

软件工程

第14章 软件项目管理



面向规模的度量

- 软件规模通常是指软件的大小(*size*)，一般用代码行度量
 - 优点：方便、直观
 - 缺点：很大程度上取决于程序设计语言以及软件设计的质量
- 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

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- 项目进度管理
- 风险管理
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- 小结

内容摘要

- **软件项目管理概述**
- 软件度量
- 软件项目估算
- 项目进度管理
- 风险管理
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- 小结

软件项目管理

- 项目管理是通过项目经理和项目组织的努力，运用系统理论的方法对项目及其资源进行计划、组织、协调、控制，旨在实现项目的特定目标的管理方法体系
- (软件)项目管理的基本内容：
项目定义、项目计划、项目执行、项目控制、项目结束

软件项目管理的关注点(4P)

- **人员(People)**

- 人员是软件工程项目的基本要素和关键因素
- 在对人员进行组织时，有必要考虑参与软件过程(及每一个软件项目)的人员类型

- **产品(Product)**

- 定义项目范围，其中包括建立产品的目的和范围、可选的解决方案、技术或管理的约束等

- **过程(Process)**

- 通常将项目分解为任务—子任务等，其分解准则是基于软件工程的过程

- **项目(Project)**

- 采用科学的方法及工具对项目基本内容进行管理

软件项目管理中的五类人员

- **项目管理人员**
 - 负责软件项目的管理工作，其负责人通常称为项目经理
- **高级管理人员**
 - 可以是领域专家，负责提出项目的目标并对业务问题进行定义
- **开发人员**
 - 掌握了开发一个产品或应用所需的专业技术，可胜任包括需求分析、设计、编码、测试、发布等各种相关的开发岗位
- **客户**
 - 一组可说明待开发软件的需求的人，也包括与项目目标有关的其它风险承担者
- **最终用户**
 - 产品或应用提交后与产品/应用进行交互的

软件项目管理中的产品

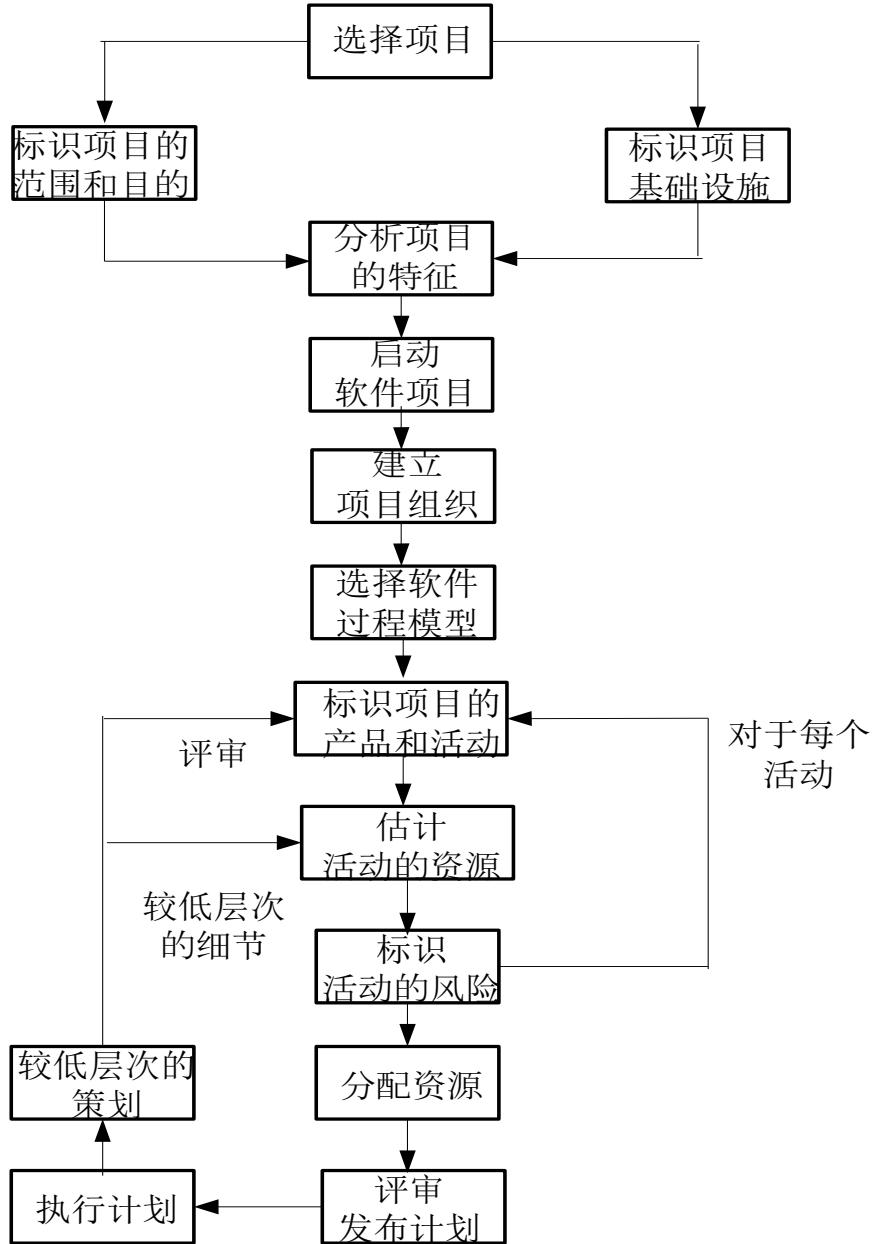
- 定义项目范围，其中包括建立产品的目的和范围、可选的解决方案、技术或管理的约束
- 目的：从客户的角度定义该产品的总体目标，但不必考虑这些目标如何实现
- 软件范围定义了与软件产品相关的数据、功能和行为，及其相关的约束：
 - 语境(context)：说明待建造的软件与其它相关系统、产品或环境的关系，以及相关的约束条件
 - 信息目标：说明目标系统所需要的输入数据及应产生的输出数据
 - 功能和性能：说明软件应提供的功能来完成输入数据到输出数据的变换以及给出对目标软件的性能要求

软件项目方法

• 对项目进行有计划和可控制的管理

- 明确目标及过程：充分理解被解决的问题，明确定义项目目标及软件范围，为项目小组及活动设置明确、现实的目标，并充分发挥相关小组的自主性
- 保持动力：提供激励措施使人员变动最小
- 跟踪进展：对每个任务的进展进行跟踪，并对其软件过程和质量进行度量
- 做出聪明的决策：项目管理者和软件小组的决策应该“保持其简单”
- 项目总结：从每个完成的项目中获取可学习的经验

软件项目管理过 程示例



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提高产品的质量，
- **度量的作用是为了有效地采用定量的方式来进行管理**
- **管理人员利用度量来了解软件工程过程的执行情况和产品质量**
- **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

项目估算

- **项目估算**是制定项目计划的基础
 - 项目所需的人力(以人月为单位)、项目持续时间(以年份或月份为单位)、成本(以元为单位)等
- 参照以前类似项目中的相关数据进行估算
 - 若存在类似历史项目则可进行类比估算
 - 若缺少可类比的项目数据则采用特定的估算技术(例如功能点估算方法等)
- 通常采用多种估算技术进行交叉检查

风险管理

- 风险：人员、经费、进度及需求等方面存在的可能影响项目按计划完成的不确定因素
- 风险管理：标识软件项目中的风险，预测风险发生的概率以及风险造成的影响，并对风险进行评估，找出那些可能导致项目失败的风险，然后采取相应的措施来缓解风险
- 风险管理贯彻于整个软件工程过程中

进度安排

- **进度安排**
 - 将项目划分成可管理的子项目、任务和活动
 - 确定任务之间的依赖关系，找出影响项目按期完成的关键任务
 - 为每个任务分配时间、工作量以及指定责任人，定义每个任务的输出结果及其关联的里程碑
- **在项目实施过程中将在进度计划基础上跟踪实际执行情况，从而及时发现偏差并采取措施加以调整以确保项目按期完成**

跟踪与控制

- 跟踪是控制的前提，它实际上是在项目实施过程中对影响项目进展的内外部因素进行及时的、连续的、系统的记录和报告的活动，其核心在于反映项目变化、提供相关信息的报告
- 控制是通过工具和技术对项目计划与实际执行进行对比，并对项目的未来走向进行预测，再此基础上进行项目的各种调整

软件配置管理

- Software Configuration Management(SCM)
- 任务：标识和确定系统中的配置项，在系统整个生命期内控制这些项的发布和变更，记录并报告配置的状态和变更要求，验证配置项的完整性和正确性
- SCM存在于整个软件过程中，是一种保护性活动

内容摘要

- 软件项目管理概述
- **软件度量**
- 软件项目估算
- 项目进度管理
- 风险管理
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- 小结

术语定义(ISO/IEC 9126-1) 《信息技术 软件产品评价 质量特性及其使用指南》

- Metric(度量) : 定义测量方法和测量标度
- Measurement(测量) : 使用一种度量把标度值(可以是数字或类别)赋予实体的某个属性
- Measure(verb 测量) : 执行一次测量(measurement)
- Measure(noun 测度) : 通过执行一次测量赋予实体属性的数字或类别
- direct measure(直接测量) : 不依赖于任何其它属性的测量导出的属性测量
- indirect measure(间接测量) : 从一个或多个其它属性的测量导出的属性测量
- internal measure(内部测量) : 一种对产品本身的直接或间接的测量
- external measure(外部测量) : 一种通过对外系统的测量导出对产品(作为系统的一部分)的间接测量

软件度量

- 度量对象：软件产品、软件过程、资源
 - 外部属性：面向管理者和用户的属性
 - 体现了软件产品/软件过程与相关资源和环境的关系，如成本、效益、开发人员的生产率
 - 通常可采用直接测量的办法进行
 - 内部属性：软件产品或过程本身的属性
 - 如软件产品的结构、模块化程度、复杂性、程序长度等
 - 有些内部属性只能用间接测量的方法度量，需要特定的测量方法或模型

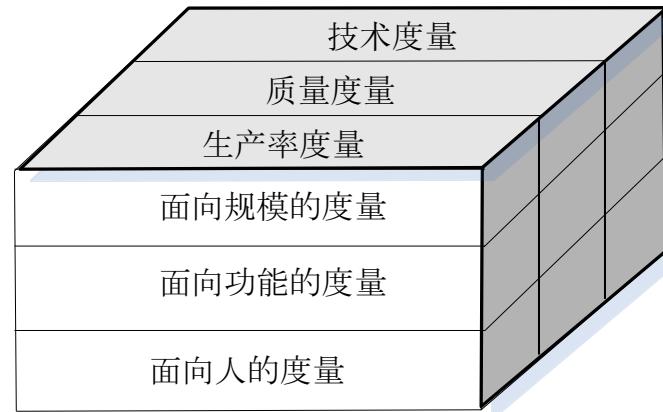
软件度量分类

• 分类1：

- 面向规模的度量用于收集与直接度量有关的软件工程输出信息和质量信息
- 面向功能的度量则集中在程序的“功能性”和“实用性”
- 面向人的度量则收集有关人们开发计算机软件所用方式的信息和人员理解有关工具的方法和效率的信息

• 分类2：

- 软件生产率度量集中在软件工程过程的输出
- 软件质量度量可指明软件满足明确的和隐含的用户需求的程度
- 技术度量主要集中在软件产品的某些特征(如逻辑复杂性、模块化程度)上，而不是软件开发的全过程



