软件工程

第一章 绪论

北京大学软件工程国家工程研究中心

刘学洋

liuxueyang@pku.edu.cn

北京大学

一、绪论

--试图回答软件开发的本质及开发的基本手段

二、软件过程

--试图回答开发所涉及的活动及活动组织

三、软件需求及系统/产品(需求)规约

--试图回答软件开发的起始点及其工作产品

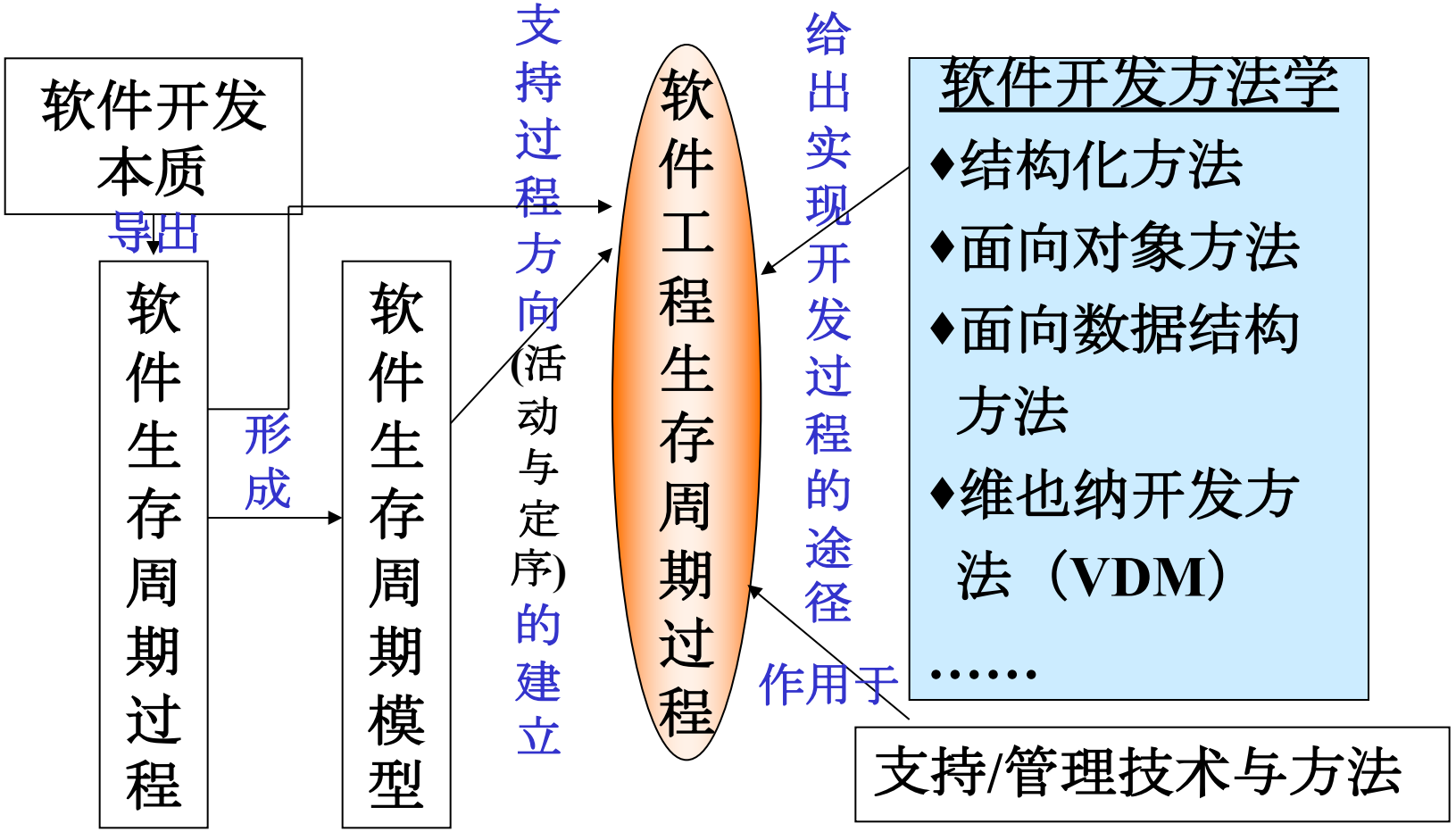
是产品/系统确认(测试)的标尺

四、软件开发方法学

--试图回答如何从事开发活动

五、CMMI

--试图回答获得正确产品/系统的过程能力保障



软件工程基本知识结构

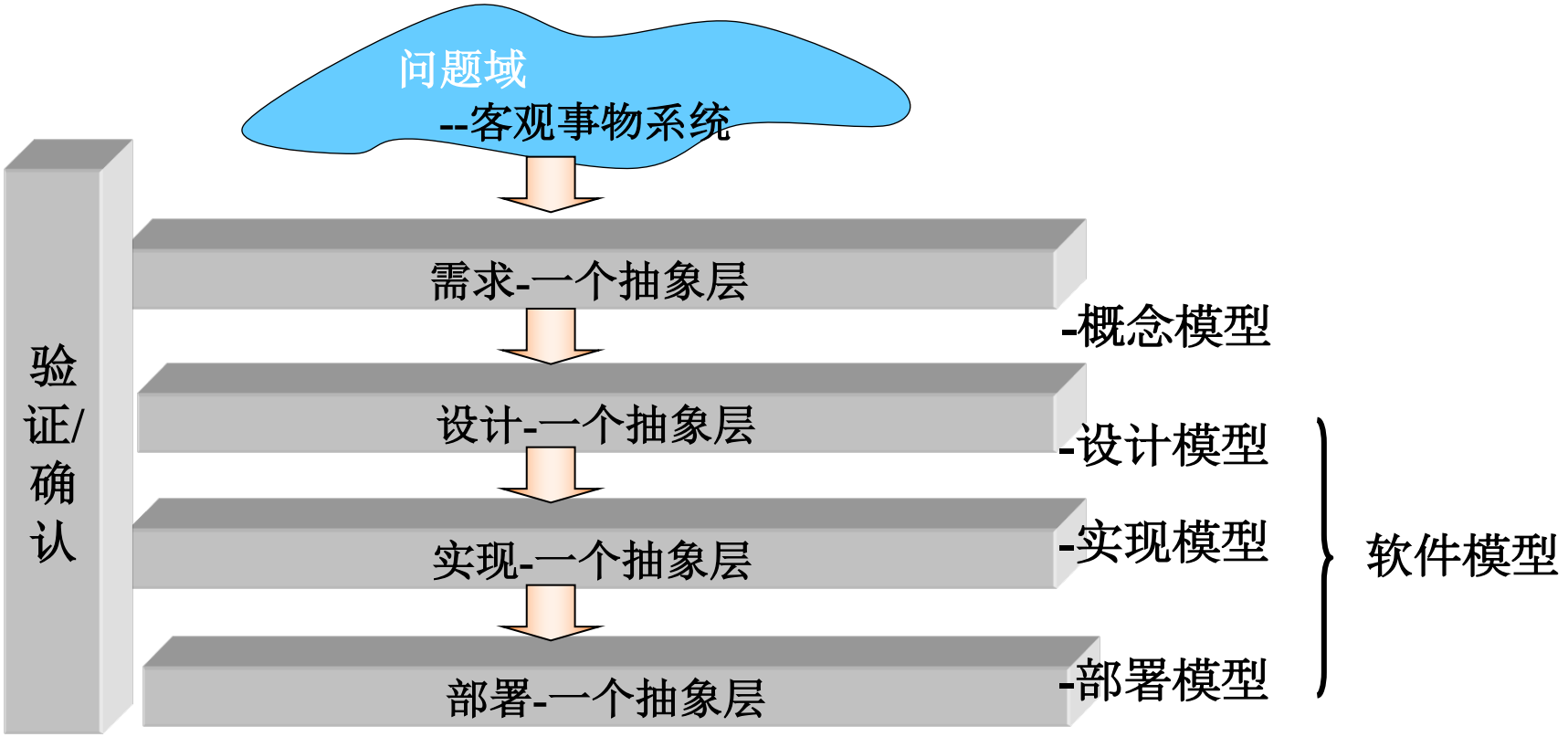
第一章 绪论

何谓模型

any abstraction that includes all essential capabilities, properties,or aspects of what is being modeled without any extraneous details. Firesmith,Henderson-Sellers]

具体地说，模型是在特定意图下所确定的角度和抽象层次上对物理系统的描述，通常包含对该系统边界的描述，给出系统内各模型元素以及它们之间的语义关系。

3 软件系统或项的模型分类



分层的基本动机是控制开发的复杂性，

一个抽象层是由一组确定的术语定义的。

软件工程及软件工程框架基本概念

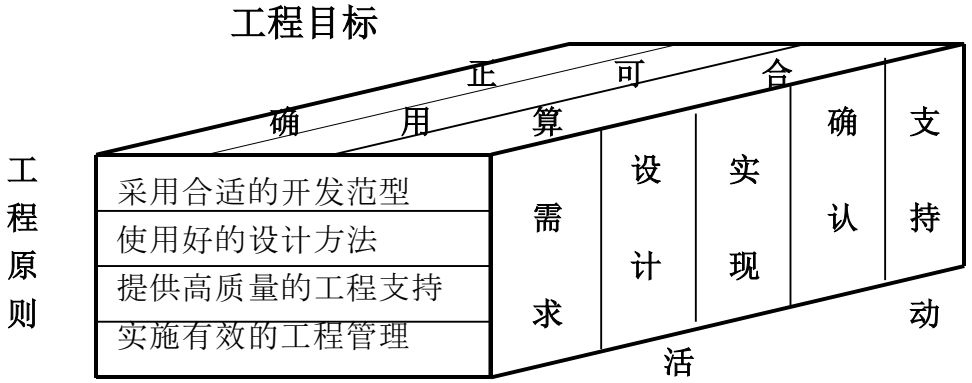
软件－计算机系统中的程序及其文档。程序是计算任务的处理对象和处理规则的描述；文档是为了便于了解程序所需的阐明性资料。

工程－将理论和所学的知识应用于实践的科学。

软件工程－应用计算机科学、数学及管理科学等原理，开发软件的工程。它借鉴传统工程的原则、方法，以提高质量，降低成本为目的。其中，计算机科学、数学用于构造模型与算法，工程科学用于制定规范、设计范型、评估成本及确定权衡，管理科学用于计划、资源、质量、成本等管理。

软件工程是一门交叉性学科。

软件工程框架



软件工程目标　生产具有正确性、可用性和开销合宜的产品。正确性是指软件产品达到预期功能的程度。可用性是指软件基本结构、实现以及文档为用户可用的程度。开销合宜是指软件开发、运行的整个开销满足用户要求的程度。

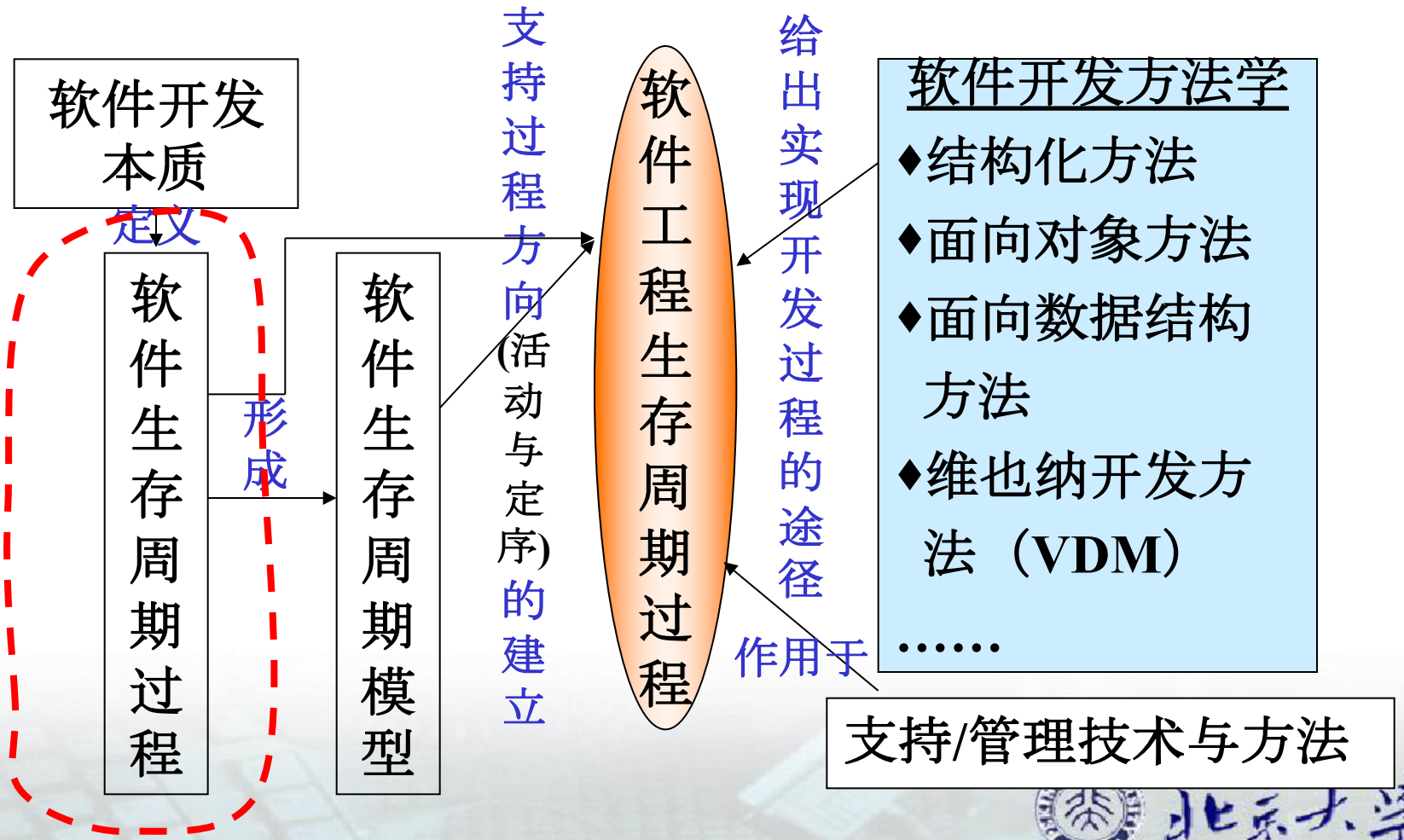
软件开发活动　　生产一个最终满足需求且达到工程目标的软件产品所需要的活动。软件开发的基本活动包括：需求分析（requirement analysis），设计（design），实现（implementation），验证/确认（verification/validation）和维护（maintenance）。

第二章

软 件 过 程

1 开发所涉及的活动

-- 软件生存周期过程



1)基本概念

为了表述软件开发需要做“什么事情(映射)”,引入了以下三个概念：

- 软件过程(process)：活动的一个集合；
- 活动(activity)：任务的一个集合；

  注:”软件过程”和”活动”相当于复合映射.
- 任务(task)： 将输入转换为输出的操作。

  注:”任务”相当于原子映射.

