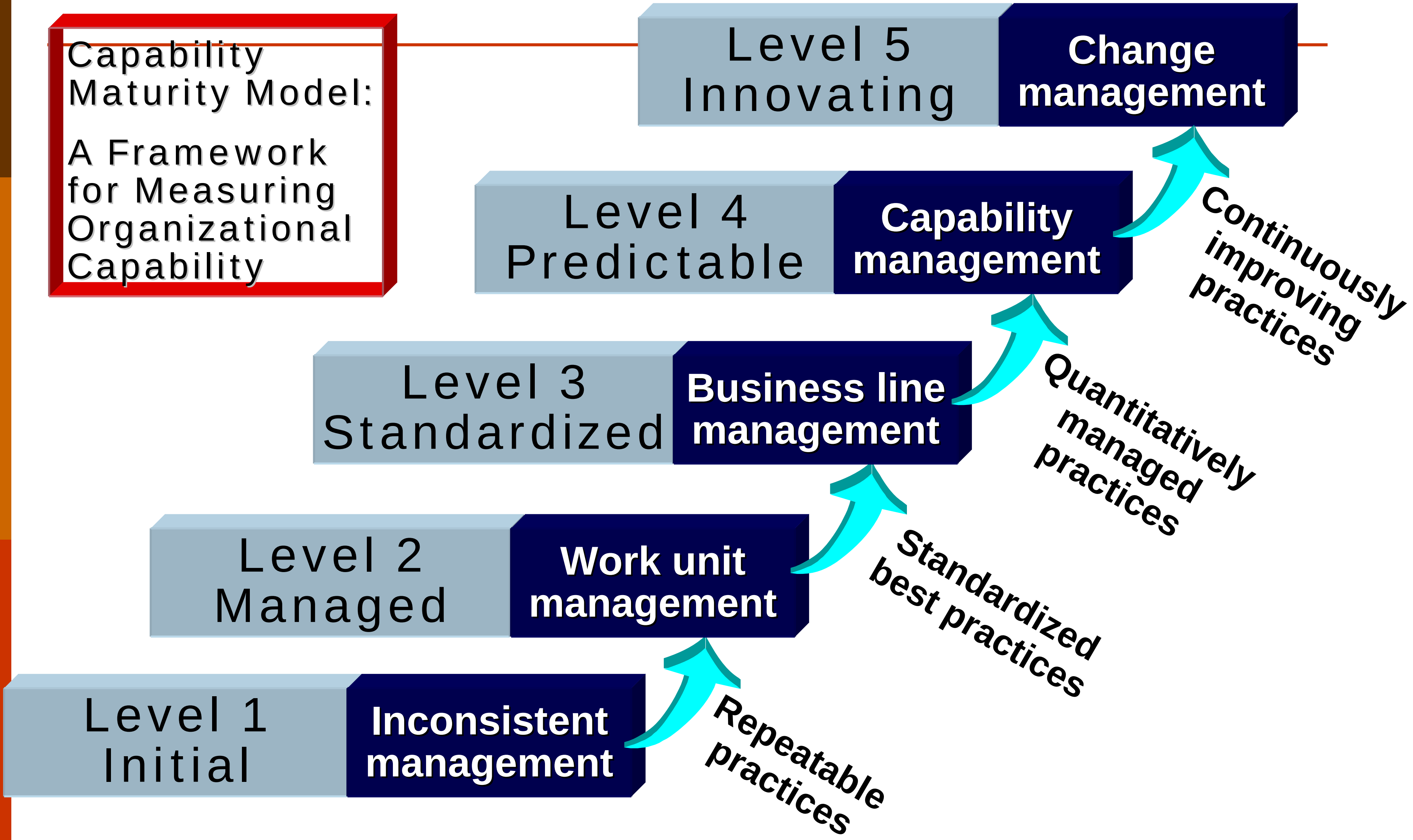
管理涉及的范围

- 软件项目管理涉及的几个主要方面是人员、产品、过程和项目，即所谓4P（People、Product、Process、Project）。
- 人员管理
  - 美国卡内基·梅隆大学软件工程研究所的Bill Curtis在1994年发表了“人员管理能力成熟度模型”（people capability maturity model，P-CMM）。该模型力图通过吸引、培养、激励、部署和聘用高水平的人才来提升软件组织的软件开发能力。

P-CMM: Five Capability Levels

Capability Maturity Model:

A Framework for Measuring Organizational Capability



Low Maturity Organizations

Rework
Rework

Firefighting

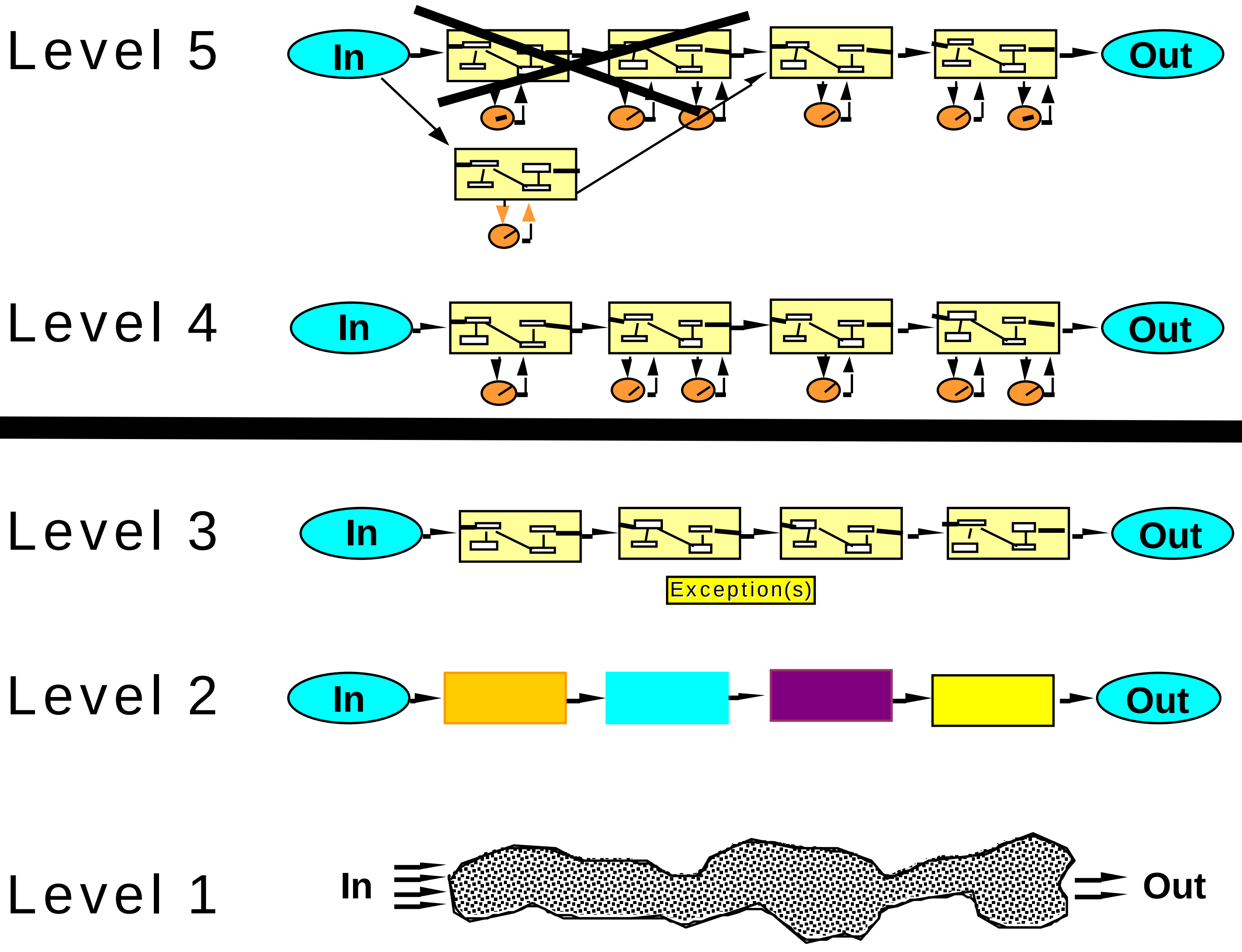
Heroes

<u>People capability variation</u>

Transaction workers: +/- 100%

Knowledge workers: +/- 1000%

Management Visibility



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

How the MM's Works-General

Level 5 Innovating	Implement continual proactive improvements to achieve business targets	▪Capable processes ▪Perpetual innovation ▪Change management
Level 4 Predictable	Manage process and results quantitatively and exploit benefits of standardization	▪Predictable results ▪Reuse/knowledge mgt. ▪Reduced variation
Level 3 Standardized	Develop standard processes, measures, and training for product & service offerings	▪Productivity growth ▪Effective automation ▪Economies of scale
Level 2 Managed	Build disciplined work unit management to stabilize work and control commitments	▪Reduced rework Repeatable practices ▪Satisfied schedules
▪Level 1 ▪Initial	▪Motivate people to overcome problems and just "get the job done"	▪Mistakes, bottlenecks ▪Ad hoc methods ▪Hero worship

How the MM's Work-Measurement

Level 5 Innovating	Implement continual proactive improvements to achieve business targets	▪Deep/unique process insight supported by numbers/analytics
Level 4 Predictable	Manage process and results quantitatively and exploit benefits of standardization	▪Manage by the numbers/analytics
▪Level 3 ▪Standardized	▪Develop standard processes, measures, and training for product & service offerings	▪Process and data standards ▪Aggregation possible
▪Level 2 ▪Managed	▪Build disciplined work unit management to stabilize work and control commitments	▪Work group/project quality ▪Aggregation difficult/costly
▪Level 1 ▪Initial	▪Motivate people to overcome problems and just "get the job done"	▪Ad hoc methods ▪No data standards ▪No aggregation

人员管理涉及：

① 共利益者。 包括：

- 项目的高级管理者——负责项目商务问题的决策；
- 项目经理——负责项目的计划与实施以及开发人员的组织与管理；
- 开发人员——项目开发的实施者；
- 客户——提出需求并代表用户与开发人员交往的人员；
- 最终用户——直接使用项目成果（产品）的人员。

② 团队负责人。在小项目的情况下，项目经理就是团队负责人。而大型项目也许会有若干个设计、编程团队或是若干个测试团队。团队负责人除去负有团队日常工作的安排、组织和管理之外，还应特别注意发挥团队成员的潜能。

③ 团队集体。团队内部有分工是必要的，但必须很好地配合，做到步调一致，为此必须强调以下3点。

- 个人的责任心，这是团队完成工作的基本条件。
- 互相信任、尊重以及互相支持。
- 充分的交流与沟通。

- 产品管理

  项目经理必须在项目开始时就明确项目的以下三个目标：

  - 产品的工作环境。
  - 产品的功能和性能。
  - 产品工作处理的是什么数据，经它处理后得到什么数据。

  只有明确了项目的这些基本要求才能着手项目管理的各项工作，如项目估算、风险分析、项目计划的制定等。

- 过程管理

  过程在软件工程项目中是重要的因素，它决定着项目中开展哪些活动以及对活动的要求和开展活动的顺序。

- 项目管理

  项目管理的任务是如何利用已有的资源，组织实施既定的项目，提交给用户适用的产品。

项目管理要开展的主要工作可分为3类。

① 计划及计划管理。包括项目策划及计划制定、项目估算、风险分析及风险管理、进度管理、计划跟踪与监督。

② 资源管理。包括人员管理（人员安排、使用）、成本管理、信息管理。

③ 成果要求管理。包括需求管理、配置管理、质量管理。

2 项目估算

通常在项目的目标确定和软件基本功能确定之后，就应该着手项目计划的制定工作。项目估算是制订计划的基础和依据。

- 项目策划与项目估算

项目策划是项目开展初期阶段的重要工作，其主要目标是得到项目计划，或者说计划（plan）是策划（planning）的结果。

项目策划中需要开展的活动

(1) 确认并分析项目的特征。

(2) 选择项目将遵循的生存期模型，确定各阶段的任务。

(3) 确定应得到的阶段性工作产品以及最终的产品。

(4) 开展项目估算，包括估算产品规模、工作量、成本以及所需的关键计算机资源。

(5) 制订项目进度计划。

(6) 对项目风险进行分析。

(7) 制订项目计划。

在项目估算中，要解决的问题是项目实施的几个主要属性，即将要开发产品的规模（size）、项目所需的工作量（effort）以及项目的成本（cost）。

（1）规模。项目的规模指的是得到最终软件产品的大小。一般以编程阶段完成以后得到程序的代码行表示，如以1千行代码为单位，记为KLOC。

当然，在项目的开始只是对代码行的估计值。另一表示方法是功能点，记为FP (Functional Point)，它是根据软件需求中的功能估算的。

（2）工作量。项目的工作量按项目将要投入的人工来考虑，以一个人工作一个月为单位，记为“人月”。

（3）成本。软件项目的成本通常只考虑投入的人工成本，如某项目投入的总人工费用为12万元。

● 成本计算

一个软件公司在完成多个项目以后积累了一些数据，进行成本分析后便可得到自己的生产率数值和人工价格。

- 生产率是平均每个人月完成的源程序行数，可记为KLOC/人月，或：FP/人月。
- 人工价则为每人月的价值。

有了这两个数值，如果在估出项目规模以后就可以很容易得到项目的工作量和成本，即

工作量=规模/生产率

成本=工作量×人工价

项目估算方法：功能点方法

- 克服了项目开始时无法得知源程序行数的实际困难，从软件产品的功能度（functionality）出发估算出软件产品的规模。
- 功能度

　　功能点方法是以项目的SRS中已经得到确认的软件功能为依据，着重分析要开发系统的功能度，并且认为，软件的大小与软件的功能度相关，而与软件功能如何描述无关，也与功能需求如何设计和实现无关。