面向规模的度量

- 软件规模通常是指软件的大小(*size*)，一般用代码行度量
 - 优点：方便、直观
 - 缺点：很大程度上取决于程序设计语言以及软件设计的质量
- 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

面向功能的度量

- 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CT “总计数值”
 - 计算复杂度调整值F
 - 计算功能点FP

CT的计算方法

测量参数	计数	加权因子			=	=
		简单	平均	复杂		
用户输入数	<input type="text"/>	×	3	4	6	<input type="text"/>
用户输出数	<input type="text"/>	×	4	5	7	<input type="text"/>
用户查询数	<input type="text"/>	×	3	4	6	<input type="text"/>
文件数	<input type="text"/>	×	7	10	15	<input type="text"/>
外部接口数	<input type="text"/>	×	5	7	10	<input type="text"/>
总计数值	<hr/>			→	<input type="text"/>	

计算复杂度调整值

复杂度调整值 F_i ($i=1$ 到 14)是基于对左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，见右表

	问 题	F_i (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		

值	定义
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

计算功能点FP

- 功能点计算公式 $FP = CT * (0.65 + 0.01 * F)$
- 其中：CT是“总计数值”，F是 F_i 之和
- 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

扩展的功能点度量

- 功能点度量的不足：最初主要用于商业信息系统的度量，强调数据维，即信息域特征值，而忽略了功能和行为（控制）
- Jones提出了称为特征点(Feature Point)的扩展的功能点度量方法
 - 在功能点信息域特征中增加了一个算法特征，并将算法定义为“特定计算机程序中所包含的一个界定的计算问题”

测量参数	计数	加权因子	结果
用户输入数		×4	
用户输出数		×5	
用户查询数		×4	
文件数		×7	
外部接口数		×7	
算法		×3	
总计CT			

功能点与LOC的换算(部分)

程序语言	每FP之LOC值			
	平均	中等	低	高
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Java	63	53	77	70
COBOL	77	77	14	400
SQL	40	37	7	110
VBScript	34	27	50	—
Visual Basic	47	42	16	158

软件质量

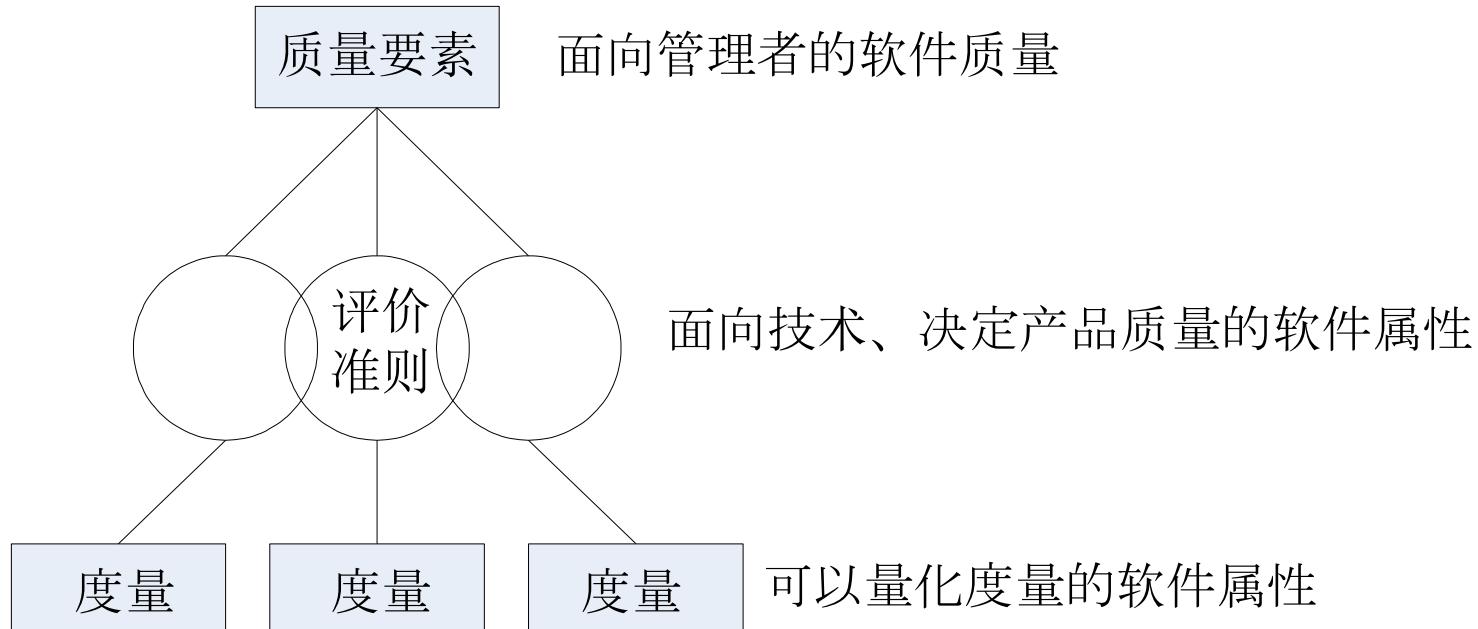
- **软件质量定义**

- ISO/IEC 9126：与软件产品满足明确或隐含需求的能力有关的特征和特性的总和
- GB/T 13423：软件产品中能满足给定需求的性质和特性的总和，例如符合规格说明的程度；软件具有所期望的各种属性的组合程度；客户或用户觉得软件满足其综合期望的程度；软件的综合特性，它确定软件在使用中将满足客户预期要求的程度

- **典型的软件质量模型：McCall模型、Boehm模型和ISO/IEC 9126质量模型**

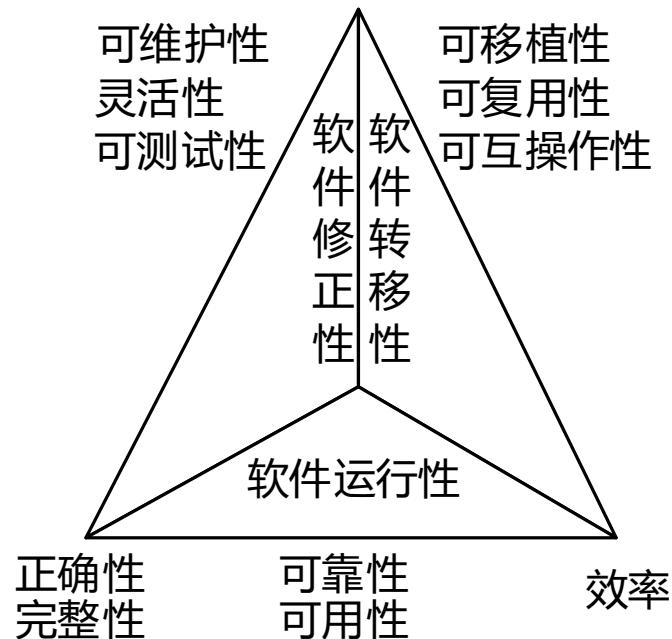
McCall模型

**质量要素反映软件的质量，决定产品品质
量的软件属性用作评价准则，量化的度
量体系可测量软件质量属性的优劣**



McCall软件质量要素

- 软件产品的运行、修改和迁移三个方面
- 11个软件质量要素



McCall软件质量要素定义

- 正确性(correctness)：一个程序满足它的需求规约和实现客户任务目标的程度
- 可靠性(reliability)：一个程序期望以所需的精确度完成它的预期功能的程度
- 效率(efficiency)：一个程序完成其功能所需的计算资源和代码的数量
- 完整性(integrity)：对未授权人员访问软件或数据的可控制程度
- 可用性(usability)：学习、操作、准备输入和解释程序输出所需的工作量
- 可维护性(maintainability)：定位和修复程序中一个错误所需的工作量
- 灵活性(flexibility)：修改一个运作的程序所需的工作量
- 可测试性(testability)：测试一个程序以确保它完成所期望的功能所需的工作量
- 可移植性(portability)：把一个程序从一个硬件和/或软件系统环境移植到另一个所需的工作量
- 可复用性(reusability)：一个程序(或一个程序的部分)可以在另外一个应用程序中复用的程度，与程序完成的功能的包装和范围相关
- 可互操作性(interoperability)：连接一个系统和另一个系统所需的工作量。

质量要素之间的关系

其中 Δ 表示正相关， \blacktriangledown 表示负相关

关 系 要 素	要 素	正 确 性	可 靠 性	效 率	完 整 性	可 用 性	可 维 护 性	可 测 试 性	灵 活 性	可 移 植 性	可 复 用 性	可 互 操 作 性
正确性												
可靠性		Δ										
效率												
完整性				\blacktriangledown								
可用性	Δ	Δ	\blacktriangledown	Δ								
可维护性	Δ	Δ	\blacktriangledown		Δ							
可测试性	Δ	Δ	\blacktriangledown		Δ	Δ						
灵活性	Δ	Δ	\blacktriangledown	\blacktriangledown	Δ	Δ	Δ					
可移植性			\blacktriangledown			Δ	Δ					
可复用性		\blacktriangledown	\blacktriangledown	\blacktriangledown		Δ	Δ	Δ	Δ	Δ		
可互操作性			\blacktriangledown	\blacktriangledown						Δ		

软件质量属性

- 软件质量要素难以直接测量，因此需要为每个质量要素定义一组软件质量属性用作质量要素的评价准则，要求
 - 能够完整、准确地描述软件质量要素
 - 容易量化和测量
- McCall定义了21种软件质量属性

软件质量要素评价准则 - 1

- (1) 可审计性(auditability)
和标准的符合性可被检查的容易程度。
- (2) 准确性(accuracy)
计算和控制的准确度。
- (3) 通信共性(communication commonality)
标准接口、协议和带宽的使用程度。
- (4) 完备性(completeness)
所需功能完全实现的程度。
- (5) 简洁性(conciseness)
以代码行数来评价的程序的简洁程度。
- (6) 一致性(consistency)
在软件开发项目中一致的设计和文档技术的使用。
- (7) 数据共性(data commonality)
在整个程序中对标准数据结构和类型的使用。

软件质量要素评价准则-2

- (8)容错性(error tolerance)
当程序遇到错误时所造成的损失。
- (9)执行效率(execution efficiency)
一个程序的运行性能。
- (10)可扩展性(expandability)
结构、数据或过程设计可被扩展的程度。
- (11)通用性(generality)
程序构件潜在的应用宽度。
- (12)硬件独立性(hardware independence)
软件独立于其运行于之上的硬件的程度。
- (13)自检测性(instrumentation)
程序监视它自身操作并且标识产生的错误的程度。
- (14)模块性(modularity)
程序部件的功能独立性。

软件质量要素评价准则 - 3

- (15)可操作性(operability)
程序操作的容易度。
- (16)安全性(security)
控制和保护程序和数据的机制的可用度。
- (17)自文档性(self-documentation)
源代码提供有意义的文档程度。
- (18)简单性(simplicity)
一个程序可以没有困难地被理解的程度。
- (19)软件系统独立性(software system independence)
程序独立于非标准编程特性、操作系统特性和其他环境限制的程度。
- (20)可追踪性(traceability)
从一个设计表示或实际程序部件跟踪到需求的能力。
- (21)易培训性(training)
软件支持使得新用户使用系统的能力。