2) 过程分类 *（注：这里按照考纲要求给出的老版本12207内容）*

按过程的主体,可分为三类过程：

(1)基本过程(primary processes)

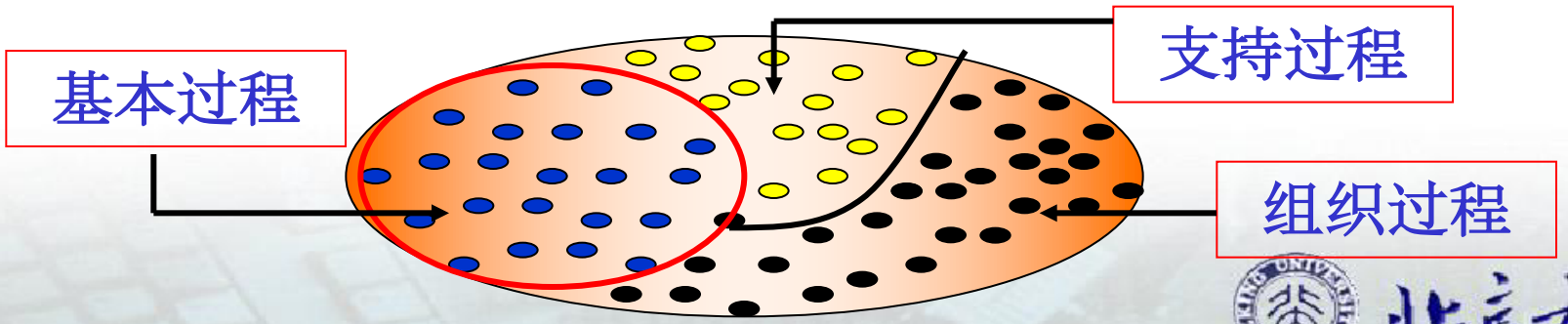
是指那些与软件生产直接相关的活动集。

(2) 支持过程(supporting processes)

是有关各方按其目标所从事的一系列支持活动集。

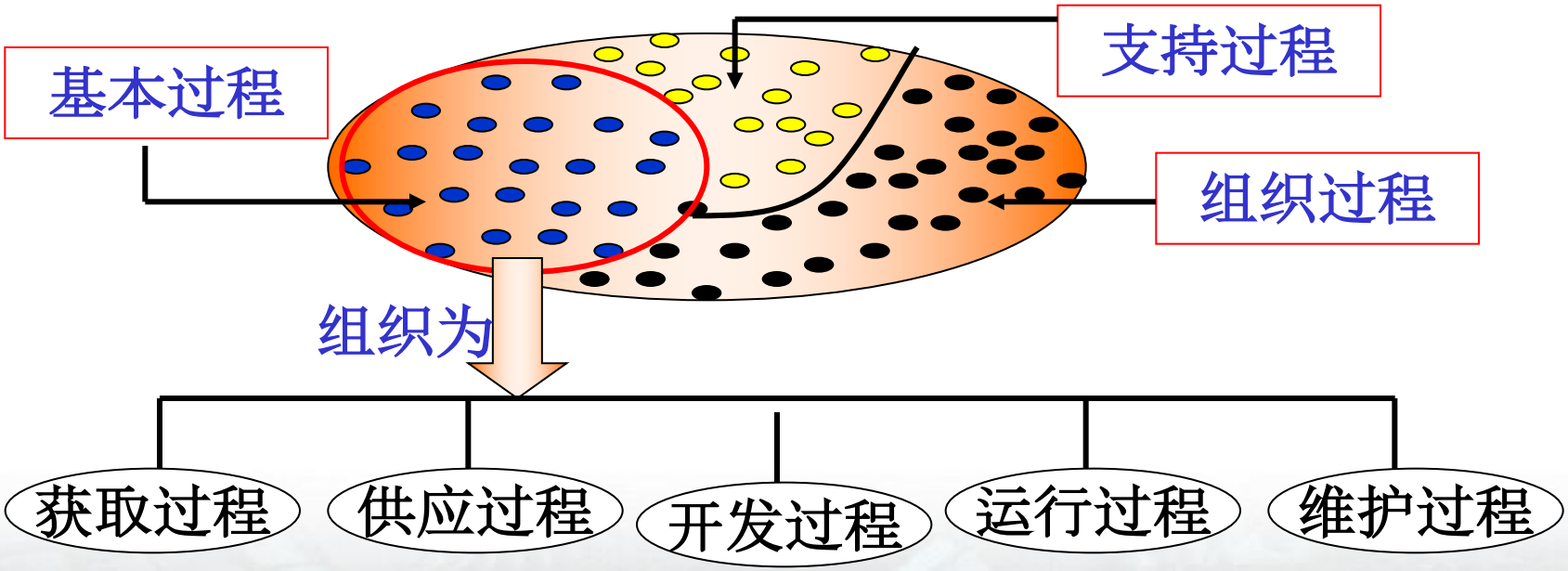
(3) 组织过程(institutional processes)

是指那些与软件生产组织有关的活动集。



(1）基本过程

又按过程中活动的不同主体，将基本过程（类）分为5个过程：获取过程、供应过程、开发过程、运行过程、维护过程



例如：开发过程

是软件开发者所从事的一系列活动。

包括13个活动：

过程的实施准备

系统需求分析　系统结构设计

软件需求分析　软件体系结构设计

软件详细设计　软件编码和测试

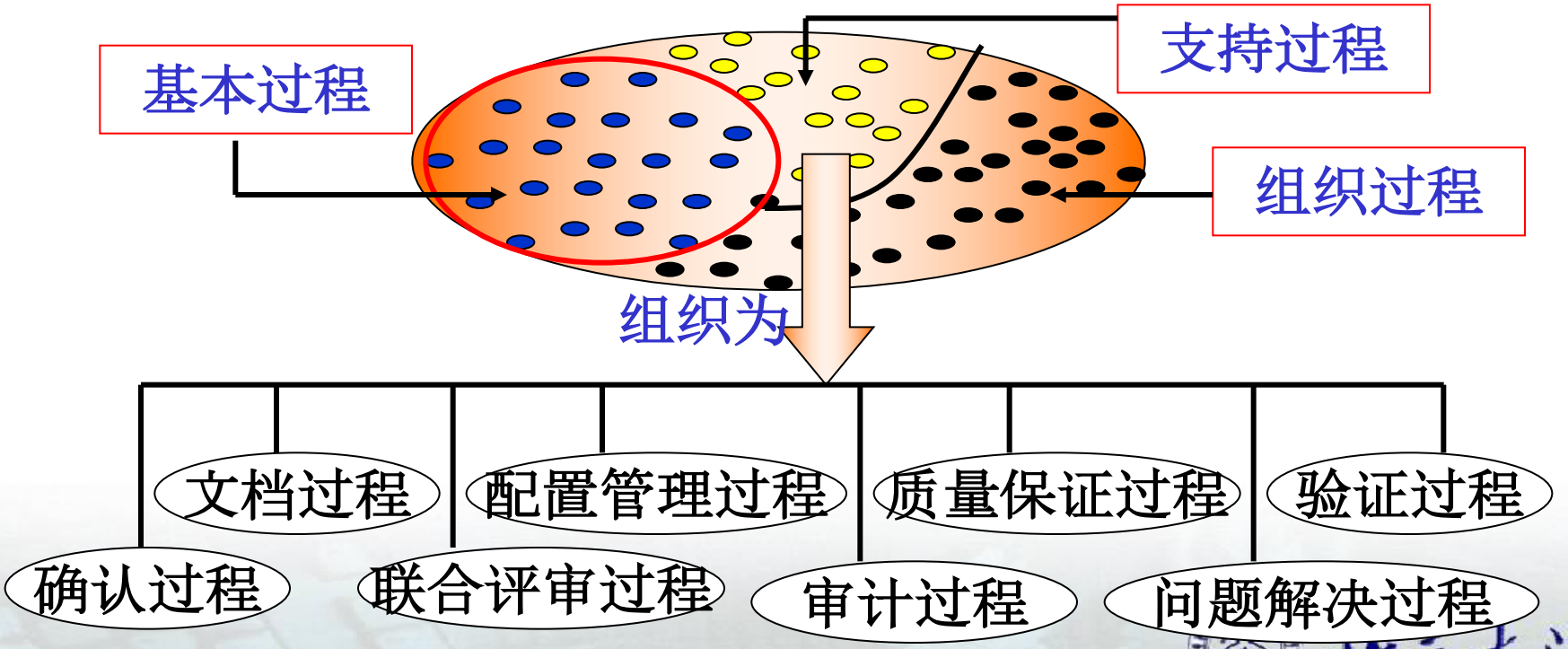
软件集成　　软件合格测试

系统集成　　系统合格测试

软件安装　　软件验收支持

（2）支持过程

又按过程中活动的不同主体，将支持过程（类）分为8个过程：文档过程、配置管理过程、质量保证、验证过程、确认过程、联合评审、审计过程、 问题解决等。



例如：验证过程

①为什么要进行验证和确认?

All models are wrong, some models are useful.
--Fox

出现这一问题的基本原因:

由于人们的认知能力,在抽象一个事物或一个问题时,往往只描述其一些必要性,但不是充分的.例如:

“操作系统是一组管理计算机系统资源的程序”.

②何谓验证过程

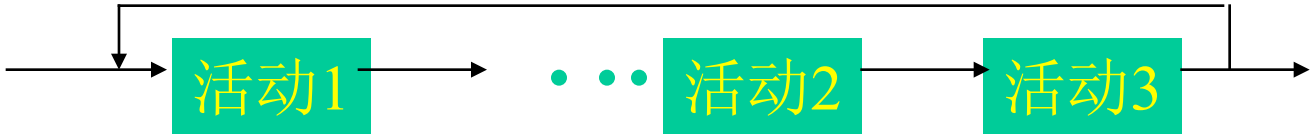
是一个确定某项活动的软件产品是否满足在以前的活动中施加于它们的需求和条件的过程。

可见，该过程的目的是：证实一个过程和/或项目的每一软件工作产品和/或服务恰当地反映了已规定的需求。

验证过程可应用于供应、开发、运行或维护等过程。该过程可以由来自同一组织一个人或多个人来实施，也可以由来自另一组织的人员来实施。

例如5：**确认过程**

定义：确认过程是一个确定需求和最终的、已建成的系统或软件产品是否满足特定预期用途的过程。

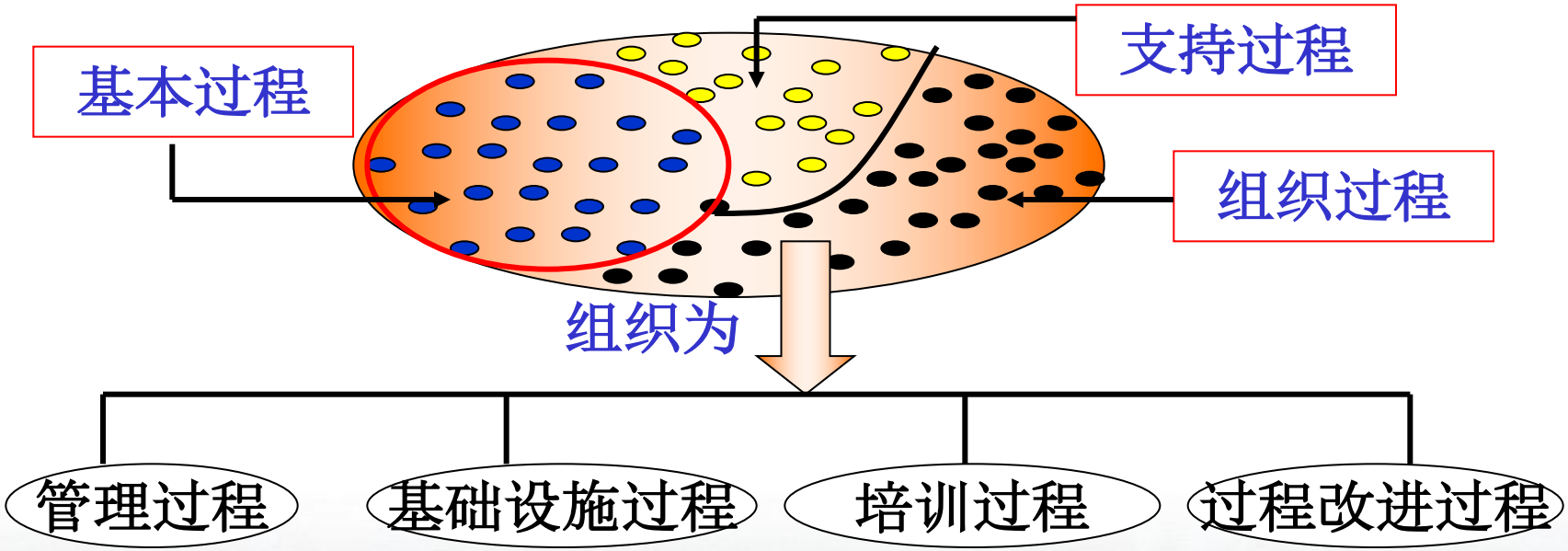


可见，该过程的目的是：证实对软件工作产品特定预期使用的需求已予实现。该过程可以作为开发过程中软件验收支持活动的一个部分来执行。该过程可以由来自同一组织一个人或多个人来实施，也可以由来自另一组织的人员来实施。

(3) 组织过程（Organizational life cycle processes）

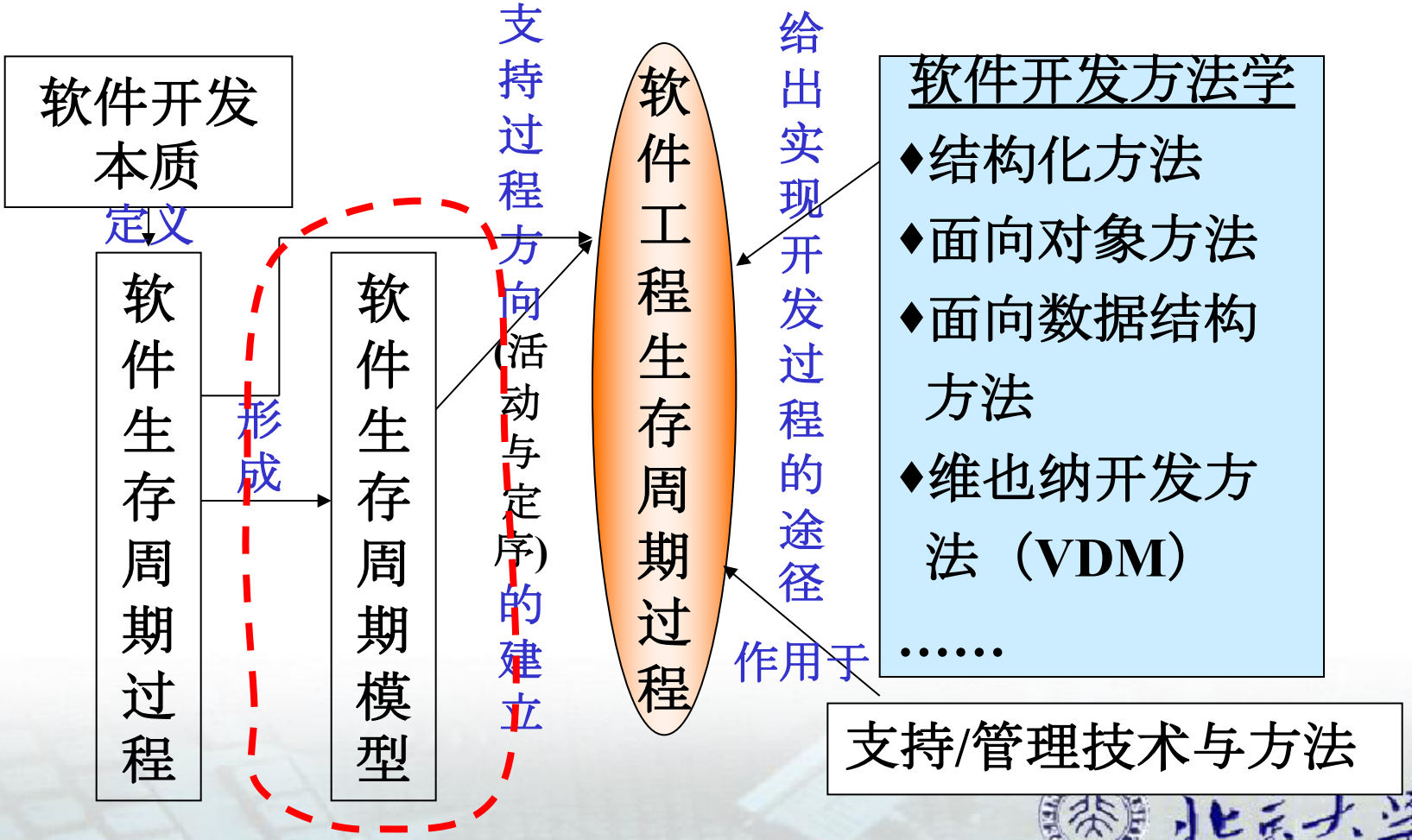
分为4个过程：

管理过程、基础设施过程、培训过程、改进过程



2 开发活动的组织框架

--软件生存周期模型



1） 基本概念

软件生存周期模型 **IEEE Standard 12207.0-1996**

把一个软件生存周期模型描述为：一个包括软件产品开发、运行和维护中有关过程、活动和任务的框架，覆盖了从该系统的需求定义到系统的使用终止。

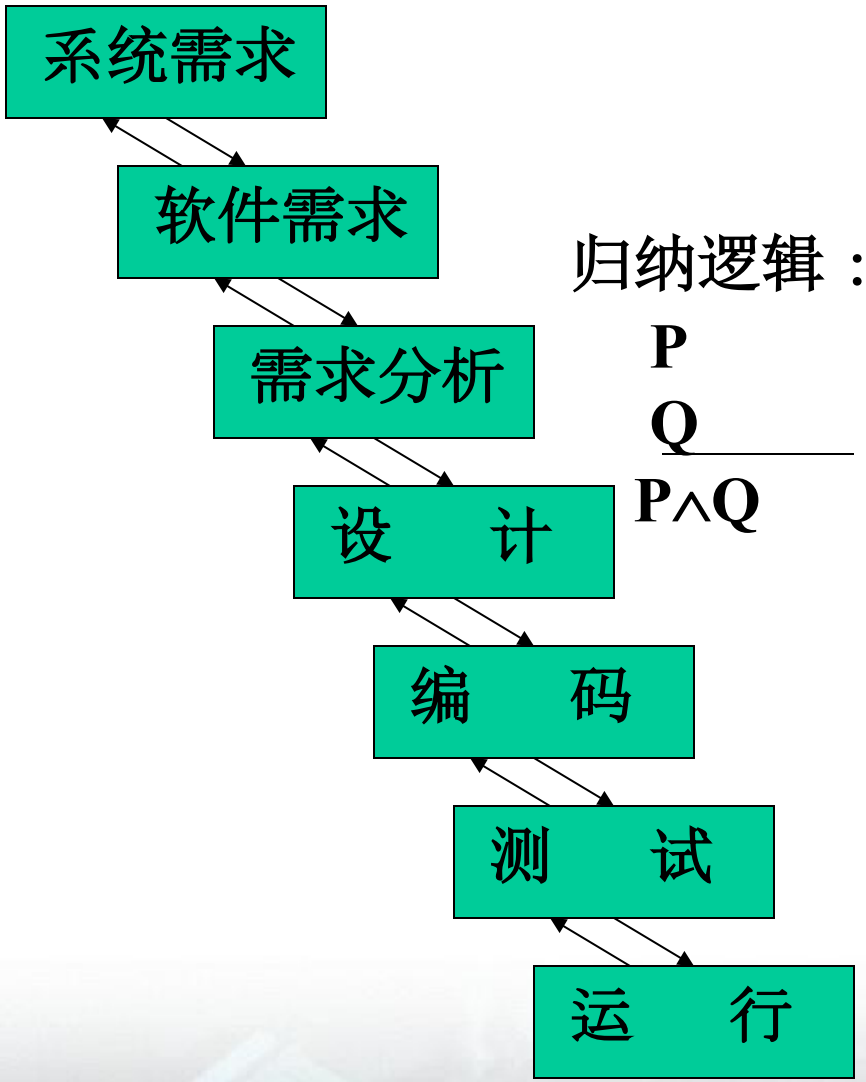
中国计算机科学与技术百科全书

称软件生存周期模型为“软件开发模型”，并把它定义为：软件过程、活动、任务的结构框架。

2） 瀑布模型

1970年，W.Royce

The basis for most current practice and has many variations. Its name come from the progression of activities based on the output of one phase “falling” as input to the following phase. It is driven by the needs to schedule project milestones which are provided by the completion of documents at each level or phase.