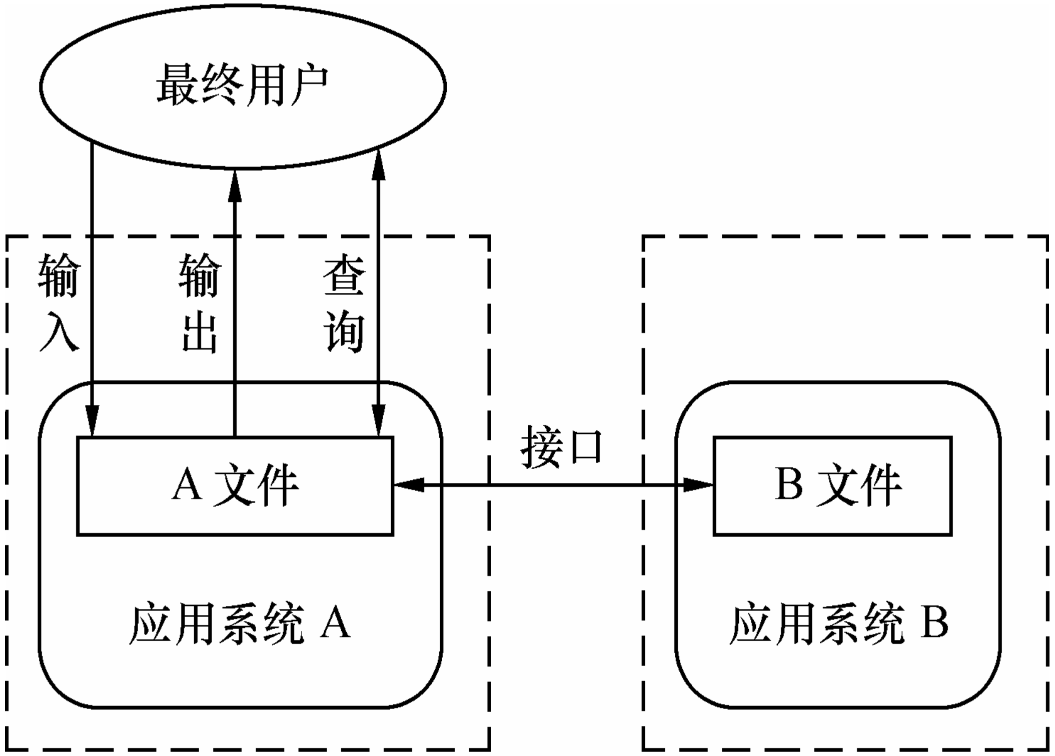
1．功能度

为具体说明功能点方法，区分各种不同的功能，需要建立应用系统边界的概念。

- 应用系统边界把目标应用系统与用户和与其相关的应用系统分割开来。
- 内部功能仅限于应用系统的边界之内，而外部功能则是跨边界的。

- 系统边界

  图中系统A有4项功能都是跨越边界的，称其为外部功能。



应用系统 A 的边界

- 五种类型的功能：

（1）外部输入。处理那些进入应用系统边界的数据或是控制信息。经特定的逻辑处理后，形成内部逻辑文件。

（2）外部输出。处理离开应用系统边界的数据或控制信息。

（3）内部逻辑文件。是用户可识别的逻辑相关数据或控制信息组，它可在应用系统边界之内使用。内部逻辑文件代表应用系统可支持的数据存储需求。

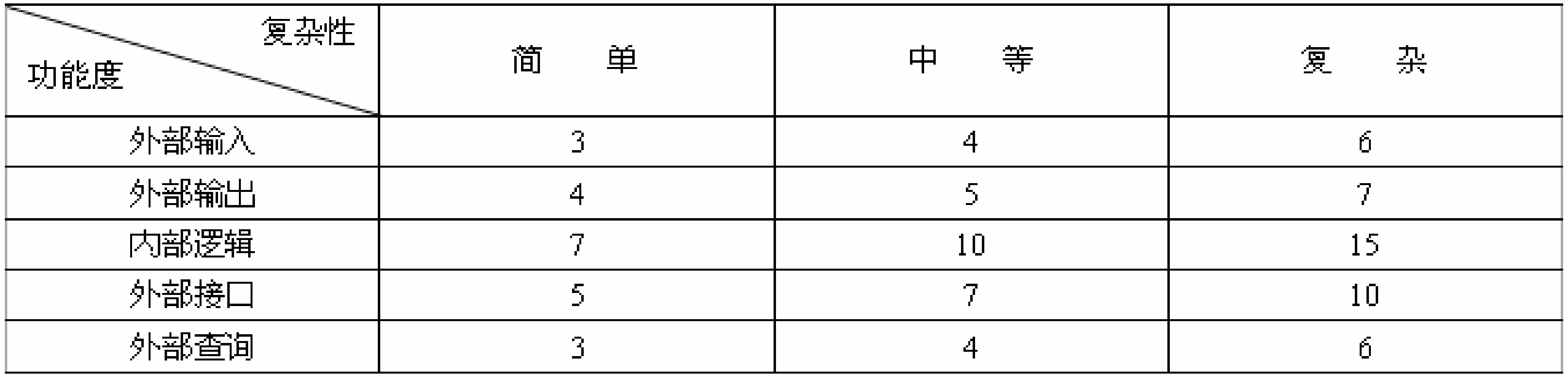
（4）外部接口文件。用户可识别的逻辑相关数据或控制信息构成的集合，该控制信息为应用系统所使用，却被另一应用系统所支持。外部接口文件代表应用系统外部支持的数据存储需求。

（5）外部查询。唯一的输入/输出组合，它为实现即时输出引起所需数据的检索，代表了应用系统查询处理的需求。

2. 功能复杂性

软件项目每类功能的复杂程度可能各不相同，为表明功能复杂性的差别，将其分为简单、中等和复杂3个等级。同时为表示其差异程度，分别给予不同的影响参数。下表列出了功能复杂性的影响参数值。

复杂性 / 功能度	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



复杂性 功能度	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

3．未调节功能点

只要能够从SRS中得到了以上5种功能度的各级复杂性功能点的个数C，不难计算出未调节功能点的值 (UFP)。

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

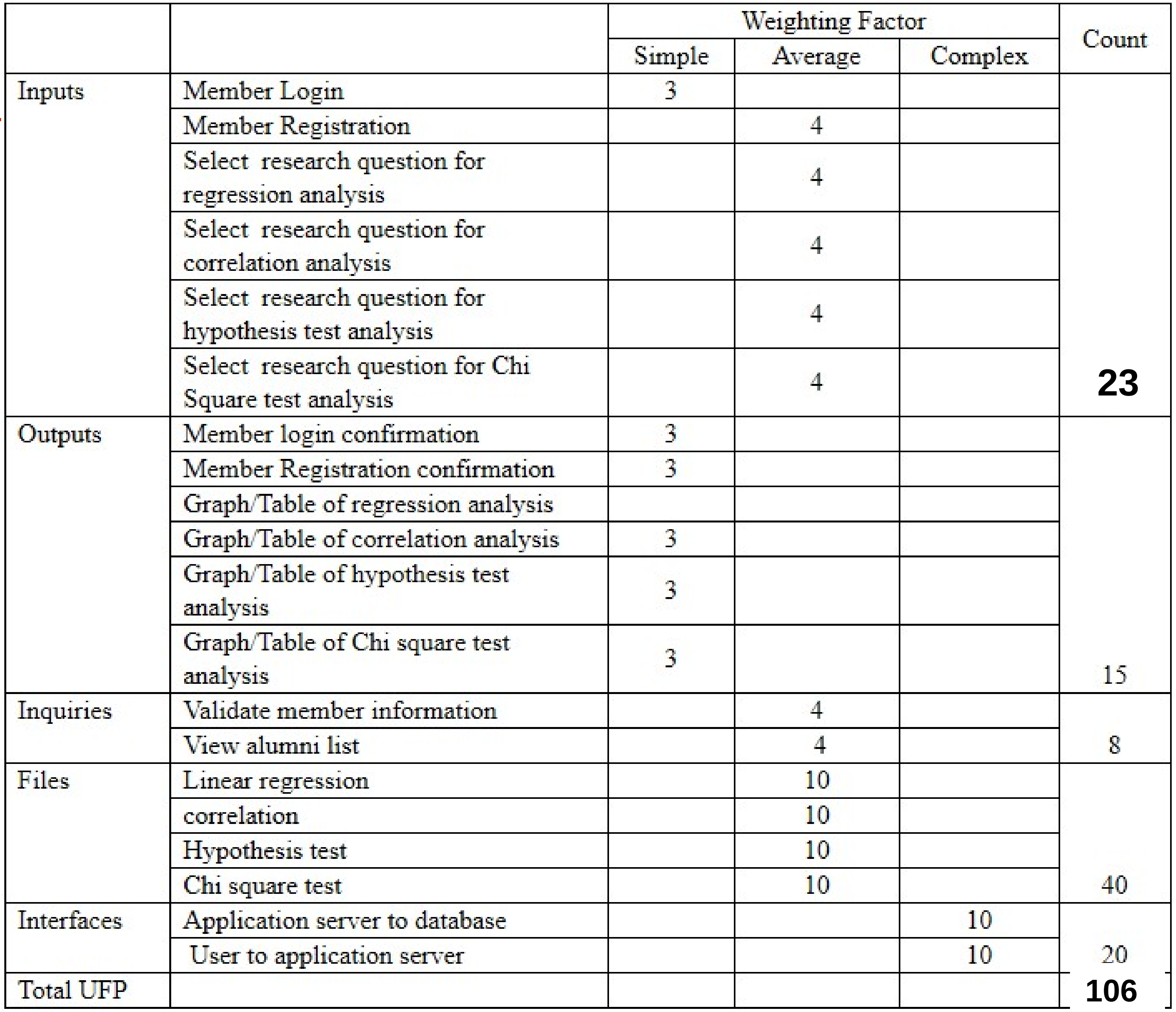
其中：i 代表功能度类型号；i=1，2，…，5；

j 代表复杂性的等级；j=1，2，3；

ω_{ij}是第i类功能度和第j级复杂性的影响参数，即上表中第i行，第j列的参数值；

C_{ij}是第i类功能度和第j级复杂度功能点的个数。

Example



		Weighting Factor			Count
		Simple	Average	Complex	
Inputs	Member Login	3			
	Member Registration		4		
	Select research question for regression analysis		4		
	Select research question for correlation analysis		4		
	Select research question for hypothesis test analysis		4		
	Select research question for Chi Square test analysis		4		23
Outputs	Member login confirmation	3			
	Member Registration confirmation	3			
	Graph/Table of regression analysis				
	Graph/Table of correlation analysis	3			
	Graph/Table of hypothesis test analysis	3			
	Graph/Table of Chi square test analysis	3			15
Inquiries	Validate member information		4		
	View alumni list		4		8
Files	Linear regression		10		
	correlation		10		
	Hypothesis test		10		
	Chi square test		10		40
Interfaces	Application server to database			10	
	User to application server			10	20
Total UFP					106

4. 调节因子

任何软件都会有其自身特性，从以下两个方面分解功能点计算的调节因子。

（1）影响因子：

- 数据通信
- 分布数据处理
- 性能目标
- 系统配置要求
- 事务率
- 联机数据录入
- 最终用户效率
- 联机更新
- 复杂的处理逻辑
- 可复用性
- 易安装性
- 易操作性
- 多工作场所
- 设施变更

（2）影响级。上述影响因子对软件功能度的影响有多大必须加以区分，于是将影响因子的影响程度分为6级，即

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

综合考虑14类影响因子的影响度N，应是将14种影响叠加起来，其值为0～70（14×5）。由此得到复杂度调节因子(Complexity Adjustment Factor, CAF)

$$CAF=0.65+0.01N$$

其值应在0.65～1.35，其中基本调节常数是0.65，可见最大的调节量为35%。

5. 交付功能点

- 经过调节因子调节后的功能点值被称为交付功能点 (Delivered function point, DFP)
- DFP = CAF × UFP

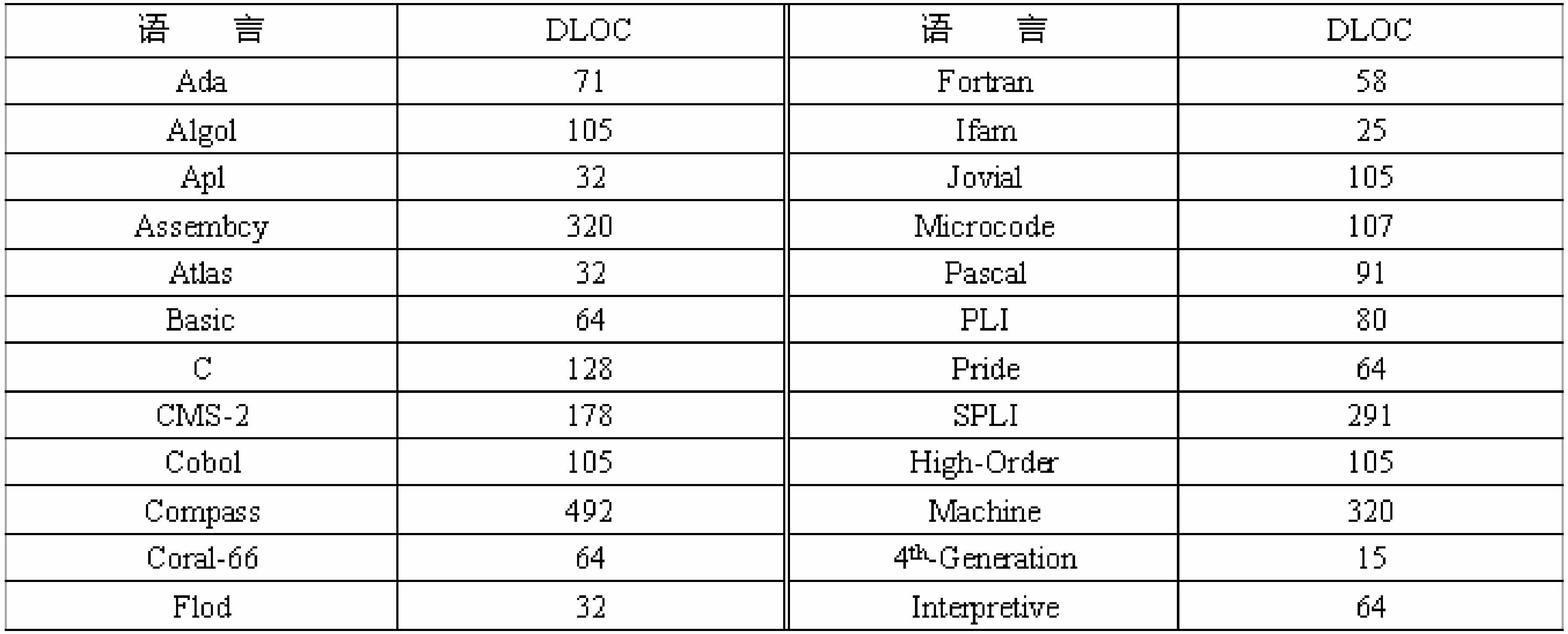
6. 交付功能点与软件规模

- 一些研究表明，上述计算出的功能点的值可以代表软件的规模，也可作为估算成本的依据。
- 软件的规模(Size)可用交付的源代码行数 (Delivered Lines of Code, DLOC)来表示。

功能点与DLOC的对应关系如下表所示。例如：

1 DFP相当于105 DLOC（COBOL程序）

1 DFP相当于128 DLOC（C程序）



语　　言	DLOC	语　　言	DLOC
Ada	71	Fortran	58
Algol	105	Ifam	25
Apl	32	Jovial	105
Assembcy	320	Microcode	107
Atlas	32	Pascal	91
Basic	64	PLI	80
C	128	Pride	64
CMS-2	178	SPLI	291
Cobol	105	High-Order	105
Compass	492	Machine	320
Coral-66	64	4th-Generation	15
Flod	32	Interpretive	64

7. 功能点方法的优点

（1）DFP只与SRS信息相关，而交付代码的行数若不通过功能点计算是不能直接从规格说明中得到的。

（2）DFP与实现软件的语言无关。

8. 功能点方法的不足之处

（1）针对需求规格说明进行分析时，主观因素难以完全排除：

● 对于SRS每人可能有不同的解释；

● 对于功能度的复杂性估计也可能因人而异；

● CAF计算时会有主观因素。

（2）实时软件、系统软件、科学计算软件等功能点的上述计算方法并不适用。

（3）DFP的计算目前尚不能借助工具自动完成。

软件开发成本估算

1．专家判定——Delphi方法

专家判定技术就是由多位专家进行成本估算，取得多个估算值。有多种方法把这些估算值合成一个估算值，Read公司提出了Delphi技术，作为统一专家意见的方法。可得到极为准确的估算值。

标准Delphi技术的步骤：

① 组织者发给每位专家一份SRS（略去名称和单位）和一张记录估算值的表格，请他们进行估算。

② 专家详细研究SRS内容，然后组织者召集小组会议，在会上，专家们与组织者一起对估算问题进行讨论。

③ 各位专家对该软件提出3个软件规模的估算值，即

a_i——该软件可能的最小规模（最少源代码行数）；

m_i——该软件最可能的规模（最可能的源代码行数）；

b_i——该软件可能的最大规模（最多源代码行数）。

无记名地填写表格，并说明做此估算的理由。

④ 组织者对各位专家在表中填写的估算值进行综合和分类

- 计算各位专家（序号为i, i=1,2,...,n）的估算期望值E_i和估算值的期望中值E。

$$E_i = \frac{a_i + 4m_i + b_i}{6}; \qquad E = \frac{1}{n}\sum_{i=1}^{n} E_i$$ （n 为专家人数）

- 对专家的估算结果进行分析。

⑤ 组织者召集会议，请专家们对其估算值有很大差异之处进行讨论。专家对此估算值另做一次估算。

⑥ 在综合专家估算结果的基础上，组织专家再次无记名填表。

从步骤④到步骤⑥适当重复几次，最终可获得一个得到多数专家共识的软件规模（源代码行数）。

最后，通过与历史资料进行类比，根据过去完成项目的规模和成本等信息，推算出该软件每行源代码所需成本。然后再乘以该软件源代码行数的估算值，得到该软件的成本估算值。

2．COCOMO模型

- 软件工程专家Barry Boehm在其著作《软件工程经济学》中提出了软件估算模型层次结构，称为构造式成本模型COCOMO
- COnstructive COst MOdel
- 软件界影响最为广泛、最为著名的估算模型。