质量要素与评价准则的关系

量化的度量

- 处于软件质量度量模型的最底层是
- 定义了每个质量属性(评价准则)的可量化的度量指标
- 通过对这些指标的测量(可以是主观的，也可以是客观的)和加权计算得到质量属性的测量值
- 在McCall的模型中未给出具体的度量指标，度量者可根据不同的软件类型定义不同的度量指标体系

质量要素值的计算

在计算质量要素值之前，首先要将质量属性的测量值归一化，即将其变换到0到1范围内的实数

假设： F_j 是第j个质量要素， M_k 是第k个质量属性（评价准则）， C_{jk} 是 M_k 在 F_j 中的加权系数。那么， F_j 可用下列公式计算：

$$F_j = \sum_{k=1}^{21} C_{jk} M_k$$

其中： $1 \leq j \leq 11$ $1 \leq k \leq 21$ $0 \leq M_k \leq 1$ $\sum_{k=1}^{21} C_{jk} = 1$ $C_{jk} \geq 0$

当 $C_{jk} = 0$ 时表示第j个质量要素与第k个质量属性无关

ISO/IEC 9126质量模型

- 由质量特性、子特性和度量三个层次组成
- 第一层有6个质量特性
- 第二层有21个质量子特性
- 第三层是由度量者定义的可定量化度量指标

ISO/IEC 9126的6个质量特性

- 功能性(functionality)：与一组功能及其指定的性质有关的一组属性
- 可靠性(reliability)：与在规定的一段时间和条件下，软件维护其性能水平的能力有关的一组属性
- 易用性(usability)：与一组规定或潜在的用户为使用软件所需作的努力和对这样的使用所作的评价有关的一组属性
- 效率(efficiency)：与在规定的条件下，软件的性能水平与所使用资源之间有关的一组属性
- 可维护性(maintainability)：与进行指定的修改所需的努力有关的一组属性
- 可移植性(portability)：与软件可从某一个环境移植到另一个环境的能力有关的一组属性

ISO/IEC 9126质量子特性-功能性

- **适合性(suitability)**：与规定任务能否提供一组功能以及这组功能的适合程度有关的软件属性
- **准确性(accuracy)**：与能否得到正确或相符的结果或效率有关的软件属性
- **互操作性(interoperability)**：与同其它指定系统进行交互的能力有关的软件属性
- **依从性(compliance)**：使软件遵循有关的标准、约定、法规及类似规定的软件属性
- **安全性(security)**：与防止对程序及数据的非授权的故意或意外访问的能力有关的软件属性

ISO/IEC 9126质量子特性-可靠性

- 成熟性(maturity)：与由软件故障引起失效的频率有关的软件属性
- 容错性(fault tolerance)：与在软件故障或违反指定接口的情况下，维持规定的性能水平的能力有关的软件属性
- 易恢复性(recoverability)：与在失效发生后，重建其性能水平并恢复直接受影响数据的能力以及为达此目的所需的时间和努力有关的软件属性

ISO/IEC 9126质量子特性-易用性

- 易理解性(understandability) : 与用户为认识逻辑概念及其应用范围所花的努力有关的软件属性
- 易学性(learnability) : 与用户为学习软件应用(例如运行控制、输入、输出)所花的努力有关的软件属性
- 易操作性(operability) : 与用户为操作和运行控制所花努力有关的软件属性

ISO/IEC 9126质量子特性-效率

- **时间特性(time behaviour)**：与软件执行其功能时响应和处理时间以及吞吐量有关的软件属性
- **资源特性(resource behaviour)**：与在软件执行其功能时所使用的资源数量及其使用时间有关的软件属性

ISO/IEC 9126质量子特性- 可维护性

- 易分析性(analysability)：与为诊断缺陷或失效原因及为判定待修改的部分所需努力有关的软件属性
- 易改变性(changeability)：与进行修改、排除错误或适应环境变化所需努力有关的软件属性
- 稳定性(stability)：与修改所造成的一次性未预料结果的风险有关的软件属性
- 易测试性(testability)：与确认已修改软件所需的努力有关的软件属性

ISO/IEC 9126质量子特性-可移植性

- **适应性(adaptability)**：与软件无需采用有别于为该软件准备的活动或手段就可能适应不同的规定环境有关的软件属性
- **易安装性(installability)**：与在指定环境下安装软件所需努力有关的软件属性
- **遵循性(conformance)**：使软件遵循与可移植性有关的标准或约定的软件属性
- **易替换性(replaceability)**：与软件在该软件环境中用来替代指定的其他软件的机会和努力有关的软件属性

程序复杂性度量

- 软件复杂性是指理解和处理软件的难易程度，包括程序复杂性和文档复杂性，主要体现在程序复杂性中
- 程序复杂性的6个方面
 - 程序理解的难度
 - 纠错、维护程序的难度
 - 向他人解释程序的难度
 - 按指定方法修改程序的难度
 - 根据设计文件编写程序的工作量
 - 执行程序时需要资源的程度
- 典型的程序复杂性度量：McCabe环形复杂性度量、Halstead的复杂性度量

程序复杂性度量的基本原则

- 1. 程序复杂性与程序大小的关系不是线性的
- 2. 控制结构复杂的程序较复杂
- 3. 数据结构复杂的程序较复杂
- 4. 转向语句使用不当的程序较复杂
- 5. 循环结构比选择结构复杂，选择结构又比顺序结构复杂
- 6. 语句、数据、子程序和模块在程序中的次序对复杂性有影响
- 7. 全局变量、非局部变量较多时，程序较复杂
- 8. 参数按地址调用比按值调用复杂
- 9. 函数副作用比显式参数传递难以理解
- 10. 具有不同作用的变量共用一个名字时较难理解
- 11. 模块间、过程间联系密切的程序比较复杂
- 12. 嵌套深度越深，程序越复杂

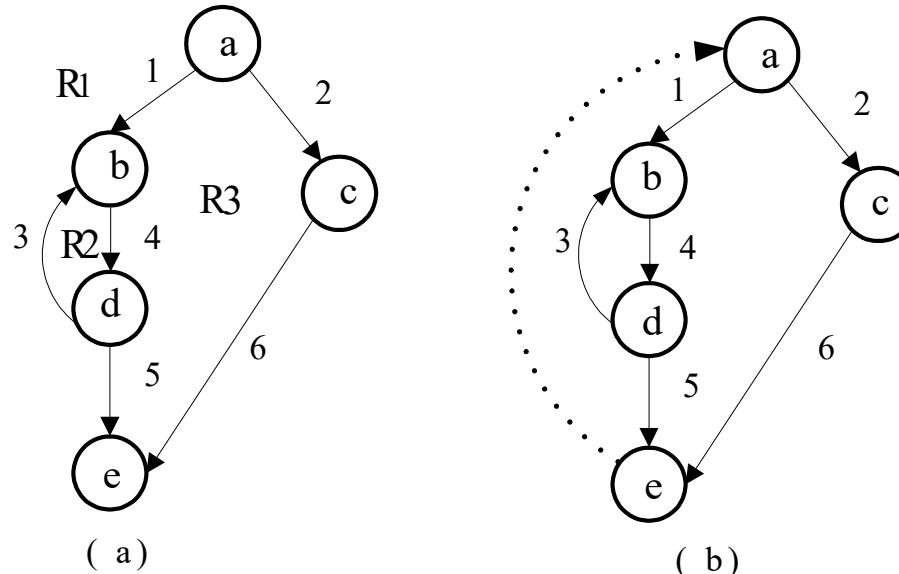
McCabe环形复杂度量

- **一种基于程序图的程序复杂度量方法**
 - 程序图：是一种退化的程序流程图，它将程序流程图中的每个处理符号(包括处理框、判断框、起点、终点等)退化成一个结点(若干个连续的处理框可合并成一个结点)，流程图中连接处理符号的控制流变成程序图中连接结点的有向弧
- **建立在图论的基础之上**
 - 对于一个强连通的有向图 G ，若 e 是图中的弧数， n 是图中的结点数， p 是强连通分量的个数，则图 G 的环数计算公式为：

$$V(G) = e - n + p$$

程序控制结构图的扩充

- 一个单入口和单出口的程序(或模块)的程序图是连通的，但通常不是强连通的
- 为此在程序图中增加一条从出口结点到入口结点的弧，使程序图变成强连通(连通分量只有一个，即 $P=1$)
 - 下图中，当增加了出口结点到入口结点的弧后成为图b后： $e=7$ 、 $n=5$ 、 $V(G)=7-5+1=3$
 - 为了简化环形复杂性的计算，我们通常用下列公式直接对图a进行计算： $V(G)=e-n+2$ ，此时， $e=6$ ， $n=5$ ， $V(G)=6-5+2=3$



环形复杂度量的含义

- 环形复杂度量反映了程序(或模块)控制结构的复杂性
- McCabe发现 $V(G)=10$ 是一个实际模块的上限，当模块的环复杂度超过10时，要充分测试这个模块变得特别难

软件可靠性度量

- 软件可靠性是指在规定的条件下和规定的时间内软件按规格说明要求不引起系统失效的概率
- 它是软件质量的一项重要指标，特别是对于一些实时系统、嵌入式系统和关键系统
- 软件可靠性通常用下列公式进行计算：
 - $MTBF = MTTF + MTTR$
 - 其中：MTBF(mean time between failer)是平均故障间隔时间，MTTF(mean time to failer)是平均故障时间，MTTR(mean time to repair)是平均修复时间
- 软件可用性(availability)是指软件在投入使用时能实现其指定的系统功能的概率。可用下式计算：

$$\frac{MTTF}{MTTF + MTTR} \times 100\%$$

内容摘要

- 软件项目管理概述
- 软件度量
- **软件项目估算**
- 项目进度管理
- 风险管理
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- 小结

软件项目估算

- 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。典型的经验估算模型有IBM估算模型、CoCoMo模型和Putnam模型。
- 上述方法可以组合使用以提高估算的精度

一种简单有效的估算方法

1. 请若干名有经验的技术人员或管理人员，采用上述估算办法的一种或多种，分别估算出代码行LOC或功能点FP的乐观值 a_i ，悲观值 b_i 及最有可能的值 m_i
2. 计算出平均值 a ， b ， m
3. LOC或FP的规模估算值： $e = (a + 4m + b)/6$
4. 根据以前该组织软件开发的平均生产率(规模/人月数)和平均成本(资金/规模)计算工作量估算值和成本估算值
 - 工作量估算值 = e / 平均生产率
 - 成本估算值 = e^* 平均成本

IBM估算模型

- 基于代码行LOC的静态单变量模型：

设L为源代码行数(KLOC) , 则

工作量E=5.2×L^{0.91}人月

项目持续时间D=4.1×L^{0.36}=14.47×E^{0.35}

人员数S=0.54×E^{0.6}

文档数量DOC = 49×L^{1.01}

- 一条机器指令为一行源代码 , 不包括程序注释及其它说明
- 非机器指令编写的程序应转换成机器指令代码行数来考虑 , 转换关系为 :

语言	转换系数
简单汇编	1
宏汇编	1.2~1.5
FORTRAN	4~6
PL/I	4~10

CoCoMo模型

- Boehm提出的“构造性成本模型”
Constructive Cost Model, CoCoMo
- 按详细程度分：基本模型、中间模型和详细模型
- 将软件项目类型划分为三类：
 - 组织型
 - 半独立型
 - 嵌入型
- 用于估算E (Effort , 人天、人月) 、 L (KLOC)