归纳逻辑：

P

Q

P∧Q

（1）项目的开发依次经过：需求、设计、编码和单元测试、集成以及维护 –这一基本路径。

（2）通过每一阶段,提交以下产品：软件需求规约、设计文档、实际代码、测试用例、最终产品等。工作产品（又称可提交的产品，**Deliverables**）流经“正向”开发的基本步骤路径

（3）“反向”步骤流表示对前一个可提交产品的重复变更（又称为“返工”**(Rework)**）。

- 由于所有开发活动的非确定性，因此是否需要重复变更，这仅在下一个阶段或更后的阶段才能认识到。
- 返工不仅在以前阶段的某一地方需要，而且对当前正在进行的工作也是需要的。

关于瀑布模型的几点说明

(1）瀑布模型的优点

虽然瀑布模型是一个比较“老”的、甚至过时的开发模型，但其优点为：

❶ 在决定系统怎样做之前，存在一个需求阶段，鼓励对系统“做什么”进行规约（即设计之前的规约）。

❷ 在建造构件之前，存在一个设计阶段，鼓励规划系统结构（即编码之前的设计）。

❸ 在每一阶段结束时进行复审，允许获取方和用户的参与。

❹ 前一步工作产品可作为下一步被认可的、文档化的基线。允许基线和配置早期接受控制。

（2）瀑布模型存在的不足

❶客户必须能够完整、正确和清晰地表达他们的需求；开发人员一开始就必须理解其应用。

❷在开始的两个或三个阶段中，很难评估真正的进度状态；设计、编码和测试阶段都可能发生延期。

❸在一个项目的早期阶段，过分地强调了基线和里程碑处的文档,可能要花费更多的时间，用于建立一些用处不大的文档。

❹当接近项目结束时，出现了大量的集成和测试工作。

❺直到项目结束之前，都不能演示系统的能力。

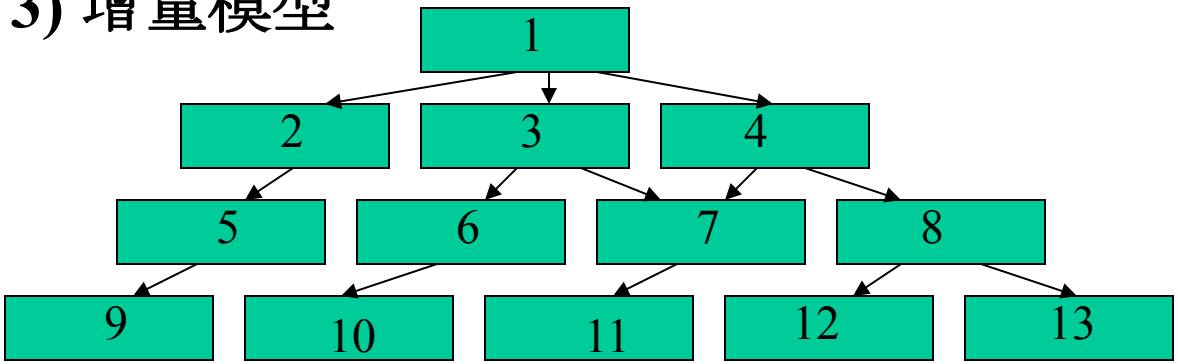
（3）瀑布模型适用的情况

❶在开发中，向下、渐进的路径占支配地位。也就是说，需求已被很好地理解；并且

❷过程设计人员也很清楚：开发组织非常熟悉为实现这一模型所需要的过程（或经过培训后，熟悉什么时候来支持这一项目，以实现这一模型所需要的过程）。

因此为了避免产生过多的反复迭代工作，增加开发成本，一般在准备采用瀑布模型(也包括其他模型)时，需要考虑以下2个问题：第一个问题是，过程设计人员必须对初始产品(通常是软件需求规约，SRS)的不确定性进行评估。另一个问题是，组织是否具有熟练实施每个活动和任务的历史经验。

3) 增量模型



增量1

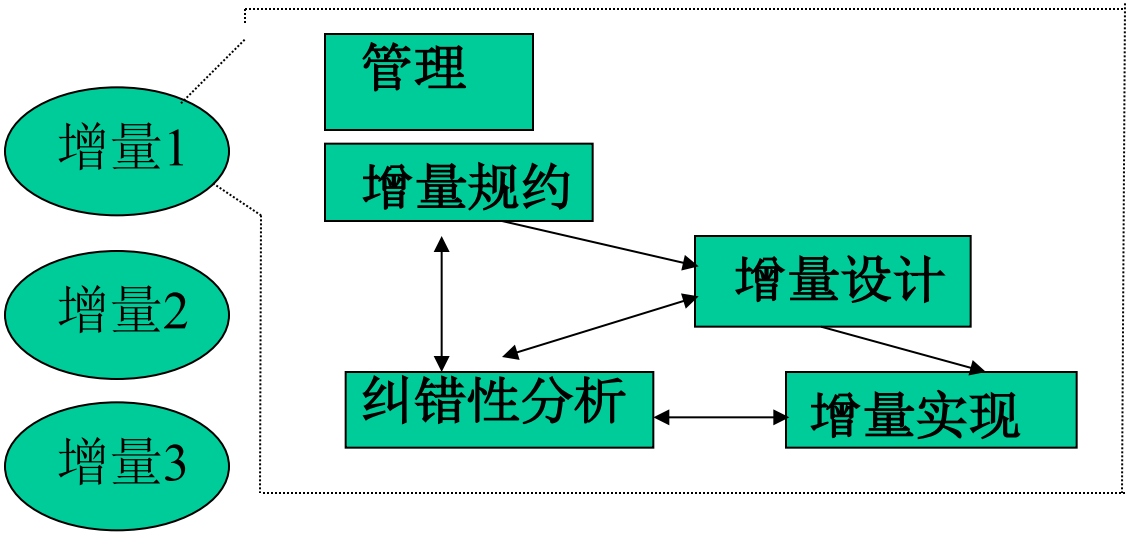
{ 1,2,5,9 }

增量2

{ 3,6,7,4,10,11 }

增量3

{ 8,12,13 }



该模型有一个假设，即需求可以分段，成为一系列增量产品，每一增量可以分别地开发。

关于增量模型的几点说明：

(1) 增量模型的优点

作为瀑布模型的第一个变体，具有瀑布模型的所有优点。

此外，它还有以下优点：

❶第一个可交付版本所需要的成本和时间是很少的；

❷开发由增量表示的小系统所承担的风险是不大的；

❸由于很快发布了第一个版本，因此可以减少用户需求的变更；

❹允许增量投资，即在项目开始时，可以仅对一个或两个增量投资。

注：如果采用增量投资方式，那么客户就可以对一些增量进行招标。然后，开发人员按提出的截止期限进行增量开发，这样客户就可以用多个契约来管理组织的资源和成本。

(2）缺点：

如果增量模型不适于某些项目，或使用有误，则有以下缺点：

❶如果没有对用户的变更要求进行规划，那么产生的初始增量可能会造成后来增量的不稳定；

❷如果需求不像早期思考的那样稳定和完整，那么一些增量就可能需要重新开发，重新发布；

❸管理发生的成本、进度和配置的复杂性，可能会超出一些组织的能力。

(3）该模型的适用情况

❶在开始开发时，需求很明确，且产品还可被适当地分解为一些独立的、可交付的软件（构造增量：**Build increments**．若一个增量不需要交付给客户的话，那么这样的增量通常称为一个“构造”**(Build)**。若增量需要被交付的话，那么它们就被认为是发布版本**(Released version)**。）；

❷在开发中，期望尽快提交其中的一些增量产品。

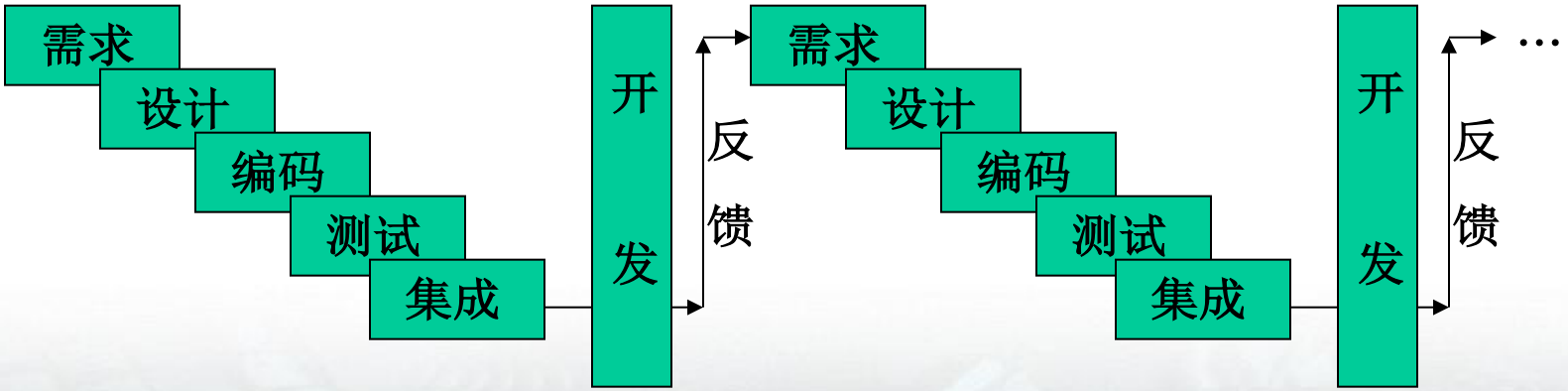
例如：

一个数据库系统，它必须通过不同的用户界面，为不同类型的用户提供不同的功能。在这一情况下，首先实现完整的数据库设计，并把一组具有高优先级的用户功能和界面作为一个增量；以后，陆续构造其它类型用户所需求的增量。

4) 演化模型（Evolutionary model）

是一种有弹性的过程模式，由一些小的开发步组成，每一步历经需求分析、设计、实现和验证，产生软件产品的一个增量。通过这些迭代，完成最终软件产品的开发。

- 针对事先不能完整地定义需求
- 针对用户的核心需求,开发核心系统
- 根据用户的反馈,实施活动的迭代



关于演化模型的几点说明

(1) 主要特征

该模型显式地把增量模型扩展到需求阶段。由图可以看出，为了第二个构造增量，使用了第一个构造增量来精化需求。这一精化可以有多个来源和路径:

首先，如果一个早期的增量已向用户发布，那么用户会以变更要求的方式提出反馈，以支持以后增量的需求开发。

第二，通过实实在在地开发一个构造增量，为以前还没有认识到的问题提供了可见性，以便实际地开始这一增量的工作。

(2）与瀑布模型的关系

在演化模型中，仍然可以使用瀑布模型来管理每一个演化的增量。一旦理解了需求，就可以像实现瀑布模型那样开始设计阶段和编码阶段。

(3）使用演化模型应注意的问题

不能弱化需求分析阶段的工作。其原因是：在项目开始时，需要考虑所有可能需求源的重要性和风险，并对这些需求源的可用性进行评估,只有这样才能识别和界定不确定的需求，并识别第一个增量中所包含的需求。

(4）演化模型的长处和不足

其长处与增量模型是类似的,但还具有以下优点：

❶在需求不能予以规约时，可以使用这一演化模型。

❷用户可以通过运行系统的实践，对需求进行改进。

❸与瀑布模型相比，需要更多用户/获取方的参与。

不足有：

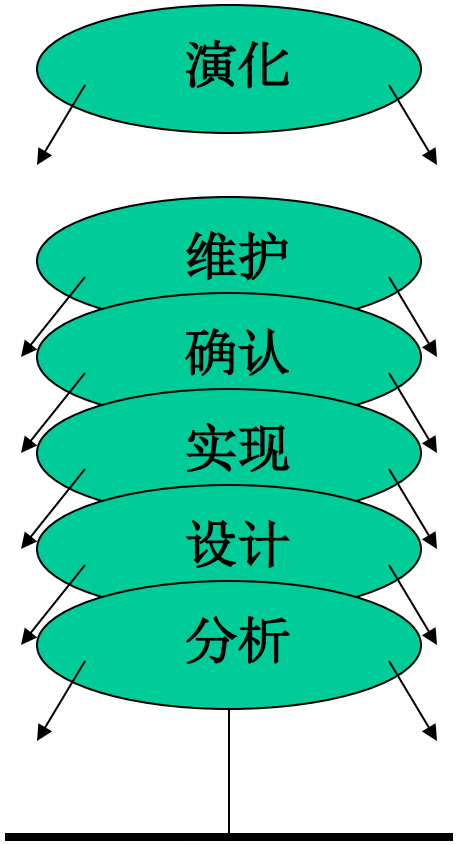
❶演化模型的使用需要有力的管理。

❷演化模型的使用很容易成为不编写需求或设计文档的借口，即使很好地理解了需求或设计。

❸用户/获取方不易理解演化模型的自然属性，因此当结果不够理想时，可能产生抱怨。

5) 喷泉模型

- 特征：迭代
  无缝
- 与面向对象技术的关系

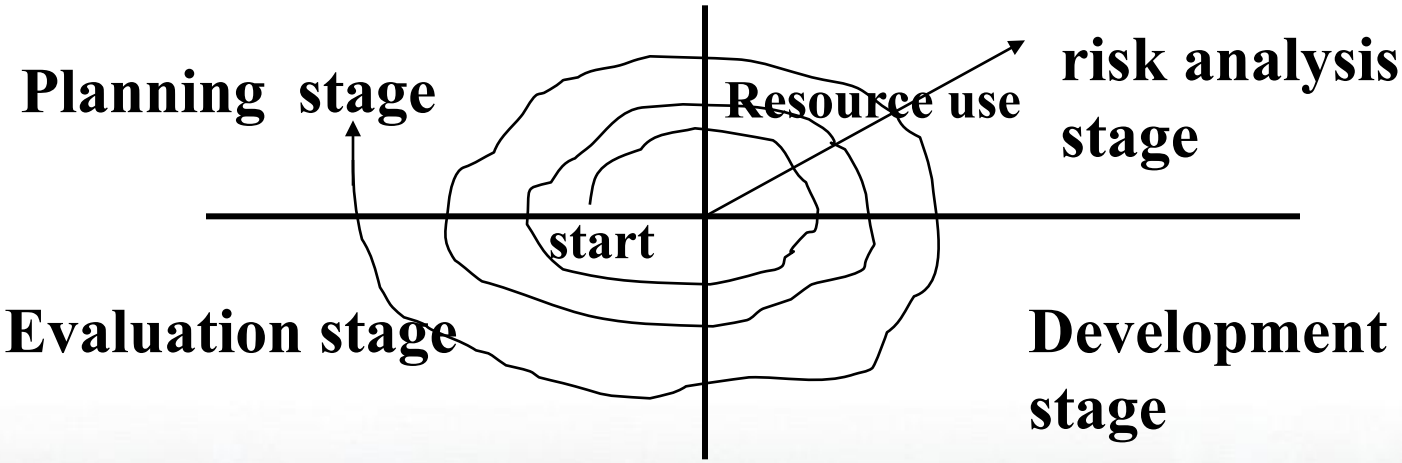


6) 螺旋模型

该模型是由Dr. Barry Boehm [Boehm 1988]开发的。

该模型将软件生存周期的活动分为四个可重复的阶段：

规划、风险分析、开发和评估：



项目的进度是“螺旋”式的。

其中：

① 评估和风险分析阶段都可作出一个决策：项目是否继续。

② 螺旋循环的次数指示了已消耗的资源；

③在规划阶段、风险分析阶段和开发阶段均进行需求规约活动；

④在早期螺旋循环中，为了为最终的实现给出一些指导性决策，经常使用原型构造；

⑤设计和实现活动一般是在开发阶段进行；

⑥ V&V 活动在开发阶段和评估阶段进行；

关于螺旋模型的几点说明：

(1) 该模型关注解决问题的基本步骤:♦标识问题;♦标识一些可选方案，选择一个最佳方案;♦遵循动作步骤，并实施后续工作。其中只要完成了开发的一个迭代，开发的另一个迭代就开始。