(1) 3种类型的项目

COCOMO是针对Boehm划分的3种类型软件进行估算的。

① 固有型（organic mode）项目。规模较小，较为简单的项目，开发人员对项目有较好的理解和较为丰富的经验。

② 嵌入型（embedded mode）项目。这类项目的开发工作紧密地与系统中的硬件、软件和运行限制联系在一起，如飞机的飞行控制软件。

③ 半独立性（semi-detached mode）项目。项目的性质介于上述两个类型之间，其规模与复杂性均属中等，如事务处理系统，数据库管理系统等。

（2）COCOMO的3级模型

① 基本COCOMO模型（basic model）。

- 该模型为静态、单变量，以估算出的源代码行数计算。
- 开发工作量: $E = a_b(\text{KLOC})^{b_b}$ （人月）

其中，KLOC为交付的千行代码数。a_b、b_b为模型系数。

- 开发周期： T = c*E^d （月）
- 系数，如下表所示。

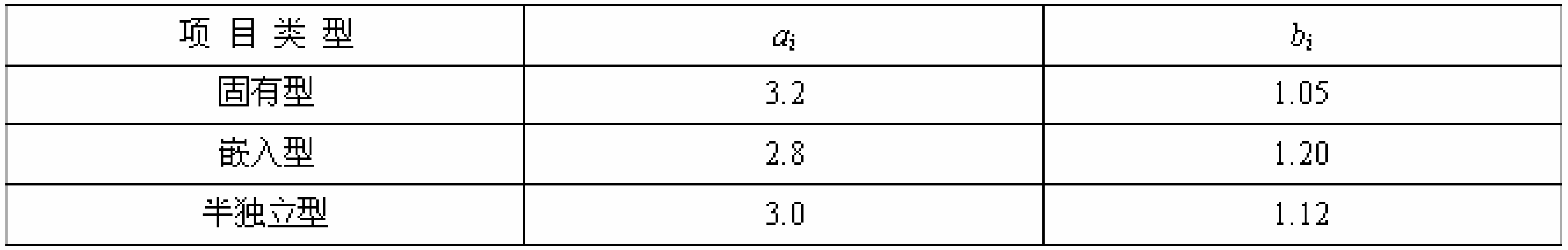
项 目 类 型	a_b	b_b	c_b	d_b
固有型	2.4	1.05	2.5	0.38
嵌入型	3.6	1.20	2.5	0.32
半独立型	3.0	1.12	2.5	0.35

② 中级COCOMO模型（Intermediate）

- 该模型除考虑源代码行数外，还考虑调节因子（EAF），用其体现产品、软件、人员和项目等因素。
- 开发工作量

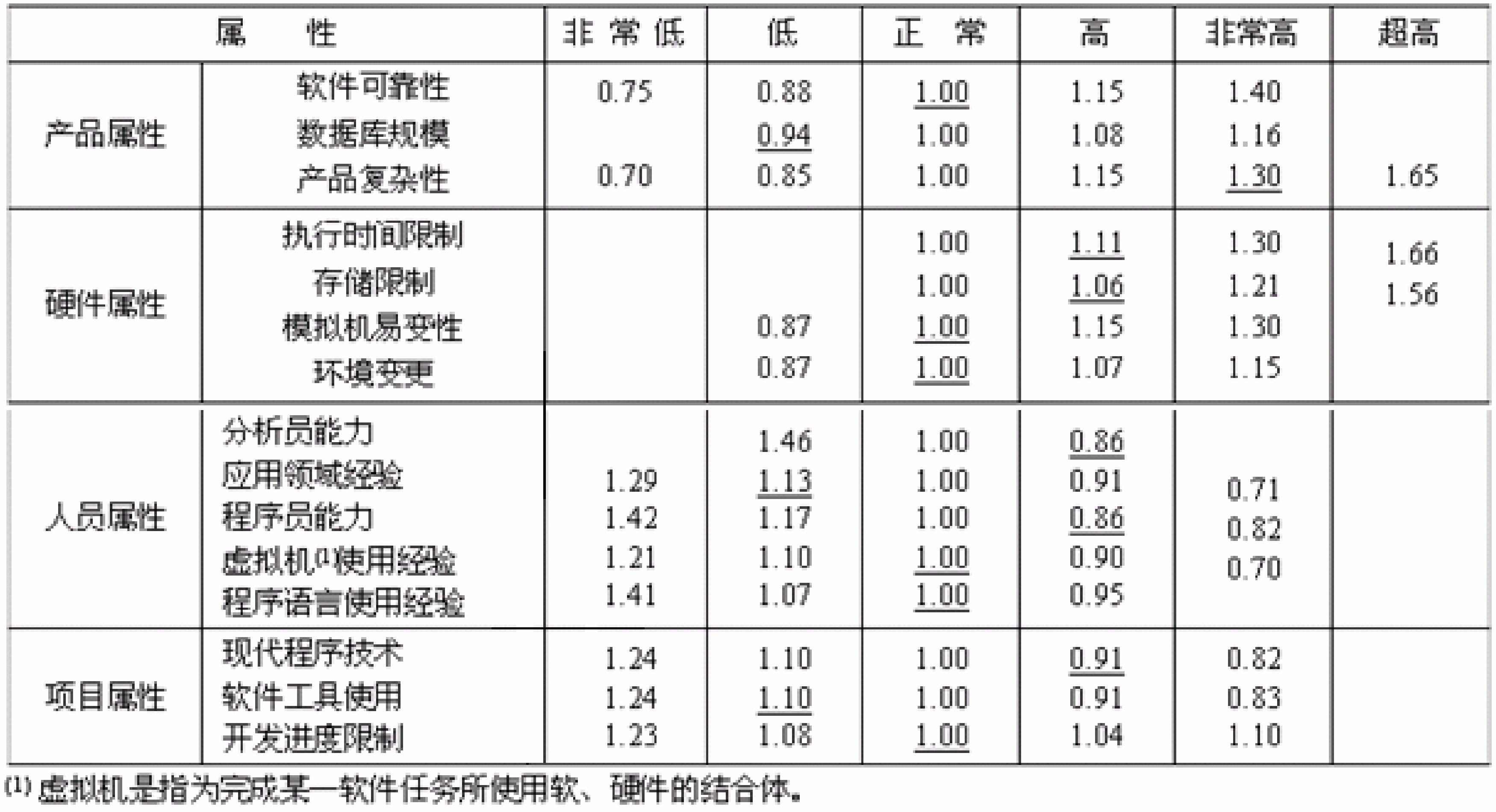
$$E = a_i (\mathrm{KLOC})^{b_i} \times \mathrm{EAF}$$

- 系数



项 目 类 型	a_i	b_i
固有型	3.2	1.05
嵌入型	2.8	1.20
半独立型	3.0	1.12

● 调节因子EAF（Effort Adjustment Factor）。包含了4类15种属性，其值为0.7～1.6。



属性		非常低	低	正常	高	非常高	超高
产品属性	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
硬件属性	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
	模拟机易变性		0.87	1.00	1.15	1.30	
	环境变更		0.87	1.00	1.07	1.15	
人员属性	分析员能力		1.46	1.00	0.86	0.71	
	应用领域经验	1.29	1.13	1.00	0.91	0.82	
	程序员能力	1.42	1.17	1.00	0.86	0.70	
	虚拟机(1)使用经验	1.21	1.10	1.00	0.90		
	程序语言使用经验	1.41	1.07	1.00	0.95		
项目属性	现代程序技术	1.24	1.10	1.00	0.91	0.82	
	软件工具使用	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	

(1) 虚拟机是指为完成某一软件任务所使用软、硬件的结合体。

Example

- The project is ***a flight control system*** (mission critical) with *319,000 DSI* in *embedded* mode
- Reliability must be very high (RELY=1.40).

So we can calculate:

- **E** = 1.40*3.6*(319)^1.20 = 5093 MM
- **T** = 2.5*(5093)^0.32 = 38.4 months
- **Average Staffing** = 5093 MM/38.4 months = 133 FSP

③ 高级COCOMO模型（advanced）

- 除保留中级模型的因素外，还涉及软件工程过程不同开发阶段的影响，以及系统层、子系统层和模块层的差别。
- 软件可靠性在子系统层各开发阶段有不同的调节因子。

开发阶段 / 可靠性级别	需求和产品设计	详 细 设 计	编程及单元测试	集成及测试	综　　合
非常低	0.80	0.80	0.80	0.60	0.75
低	0.90	0.90	0.90	0.80	0.88
正常	1.00	1.00	1.00	1.00	1.00
高	1.10	1.10	1.10	1.30	1.15
非常高	1.30	1.30	1.30	1.70	1.40

3 进度管理

- 进度控制问题
- 甘特图
- 时标网状图
- PERT图

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

进度控制问题

1．值得重视的现象

软件项目能否按计划的时间完成，及时提交产品是项目管理的一个重要课题。我们都希望按计划及时完成，但项目未能按预期的进度提交产品，延误工期的现象经常会出现。

2．制订项目进度安排的条件

制订项目进度安排计划是为了实施，自然希望越准确，越符合实际越好，但是怎样才能做到这一点，需要在这以前做些工作，创造良好的条件，使得进度安排的确定是有根据的。

这些条件包括以下7条：

（1）项目分解。无论多么大、多么复杂的项目首先将其划分成能够管理的若干活动和任务，并且往往这种分解是多个层次的。

（2）确定各部分之间的相互关系。划分后的活动和任务必定存在一定的相互依赖关系，如谁先谁后，或是两者应该并行互不依赖等。

（3）时间分配。为每项活动和任务分配需要的时间，如人天工作量。

（4）确认投入的工作量。应确认按项目要求的人力投入工作量在实际工作中能够予以满足，而不致出现某些工作阶段人力投入不足的现象。

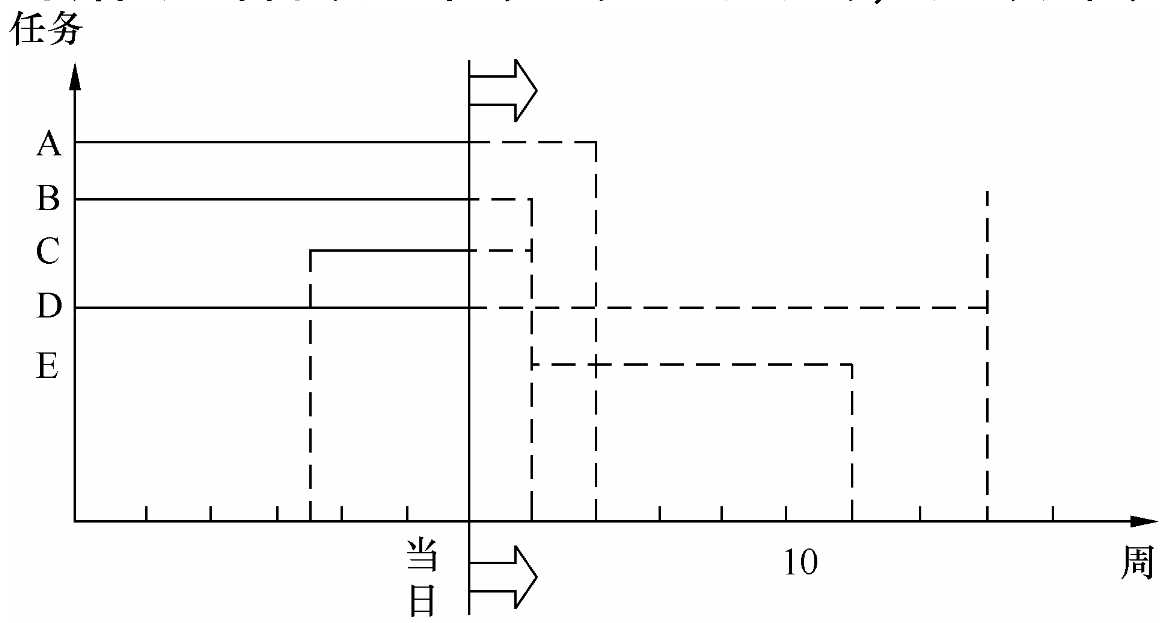
（5）确定人员的责任

（6）规定工作成果。任何分配的任务都应给出符合要求的工作成果，它应该是整个项目的一个组成部分。

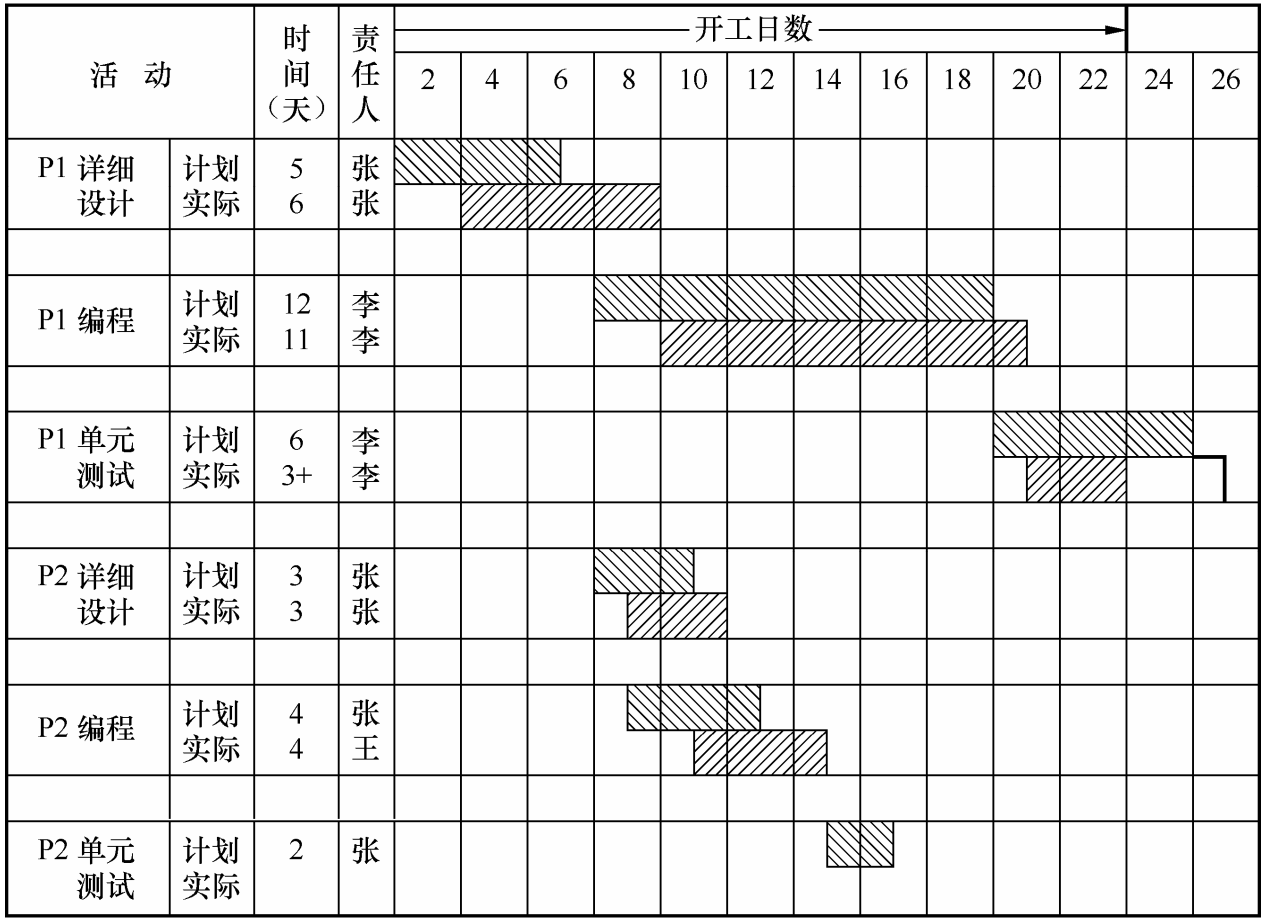
（7）规定里程碑。任何一项工作完成后需经过一定形式的检验，如经过评审或审核（批准）得到认可，被认为确已完成，表示一个里程碑已经完成。里程碑也被称为基线。