基本CoCoMo模型

$$E = aL^b \quad D = cE^d \quad \text{其中} :$$

E表示工作量，单位是人月

D表示开发时间，单位是月

L是项目的源代码行估计值，单位是千行代码

a、b、c、d是常数，其取值如下表所示

项目类型	a	b	c	d
组织型	2.4	1.05	2.5	0.38
半独立型	3.0	1.12	2.5	0.35
嵌入型	3.6	1.20	2.5	0.32

中间CoCoMo模型

- 在基本CoCoMo (*Constructive Cost Model*)模型基础上考虑了**15**种影响软件工作量的因素
- 通过工作量调节因子(EAF)修正对工作量的估算，从而使估算更合理
- 公式如下： $E=a(L)^b EAF$
 - 其中：L是软件产品的目标代码行数，单位是千行代码数，
a、b是常数，取值如下表所示

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	1.20

工作量调节因子的计算

- 每个调节因子 F_i 的取值分为很低、低、正常、高、很高、极高六级，正常情况下 $F_i=1$
- 当15个 F_i 选定，可得： $EAF = \prod_{i=1}^{15} F_i$

	工作量因素 F_i	很低	低	正常	高	很高	极高
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
	虚拟机易变性		0.87	1.00	1.15	1.30	
	环境周转时间		0.87	1.00	1.07	1.15	
人员的因素	分析员能力		1.46	1.00	0.86		
	应用领域实际经验	1.29	1.13	1.00	0.91	0.71	
	程序员能力 (软硬件结合)	1.42	1.17	1.00	0.86	0.82	
	虚拟机使用经验	1.21	1.10	1.00	0.90	0.70	
	程序语言使用经验	1.41	1.07	1.00	0.95		
项目因素	现代程序设计技术	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	

详细COCOMO模型

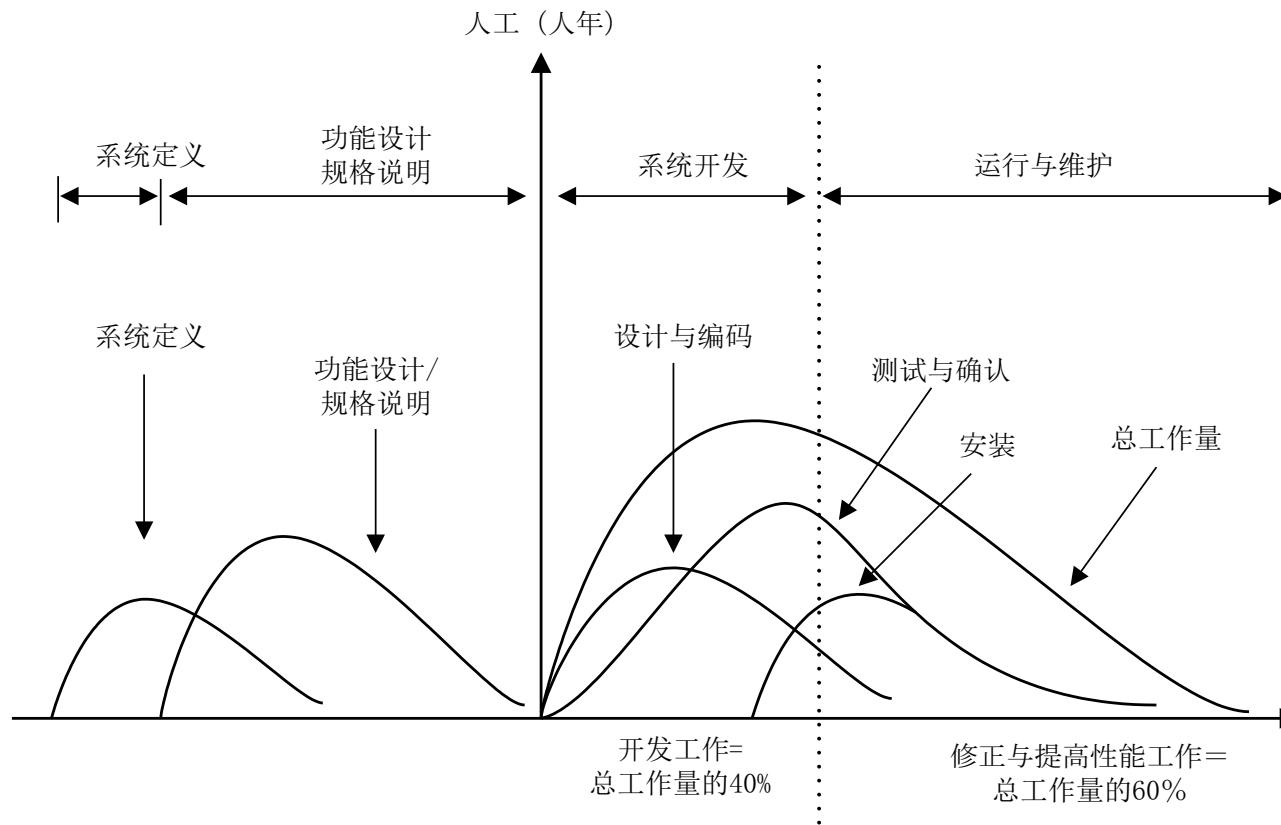
- 估算公式与中间CoCoMo模型相同，并按分层、分阶段的形式给出其工作量影响因素分级表
 - 针对每一个影响因素，按模块层、子系统层、系统层，有三张工作量因素分级表，供不同层次的估算使用
 - 每一张表中又按开发各个不同阶段给出
- 例如软件可靠性在子系统层的工作量因素分级表如下所示

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

Putnam模型

- **一种软件项目工作量估算的动态多变量模型**

- 他根据一些大型软件项目(30人年以上)的工作量分布情况，推导出软件项目在软件生存周期各阶段的工作量分布，如图所示
- 图中的工作量分布曲线与著名的Reyleigh-norden曲线相似



Putnam模型计算公式

- 根据Reyleigh-norden曲线给出代码行数、工作量和开发时间之间的关系，如下所示：

$$L = C_K E^{1/3} t_d^{4/3}$$

其中：L表示源程序代码行数(LOC)

t_d 表示开发持续时间(年)

E是包括软件开发和维护在整个生存期所花费的工作量(人年)

C_k 表示技术状态常数，其值依赖于开发环境

$$C_k = \begin{cases} 2\,000 & \text{比较差的软件开发环境} \\ 8\,000 & \text{一般的软件开发环境} \\ 11\,000 & \text{比较好的软件开发环境} \end{cases}$$

$$\text{由此可得: } E = L^3 / (C_k^3 t_d^4)$$

软件可靠性估算

- 与软件可靠性密切相关的程序中残留错误数的估算和平均故障间隔时间的估算
 - 错误植入法
 - 分别测试法
 - 软件平均故障间隔时间估算

错误植入法

假设程序中测试前残留的错误数为 N ，然后人为地在程序中植入 N_s 个错误，这些植入的错误对测试人员来说是未知的。经过一段时间的测试，如果发现的错误数为 n ，其中植入的错误数为 n_s ，则原程序中残留的错误估算值 N' 可用下式计算：

$$N' = \frac{nN_s}{n_s}$$

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- **项目进度管理**
- 风险管理
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- 小结

指导软件项目进度安排的基本原则 - 1

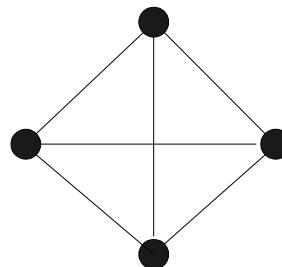
- **划分**：项目必须被划分成若干可以管理的活动和任务，为了实现项目的划分，对产品和过程都需要进行分解
- **相互依赖性**：确定各个被划分的活动或任务之间的相互关系，有些任务必须是串行的，有些可能是并行的
- **时间分配**：为每个被调度的任务分配一定数量的工作单位，为每个任务制定开始和结束日期；
- **工作量确认**：确保在任意时段中分配给任务的人员数量不会超过项目组中的人员数量

指导软件项目进度安排的基本原则

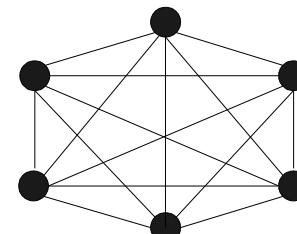
- **定义责任**：每个被调度的任务都应该指定某个特定的小组成员来负责
- **定义结果**：每个被调度的任务都应该有一个确定的输出结果
- **定义里程碑**：每个任务或任务组都应该与一个项目里程碑相关联(当一个或多个工作产品经过质量评审并且得到认可时，标志着一个里程碑的完成)

人员与生产率之间的关系

- **人员之间的交流开销**：一个由 n 个人组成的项目组内共存在 $n(n - 1)/2$ 条通信路径
- **对于生产率的影响**：
 - 增加一个人并不等于净增了一个人的工作量，应扣除相应的通信代价
 - 每个开发小组的成员不宜太多，通过合理的组织形式减少组内的通信路径数
 - 在开发过程中尽量不要中途加人，避免太多的生产率损失
- **参与项目的人员数与整体生产率之间的关系并非是线性的**