(2) 螺旋模型的一个特征是，实际上只有一个迭代过程真正开发可交付的软件。因此,如果

♦项目的开发风险很大，或

♦客户不能确定系统需求，在更广泛的意义上来讲，还包括系统或系统类型的要求，

这时螺旋模型就是一个好的生存周期模型。

(3) 与其它模型的关系

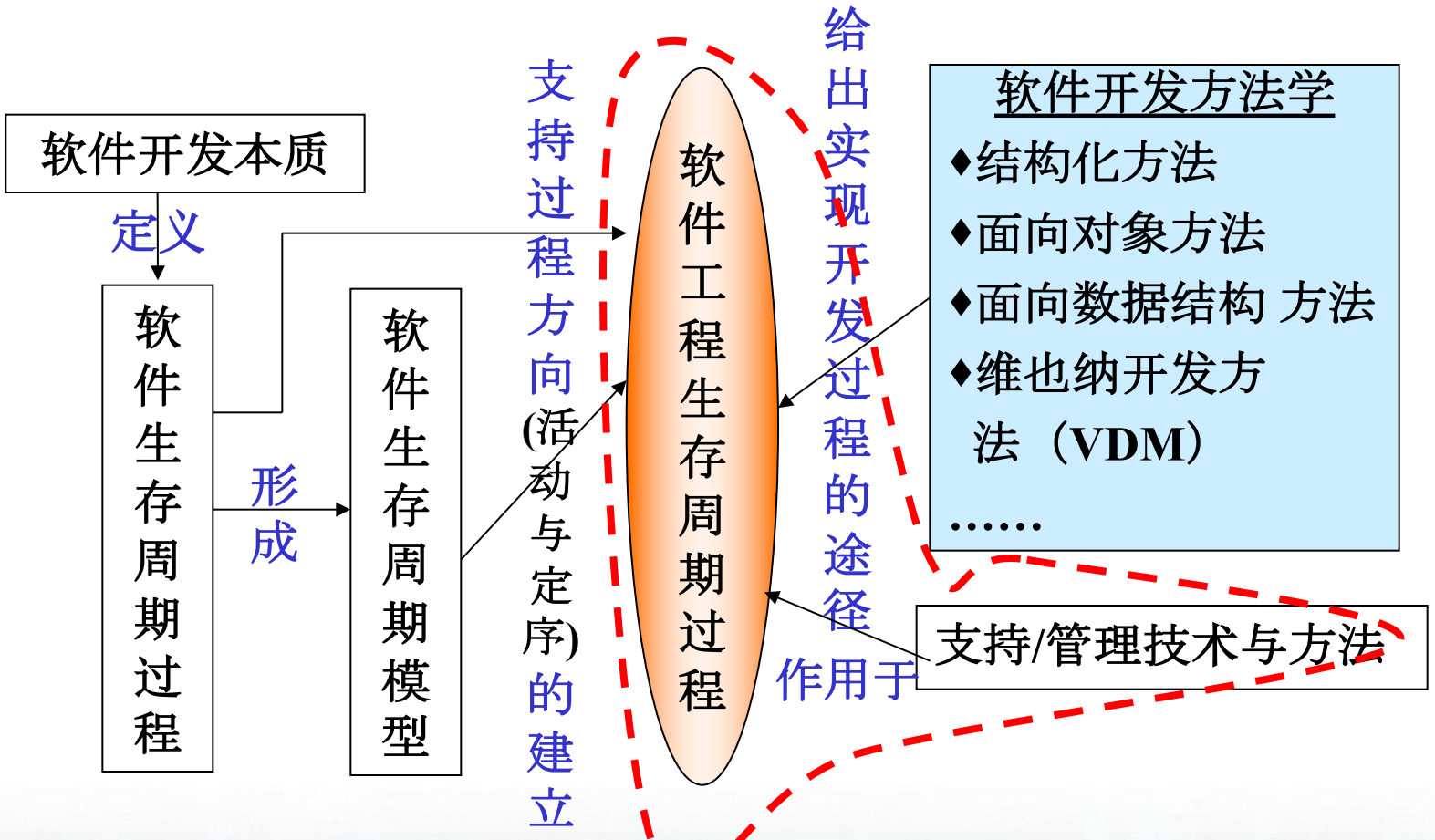
①与演化模型一样，螺旋模型也使用瀑布模型作为一个嵌入的过程-即分析、设计、编码、实现和维护的瀑布过程，是螺旋一周的组成部分。

②尽管螺旋模型和一些迭代模型在框架和全局体系结构方面是等同的，但所关注的阶段以及它们的活动是不同的。

具体地说：❶标识客户想要的是一个什么样的系统；❷确定风险和效益的可选路线；❸选择最优方案；❹开发系统；❺评估完成情况等；❻重新开始。

即 螺旋模型扩展了增量模型的管理任务范围。而增量模型是基于以下假定：需求是最基本的、并且是唯一的风险源。而在螺旋模型中，决策和降低风险的空间是相当广泛的。

3、软件工程生存周期过程管理

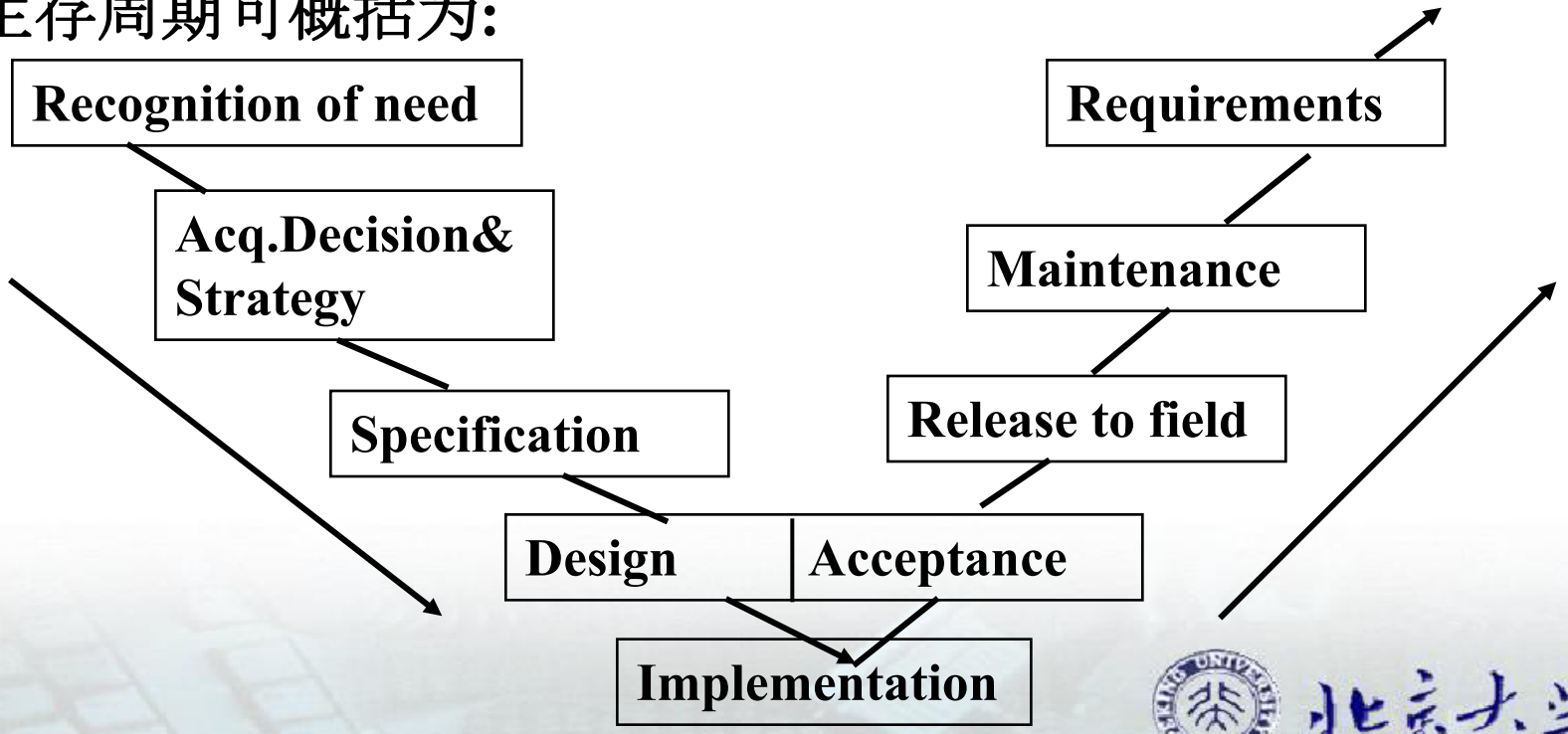


回答:如何建立一项软件工程的生存周期过程并管理之.

1)引言

何谓一个项目的软件生存周期?

无论是软件项还是硬件项，在其开发上的演化一般被称为该项的生存周期。通常，一个项的开发往往始于一个想法，依其服务情况，不断地进行改进。因此,项目的生存周期可概括为:



何谓一个项目的软件生存周期过程?

在一个项目的生存周期中，每一个任务(例如Design)都通过一个或多个过程的方式来完成的, 所有这些相关过程的组合, 称为项目的软件生存周期过程。

注:在这一定义中,关注开发产品所需要的工程技术和管理技术活动,从规约 (Specification)一直到验收(Acceptance).

关于软件开发风范－过程模式

-开发活动的组织:风格与准则

尽管在实践中，每一项软件工程都有自己特定的软件开发过程，但软件开发风范（paradigms）主要有五种，它们是：

❶瀑布（waterfall）风范，即以瀑布模型为基础而形成的项目的生存周期过程；

❷迭代（iterative）风范（也称为演化（evolutionary）风范），即以演化模型、增量模型和喷泉模型为基础而形成的项目的生存周期过程；

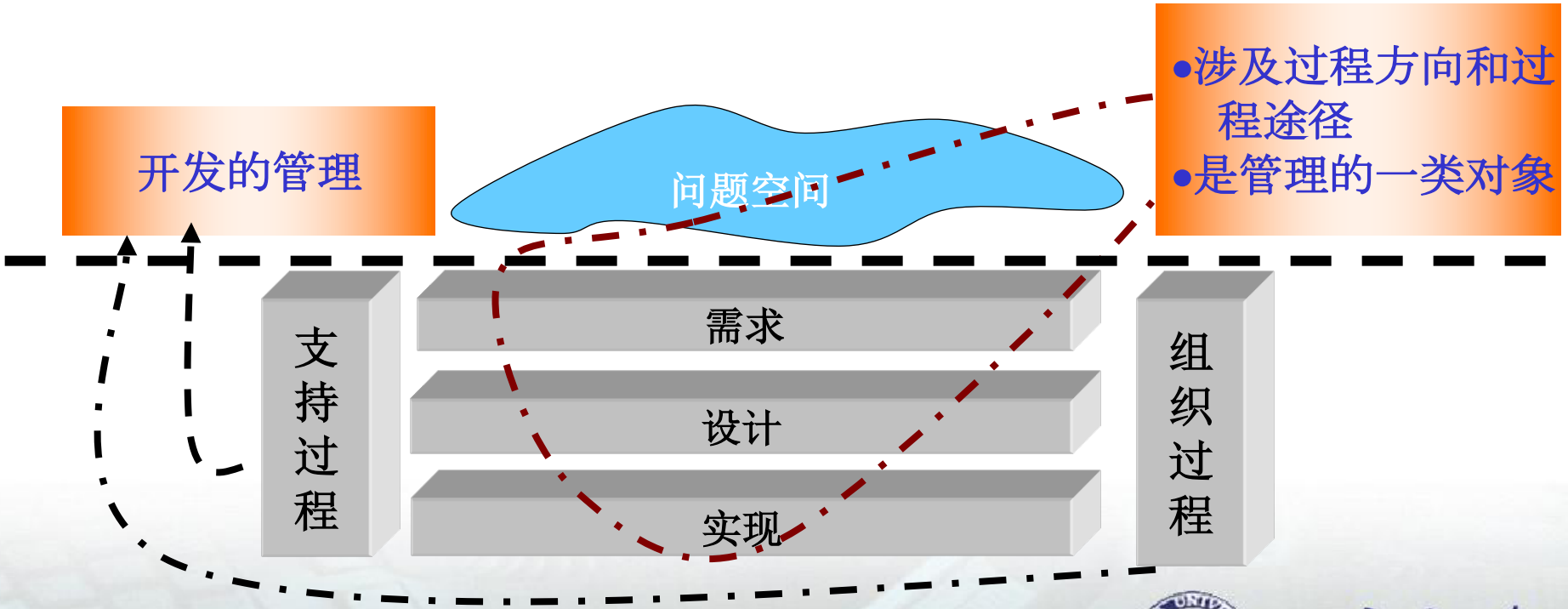
❸螺旋（spiral）风范，即以螺旋模型为基础而形成的项目的生存周期过程；

❹转换（transformational）风范，即基于待开发系统的形式化需求规约为基础，通过一系列转换，将需求规约转化为它的实现而形成的项目的生存周期过程；其中，如果需求规约发生变化的话,可以重新应用这些转换, 对其实现进行更新.

❺第四代（fourth generation）风范，即围绕特定语言和工具，描述待开发系统的高层，并自动生成代码的项目生存周期过程。

软件过程小结：

1）软件开发过程中的活动分为三类,基本过程,支持过程和组织过程,支持过程和组织过程作用于基本过程,以保障开发活动的有效实施.



2）所有软件生存周期模型的基本内在特征为:

❶ 描述了开发的主要阶段；

❷ 定义了每一个阶段要完成的主要活动；

❸ 规约了每一个活动的输入和输出（产品）；

因此可以说,软件生存周期模型提供了一个求解软件的逻辑框架，可以把必要的活动映射到该框架中。

3）软件开发诸过程中的活动及其定序，确定了软件工程的基本过程方向，是创建软件项目产品的“机制”,是求解软件的逻辑。

第三章

需 求 规 约

北京大学

1 如何认识需求工作

首先,了解现代系统中软件的作用

在许多现代技术产品中，最重要的技术是软件技术。软件通常为一些产品：

--提供了控制功能

--提供耦合功能,

--提供一些由软件本身所实现功能,

即软件是现代系统中的重要元素, 使这些产品/系统成为用户的解决方案。

2 需求与需求获取

1）定义

一个需求是一个有关“要予构造”的陈述，描述了待开发产品／系统（或项）功能上的能力、性能参数或者其它性质。

A requirement is a statement that has been constructed to describe a necessary functional capability,performance parameter, or other property of the intended product(or item).

2)什么样的陈述可以作为需求

--需求的基本性质

IEEE标准830-1998要求单一需求必须具有5个基本性质：

①必要的(Necessary)。是要求的吗？

②无歧义的(Unambiguous)。只能用一种方式解释吗？

③可测的(testable)。可以对它进行测试吗？

④可跟踪的(Traceable)。可以从一个开发阶段到另一个阶段对它进行跟踪吗？

⑤可测量的(Measurable)。可以对它进行测量吗？

注:确定一个需求是否满足以上五个性质是复杂耗时的过程.

3) 需求分类

功能； 性能；外部接口；设计约束； 质量属性。

❶ 功能需求

功能需求规约了系统或系统构件必须执行的功能。

例如

系统应对所有已销售的应纳税商品计算销售税。

系统应提供一种方法，使系统用户可根据本地利率调整销售税比例.