甘特图（Gantt Chart）

- 表示工作进度计划以及实际进度情况
- 横坐标表示时间，水平线段表示子任务的工作阶段，可以为其命名。
- 线段的起点和终点分别对应着该项子任务的开工时间和完成时间，线段的长度表示完成它所需的时间，有实线和虚线之分，一开始做出各项子任务的计划时间，应该都以虚线表示。



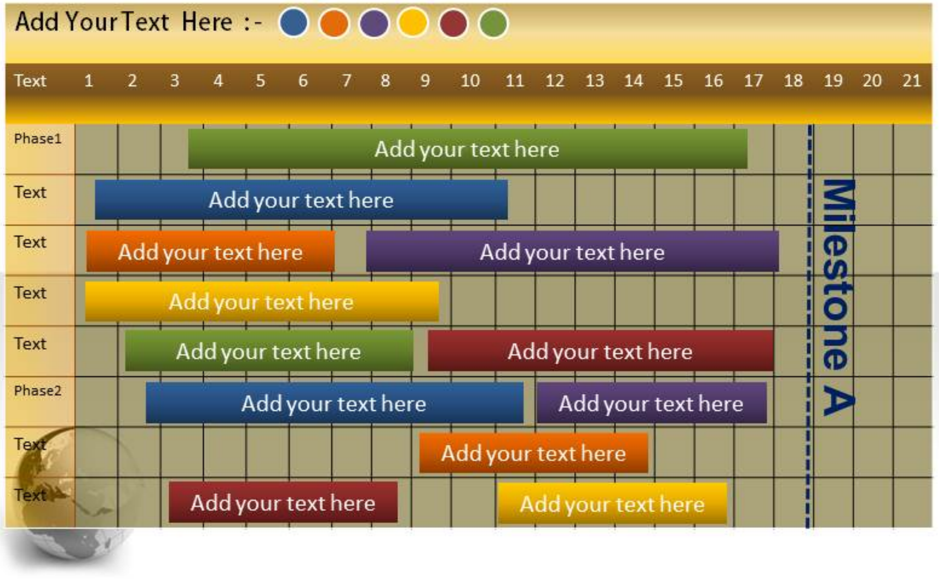
清楚地表示各项子任务在时间对比上的关系，但无法表达多个子任务之间更为复杂的衔接关系。

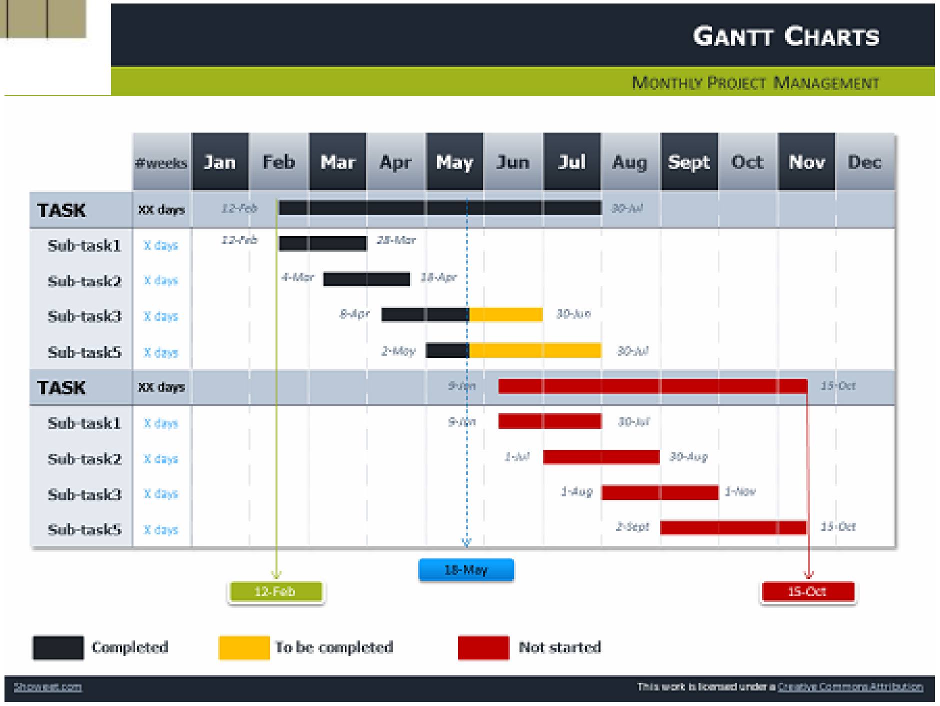


活　动		时间（天）	责任人	开工日数 2	4	6	8	10	12	14	16	18	20	22	24	26
P1 详细设计	计划	5	张	▧	▧	▧										
	实际	6	张		▨	▨	▨									
P1 编程	计划	12	李				▧	▧	▧	▧	▧	▧				
	实际	11	李					▨	▨	▨	▨	▨	▨			
P1 单元测试	计划	6	李										▧	▧	▧	
	实际	3+	李										▨	▨		
P2 详细设计	计划	3	张				▧	▧								
	实际	3	张				▨	▨								
P2 编程	计划	4	张				▧	▧	▧							
	实际	4	王					▨	▨	▨						
P2 单元测试	计划	2	张							▧	▧					
	实际															

Gantt Chart

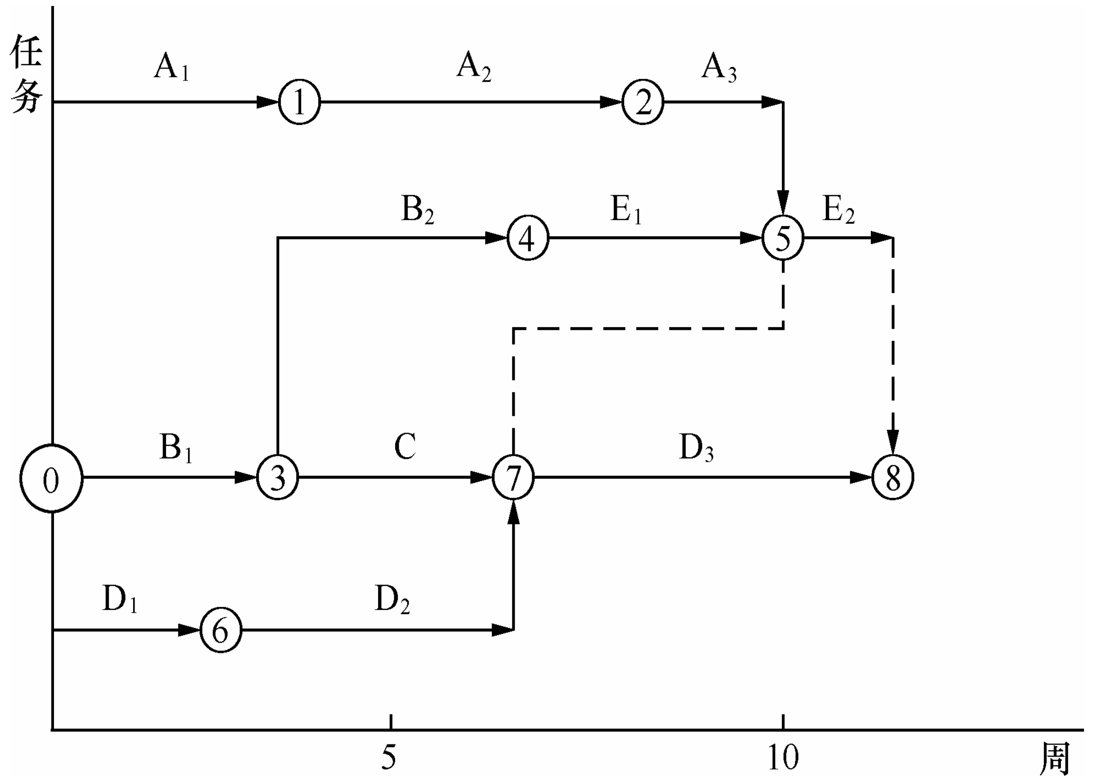
北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY





时标网状图（time scalar network）

为克服甘特图的缺点，形成了时标网状图。任务以有向线段表示，其指向点表示任务间的衔接点，并且都给予编号，可以显示出各子任务间的依赖关系。



PERT图

计划评审技术（program evaluation and review technique，PERT）也称网络图方法，或简称PERT图方法，它的另一名称是关键路径法（critical path method，CPM）。

- PERT图中以有向箭头作为边表示子任务，它有名称、长度（即完成此项子任务所需的时间）；
- 以有编号的圆圈作为结点，它应该是子任务向量的始发点或指向点；
- 由若干条边和若干个结点构成了网状图，于是可以沿相互衔接的子任务形成的路径，进行路径长度的计算、比较和分析，从而实现项目工期的控制。

实例

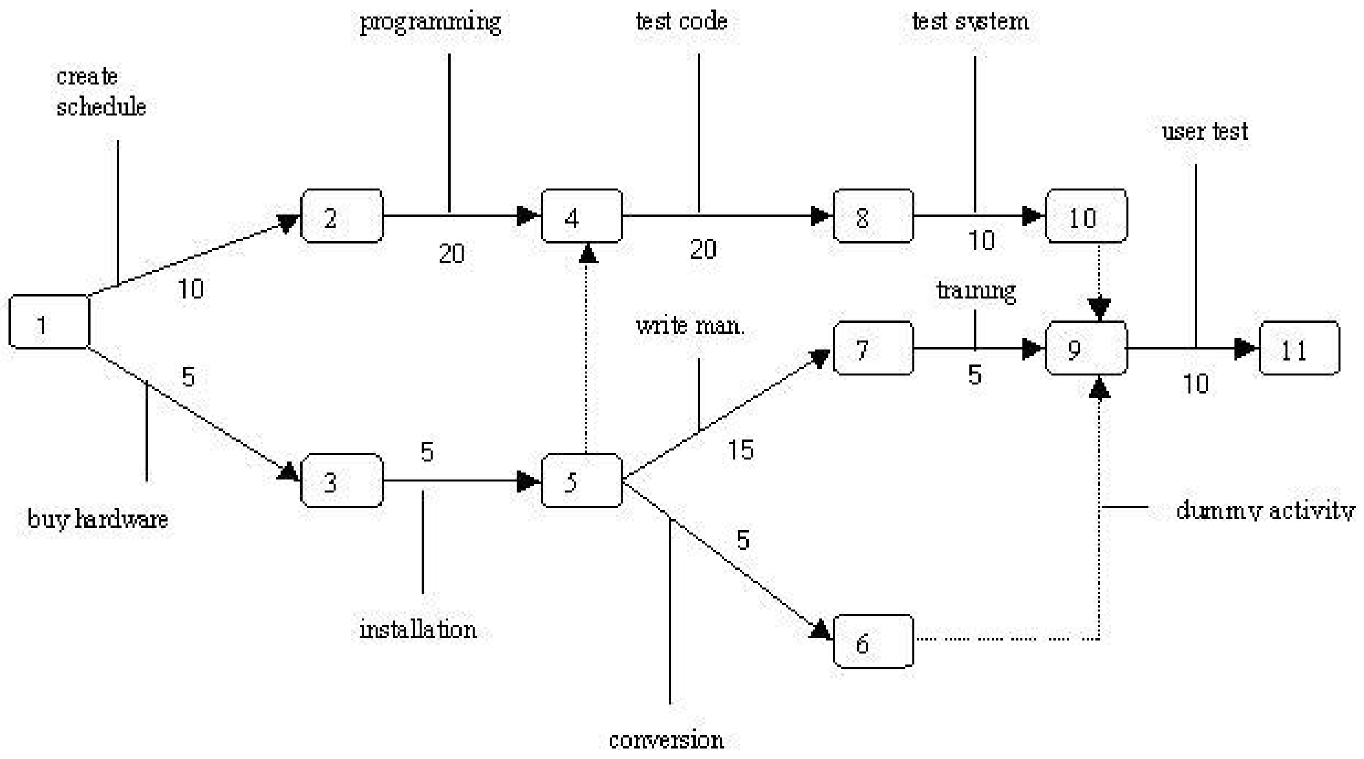
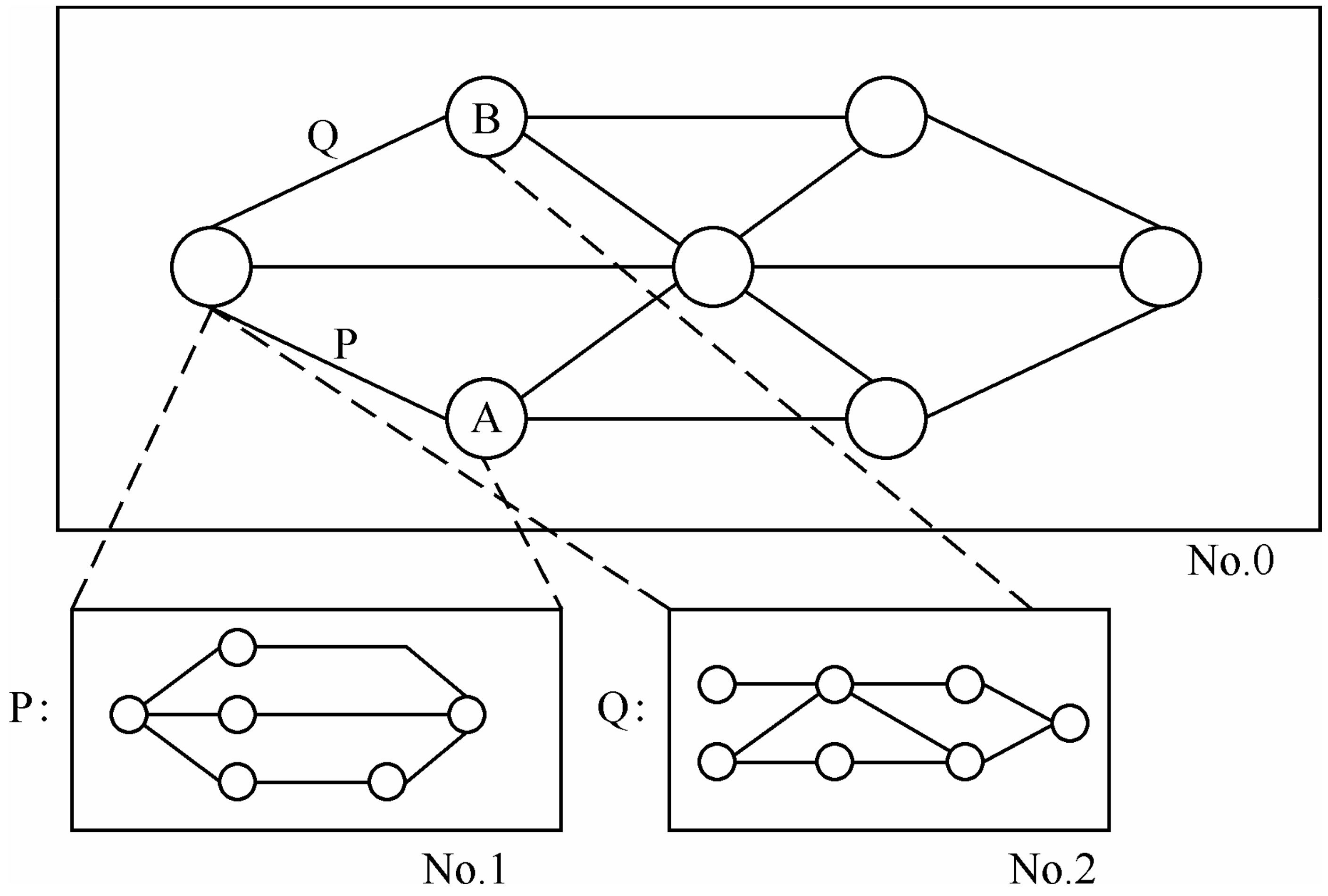


Fig. 1:
PERT Chart

* Numbered rectangles are nodes and represent events or milestones.
* Directional arrows represent dependent tasks that must be completed sequentially.
* Diverging arrow directions (e.g. 1-2 & 1-3) indicate possibly concurrent tasks
* Dotted lines indicate dependent tasks that do not require resources.