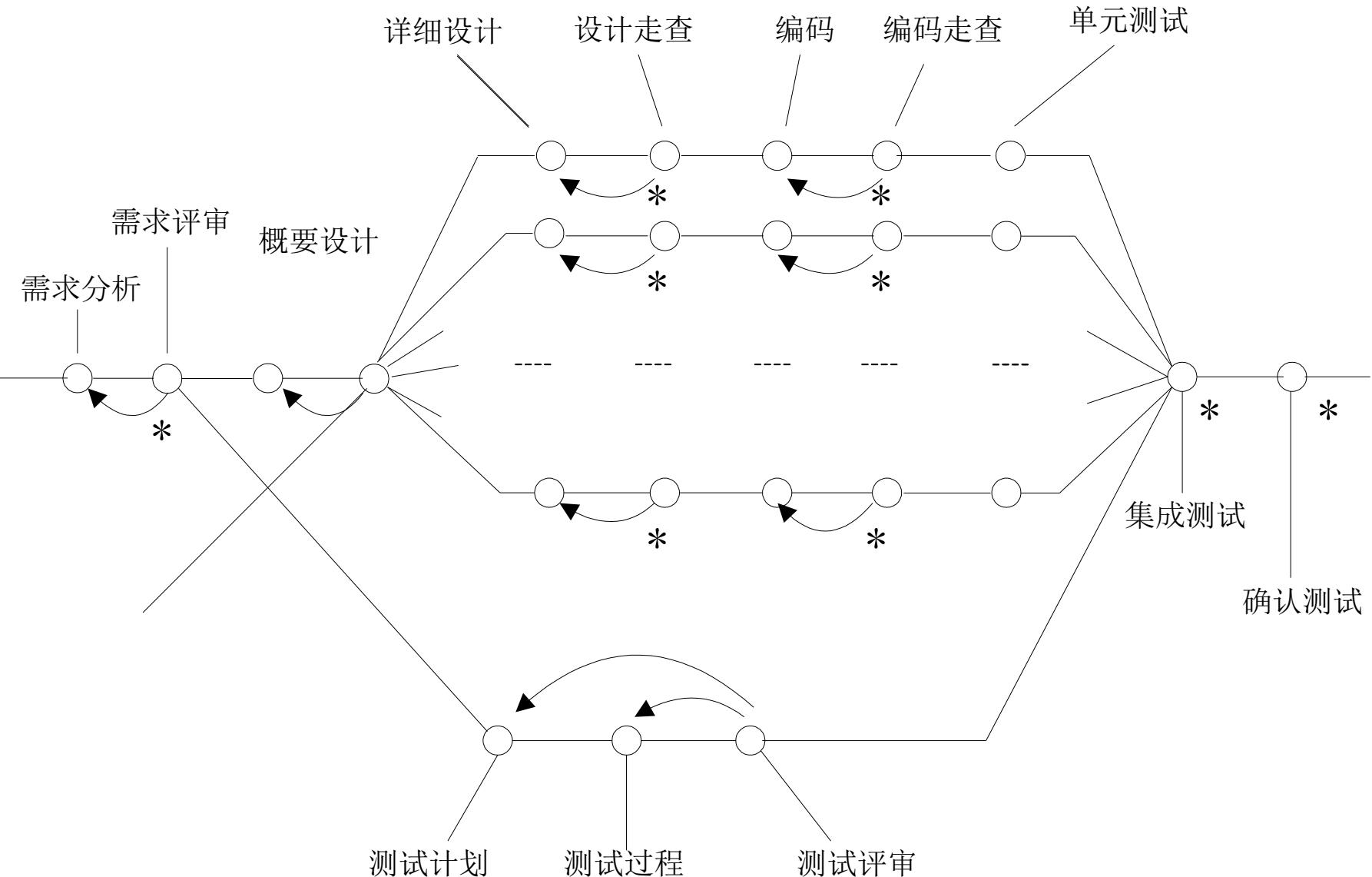


(a) 四人之间所有通信路径



(b) 六人之间所有通信路径

基于瀑布模型的任务网络示例



任务工作量的确定

- 根据软件工程过程的不同，可确定其相应的任务的工程量分配
 - 常用的有40-20-40规则：在整个软件开发过程中，编码工作量仅占20%，编码前工作量占40%，编码后工作量占40%

CoCoMo任务工作量分配比例

项目类型	阶段分配	规模 (千行)				
		微型 <2	小型 8	中型 32	大型 128	特大型 512
组织型	计划与需求	10	11	12	13	
	设计	19	19	19	19	
	编码与单元测试	63	59	55	51	
	组装与测试	18	22	26	30	
半独立型	计划与需求	16	18	20	22	24
	设计	24	25	26	27	28
	编码与单元测试	56	52	48	44	40
	组装与测试	20	23	26	29	32
嵌入型	计划与需求	24	28	32	36	40
	设计	30	32	34	36	38
	编码与单元测试	48	44	40	36	32
	组装与测试	22	24	26	28	30

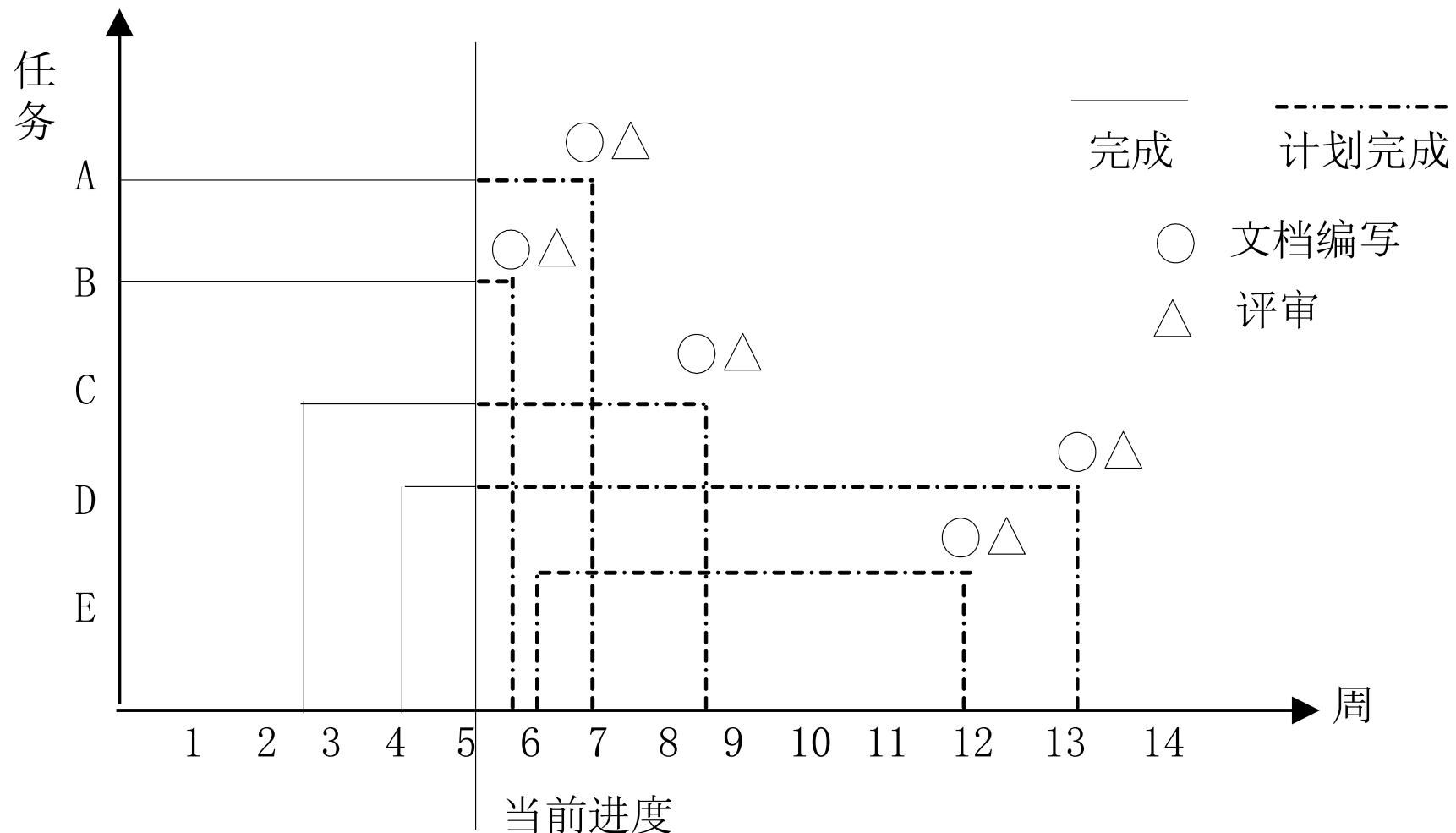
进度安排

- 通用的项目进度安排工具和技术可以应用于软件项目
- 为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，需要采用**图示的方法明确标识**：
 - 各个任务的计划开始时间和完成时间
 - 各个任务的完成标志
 - 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**和**网络图**是两种常用的图示方法

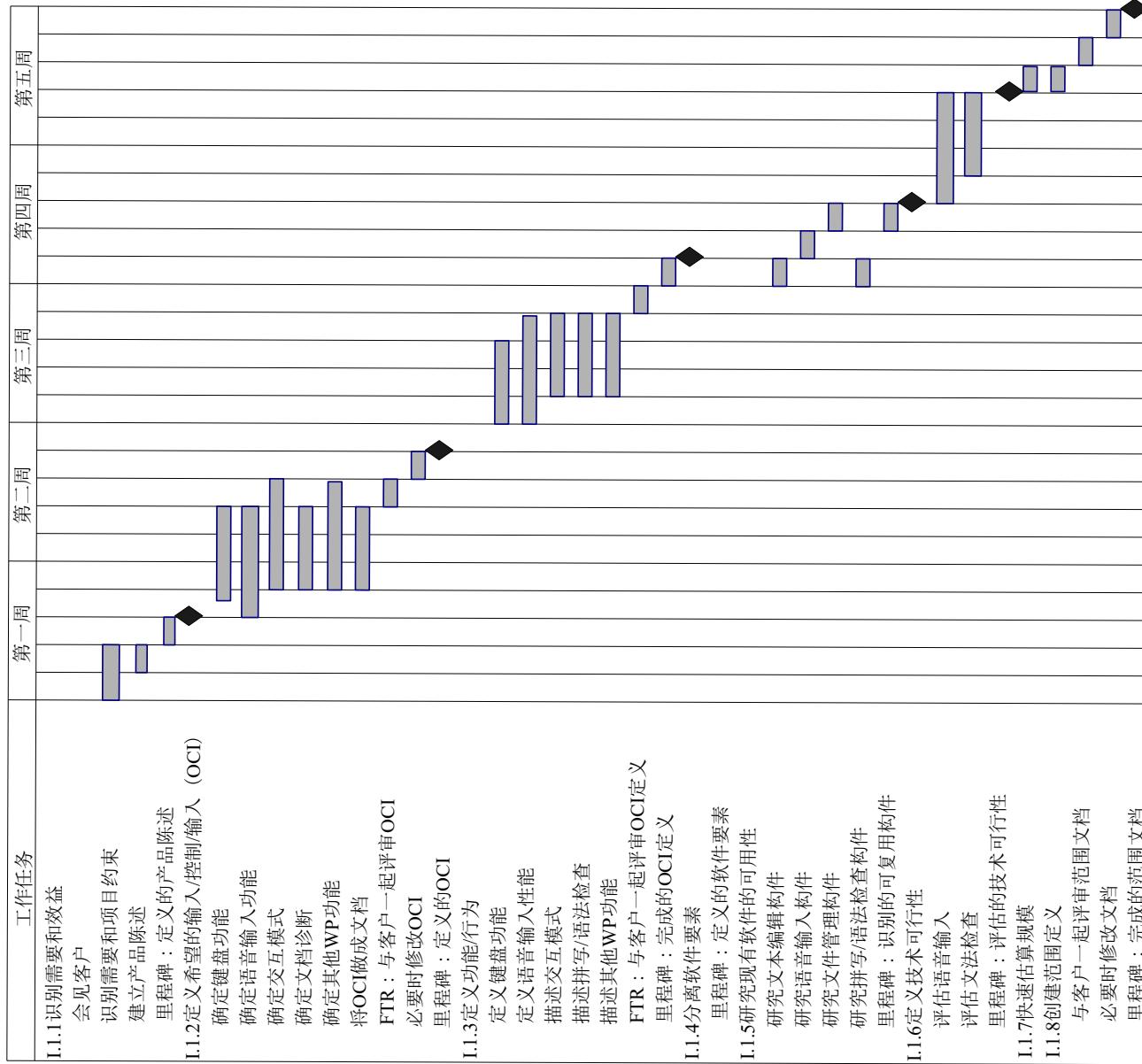
甘特图(Gantt Chart)

- 也称时间表(Timeline chart)，用来建立项目进度表
- 在甘特图中，每项任务的完成以必须交付的文档和通过评审为标准
- 因此在甘特图中，文档编制与评审是软件开发进度的里程碑

甘特图示例-1



甘特图示例 - 2



跟踪进度

- 根据项目进度表，跟踪和控制各任务的实际执行情况
- 一旦发现某个任务(特别是关键路径上的任务)未在计划进度规定的时间范围内完成，那么就要采取措施进行调整
 - 增加额外的资源、增加新的员工或调整项目进度表
- 可以通过以下方式来实现项目跟踪：
 - 定期举行项目状态会议，由项目组中的各个成员分别报告进度和问题
 - 评价在软件工程过程中产生的所有评审结果
 - 确定正式的项目里程碑是否在预定日期内完成
 - 比较项目表中列出的各项任务的实际开始日期与计划开始日期
 - 非正式与开发人员会谈，获取他们对项目进展及可能出现的问题的客观评价