系统应能够产生月销售报表。

❷ 性能需求

性能需求(Performance requirement)规约了一个系统或系统构件必须具有的性能特性。例如：

系统应该在5分钟内计算出给定季度的总销售税。

系统应该在1分钟内从100000条记录中检索出一个销售定单。

该应用必须支持100个Windows 95/NT工作站的并行访问。

注：性能需求隐含了一些满足功能需求的设计方案，经常对设计产生一些关键的影响。例如：排序，关于花费时间的规约将确定哪种算法是可行的。

❸外部接口需求

外部接口需求(External interface requirement)规约了系统或系统构件必须与之交互的硬件、软件或数据库元素。它也可能规约其格式、时间或其他因素。

例如：

账户接收系统必须为月财务状况系统提供更新信息，如在“财务系统描述”第4修订版中所描述的。

引擎控制系统必须正确处理从飞行控制系统接收来的命令，符合接口控制文档B2-10A4，修订版C的1到8段的规定。

--用户接口(User interfaces)：规约了软件产品和用户之间接口的逻辑特性。即规约 对给用户所显示的数据，对用户所要求的数据以及用户如何控制该用户接口。

--硬件接口(Hardware interfaces)：如果软件系统必须与硬件设备进行交互，那么就应说明所要求的支持和协议类型。

--软件接口(Software interfaces)：允许与其它软件产品进行交互，如，数据管理系统、操作系统或数学软件包。

--通讯接口(Communications interfaces)：规约待开发系统与通讯设施(如，局域网)之间的交互。如果通讯需求包含了系统必须使用的网络类型（TCP/IP，WindowsNT,Novell），那么有关类型的信息就应包含在SRS中。

--内存约束(Memory constraints)：描述易失性存储和永久性存储的特性和限制，特别应描述它们是否被用于与一个系统中其它处理的通讯。

--操作(Operation)：规约用户如何使系统进入正常和异常的运行以及在系统正常和异常运行下如何与系统进行交互。应该描述在用户组织中的操作模式，包括交互模式和非交互模式；描述每一模式的数据处理支持功能；描述有关系统备份、恢复和升级功能方面的需求。

--地点需求(Site adaptation requirements)：描述系统安装以及如何调整一个地点，以适应新的系统。

❹设计约束

设计约束限制了系统或系统构件的设计方案。就约束的本身而言，对其进行权衡或调整是相当困难的，甚至是不可能的。它们必须予以满足。这一性质，是与其它需求的最主要差别。为了满足功能、性能和其它需求，许多设计约束将对软件项目规划、所需要的附加成本和工作产生直接影响。例如：

系统必须用C++或其他面向对象语言编写。 系统用户接口需要菜单。

任取10秒，一个特定应用所消耗的可用计算能力平均不超过50%。

必须在对话窗口的中间显示错误警告，其中使用红色的、14点加粗Arial字体。

针对产品开发，为确定其相关的设计约束，一般需要考虑以下10个方面：

--法规政策(Regulatory policies)；

--硬件限制(Hardware limitations)，例如：处理速度、信号定序需求、存储容量、通讯速度以及可用性等；

--与其它应用接口(Interfaces to other applications)，如，当外部系统处于一个特定状态时，禁止新系统某些操作

--并发操作(Parallel operations)，例如，可能要求从/自一些不同的源，并发地产生或接收数据。对此，必须清晰地给出有关时间的描述。

--审计功能(Audit functions)，规约软件系统必须满足的数据记录准则或事务记录准则。如，如果用户察看或修改数据，那么就可能要求该系统为了以后复审，记录该系统的动作。

--控制功能(Control functions)：可以对系统的管理能力进行远程控制、可以对其他外部软件以及内部过程进行控制。

--高级语言需求(Higher order language requirements)：

--握手协议(Signal handshake protocols)：通常用于硬件和通讯控制软件，特别当给出特定的时间约束时，一般就要把“握手协议”作为一项约束。

--应用的关键程度(Criticality of the application)，许多生物医学、航空、军事或财务软件属于这一类。

--安全考虑(Safety and security considerations)。

❺ 质量属性

质量属性(Quality attribute)规约了软件产品必须具有的一个性质是否达到质量方面一个所期望的水平。例如：

属性	描述
可靠性	软件系统在指定环境中没有失败而正常运行的概率。
存活性	当系统的某一部分系统不能运行时，该软件继续运行或支持关键功能的可能性。
可维护性	发现和改正一个软件故障或对特定的范围进行修改所要求的平均工作。
用户友好性	学习和使用一个软件系统的容易程度。
安全性	在一个预定的时间内，使软件系统安全的可能性。
可移植性	软件系统运行的平台类型。

4）需求发现

常用的发现初始需求的技术，包括：

❶自悟(Introspection)

需求人员把自己作为系统的最终用户，审视该系统并提出问题："如果是我使用这一系统，则我需要…"

适用条件：需求工程师不能直接与用户进行交流，自悟是乎是一种比较有吸引力的方法，可能确实是必须的。

成功条件：若使自悟是成功的，需求人员必须具有比最终用户还要多的应用领域和过程方面的知识，并具有良好的想象能力。

❷交谈(Individual interviews)

为了确定系统应该提供的功能，需求人员通过提出问题，用户回答，直接询问用户想要的是一个什么样的系统。

成功条件：交谈通常是一种比自悟更好的技术。这种途径成功与否依赖于：

——需求人员是否具有“正确提出问题”的能力，

——回答人员是否具有“揭示需求本意”的能力。

存在的风险：在交谈期间需求可能不断增长，或是以前没有认识到的合理需求的一种表现，说是“完美蠕行” (Creeping elegance)病症的体现，以至于很难予以控制，可能导致超出项目成本和进度的限制。

❸观察(Observation)

通过观察用户执行其现行的任务和过程，或通过观察他们如何操作与所期望的新系统有关的现有系统，了解系统运行的环境，特别是了解要建的新系统与现存系统、过程以及工作方法之间必须进行的交互。尽管了解的这些信息可以通过交谈获取，但“第一手材料”一般总是能够比较好的“符合现实”的。

存在的风险：

——客户可能抵触这一观察。其原因是他们认为开发者打扰了他们的正常业务。

——客户还可能认为开发者在签约之前，就已经熟悉了他们的业务。

❹小组会(Group session)

举行客户和开发人员的联席会议，与客户组织的一些代表共同开发需求。其中：

－通常是由开发组织的一个代表作为首席需求工程师或软件工程项目经理，主持这一会议。但还可以采用其它形式，这依赖于其应用领域和主持人的能力。主持人的作用主要是掌握会议的进程。

－必须仔细地选择该小组的成员，不仅要考虑他们对现存的和未来运行环境的理解程度，还要考虑他们的人品。

❺提炼(Extraction)

复审技术文档（例如，有关需要的陈述，功能和性能目标的陈述，系统规约接口标准，硬件设计文档以及ConOps文档），并提出相关的信息。

适用条件：提炼方法是针对已经有了部分需求文档的情况。依据产品的本来情况，可能有很多文档需要复审，以确定其中是否包含相关联的信息。在有的情况，也可能只有少数文档需要复审。

在许多项目中，在任何交谈、观察、小组会或自悟之前，应该对该项目的背景文档进行复审，还应对系统规约进行复审，同时了解相关的标准和政策。

2 需求规约（ＳＲＳ）及其格式

1)概念

一个需求规约是一个软件项/产品/系统所有需求陈述的正式文档， 是一个软件产品／系统的概念模型。

2) 基本性质

一般来说，SRS应必须具有以下4个性质：

①重要性和稳定性程度(Ranked for importance and stability)。

②可修改的(Modifiable)。在不过多地影响其它需求的前提下， 可以容易地修改一个单一需求.

③完整的(Complete)。没有被遗漏的需求.

④一致的(Consistent)。不存在互斥的需求.

SRS所不能实现的作用

第一，它不是一个设计文档。它是一个“为了”设计的文档

第二，它不是进度或规划文档，不应该包含更适宜包含在

工作陈述（SOW）、软件项目管理计划(SPMP)、软件生存周

期管理计划(SLCMP)、软件配置管理计划(SCMP)或软件质

量保证计划(SQAP)等文档中的信息。因此，在SRS中不应给

出： 项目成本； 交付进度； 报告规程；

软件开发方法；质量保证规程；配置管理规程；

验证和确认规程；验收规程；安装规程。

第四章　系统建模技术-结构化方法

一、结构化分析方法

要回答：如何定义问题？

就如何定义问题而言，如何获得需求

如何规约需求

如何验证需求

1、关于需求获取

需求面临的挑战

- 问题空间理解
- 人与人之间的通信
- 需求的不断变化

需求获取技术- Use Case

Use Case (Ivar Jacobson,1994)

1）引言

- **Use Case主要用于促进和用户的交流、沟通。为此使用了一种用户和开发人员都能理解方式描述系统功能和行为。**
- **Use Case可以划分系统与外部实体的界限，是系统开发的起点，而最终应该落实到类和实现代码上。**
- **Use Case既然是对系统行为的动态描述，因此它是类、对象、操作的来源，是系统分析和设计阶段的输入之一，是分析和设计，制定开发计划，测试计划，设计测试用例的依据之一。**
- **Use Case Model是系统需求分析阶段的成果之一。**
- **Use Case不但有助于帮助分析员理清思路；验证用户需求。而且，也是开发人员之间进行交流的重要手段。**

2）语义与表示

一般地说，USE CASE是用户为了达到某一目标和系统进行的典型交互。例如：

“做一次拼写检查”

“对一个文档建立索引”

对一个用况而言，关键要素是：表示一种用户可以理解并对该用户有价值的功能。

用况提供了客户和开发人员在制订项目计划中进行交流的主要成分。

（1） USE CASE语义

一个USE CASE是系统或其它语义实体（例如子系统或一个类）所提供的一块(unit)高内聚的功能，显露该系统和一个或多个外部的交互者（称为操作者）交替出现的消息序列，以及该系统所执行的动作。

可见，一个USE CASE捕获了参与交互的各方关于其行为的一个约定。通过这一约定，描述了该语义实体在不同条件下的行为对参与者一个要求的响应，以实现某一目的。不同的行为序列，依赖于所给出的特定要求以及与这些要求相关的条件。

（2）表示与描述

USE CASE通常被表示为：

USE CASE包含一组操作和属性，这些操作和属性规约了该USE CASE的实例所执行的那个动作序列。动作包含状态的改变以及该USE CASE与其环境的通讯。

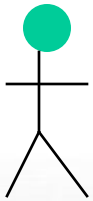
为了表明USE CASE所包含的具体内容，还应给出它的正文描述。

3）操作者语义与表示

一个操作者定义了一组高内聚的角色，当用户与该实体交互时，用户可以扮演这一角色。

对于每一USE CASE，一个操作者有一种角色，即每一USE CASE与具有一种角色的操作者进行通讯。

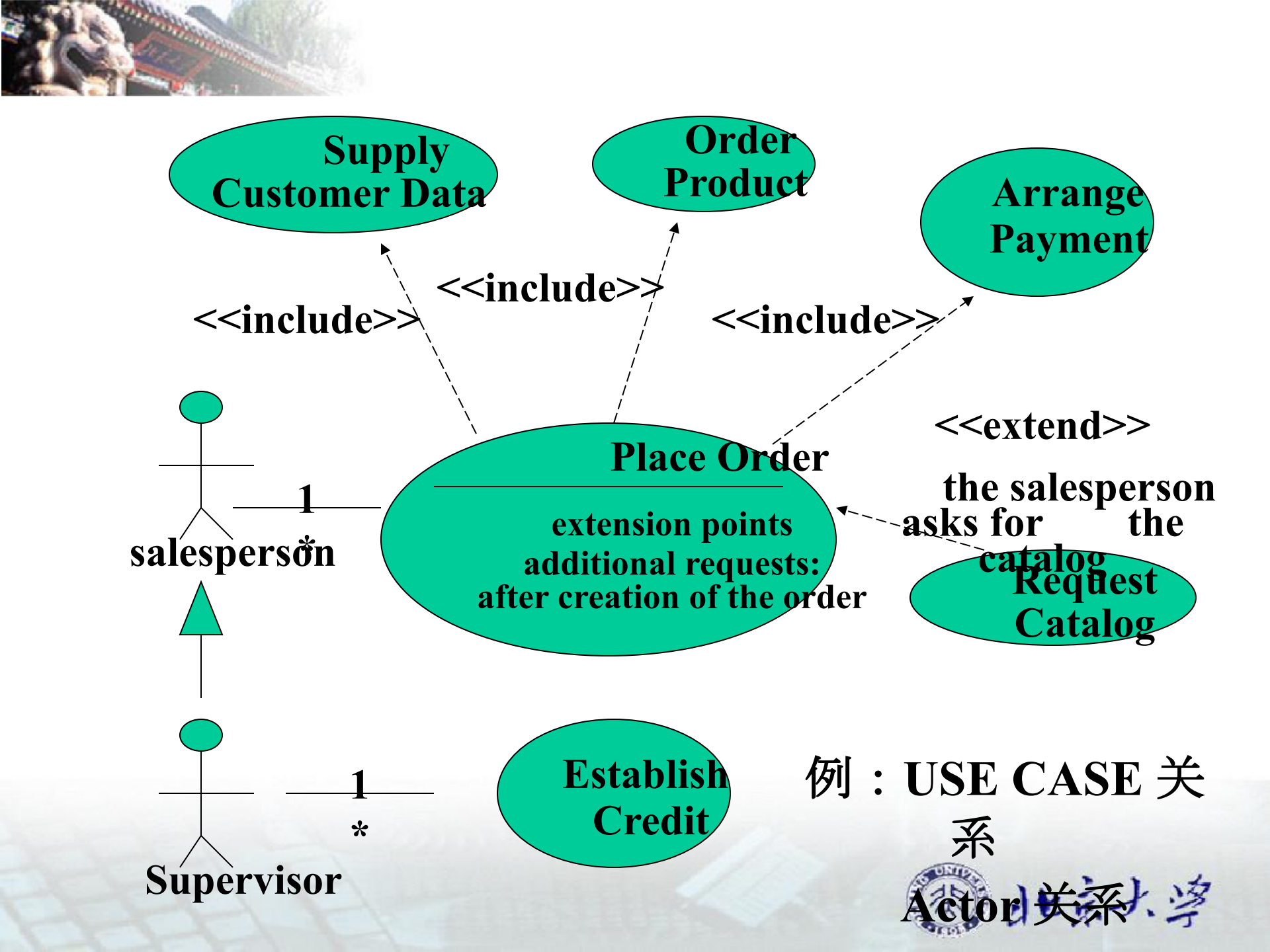
通常，一个操作者被表示为：



5）关系

在USE CASE之间，或在操作者与USE CASE之间，存在一些标准的关系：

- 关联：参与关系，即操作者参与一个USE CASE。例如，操作者的实例与USE CASE实例相互通讯。关联是操作者和USE CASE之间的唯一关系。
- 扩展：USE CASE A到USE CASE B的一个扩展关系，指出了USE CASE B的一个实例可以由A说明的行为予以扩展（根据该扩展所说明的特定条件），并依据该扩展点定义的位置，A说明的行为被插入到B中。
- 包含：USE CASE A到USE CASE B的一个包含，指出A的一个实例将包含B说明的行为，即这一行为将包含在A定义的那部分中。
- 泛化：USE CASE A到USE CASE B的泛化，指出A是B的特殊情况。



例：USE CASE 关系

Actor关系

6）USE CASE图

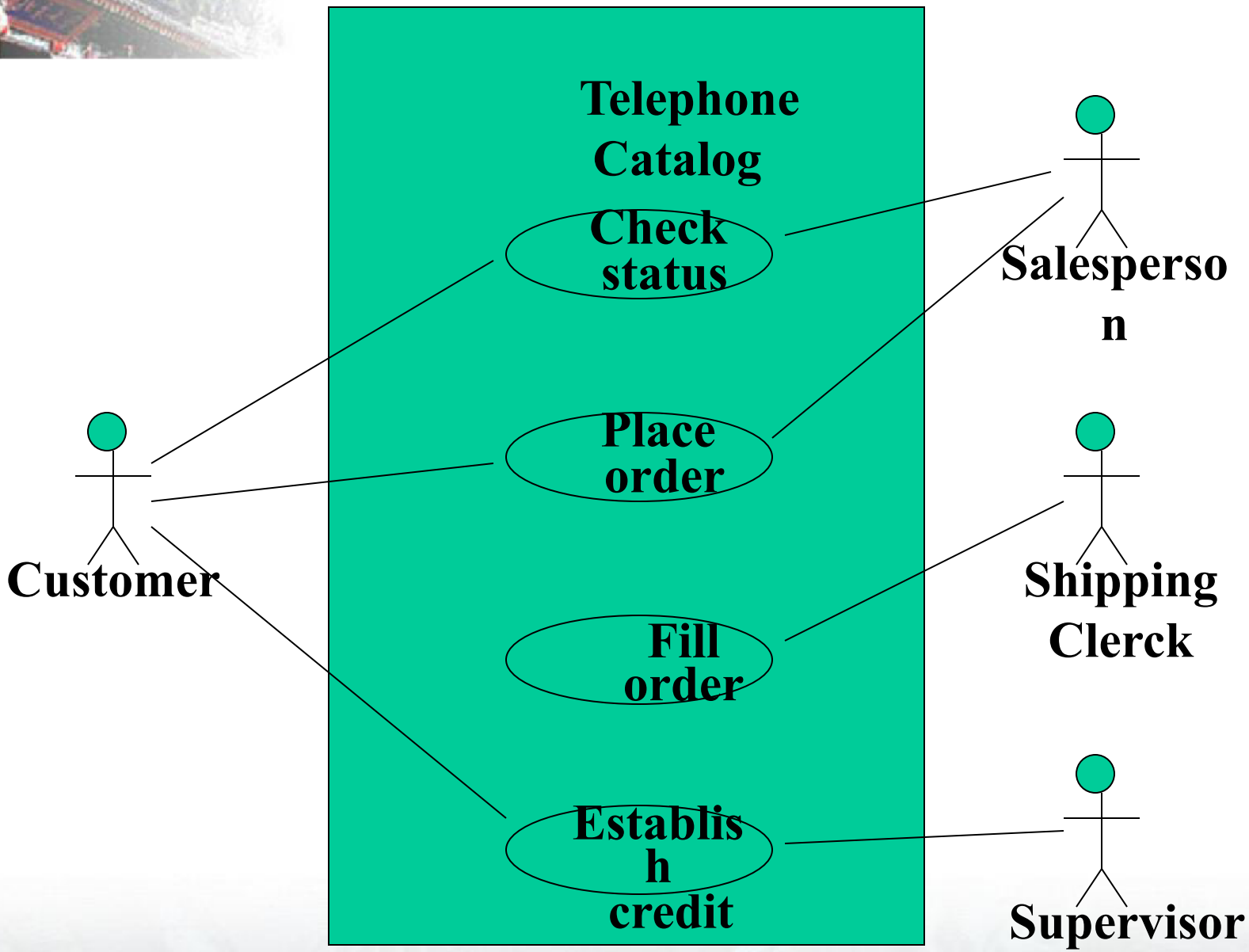
USE CASE图给出了操作者和USE CASE以及它们之间的关系。即图中给出了一些操作者、一组关系、一些接口和这些元素之间的关系。