分层PERT图



4. Software Configuration Management: WHY?

- The problem:
  - Multiple people have to work on software that is ***changing***
  - ***More than one version*** of the software has to be supported:
    - System(s) under development
    - Custom configured systems (different functionality)
    - Released systems
  - Software must run on different machines and OS

➘ ***Need for Coordination***

- Software Configuration Management (SCM)
  - manages evolving software systems
  - controls the costs involved in making changes to a system

What is SCM?

- “*SCM is the control of the evolution of complex systems,…, for the purpose to contribute to satisfying quality and delay constraints.*”

  – Jacky Estublier

- “*SCM provides the capabilities of identification, control, status accounting, audit and review, manufacture, process management, and teamwork.*”

  – Susan Dart

What is SCM? (cont'd)

- CM is a key process in Capability Maturity Model
  - **Level 1-Initial**: ad hoc/chaotic
  - **Level 2-Repeatable**: basic PM and documentation
  - **Level 3-Defined**: standard and complete process control and procedures
  - **Level 4-Managed**: predictable process performance and precise measurements
  - **Level 5-Optimizing**: continuous and recursive improvement to performance

- CM operates through the software life cycle

What is NOT SCM?

- Not just version control
- Not just for source code management
- Not only for development phase
- Selecting and using tools are important, but design and management of CM process are more crucial for project success

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

Some Simple CM Scenarios

- Developer A wants to see latest version of foo.c and its change history since last week
- B needs to revert foo-design.doc to its version two days ago
- B makes a release of the project and he needs to know what items to include and which version
- A lives in New Dehli, India and B lives in Boston, US, they want to work on HelloWorld.java together
- In the latest release, a serious bug is found and manager C wants to track what changes caused the bug, who made those changes and when
- C wants to get reports about current project progress to decide if she needs to hire more programmers and delay the alpha release

SCM Roles

- Configuration Manager
  - identify configuration items
  - define the procedures for creating promotions and releases
- Change control board member
  - approving or rejecting change requests
- Developer
  - Creates promotions triggered by change requests or the normal activities of development.
  - checks in changes and resolves conflicts
- Auditor
  - selection and evaluation of promotions for release and for ensuring the consistency and completeness of this release

Terminology

- Configuration Item (CI)
- Version, Variant, and Revision
- Configuration
- Baseline
- Workspace (Repository)

Configuration Item (CI)

- *"An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process."*

- Not only program code segments but all type of documents according to development, e.g
  - all type of code files
  - drivers for tests
  - analysis or design documents
  - user or developer manuals
  - system configurations (e.g. version of compiler used)

- In some systems, not only software but also hardware configuration items (CPUs, bus speed frequencies) exist!

Finding CIs

- Large projects typically produce thousands of entities (files, documents, data ...) which must be uniquely identified.
- Any entity can potentially be brought under configuration management control
- But not all the time.

- Two Issues:
  - What: Selection of Configuration Items
    - What should be under configuration control?
  - When: to start to place entities under configuration control?

- Conflict for the Project Manager:
  - Starting with CIs too early introduces too much bureaucracy
  - Starting with CIs too late introduces chaos

Finding CIs (cont’d)

- Some items must be maintained for the lifetime of the software.
- Sometimes after the software is no longer developed but still in use;
- Who expects proper support for lots of years.
- An entity naming scheme should be defined

Which of these Entities should be CIs?

- Problem Statement
- Software Project Management Plan (SPMP)
- Requirements Analysis Document (RAD)
- System Design Document (SDD)
- Project Agreement
- Object Design Document (ODD)
- Dynamic Model
- Object model
- Functional Model
- Unit tests
- Integration test strategy
- Source code
- API Specification
- Input data and data bases
- Test plan
- Test data
- Support software (part of the product)
- Support software (not part of the product)
- User manual
- Administrator manual

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

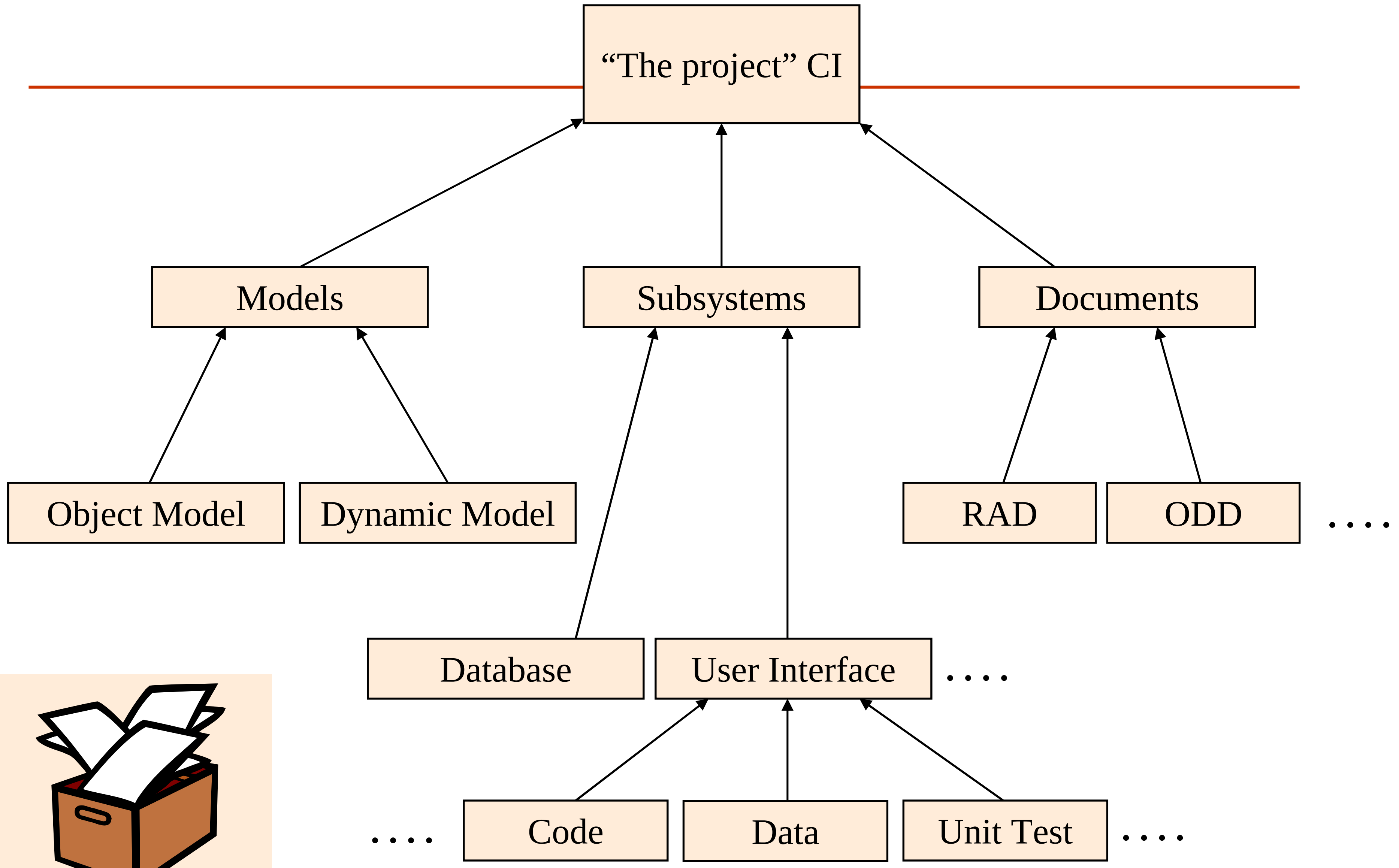
Possible Selection of CIs

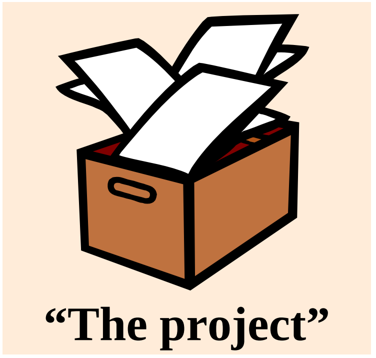
- Problem Statement
- Software Project Management Plan (SPMP)
- Requirements Analysis Document (RAD)
- S
  (
- Project Agreement
- Object Design Document (ODD)
- Dynamic Model
- Object model
- Functional Model
- Unit tests
- Integration test strategy

- Source code
- API Specification
- Input data and databases
- Test plan
- Support software (part of the product)
- Support software (not part of the product)
- User manual
- Administrator manual

Once the Configuration Items are selected, they are usually organized in a tree

CI Tree (Example)



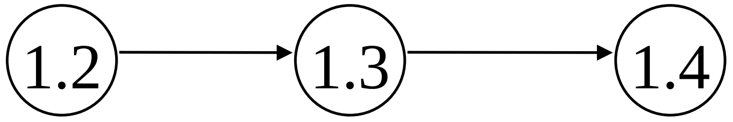


Version, Variant, and Revision

- **Version**: a CI at one point in its development, includes revision and variant

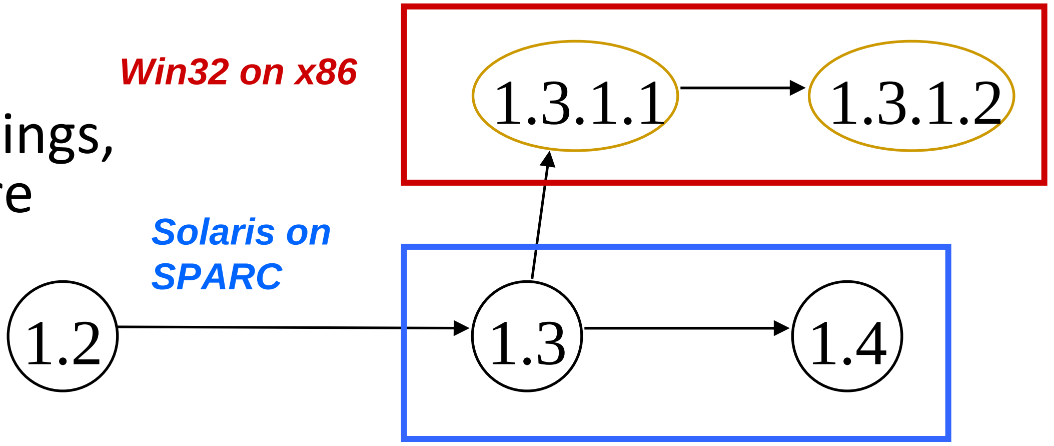
Question: Is Windows 7 a new version or a new revision compared to Windows XP?

- **Revision**: a CI linked to another via revision-of relationship, and ordered in time


$$\begin{tikzcd} 1.2 \arrow[r] & 1.3 \arrow[r] & 1.4 \end{tikzcd}$$

- **Variant**: functionally equivalent versions, but designed for different settings, e.g. hardware and software

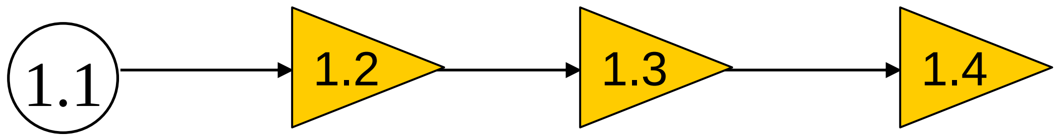
- **Branch**: a sequence of versions in the time line



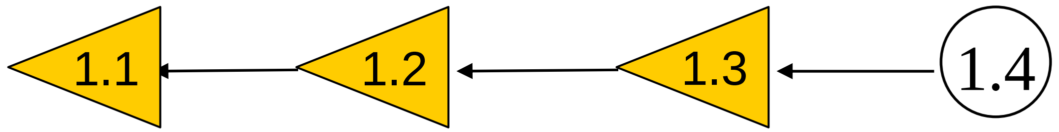
$$\begin{tikzcd} & 1.3.1.1 \arrow[r] & 1.3.1.2 \\ 1.2 \arrow[r] & 1.3 \arrow[u] \arrow[r] & 1.4 \end{tikzcd}$$

How Versions are Stored

- Full copy of each version
- **Delta** (differences between two versions)
- Forward delta



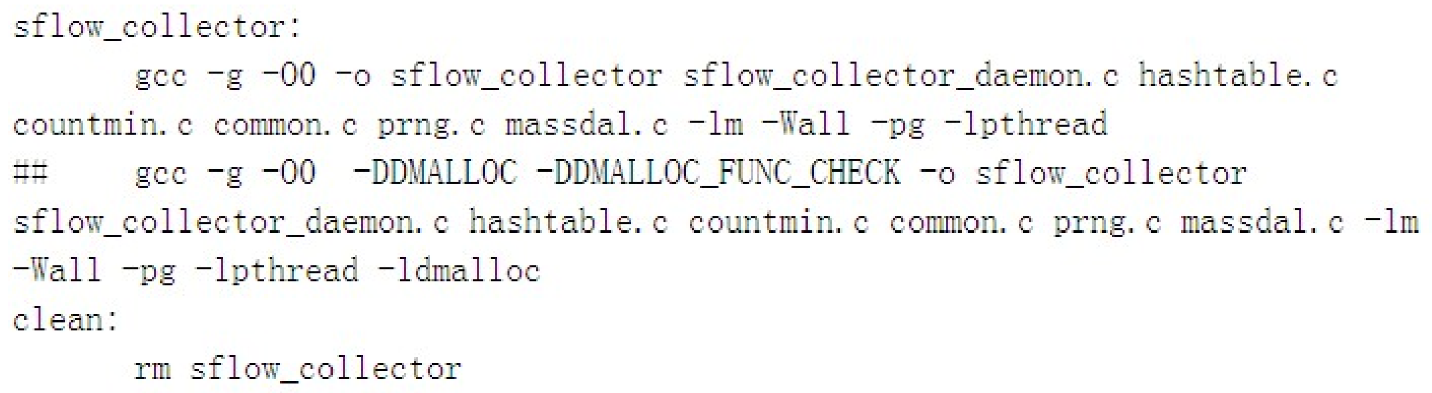
- Reverse delta



- Mixed delta

Configuration

- An arrangement of functional CIs according to their nature, version and other characteristics
- Guaranteed to recreate configurations with quality and functional assurance
- Sometimes, configuration needs to record environment details, e.g. compiler version, library version, hardware platform, etc.
- Simple examples
  - Ant buildfile, Makefile



```
sflow_collector:
	gcc -g -O0 -o sflow_collector sflow_collector_daemon.c hashtable.c
countmin.c common.c prng.c massdal.c -lm -Wall -pg -lpthread
##	gcc -g -O0  -DDMALLOC -DDMALLOC_FUNC_CHECK -o sflow_collector
sflow_collector_daemon.c hashtable.c countmin.c common.c prng.c massdal.c -lm
-Wall -pg -lpthread -ldmalloc
clean:
	rm sflow_collector
```

Baseline

- A collection of item versions that have been formally reviewed and agreed on, a version of configuration
- Marks milestones and serves as basis for further development
- Can only be changed via formal change management process
- Baseline + change sets to create new baselines

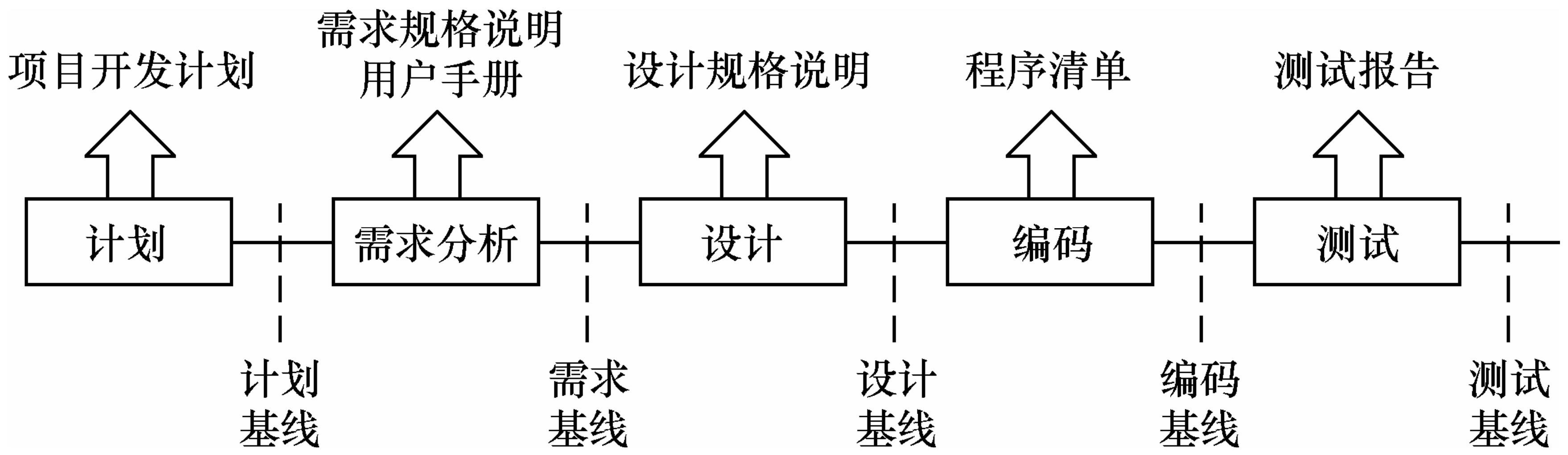
Examples:

Baseline A: All t**he API have completely been defined; the bodies of the methods are empty**.