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- 项目进度管理
- **风险管理**
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- 小结

风险与风险管理

- 现代项目管理与传统项目管理的不同之处就是引入了风险管理技术
- 风险：在给定情况下和特定时间内，那些可能产生的结果与预期结果之间的差异，差异越大，风险越大
- 风险管理就是识别评估风险，建立、选择、管理和解决风险的可选方案和组织方法
 - 包括了风险标识、风险预测、风险评估和风险管理与监控四个活动
 - 强调通过对项目目标的主动控制做到防患于未然以避免或减少损失

风险的类别

- **项目风险**：可能对项目的预算、进度、人力、资源、顾客和需求等方面产生不良影响的潜在问题
- **技术风险**：潜在的设计、实现、接口、验证和维护等方面的问题，此外，规约的二义性、技术的不确定性、陈旧或不成熟的“领先的”技术都可能是技术风险
- **商业风险**：威胁要开发的软件的生存能力
 - 开发了一个无人真正需要的产品(市场风险)
 - 开发的产品不符合公司的整体商业策略(策略风险)
 - 建造了一个销售部门不知如何销售的产品
 - 由于重点转移失去了高级管理层支持(管理风险)
 - 没有得到充分预算或人力资源保证(预算风险)

通过风险检测表标识风险

- 风险检测表
- 选用0~5来回答，值越大表示风险越大
- 例如“人员配备风险检测表”：

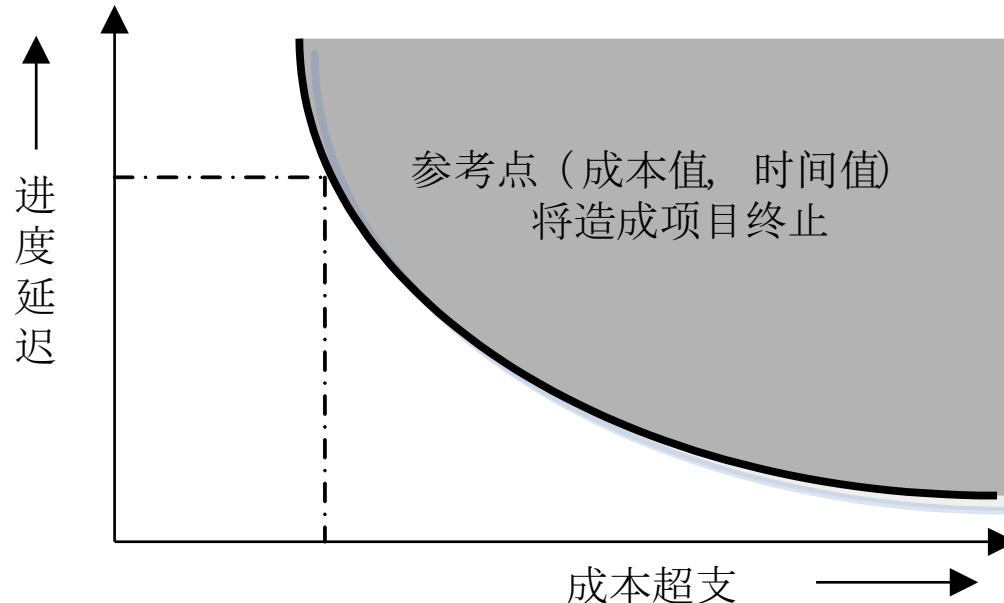
问 题	风险程度 (0...5)
开发人员的水平如何；	
开发人员在技术上是否配套；	
开发人员的数量如何；	
开发人员是否能够自始至终地参加软件开发工作；	
开发人员是否能够集中全部精力投入软件开发工作；	
开发人员对自己的工作是否有正确的期望；	
开发人员是否受过必要的培训；	
开发人员的流动是否能够保证工作的连续性；	

风险评估

- 进一步审查风险预测阶段对各种风险预测的精确度，并对每个风险因素定义一个风险参考水准
- 当性能下降、成本超支、支持困难或进度延迟超过相应的水准时会导致项目被迫终止
- 风险评估活动通常采用下列形式的三元组：

风险参考水准

- 可以为风险因素的组合定义风险参考水准。
- 下图给出了进度和成本组合的风险参考水准，图中阴影部分是导致项目终止的区域，即当项目的成本值和进度值位于该区域时将导致项目的终止



风险避免

- 对付风险的最好办法是主动地避免风险，即在风险发生前，分析引起风险的原因，然后采取措施，以避免风险的发生
- 例如为避免“频繁的人员流动”风险可采取如下策略：
 - 与现有人员探讨人员流动的原因(如恶劣的工作条件、低报酬、竞争激烈的劳务市场等)
 - 在项目开始前采取行动，缓解那些管理控制范围内的原因
 - 一旦项目启动，采取一些技术来保证在人员离开时工作的连续性
 - 对项目组进行良好的组织，使每一个开发活动的信息能被广泛的传播和交流
 - 定义文档的标准并建立相应的机制，以确保文档能被及时建立
 - 对所有工作进行详细评审，使得多个人熟悉该项工作
 - 对每一个关键的技术人员都指定一个后备人员

风险监控

- 监控可以提供风险指示(是否正在变高或变低)的因素
- 例如，对人员流动风险可监控如下因素：
 - 项目组成员对项目的态度
 - 项目组的凝聚力
 - 成员之间的关系
 - 与报酬和利益相关的问题
 - 在公司内和公司外工作的可能性

风险管理及监控计划(RMMP)

- Risk Management and Monitoring Plan
- 对于每个风险，特别对那些高概率高影响的风险应制定RMMP
- RMMP的实施会导致额外的项目开销
 - 对于一个大型项目，可能识别出30或40种风险。如果为每种风险定义3至7个风险管理步骤，则风险管理本身可作为一个子项目

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- 项目进度管理
- 风险管理
- **软件项目的组织**
- 软件质量管理
- 软件配置管理
- 小结

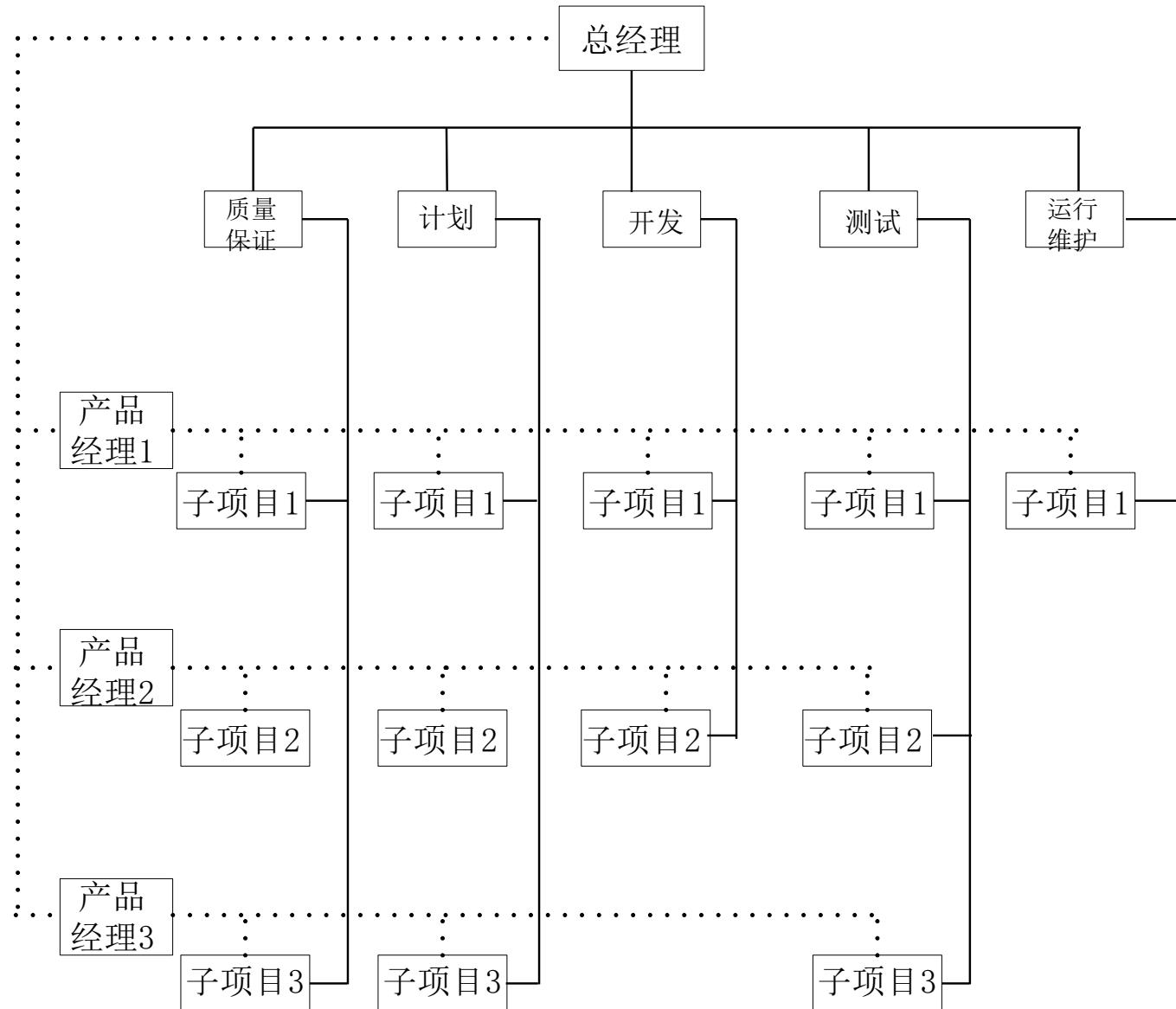
软件项目的组织

- 项目组织形式不仅要考虑软件项目的特点，还需要考虑参与人员的素质
- 在软件项目的组织原则：
 - 尽早落实责任：在软件项目开始组织时，要尽早指定专人负责，使他有权进行管理，并对任务的完成负全责
 - 减少接口：一个组织的生产率随完成任务中存在通信路径数目的增加而降低。要有合理的人员分工、好的组织结构、有效的通信，减少不必要的生产率的损失
 - 责权均衡：软件经理人员所负的责任不应比委任给他的权力还大

项目组织模式

- **按项目划分的模式**：按项目将开发人员组织成项目组，项目组的成员共同完成该项目的所有开发任务，包括项目的定义、需求分析、设计、编码、测试、评审以及所有的文档编制，甚至包括该项目的维护
- **按职能划分的模式**：按软件过程中所反映的各种职能将项目的参与者组织成相应的专业组，如开发组、测试组、质量保证组、维护组等
- **矩阵形模式**：上述两种模式的复合，每个软件人员既属于某个专业组，又属于某个项目组

矩阵型组织结构示例



程序设计小组的组织形式

- 程序设计小组主要是指从事软件开发活动的小组
- 三种常见的程序设计小组的组织形式(具有不同的通信路径数)
 - 主程序员制小组(Chief programmer team)
 - 民主制小组(Democratic team)
 - 层次式小组(Hierarchical team)