(关系是操作者和USE CASE之间的关联

是操作者之间的泛化

是USE CASE之间的泛化、扩展和包含)

可以将一些USE CASE用一矩形括起，以表示所包括的那个系统或其它语义实体的边界。



例：USE CASE图

7）使用USE CASE图的建模类型

使用USE CASE图所建造的模型类型，可以从两个层面上进行分类，它们是“整体/部分”关系

- 系统建模（system modeling ）
  - 系统建模用于描述软件系统的结构和行为
- 业务建模（business modeling）
  - 业务建模用于企业或组织过程的优化和再工程（process re-engineering）
  - 业务建模的图形元素不仅包括普通的Actor和Use Case，还包括Worker、Artifact，Business
  - 过程描述时还应结合时序图(sequence diagram)和活动图(activity diagram)

2、关于需求规约

需求规约的主要目标：

依据需求陈述（作为输入），解决其中的歧义、不一致等问题，以系统化的形式表达用户的需求，即给出问题的形式化或半形式化的描述（建立模型），形成需求规格说明书。为了实现这一目标，

（一）结构化分析方法

1〕提出的概念有：

数据流：——加工：

数据存储：——数据源：　　数据潭：

概念是完备的。

2〕建模过程

(1)建立系统的功能模型

---使用的工具为数据流图DFD

首先：建立系统环境图，确定系统边界

继之：自顶向下，逐层分解

(2)建立数据字典

定义数据流　　定义数据存储

定义数据项

(3)给出加工小说明

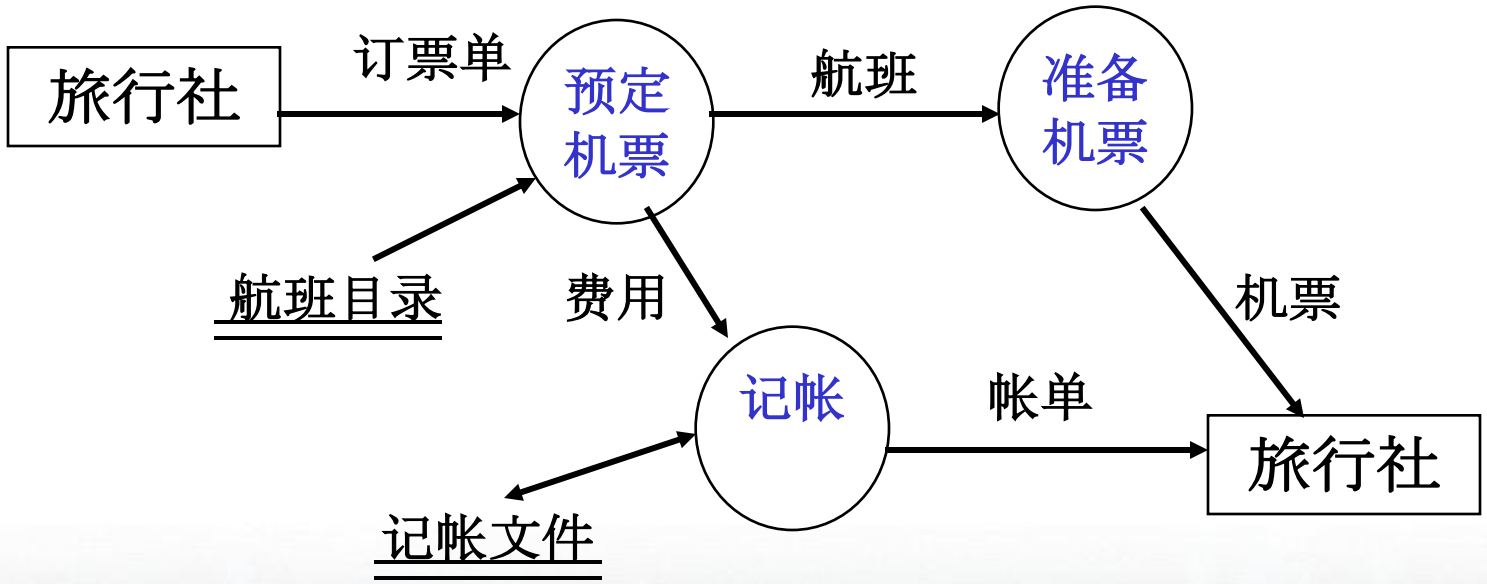
---使用的工具可以为判定表

判定树

(1)建立系统的功能模型

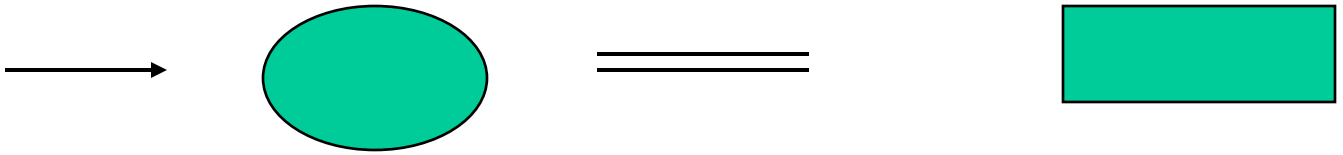
---使用的工具为数据流图DFD

数据流图：是一种描述数据变换的图形工具。例如：



数据流图由四个基本成分组成：

数据流　加工　数据存储　数据源和数据潭



其中：1 各成分的定义

2 数据流、数据存储---支持数据抽象

加工---支持过程/功能的抽象

3 关于命名问题

首先：建立系统环境图，确定系统边界

------顶层DFD

其中：1 数据流为：销售的商品， 日销售额等

3个输入流，3个输出流

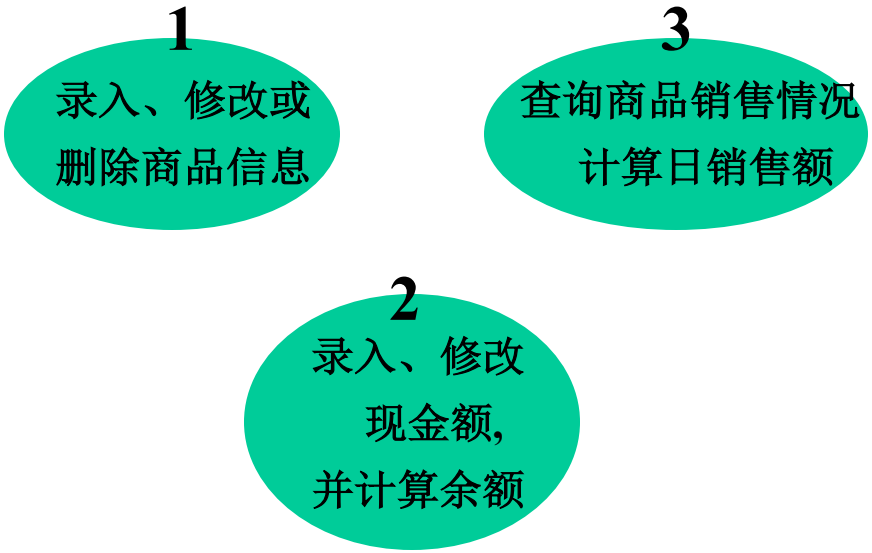
数据源为：营业员，经理，收款员

数据潭为：经理，收款员

2 加工名为：要建立的系统名字

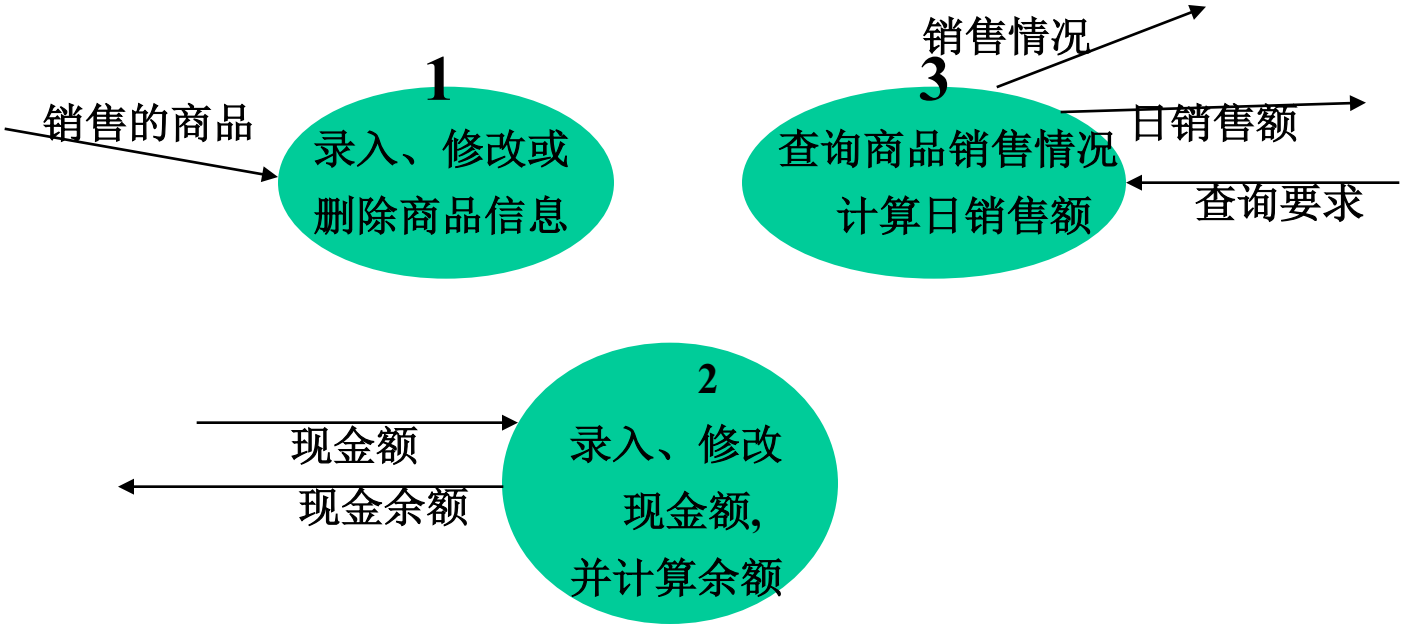
继之：自顶向下，逐层分解

A、按人或部门的功能要求，将加工“打碎”，形成：



注：需给每一加工编号；

B、“分派”数据流，形成：

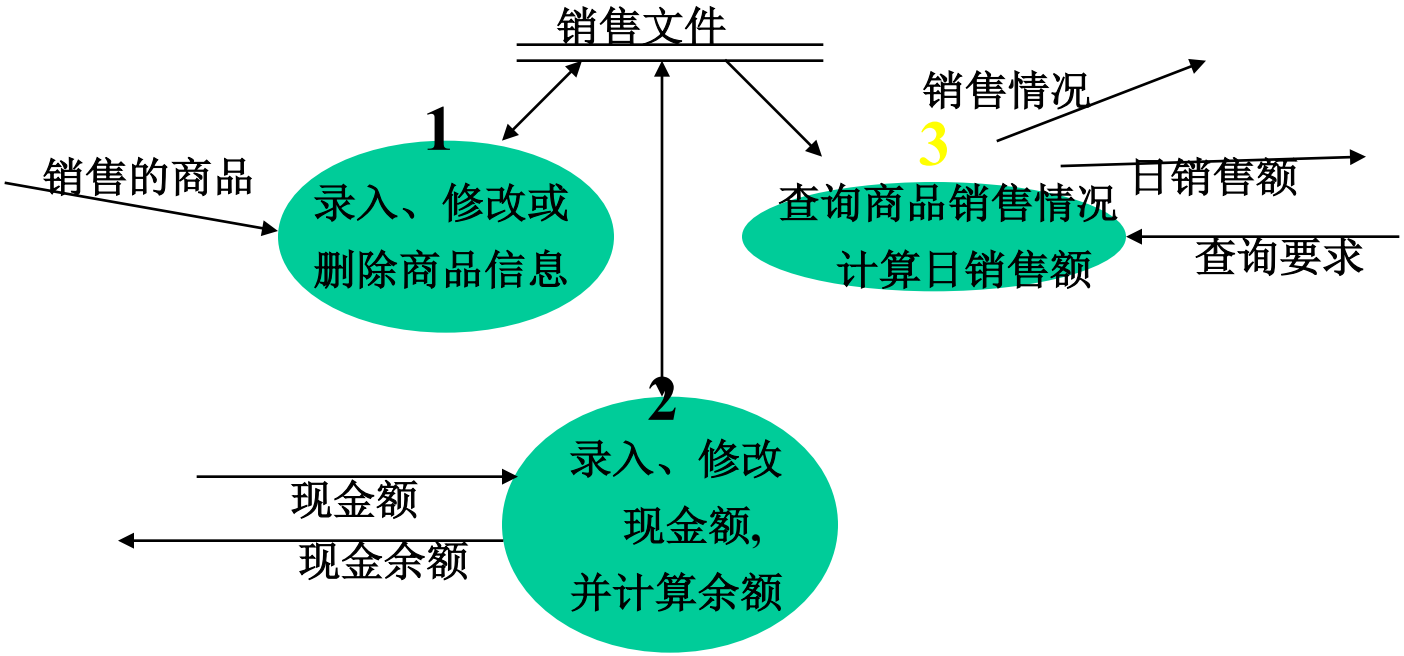


其中：要根据特定的加工要求进行分派；

保持与顶层数据流的一致；

可以不引入数据源和数据潭。

C、引入文件，使之形成一个有机整体—系统：

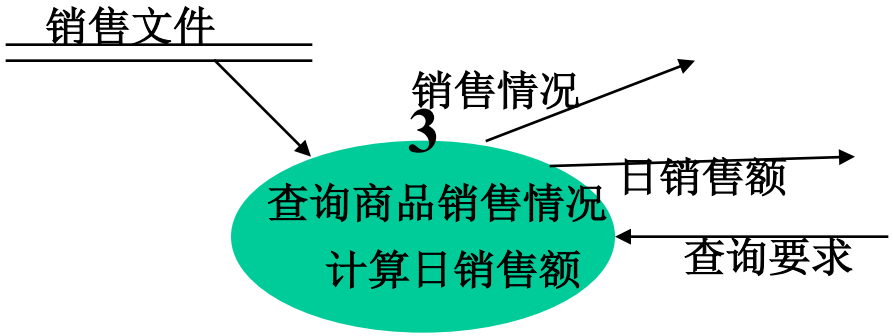


注：到一个文件，既有输入流，又有输出流，则可简化为

，并可不给出标识。

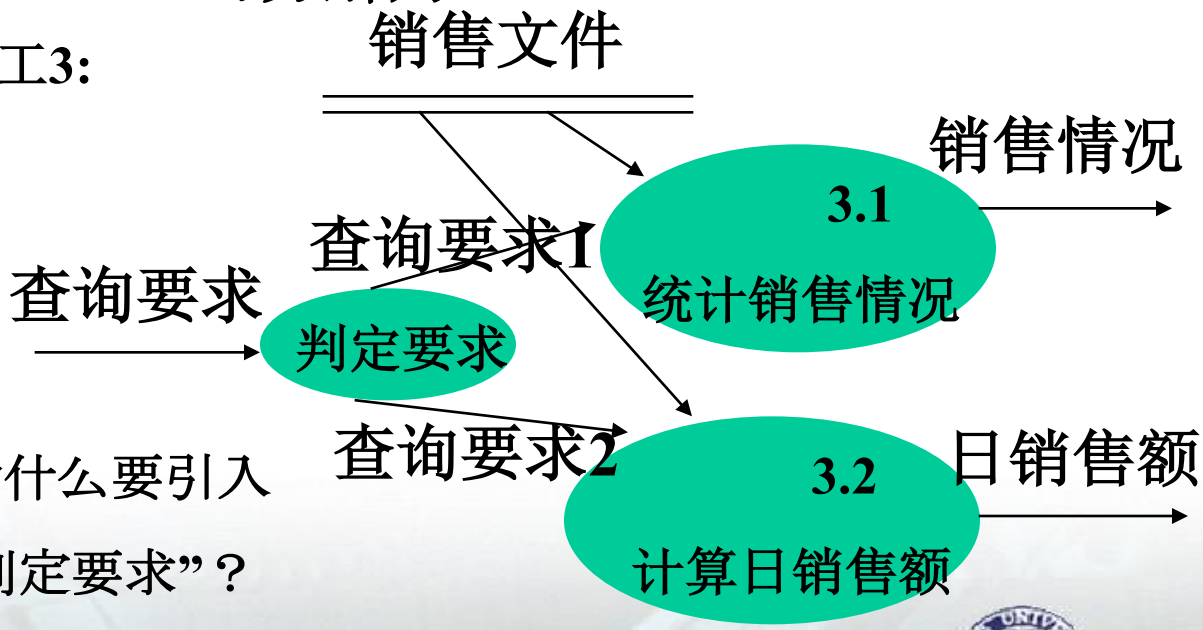
至此，体现精化，形成0层数据流图。

继续A、B、C：自顶向下，逐层分解。例如：加工3



可分解为：

加工3:



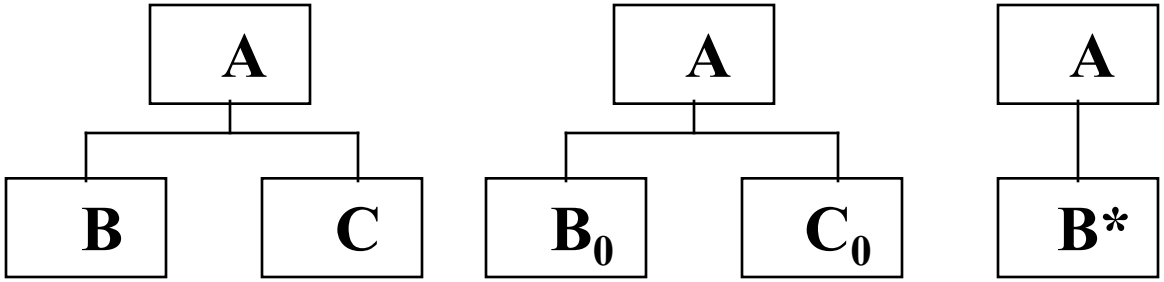
*其中为什么要引入加工“判定要求”？

(2)建立数据字典

定义数据流　定义数据存储　定义数据项

引入：结构符　＋　　|　　{ }

—用于定义数据结构



数据字典:

1、数据流:

销售的商品=商品名+商品编号+单价+数量+日期

现金额 = 余额 =日销售额=非负实数

查询要求=[商品编号|日期]

查询要求1=商品编号

查询要求2=日期

销售情况=商品名+商品编号+金额

2、数据存贮:

销售文件={销售的商品}

3、数据项

(3)给出加工小说明

---使用的工具可以为判定表

判定树

判断表　Ⅰ 条件类别　Ⅱ 条件组合

Ⅲ 操作　Ⅳ 操作执行

例如：

考试总分	>=620	>=620	<620
单科成绩	有满分	有不及格	有满分
发升级通知书	y	y	n
发留级通知书	n	n	y
发重修通知书	n	y	n

3〕建模中注意的问题

（1）模型平衡规则

· 父图和子图必须平衡

· 每个数据流和数据存储必须在数据字典中予以定义

· “叶”加工（最低层）必须给出加工小说明

· 小说明和数据流图的图形表示必须一致，例如：

在小说明中，必须说明“输入数据流”如何使用，必须说明如何产生“输出数据流”，必须说明如何选取、使用、修改“数据存储”

（2）控制复杂性规则

· 上层数据可以“打包”

上、下数据流对应关系在数据字典中给出，但包内数据流的性质（输入、输出）必须一致。

· 一幅图中的图元个数应控制在7+/-2以内

· 与每一加工相关的数据流的数目应适中

（与层次有关）

· 分析数据内容，确定是否所有的输入信息都用于产生输出信息；

分析加工，确定一个加工所产生的输出，是否都能由该加工的输入信息导出

有关SRS内容方面：

（1）正确性：指的是SRS中陈述的每个需求是否都表达了系统的某个要求。

（2）无二义性：指的是SRS中陈述的每个需求是否都只有一种解释。

（3）完整性：

· 未来系统所做的任何事情都包含在SRS的陈述中；

· 未来系统响应所有可能的输入（包括有效和无效）；

· SRS中没有被标识为“待定”的内容。

（4）可验证性：SRS中陈述的每个需求都是可验证的-即当且仅当存在一个有限代价的过程（人工或机器）可以检查构造的软件产品是否符合用户的需求。

（5）一致性：

· SRS中陈述的需求没有与以前的文档发生冲突；

· SRS中陈述的各个需求之间没有发生冲突。

（6）可理解性：

有关SRS格式与风格方面

（7）可修改性：指的是SRS的结构和风格使任何对需求的必要修改都易于完整、一致的进行。

（8）可被跟踪性：指的是SRS中的每个需求的出处都是清楚的，这意味着SRS中包含对前期支持文档的引用表。

（9）可跟踪性：指的是SRS的书写方式有助于对其中陈述的每个需求进行引用。

（10）设计无关性：指的是SRS不暗示特定的软件结构和算法。

二、结构化设计

要回答如何解决问题

—即给出软件解决方案

1〕总体设计的任务：如何将DFD转化为MSD

分二步实现：

第一步：如何将DFD转化为初始的MSD

分类：变换型数据流图

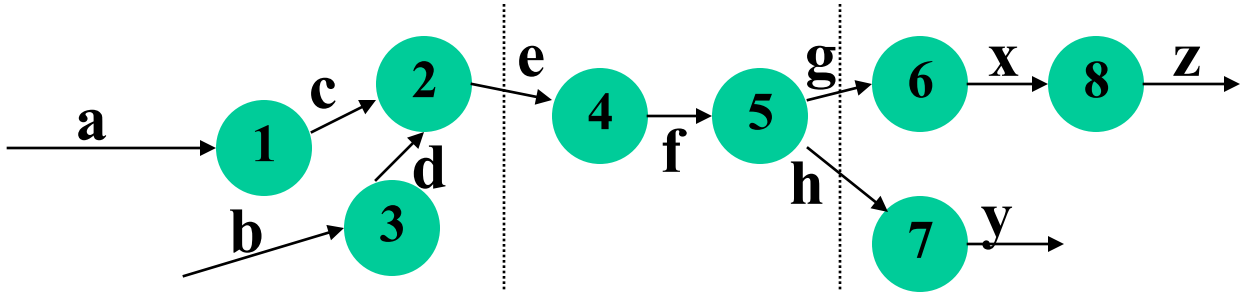
事务型数据流图

变换设计

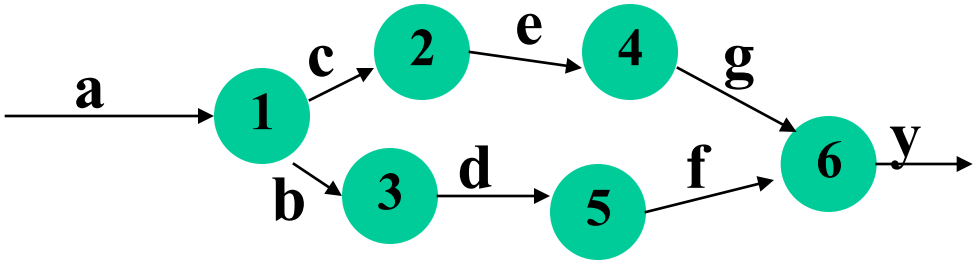
事务设计

●数据流图分类

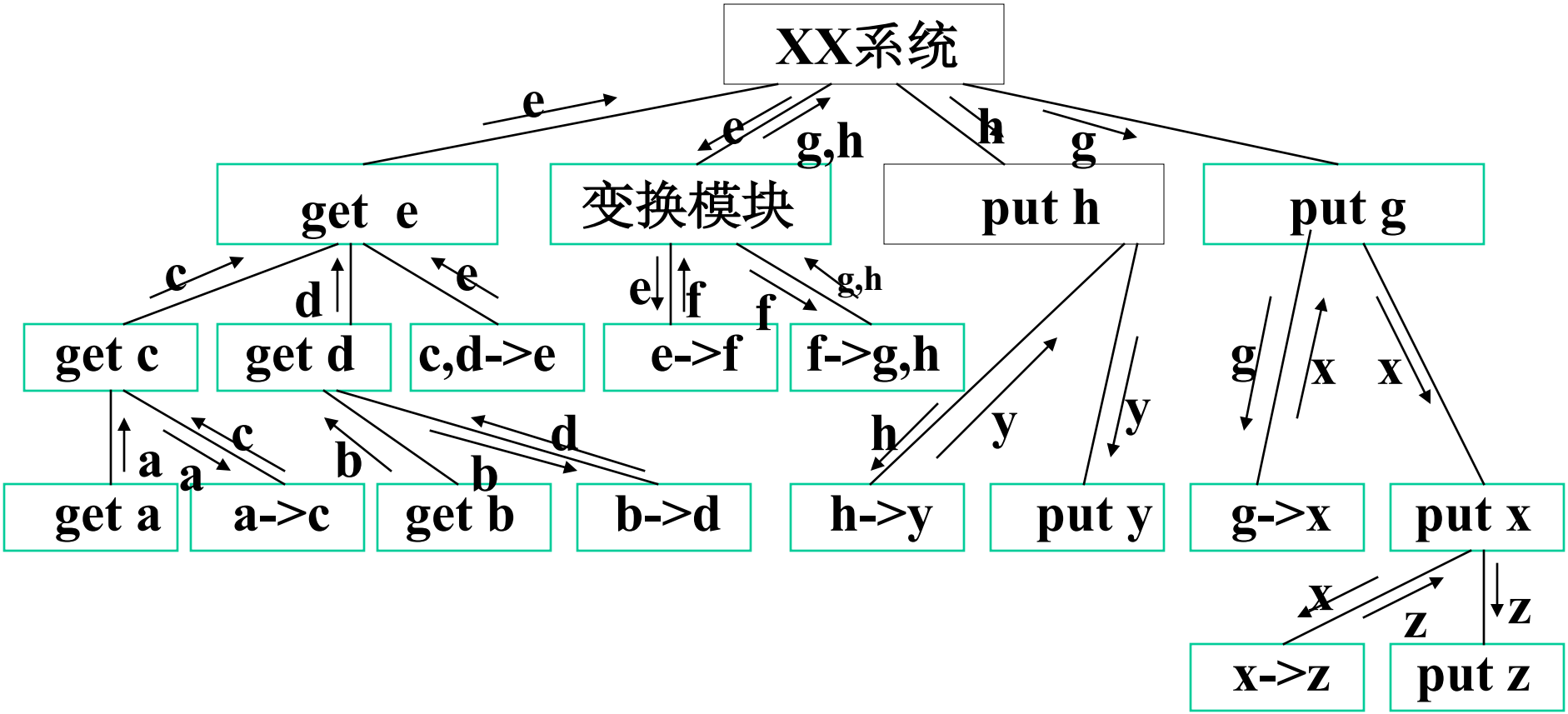
变换型:



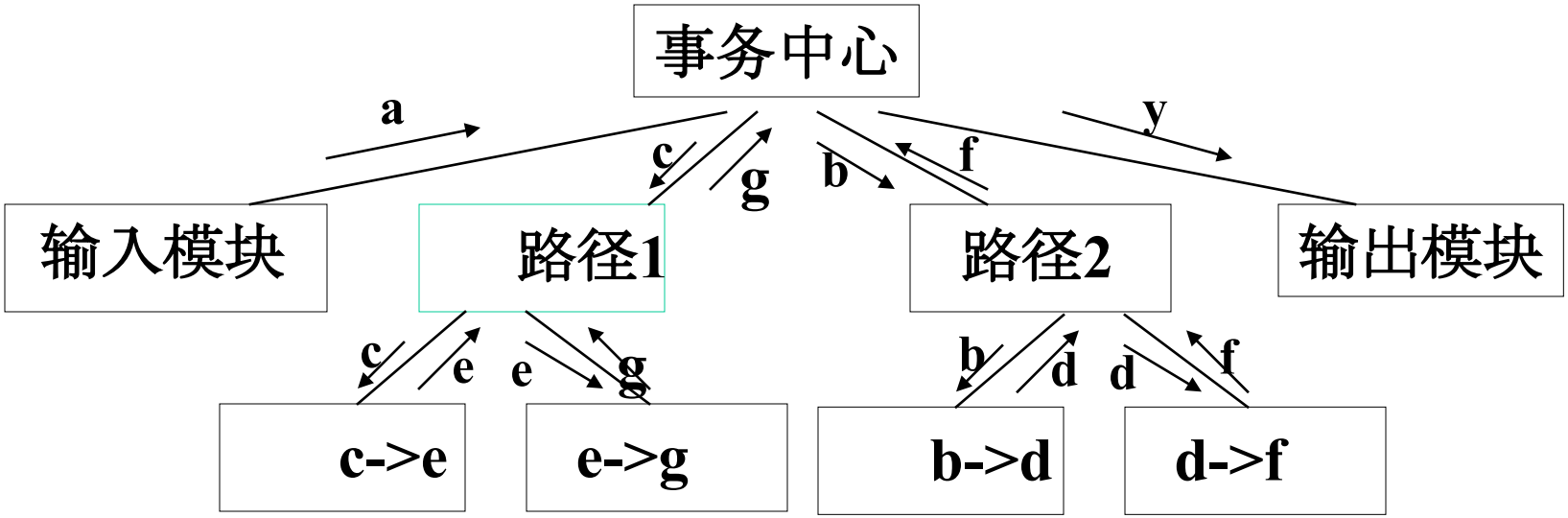
事务型



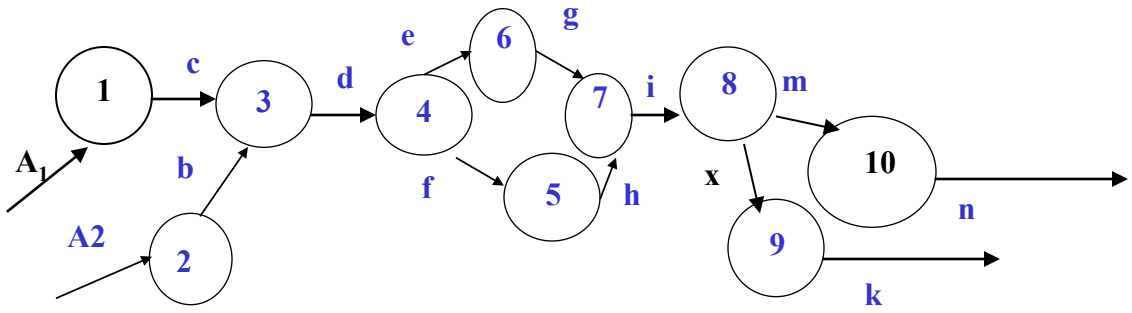
●变换设计


XX系统
e
e
g,h
h
g
get e
变换模块
put h
put g
c
d
e
e
f
f
g,h
get c
get d
c,d->e
e->f
f->g,h
g
x
x
a
a
c
b
b
d
h
y
y
get a
a->c
get b
b->d
h->y
put y
g->x
put x
x
z
z
x->z
put z

事务设计



一个系统的DFD，通常是变换型数据流图和事务型数据流图的组合。如下所示：



第二步：如何将初始的MSD转化为最终可供详细设计使用的MSD

- 概念：模块
- 模块化

  模块化度量：内聚 耦合
- 设计规则－经验规则
- 精化初始的MSD

  －体现设计人员的创造

1）耦合：不同模块之间相互依赖程度的度量。

耦合类型：

（1）内容耦合：

（2）公共耦合：两个以上的模块共同引用一个全局数据项。

（3）控制耦合：一个模块向另一模块传递一个控制信号，接受信号的模块将依据该信号值进行必要的活动。

（4）标记耦合：两个模块至少有一个通过界面传递的公共有结构的参数。

（5）数据耦合：模块间通过参数传递基本类型的数据。

2）内聚：一个模块之内各成分之间相互依赖程度的度量。

内聚类型：

（1）偶然内聚：一个模块之内各成分之间没有任何关系。

（2）逻辑内聚：几个逻辑上相关的功能放在同一模块中。

（3）时间内聚：一个模块完成的功能必须在同一时间内完成，而这些功能只是因为时间因素关联在一起。

（4）过程内聚：处理成分必须以特定的次序执行。

（5）通信内聚：各成分都操作在同一数据集或生成同一数据集。

（6）顺序内聚：各成分与一个功能相关，且一个成分的输出作为另一成分的输入。

（7）功能内聚：模块的所有成分对完成单一功能是最基本的，且该模块对完成这一功能而言是充分必要的。

总体设计小结：

1、总体设计的目标和任务；

2、总体设计的表示：层次图，HIPO图，模块结构图；

3、基本概念：模块，以及由此产生的“鸿沟”；

4、总体设计的基本思想与步骤：

通过：变换设计和事务设计

DFD--→ 初始的MSD （几乎可“机械”地进行）

使用：启发式规则

初始的MSD-→ MSD （体现设计人员的创造）

2〕详细设计的任务：定义每一模块

- 结构化程序设计

三种控制结构:顺序

begin $s_1;s_2;\ldots s_n$ end;

选择

if 条件表达式 then s_1

else s_2;