Baseline B: **All data access methods are implemented and tested**.

Baseline C: **The GUI is implemented.**

Example


项目开发计划
需求规格说明
用户手册
设计规格说明
程序清单
测试报告
计划
需求分析
设计
编码
测试
计划
基线
需求
基线
设计
基线
编码
基线
测试
基线

Workspace

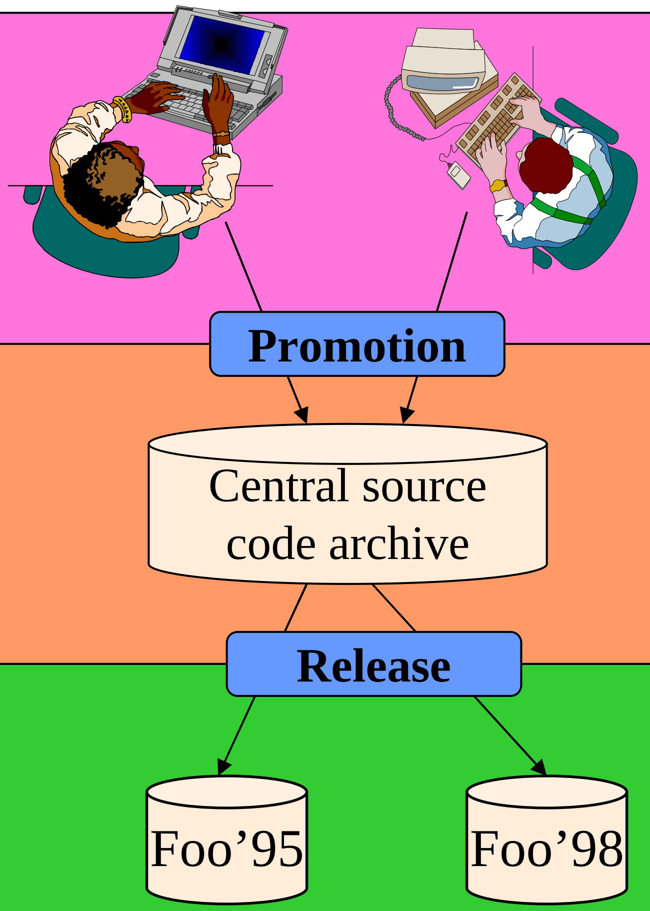
- An isolated environment where a developer can work (edit, change, compile, test) without interfering other developers

- Examples
  - Local directory under version control
  - Private workspace on the server

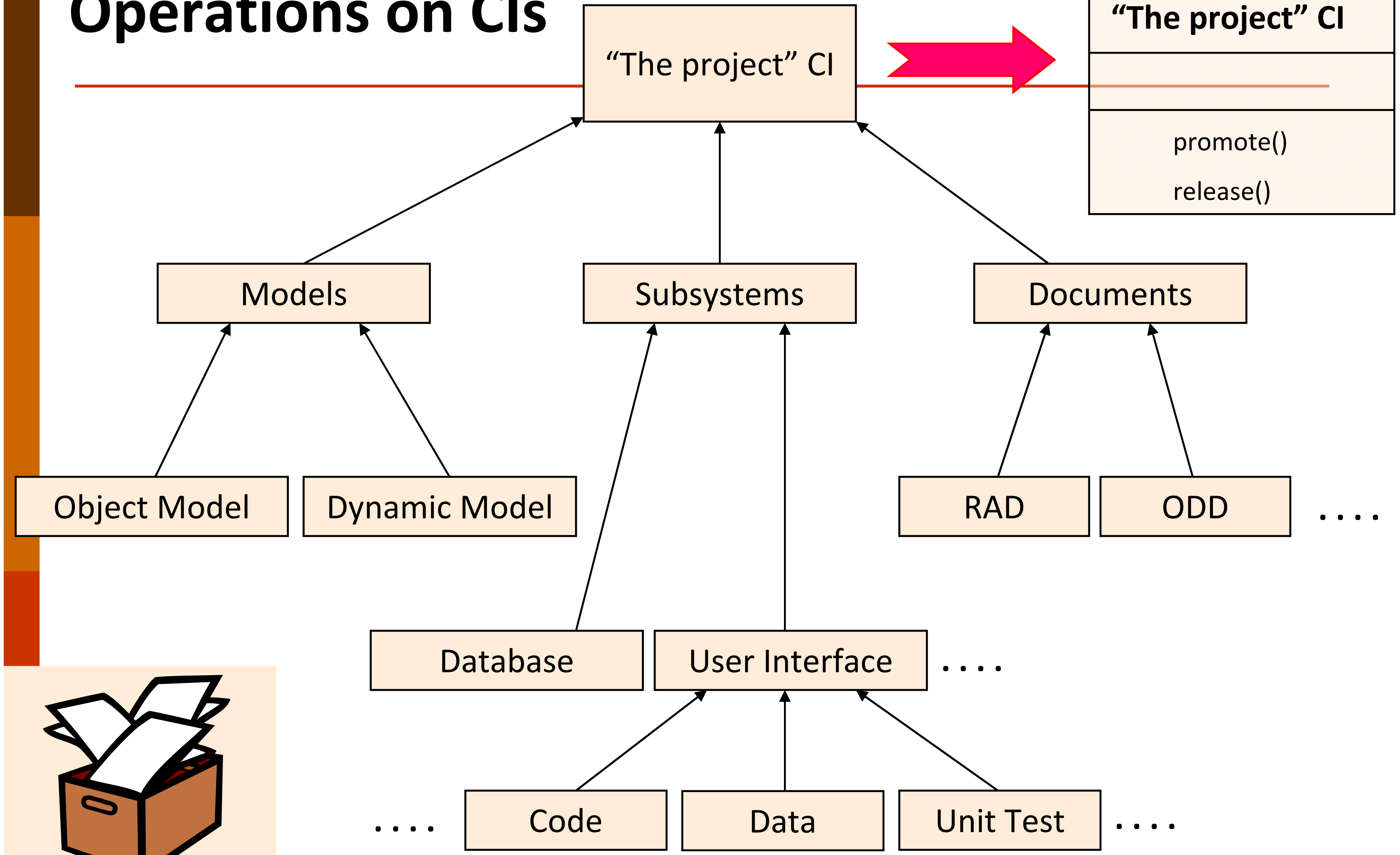
- Common Operations
  - Import: put resources into version control in repository
  - Update: get latest version on the default branch
  - Checkout: get a version into workspace
  - Checkin: commit changes to the repository

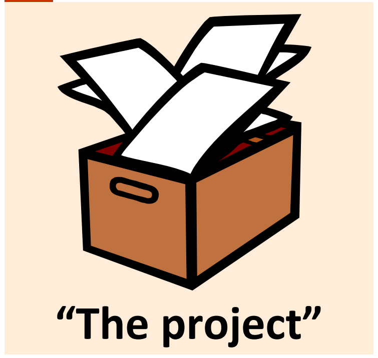
Standard SCM Directories

- Programmer's Directory
  - (IEEE Std: "Dynamic Library")
  - Completely under control of one programmer.
- Master Directory
  - (IEEE Std: "Controlled Library")
  - Central directory of all promotions.
- Software Repository
  - (IEEE Std: "Static Library")
  - Externally released baselines.



Promotion and Release are Operations on CIs



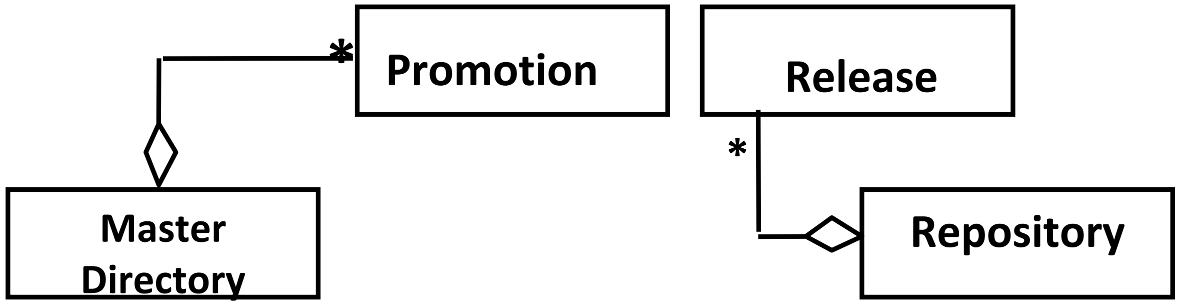


Let‘s Create a Model for Configuration Management

- We just learned that promotions are stored in the master directory and releases are stored in the repository