主程序员制小组

- **由一名主程序员、若干名程序员、一名后缓(back up)工程师和一名资料员组成**
 - 主程序员通常由高级工程师担任，负责小组的全部技术活动，进行任务的分配，协调技术问题，组织评审，必要时也设计和实现项目中的关键部分
 - 程序员负责完成主程序员指派给他的任务，包括相关的文档编写
 - 后援工程师协助主程序员工作，必要时能替代主程序员，他也做部分的开发工作
 - 资料员负责小组中所有文档资料的管理，收集与过程度量相关的数据，为评审准备资料。一个资料员可以同时服务于多个小组

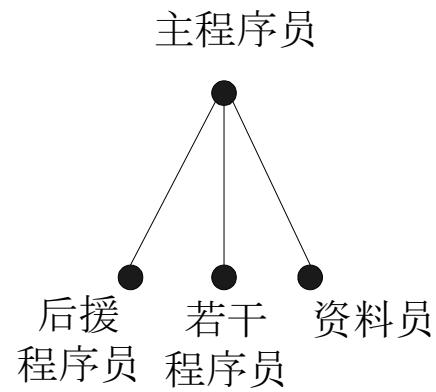
民主制小组

- 小组成员之间地位平等，虽然形式上有一位组长，但小组的工作目标及决策都是由全体成员集体决定的
 - 能充分发挥每个成员的积极性
 - 小组成员平等地交换意见，互相合作，形成一个良好的工作氛围
 - 但这种形式的组内通信路径比较多

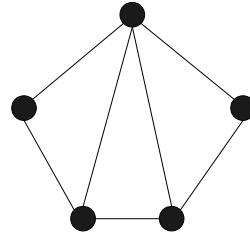
层次式小组

- **层次式小组：一名组长领导若干名高级程序员，每名高级程序员领导若干名程序员**
 - 组长通常就是项目负责人，负责全组的技术工作，进行任务分配，组织评审
 - 高级程序员负责项目的一个部分或一个子系统，负责该部分的分析、设计，并将子任务分配给程序员
 - 这种组织形式适合于具有层次结构特征的项目的开发
 - 组内的通信路径数介于主程序员制小组和民主制小组之间

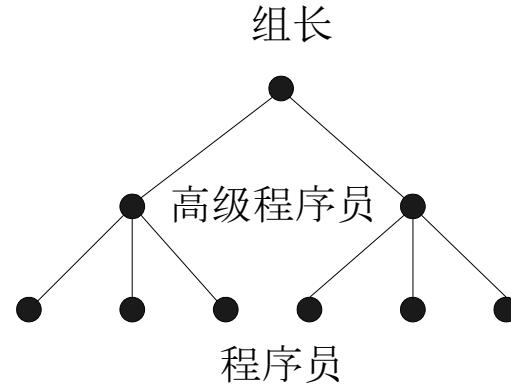
三种程序设计小组的组织结构及通信



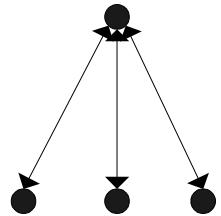
(a) 主程序员制小组



(b) 民主制小组

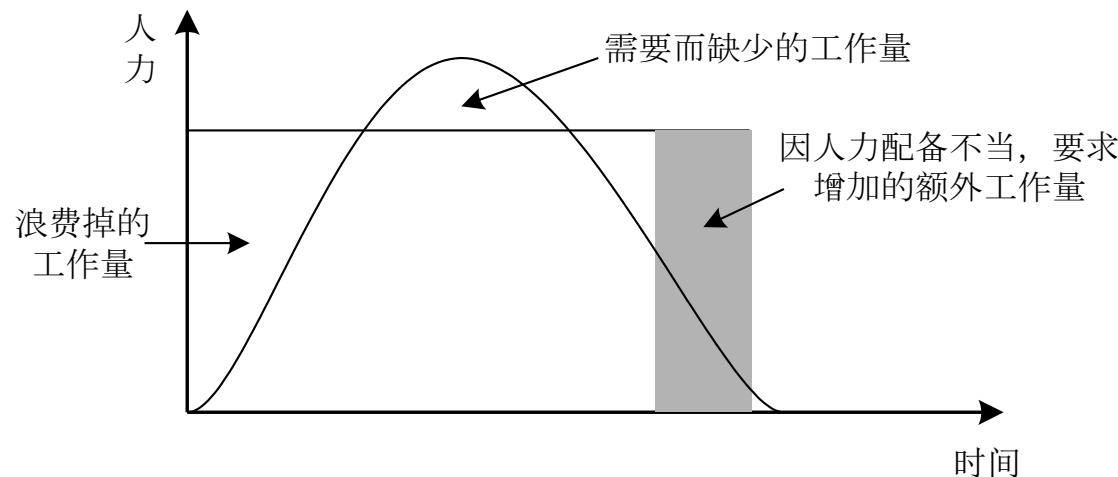


(c) 层次式小组



人员配备

- **合理地配备人员包括：对不同的开发活动指派不同的人员，并明确指出对种类人员的要求**
 - 通常在项目初期需要的人员并不太多，但其业务和技术水平要高
 - 在项目的中后期需要较多的人参与，其中大多是一些有专门技术(如编程、测试)的人
 - 在项目临近结束(试运行)时，只需少量人员参与即可
- **如果一个软件项目从开始到结束都保持一个恒定的人员配备，那么就会出现下图中的情况**



配备人员的原则

- **重质量**：软件项目组不仅需要足够的人，更需要业务和技术水平高的人
- **重培训**：培养所需技术人员和管理人员是有效解决人员问题的好方法
- **双阶梯提升**：人员提升应分别按技术职务和管理职务进行，不能混在一起

项目经理的要求

- **项目经理是项目的组织者，关系到项目的成败**
- **一个称职的项目经理应具备如下能力：**
 - 获得充分资源的能力
 - 组建团队的能力
 - 分解工作的能力
 - 为项目组织提供良好环境的能力
 - 权衡项目目标的能力
 - 应付危机，解决冲突的能力
 - 谈判及广泛沟通的能力
 - 技术综合能力
 - 领导才能

软件人员的素质要求

- **牢固掌握计算机软件的基本知识和技能**
- **善于分析和综合问题，具有严密的逻辑思维能力**
- **工作踏实、细致、不靠碰运气，遵循标准和规范，具有严格的科学作风**
- **工作中表现出有耐心、有毅力、有责任心**
- **善于听取别人的意见，善于与周围人员团结协作，建立良好的人际关系**
- **具有良好的书面和口头表达能力**

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- 项目进度管理
- 风险管理
- 软件项目的组织
- **软件质量管理**
- 软件配置管理
- 小结

软件质量管理

- 高质量的软件应该具备以下条件：
 - 满足软件**需求**定义的功能和性能
 - **文档**符合事先确定的软件开发标准
 - 软件的特点和属性遵循软件工程的**目标和原则**
 - 还应该考虑在预算和进度范围内交付;
 - 因此在项目进行过程中要对偏差进行控制

质量控制和质量保证

- 质量控制是为了保证每一件工作产品都满足对它的需求而应
用与整个开发周期中的一系列审查、评审和测试
 - 质量控制在创建工作产品的过程中包含一个**反馈循环**，通过对质量的反馈，使得我们能够在得到的工作产品不能满足其规约时调整开发过程
 - 所有工作产品都应该具有定义好的和**可度量的规约**，这样就可以将每个过程的产品与这一规约进行比较
- 质量保证由管理层的审计和报告构成，目标是为管理层提供获知产品质量信息所需的数据，从而获得产品质量是否符合预定目标的认识和信心

软件质量保证

- 软件质量保证活动由两类不同的角色承担
 - 负责技术工作的软件工程师：通过采用可靠的技术方法和措施、进行正式的技术评审、计划周密的软件测试来考虑质量问题，并完成软件质量保证和质量控制活动
 - 负责质量保证工作的*SQA* (*Software Quality Assurance*) 小组：辅助软件工程小组得到高质量的最终产品

SQA小组的活动(CMU SEI)

- 为项目准备SQA计划
- 参与开发该项目的软件过程描述
- 评审各项软件工程活动、对其是否符合定义好的软件过程中的相应部分进行核实
- 审计指定的软件工作产品、对其是否符合定义好的软件过程中的相应部分
- 确保软件工程及工作产品中的偏差已被记录在案，并根据预定规程进行处理
- 记录所有不符合的部分并报告给高级管理者。
- 此外，SQA小组还需要协调变化的控制和管理，并帮助收集和分析软件度量信息

软件评审

- 软件评审是软件质量保证的重要手段
- 通常在软件工程过程的每个活动(如需求分析、设计、编码)的后期进行正式的软件评审
- 两种主要评审活动：项目管理评审和技术评审(ISO/IEC 12207《信息技术 软件生存周期过程》中的联合评审过程定义)

正式评审和非正式评审

- 正式评审(*formal reviews*)通常在软件工程过程的每个活动的后期进行，采用正式的会议评审方式，通过正式评审的活动标志着该活动到达了一个里程碑，该活动的制品也就成为一个基线
- 非正式评审(*informal reviews*)通常是一种由同事参加的即兴聚会，大多采用“走查”(*walkthrough*)的方式

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- 项目进度管理
- 风险管理
- 软件项目的组织
- 软件质量管理
- **软件配置管理**
- 小结

软件配置管理基本概念-1

GB/T 11457 《软件工程术语》

- 软件配置项(*Software Configuration item , SCI*)：为配置管理设计的软件的集合，它在配置管理过程中作为单个实体对待
- 软件配置(*Software configuration*)：软件产品在不同时期的组合。该组合随着开发工作的进展而不断变化
- 配置管理(*Configuration management*)：应用技术的和管理的指导和监控方法以标识和说明配置项的功能和物理特征，控制这些特征的变更，记录和报告变更处理和实现状态并验证与规定的需要的遵循性
- 版本(*Version*)：与计算机软件配置项的完全编纂或重编纂相关的计算机软件配置项的初始发布或再发布