Problem: There can be many promotions and many releases

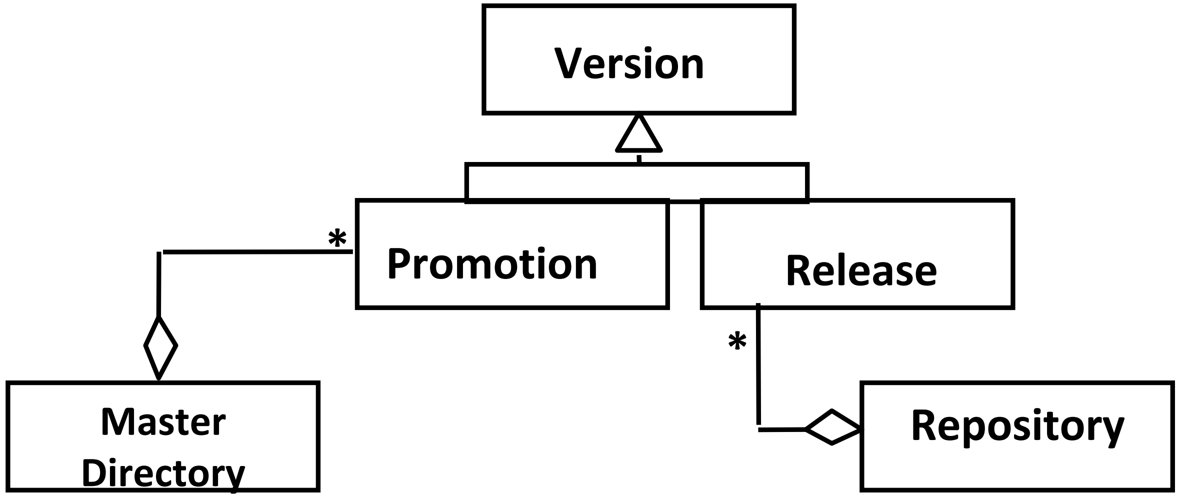
Solution: Use Multiplicity



Let‘s Create a Model for Configuration Management

- Insight: Promotions and Releases are both versions

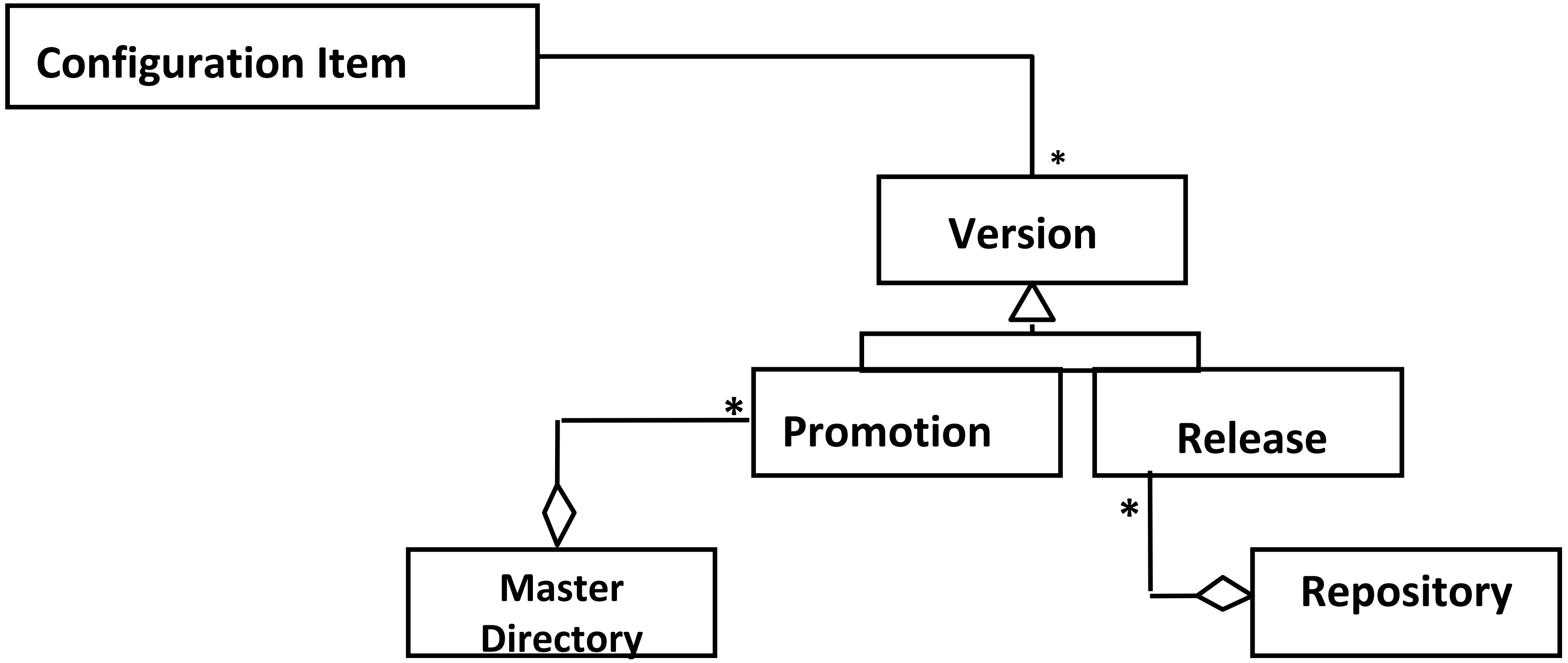
Solution: Use Inheritance



Let's Create a Model for Configuration Management

- Problem: A configuration item has many versions

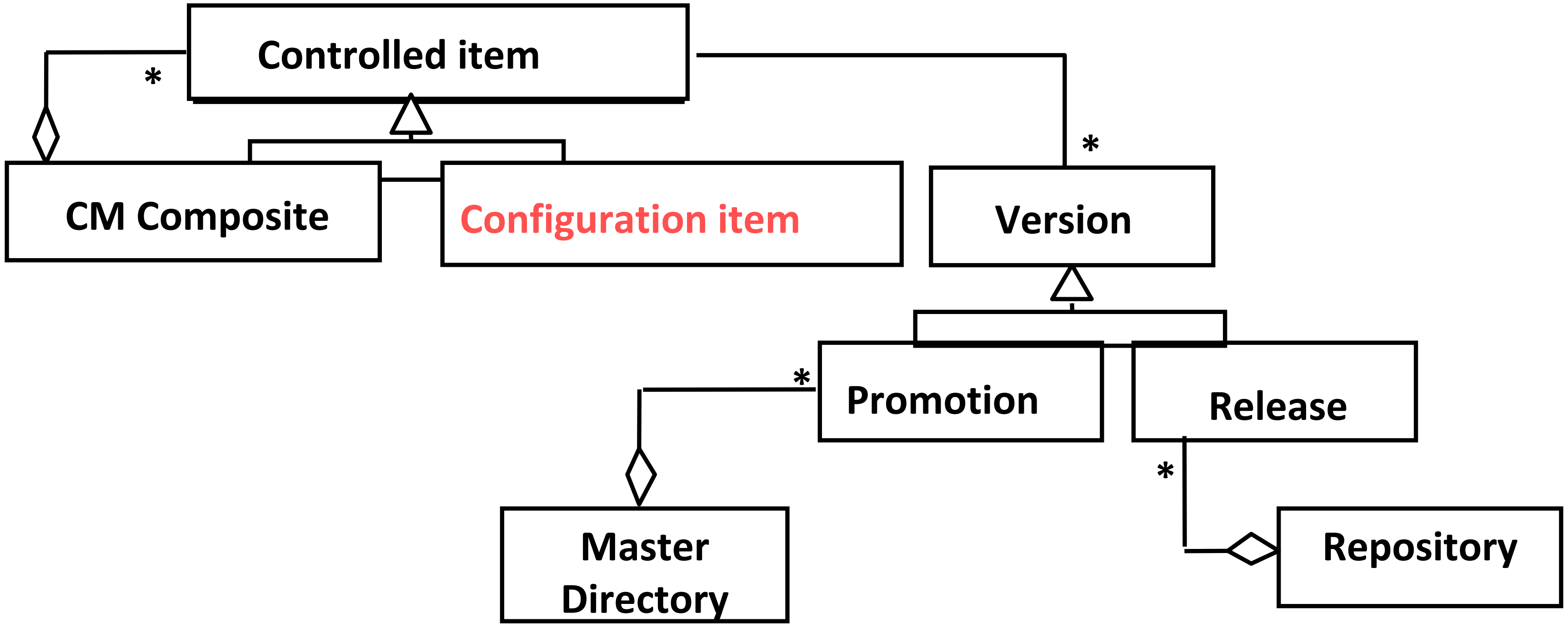
Solution: Create a 1-many association between Configuration Item and Version



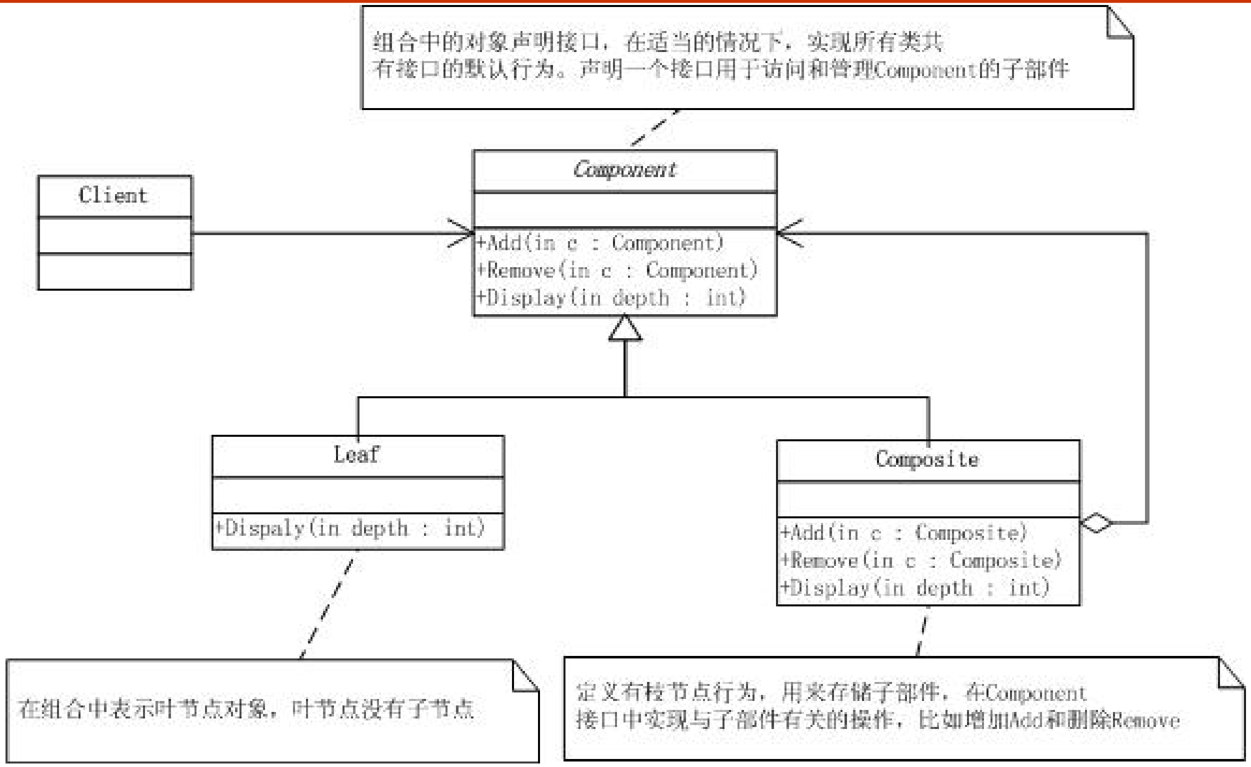
Let‘s Create a Model for Configuration Management

- Problem: Configuration items can themselves be grouped

  Solution: Use ***Composite Design Pattern***



Revisit: Composite Design Pattern


组合中的对象声明接口，在适当的情况下，实现所有类共
有接口的默认行为。声明一个接口用于访问和管理Component的子部件
Component
Client
+Add(in c : Component)
+Remove(in c : Component)
+Display(in depth : int)
Leaf
+Dispaly(in depth : int)
Composite
+Add(in c : Composite)
+Remove(in c : Composite)
+Display(in depth : int)
在组合中表示叶节点对象，叶节点没有子节点
定义有枝节点行为，用来存储子部件，在Component
接口中实现与子部件有关的操作，比如增加Add和删除Remove

- Component：组合中的对象声明接口，在适当情况下实现所有类共有的默认行为
- Leaf: 在组合中表示叶节点，叶节点没有子节点，定义对象的基本行为。
- Composite：定义有子部件的那些部件的行为，存储子部件并在Component接口实现与子部件有关的操作。
- Client:通过Component接口操作组合部件的对象。

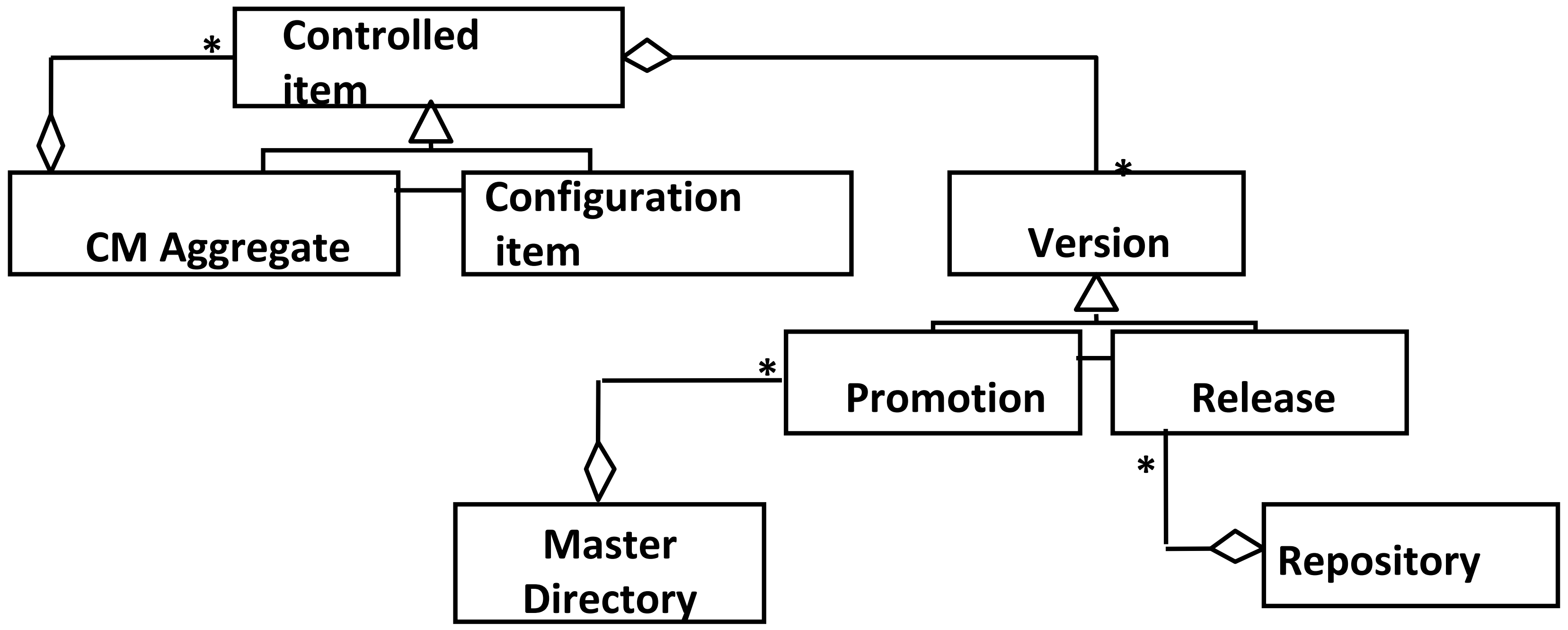
```
public abstract class Component
{
    protected string name;
    public Component(string name)
    {          this.name = name;}

    public abstract void Add(Component c);
    public abstract void Remove(Component c)
    public abstract void Display(int depth);
}
```

```
public class Leaf : Component
{
    public Leaf(string name) : base(name) { }

    public override void Add(Component c)
    {  Console.WriteLine("Cannot add to a leaf"); }

    public override void Remove(Component c)
    { Console.WriteLine("Cannot remove from a leaf"); }

    public override void Display(int depth)
    { Console.WriteLine(new String('-', depth) + name); }
}
```

```
public class Composite : Component
 {
    private List<Component> children = new Li

    public Composite(string name) : base(nam

    public override void Add(Component c)
    {  children.Add(c);}

    public override void Remove(Component c)
    { children.Remove(c);}

    public override void Display(int depth)
    {
        Console.WriteLine(new String('-', depth) + name);

        foreach (Component component in children)
        { component.Display(depth + 2);}
    }   }
```

```
public class Program
{
    static void Main(string[] args)
    {
        Composite root = new Composite("root");
        root.Add(new Leaf("Leaf A"));
        root.Add(new Leaf("Leaf B"));

        Composite comp = new Composite("Composite X");
        comp.Add(new Leaf("Leaf XA"));
        comp.Add(new Leaf("Leaf XB"));

        root.Add(comp);

        Composite comp2 = new Composite("Composite XY");
        comp2.Add(new Leaf("Leaf XYA"));
        comp2.Add(new Leaf("Leaf XYB"));

        comp.Add(comp2);

        root.Add(new Leaf("Leaf C"));

        Leaf leaf = new Leaf("Leaf D");
        root.Add(leaf);
        root.Remove(leaf);

        root.Display(1);
        Console.Read();
    }   }
```

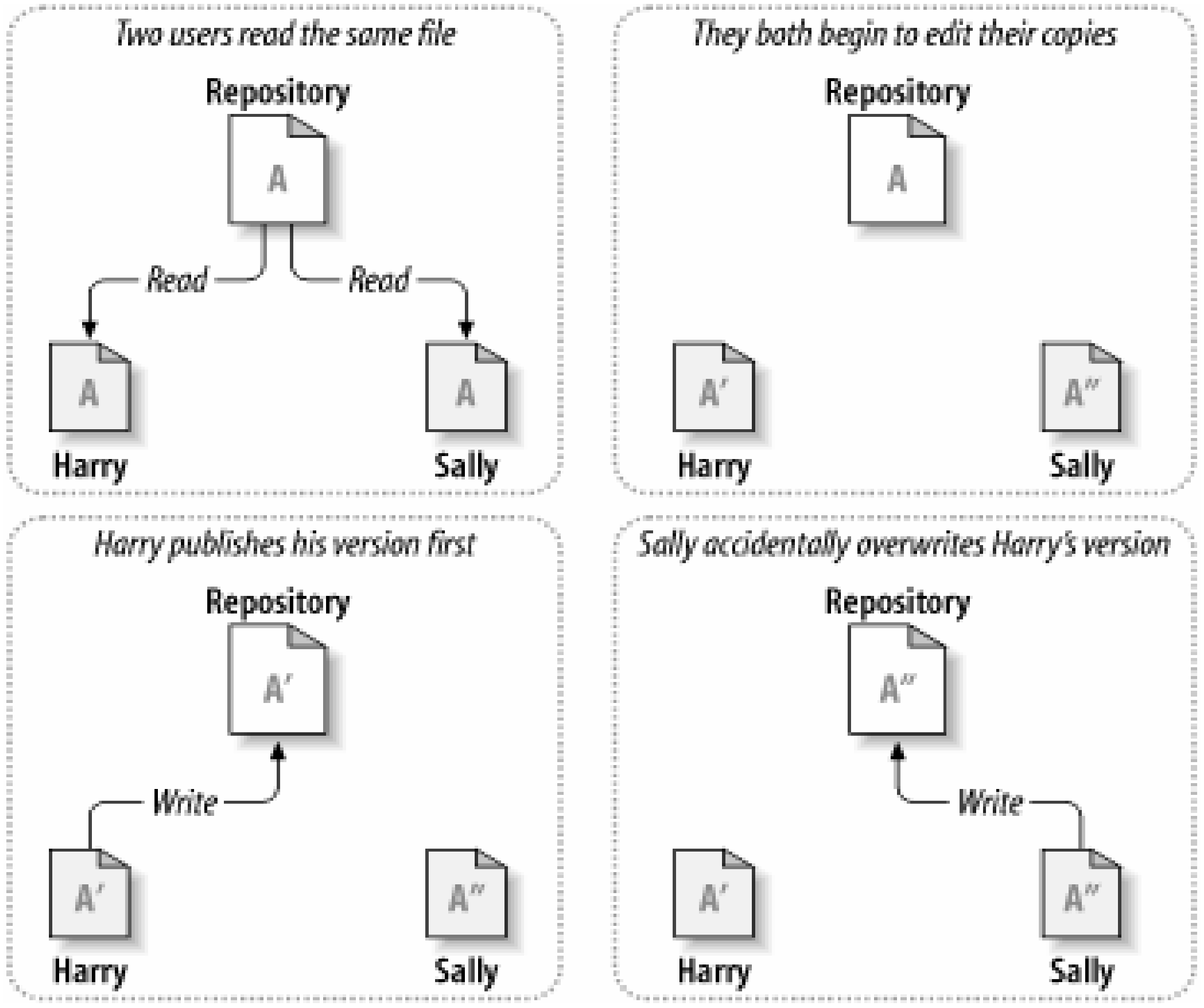
CI Model (UML Class Diagram)



北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

Version Control Models (1/3)

- Basic problem of collaborative work

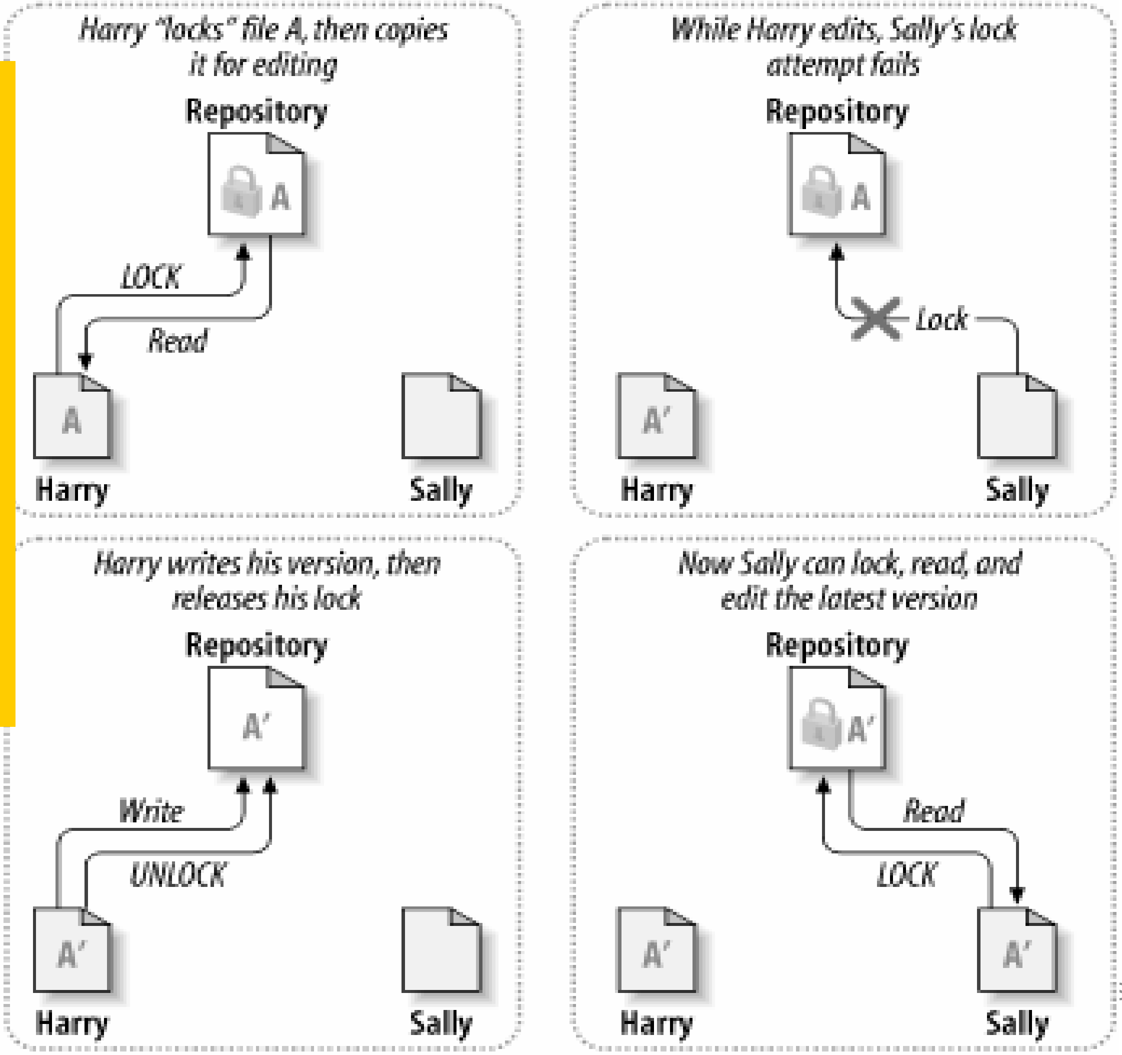

Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

Version Control Models (2/3)

- Model 1-Pessimistic: lock-modify-unlock

Problems:

- Forget to unlock
- Parallel work not possible
- Deadlock



Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
A
Harry
Sally
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
A'
Harry
Sally
Harry writes his version, then releases his lock
Repository
A'
Write
UNLOCK
A'
Harry
Sally
Now Sally can lock, read, and edit the latest version
Repository
A'
Read
LOCK
A'
Harry
A'
Sally

svn-book

Version Control Models (3/3)

Model 2-Optimistic: copy-modify-merge

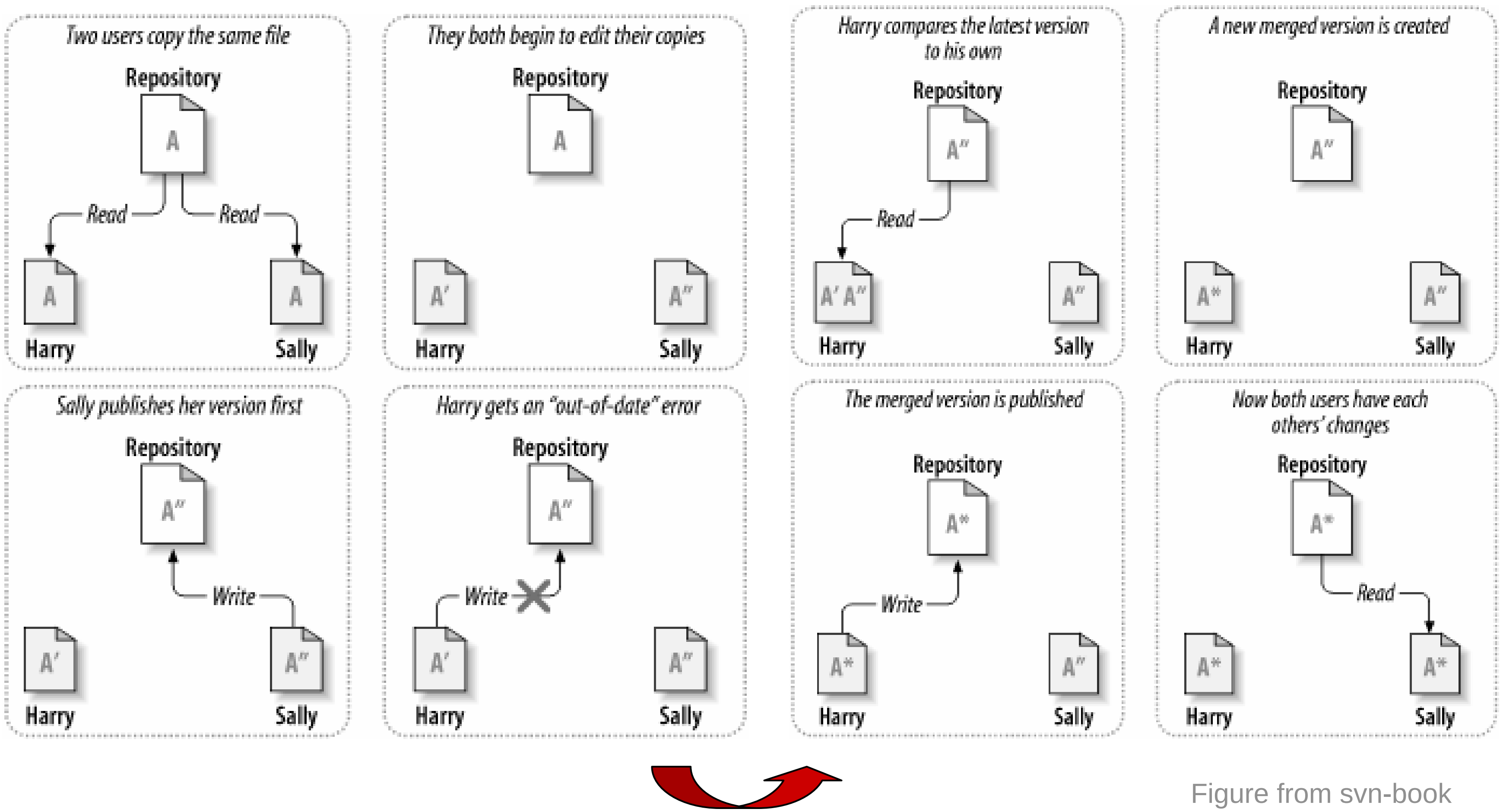

Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry compares the latest version to his own
Repository
A″
Read
A′ A″
A″
Harry
Sally
A new merged version is created
Repository
A″
A*
A″
Harry
Sally
Sally publishes her version first
Repository
A″
Write
A′
A″
Harry
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
A″
Harry
Sally
The merged version is published
Repository
A*
Write
A*
A″
Harry
Sally
Now both users have each others' changes
Repository
A*
Read
A*
A*
Harry
Sally

Figure from svn-book

SCM Processes

- Change control process
- Status accounting
- Configuration audit
- Release management
- CM planning

Change Control Process

- Submission of Change Request (CR)
- Technical and business evaluation and impact analysis
- Approval by Change Control Board (CCB)
- Engineering Change Order (ECO) is generated stating
  - changes to be made
  - criteria for reviewing the changed CI
- CI’s checked out
- Changes made and reviewed
- CI’s checked in

Status Accounting

- Administrative tracking and reporting of CIs in CM system

- Examples
  - Status of proposed changes
  - Status of approved changes
  - Progress of current version, on or behind schedule
  - Estimate of resources to finish one task
  - bugs identified by configuration audit

Configuration Audit

- Independent review or examination to assess if a product or process is in compliance with specification, standards, contractual agreement, or other criteria

- Examples
  - Verifies that CIs are tested to satisfy functional requirements
  - Verifies that baseline contains necessary and correct CI versions
  - Ensures that changes made to a baseline comply with the configuration status report

Release Management

- Creation and availability of a new version of software to the public
- Release format
  - Source code + build script + instructions
  - Executables packaged for specific platforms
  - Other portable formats: Java Web Start, plugins
  - Patches and updates: automatic, manual
- Release content
  - Source and/or binary, data files, installation scripts, libraries, user and/or developer documentation, feedback programs, etc.

Software CM Planning

- Software configuration management planning starts during the early phases of a project.

- The outcome of the SCM planning phase is the

  Software Configuration Management Plan (SCMP)

  which might be extended or revised during the rest of the project.

- Either follow a public standard like the IEEE 828, or an internal (e.g. company specific) standard.

SCMP

- Defines the *types of documents* to be managed and a document naming scheme.
- Defines *who takes responsibility* for the CM procedures and creation of baselines.
- Defines *policies for change* control and version management.
- Describes the *tools* which should be used to assist the CM process and any limitations on their use.
- Defines the *configuration management database* used to record configuration information.

Outline of a SCMP by IEEE 828-1990

1. Introduction
   - Describes purpose, scope of application, key terms and references
2. Management (WHO?)
   - Identifies the responsibilities and authorities for accomplishing the planned configuration management activities
3. Activities (WHAT?)
   - Identifies the activities to be performed in applying to the project.
4. Schedule (WHEN?)
   - Establishes the sequence and coordination of the SCM activities with project mile stones.
5. Resources (HOW?)
   - Identifies tools and techniques required for the implementation of the SCMP
6. Maintenance
   - Identifies activities and responsibilities on how the SCMP will be kept current during the life-cycle of the project.

SCMP Section 1: Introduction

- Simplified overview of the CM activities.
- Scope:
  - Overview description of the project
  - Identification of the CI(s) to which software CM will be applied.
- Identification of other software to be included as part of the SCMP (support software and test software)
- Relationship of SCM to hardware of system CM activities
- Degree of formality and depth of control for applying SCM to project.
- Limitations and time constraints for applying SCM to this project
- Assumptions that might have an impact on the cost, schedule and ability to perform defined SCM activities.

SCMP Section 2: Management

- Organization
  - Organizational context (technical and managerial) within which the SCM activities are implemented. Identifies
    - All organizational units (client, developers, managers) that participate in a SCM activity
    - Functional roles of these people within the project
    - Relationship between organizational units
- Responsibilities
  - For each SCM activity list the name or job title to perform this activity
  - For each board performing SCM activities, list
    - purpose and objectives
    - membership and affiliations
    - period of effectivity, scope of authority
    - operational procedures
- Applicable Policies
  - External constraints placed on the SCMP

SCMP Section 3: Activities

3.1 Configuration Identification

3.2 Configuration Control

3.3 Configuration Status Accounting

3.4 Configuration Audits and Reviews

3.5 Interface Control

3.2 Configuration Control

Defines the following steps

- 3.2.1 How to identify the need for a change (layout of change request form)
- 3.2.2 Analysis and evaluation of a change request (CR)
- 3.2.3 Approval or disapproval of a request
- 3.2.4 Verification, implementation and release of a change

3.2.1 Change Request

- Specifies the procedures for requesting a change to a baselined CI and the information to be documented:
  - Name(s) and version(s) of the CI(s) where the problem appears
  - Originator’s name and address
  - Date of request
  - Indication of urgency
  - The need for the change
  - Description of the requested change

3.2.2 Evaluation of a Change

- Specifies the analysis required to determine the impact of proposed changes and the procedure for reviewing the results of the analysis.

3.2.3 Change Approval or Disapproval

- This section of the SCMP describes the organization of the configuration control board (CCB).
- Configuration Control Board (CCB)
  - Can be an individual or a group.
  - Multiple levels of CCBs are also possible, depending on the complexity of the project
- Multiple levels of CCBs may be specified.
  - In small development efforts one CCB level is sufficient.
- This section of the SCMP also indicates the level of authority of the CCB and its responsibility.
  - In particular, the SCMP must specify when the CCB is invoked.

3.2.4 Implementing Change

- Specifies the activities for verifying and implementing an approved change.
- A completed change request must contain:
  - The original change request(s)
  - The names and versions of the affected configuration items
  - Verification date and responsible party
  - Identifier of the new version
  - Release or installation date and responsible party
- This section must also specify activities for
  - Archiving completed change requests
  - Planning and control of releases
  - How to coordinate multiple changes
  - How to add new CIs to the configuration
  - How to deliver a new baseline

3.3 Configuration Status Accounting

- What elements are to be tracked and reported for baselines and changes?
- What types of status accounting reports are to be generated? What is their frequency?
- How is information to be collected, stored and reported?
- How is access to the configuration management status data controlled?

3.4 Configuration Audits and Reviews

- Identifies audits and reviews for the project.
  - An audit determines for each CI if it has the required physical and functional characteristics.
  - A review is a management tool for establishing a baseline.

- For each audit or review the plan has to define:
  - Objective
  - The Configuration Items under review
  - The schedule for the review
  - Procedures for conducting the review
  - Participants by job title
  - Required documentation
  - Procedure for recording deficiencies and how to correct them
  - Approval criteria

Summary of SCMP

- Standards
  - IEEE Std 828 (SCM Plans), ANSI-IEEE Std 1042 (SCM), etc.

- CM plan components
  - What will be managed (list and organize CIs)
  - Who will be responsible for what activities (roles and tasks)
  - How to make it happen (design processes for change requests, task dispatching, monitoring, testing, release, etc.)
  - What records to keep (logs, notes, configurations, changes, etc.)
  - What resources and how many (tools, money, manpower, etc.)
  - What metrics to measure progress and success

CM Tools

- Version control
  - RCS, CVS, Subversion, Visual Source Safe, Rational ClearCase
- Bug tracking
  - Bugzilla, Mantis Bugtracker, Rational ClearQuest
- Build
  - GNU Make and many variants, Ant
- Project management
  - Sourceforge.net, freshmeat.net, GForge, DForge

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

Reference and Further Readings

Reference

- *Introduction to Configuration Management*, lecture slides for COMP3100/3500, Ian Barnes, the Australian National University.
- *Software Configuration Management*, Center for Development of Advanced Computing, Mumbai at Juhu, India.
- *Concepts in Configuration Management Systems*, Susan Dart, CMU.
- *Software Configuration Management: A Roadmap*, Jacky Estublier, CNRS, France.

Further Reading

- *Software Engineering, a Practitioner's Approach (6th), part 4*, Roger Pressman.
- *Code Complete (2nd)*, Steve McConnel.
- *http://cmcrossroads.com/*
- *Implementing and Integrating PDM and SCM*, Ivica Crnkovic et al.
- *Version Control with Subversion*, Ben Collins-Sussman et al.