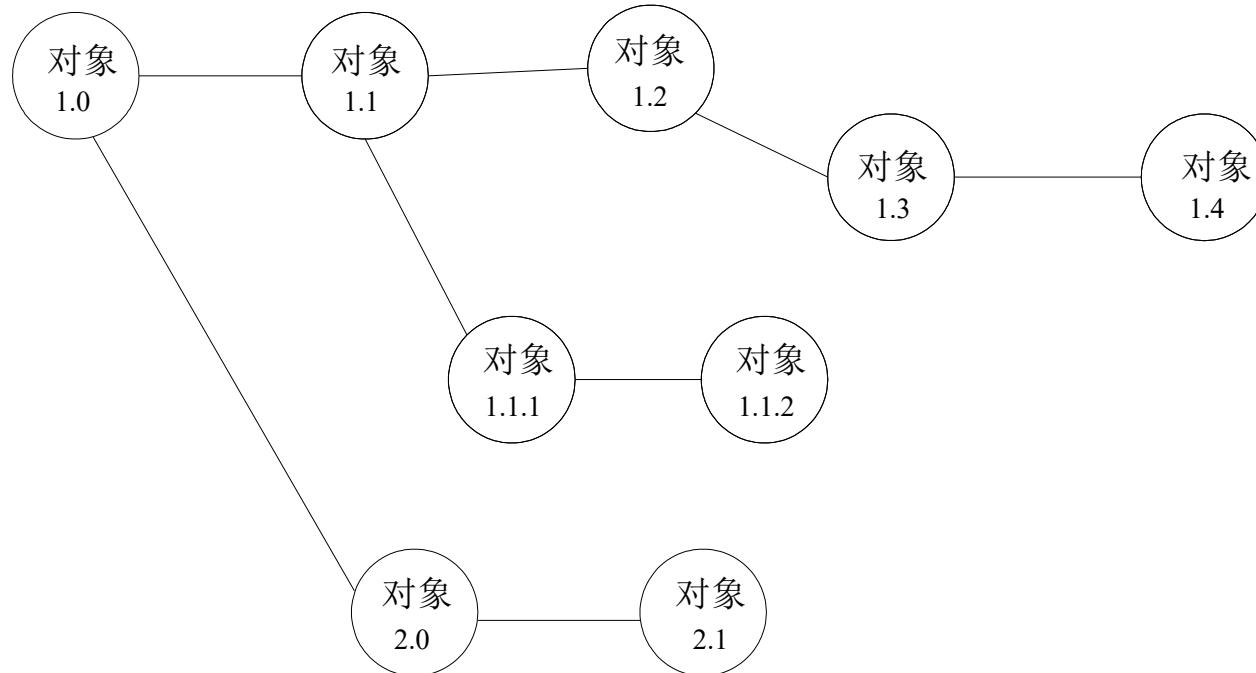
软件配置管理基本概念-2

GB/T 11457 《软件工程术语》

- **发布(Release)** : 一项配置管理行为，它说明某配置项的一个特定版本已准备好用于特定的目的(例如发布测试产品)
- **基线(baseline)** : 业已经过正式审核与同意，可用作下一步开发的基础，并且只有通过正式的修改管理过程方能加以修改的规格说明或产品
- **变更控制(change control)** : 指提议作一项变更并对其进行**估计、同意或拒绝、调度和跟踪的过程。**
- **配置审计(configuration audit)** : 指对所要求的全部配置项均已产生出来，**当前的配置与规定的需求相符所作的证明**。技术文件说明书完全而准确地描述了各个配置项目，并且曾经提出的所有变更请求均已得到解决的过程
- **配置状态记录(configuration status accounting)** : 一种配置管理的元素，它由**记录和报告为有效地管理某一配置所需的信息组成**。此信息包括列出经**批准的配置标识表、建议变更的配置状态和经批准变更的实现状态**

软件配置管理的主要活动1 - 版本控制

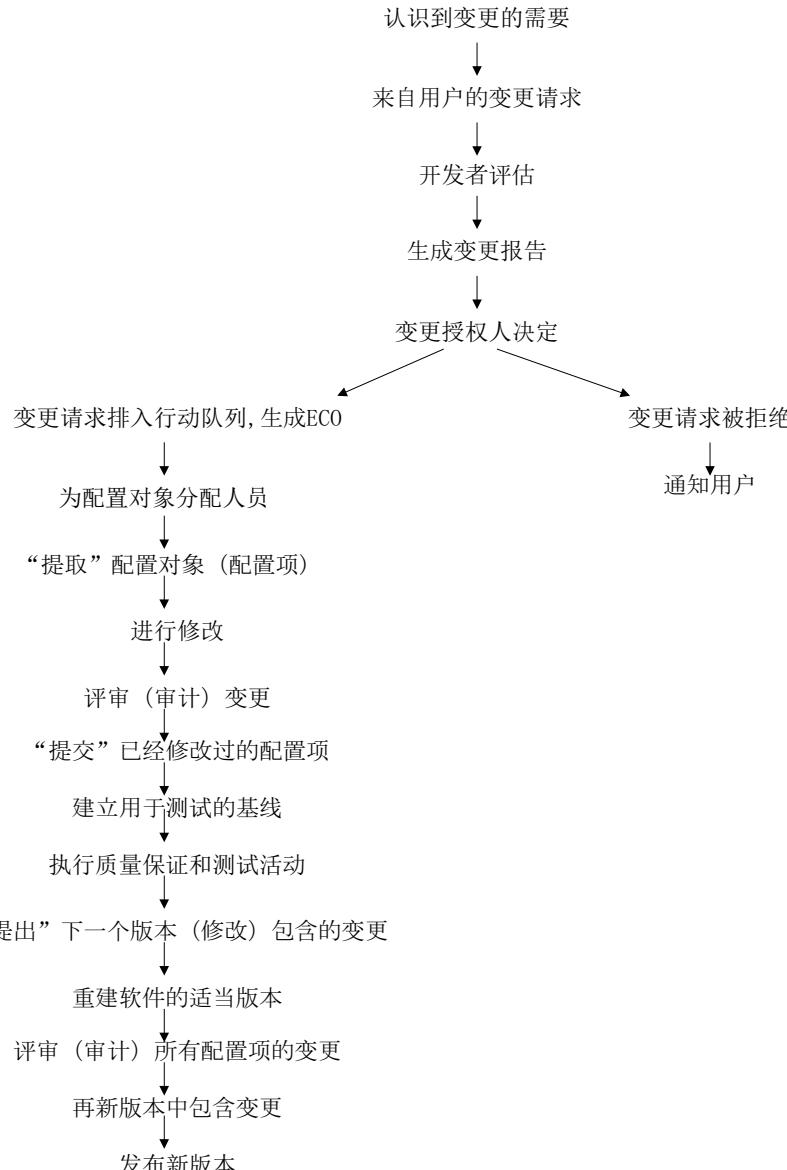
- 对系统不同版本进行标识和跟踪的过程
- 演化图：描述对象的变更历史
 - 在一个配置项被确定为基线前，它可能会变更很多次，甚至在建立基线后，变更也可能经常发生



软件配置管理的主要活动2-变更控制

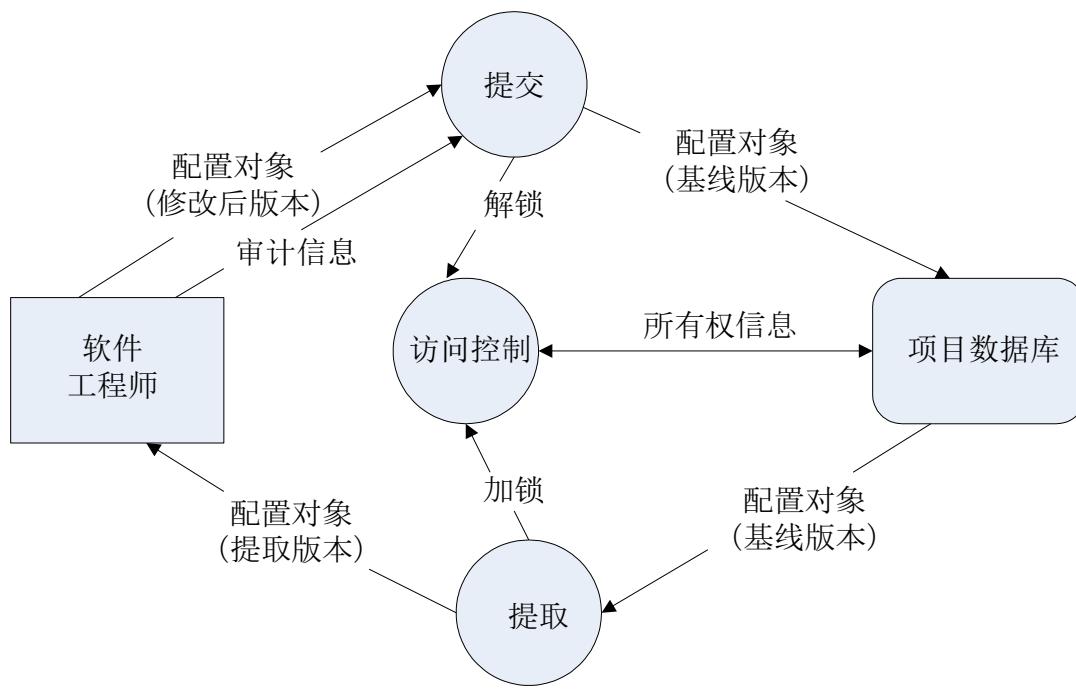
- 变更控制结合人的规程和自动化工具以提供一个管理变更的机制
- 主要过程(下页图)
 - 变更请求提交后的评估，评估对象包括技术指标、潜在副作用、对其他配置对象和系统功能的影响以及变更的成本等
 - 评估的结果以**变更报告**的形式给出，该报告由变更授权人(*change control authority, CCA*)审核批准
 - 对被批准的变更生成工程**变更工单**(描述了将要进行的变更、必须注意的约束以及评审和审计的标准)
 - 将被修改的对象检出(*check out*)并通过合适的SQA活动进行修改
 - 修改完成后对象检入，并使用版本控制机制建立软件的下一个版本

变更控制过程



访问控制和同步控制

- 通过检出(提取)和检入(提交)实现配置项的访问控制和同步控制
 - 在配置项变成基线之前，开发者可以进行非正式的变更控制
 - 一旦配置项已经经过正式的技术审评并创建了配置项的基线，则要实施项目级的变更控制
 - 为了对其进行修改，开发者必须获得项目管理者的批准(如果变更是“局部的”)或此配置项的变更授权人的批准(如果该变更影响到其他配置项)



配置审计

- 审计：通过调查研究确定已制定的过程、指令、规格说明、基线及其他特殊要求是否恰当和被遵守、以及实现是否有效
- 通过正式技术评审或软件配置审计来保证变更的有效性
 - 正式评审关注已经被修改的配置项的技术正确性
 - 评审者评估SCI以确定它与其他SCI的一致性、遗漏、及潜在的副作用
 - 原则上应该对所有变更进行正式评审
 - 软件配置审计通过评估配置项未在正式技术评审中考虑的特征，形成对正式评审的补充

软件配置审计的主要内容

- 在工程变更工单中说明的变更已经完成了吗？是否有副作用
- 是否已经进行了正式的技术评审以评估技术正确性
- 软件过程是否遵循了软件工程标准
- 变更在配置项中被“显著地强调”了吗
- 是否指出了变更的日期和变更的作者
- 配置对象的属性反映了该变更吗
- 是否遵循了标注变更、记录变更并报告变更的软件配置管理规程
- 所有相关的配置项被适当更新了吗

状态报告

- 配置状态报告，也称为状态记录(*status accounting*)
- 软件配置管理的任务之一，它回答下列问题：发生了什么事；谁做的此事；此事是什么时候发生的；将影响别的什么
 - 每次当一个SCI被赋上新的或修改后的标识时，则一个状态报告条目被创建
 - 每次当一个变更被变更授权人批准(即一个工程变更工单产生)时，一个CSR条目被创建
 - 每次当配置审计进行时，其结果作为CSR任务的一部分被报告
 - CSR的输出可以放置到一个联机数据库中，使得软件开发者或维护者可以通过关键词分类访问变更信息
 - 此外，CSR报告被定期地生成，并允许管理者和开发者评估重要的变更

内容摘要

- 软件项目管理概述
- 软件度量
- 软件项目估算
- 项目进度管理
- 风险管理
- 软件项目的组织
- 软件质量管理
- 软件配置管理
- **小结**

小结

- 软件项目管理是软件开发过程中的一项重要活动，它贯穿整个软件生存周期
- 大量工程实践表明，项目失败的一个重要原因 **是项目管理能力太弱**
- 本章在介绍项目管理的关注点和项目管理的内容的基础上，分别对**软件度量、项目估算、进度管理、风险管理、项目组织、质量保证、软件评审和软件配置管理的概念、相关技术和方法**进行介绍