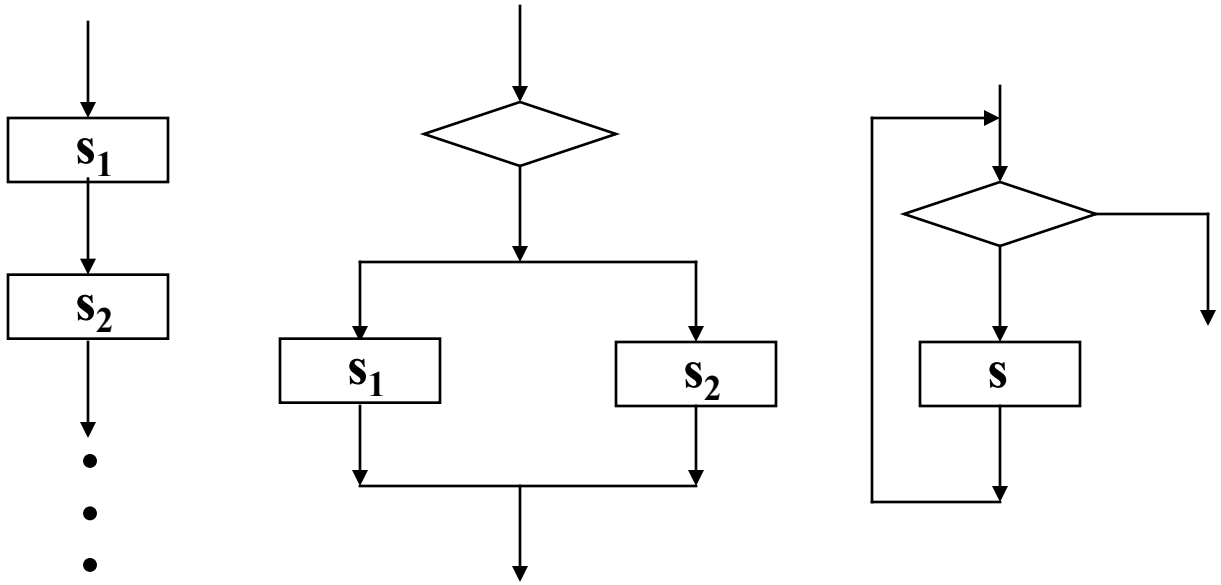
循环

while 条件表达式 do s;

几种表示工具

流程图、PAD、N－S图、伪码等

1)框图



2)伪码

伪码是一种混合语言。外部采用形式语言的控制结构，内部使用自然语言。

Begin

输入一元二次方程的系数a,b,c;

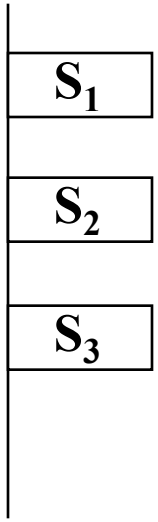
if b^2-4ac≥o then 计算两实根

else 输出无实根；

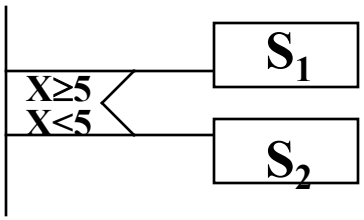
end.

3)PAD图

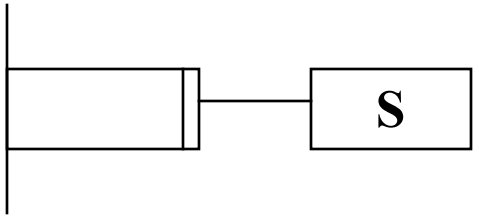
顺序:



选择 :



循环:

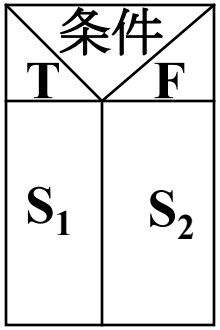


注:支持逐步求精设计

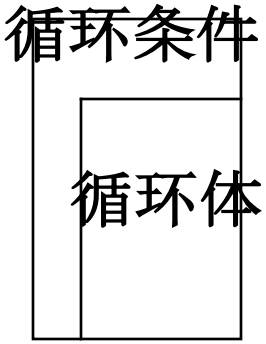
3)N-S图

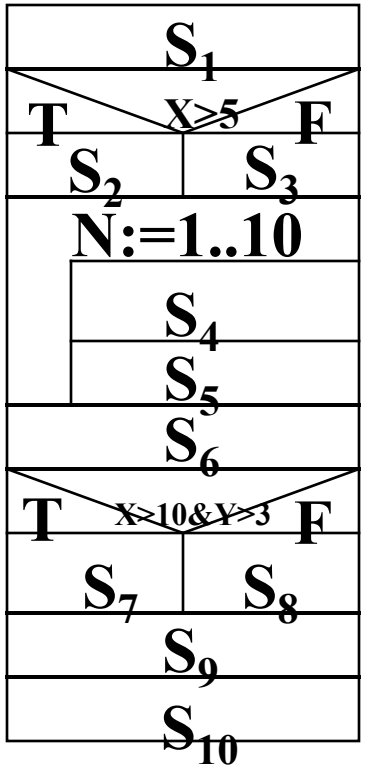
顺序:

选择：



循环:





支持逐步求精设计举例

结构化方法小结

1、结构化方法是一种比较系统的软件开发方法学。

包括：结构化分析和结构化设计

2、紧紧围绕“过程抽象”和“数据抽象”，

给出了　完备的符号体系

---概念与表示

可操作的过程

---步骤与准则

易理解的表示工具

提供了　控制信息组织复杂性的机制，例如

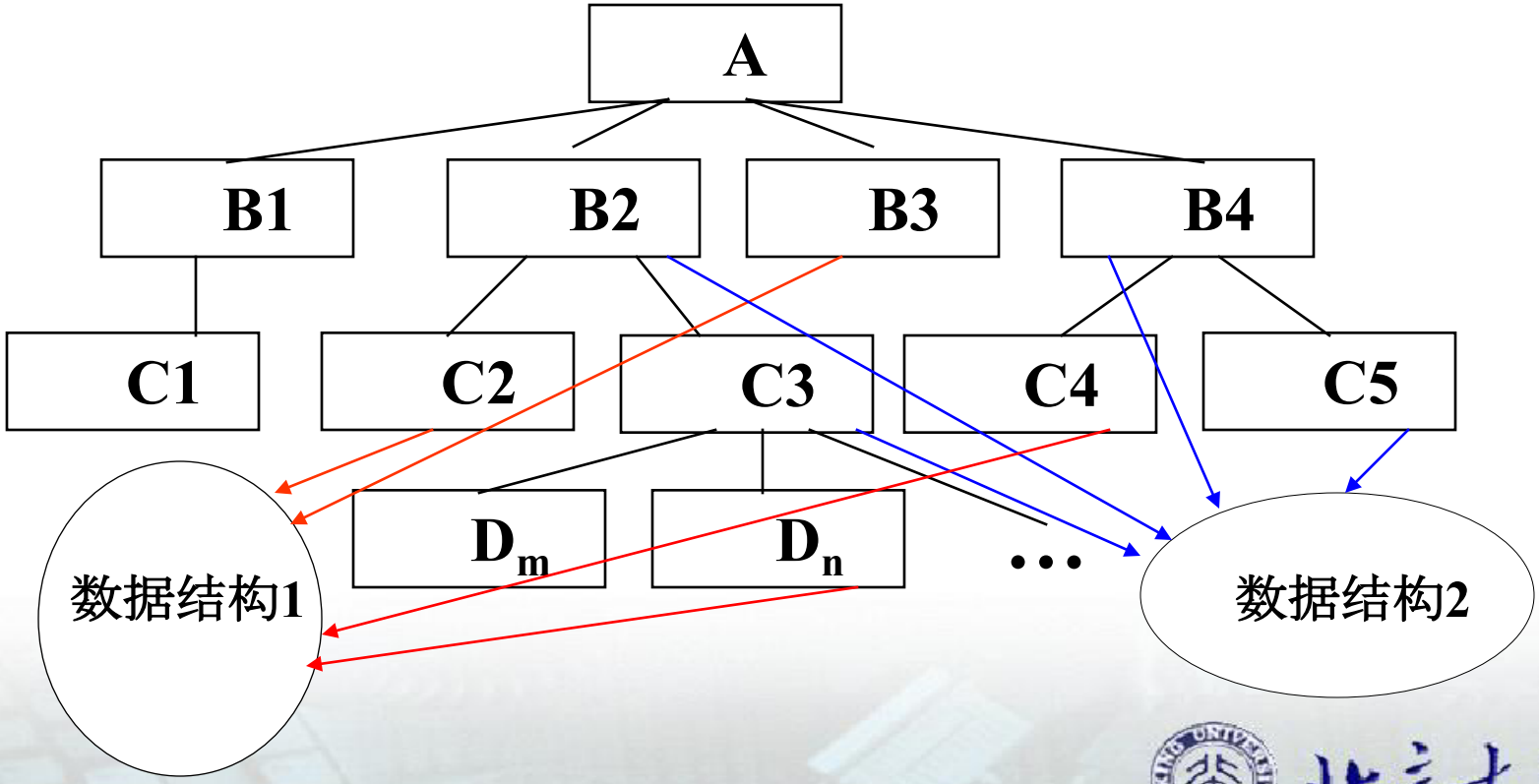
逐层分解，数据打包等

3、问题：捕获的“过程”和“数据”

恰恰是客观事物的易变性质，

解的结构也不保持原系统的结构，

从而：造成　维护，验证上的困难。



●概念

软件方法学－以软件方法为研究对象的学科。主要涉及指导软件设计的原理和原则，以及基于这些原理、原则的方法和技术。狭义的也指某种特定的软件设计指导原则和方法体系。

●从构造的角度，软件开发方法学主要由三部分组成

·NOTATION

·PROCESS

·TOOLS

- 从能力的角度，软件开发方法学应能表达：
  - ·系统的说明性信息
  - ·系统的行为信息
  - ·系统的功能信息，并要给出以下机制：
    - ·控制信息组织复杂性
    - ·控制文档组织复杂性

第五章 面向对象方法-概念与表示

一、引言

1、构造模块的四种基本观点

1）以“过程”或“函数”为基点，使每一模块实现一项功能；

2）以一个“数据结构”为基点，使每一模块实现该数据结构上的操作功能；

3）以“事件驱动”为基点，使每一模块识别一个事件并对该事件作出响应；

4）以“问题域中的一个成分”为基点，使每一模块对应现实世界中的一个事物。

2、OO方法基于的“世界观”：

- 世界是由对象构成的；
- 对象有其自己的属性和内部运动规律；
- 对象之间的相互作用，构成了大千世界的各式各样的不同系统。

面向对象建模技术

如以前所述,建模方法主要包括:建模语言和过程.

建模语言(Modeling Language)是用以表述设计方法的表示法(主要是图形的);

过程(Process)是对设计中所应采取的步骤的意见.

在建模语言方面,“UML已成为一种绘制面向对象设计图的标准工具,并已传播到非面向对象领域.面向对象以前的主要方法已经消逝.UML登场了,并且稳居宝座.” —摘自<UNL精粹>序,徐家福译

1、类图

1）类

根据OO的世界观，对象有其自己的属性和内部运动规律，并与其他对象相互作用。因此，可以在结构上把对象抽象为两个主要部分：

数据和操作

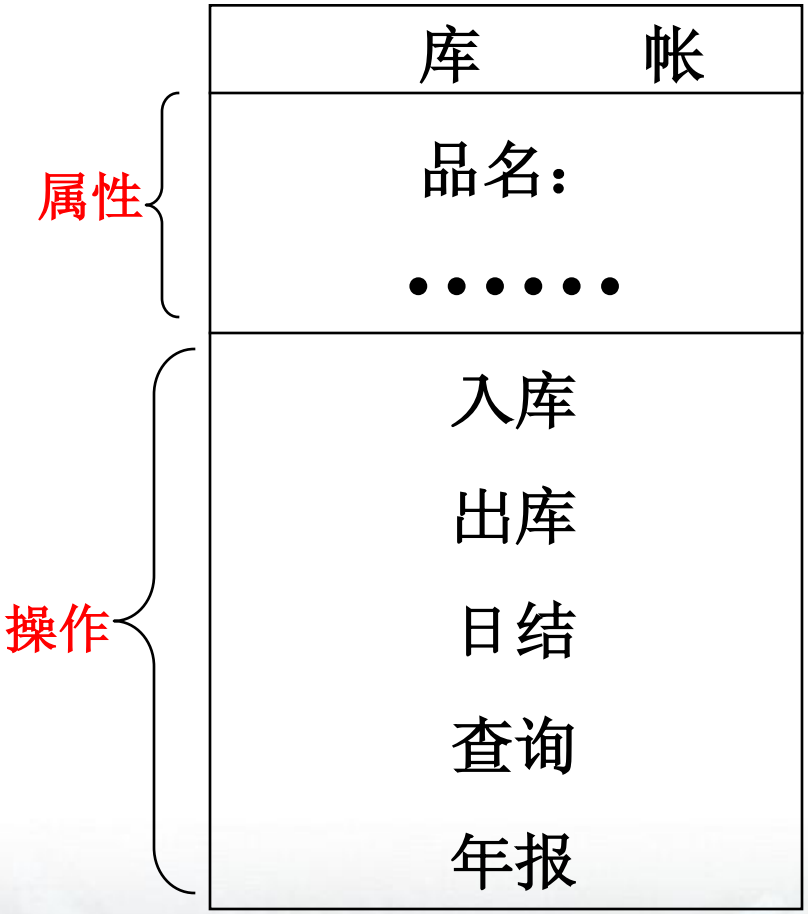
其中，数据用于表达对象的状态，而操作表达在特定状态对外提供的服务（功能）。

于是，现在可以从二个层面来理解“对象”：

概念层：对象是可标识的、可触摸的和可感知的一切事物。

结构层：对象是数据和操作(或谓属性和行为)的封装通信单位.数据表示对象的属性状态，操作(或称方法)决定了对象的行为和与其它对象进行通信的接口。

例如：



为了控制信息组织的复杂性，引入了类的概念：

（2）语义

类是具有相同结构、行为和关系的一组对象的描述符。

表示正被建模系统的一个概念。

就描述问题的方法而言，一般采用“二分法”；例如：《类，实例》

《类型，值》

type array[1..5] of integer;

《主语，特定的名词或名词短语》

……

例如：　•抽象类

•基类，超类

•聚合类：

collection(to hold component objects of a single type and its subtypes.[Firesmith])

container(to hold unrelated component objects of multiple unrelated types.[Booch,Firesmith,Rumbaugh])

structure(class)(component objects of which are interrelated.[Firesmith])

•composite class:any class that composed of other classes.

[Wirfs-Brock]

2）属性

语义

表示对象状态的一组值。

Any named property of an object that takes a literal as its value,defines the abstract state of its object,and appear within the interface rather then the implementation.

[ODMG]

Any named property used as a data abstraction to describe its enclosing object,class,or extent.

[Firesmith,Rumbaugh]

3）操作：为其它对象提供的服务。

语义

-any service that can be requested.

[Jacobson,Martin,OMG]

-Any discrete activity,action,or behavior that is performed by (I.e.,belongs to) an object or class.

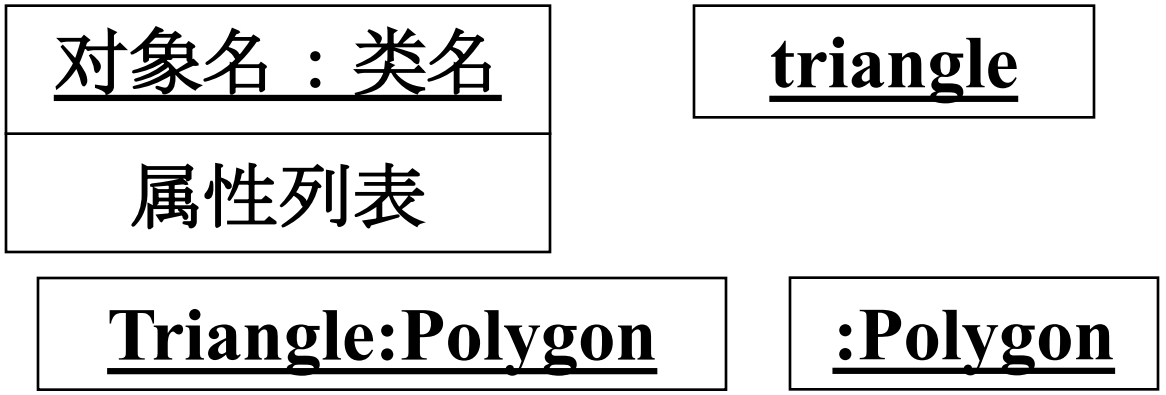
[Ada95,Firesmith,Hender-Sellers]

同义词：**function,method,service.**

对象：（1）语义

对象是类的一个实例。

（2）表示



名字的其他情况：display-window：WindowSystem::

GragicWindows::Window

a:b,c,d(规则：在一个时刻只属于一个实现类)

5）接口

(1)语义(仅以类为例)

接口描述类、构件或者子系统的外部可见操作，并不描述内部结构。

通常，一个接口仅描述一个特定类的有限行为。

接口没有实现，接口也没有属性、状态或者关联，接口只有操作。

接口只可以被其它类目使用，而其本身不能访问其它类目。

可以对接口使用泛化关系。

接口在形式上等价于一个没有属性、没有方法而只有抽象操作的抽象类。

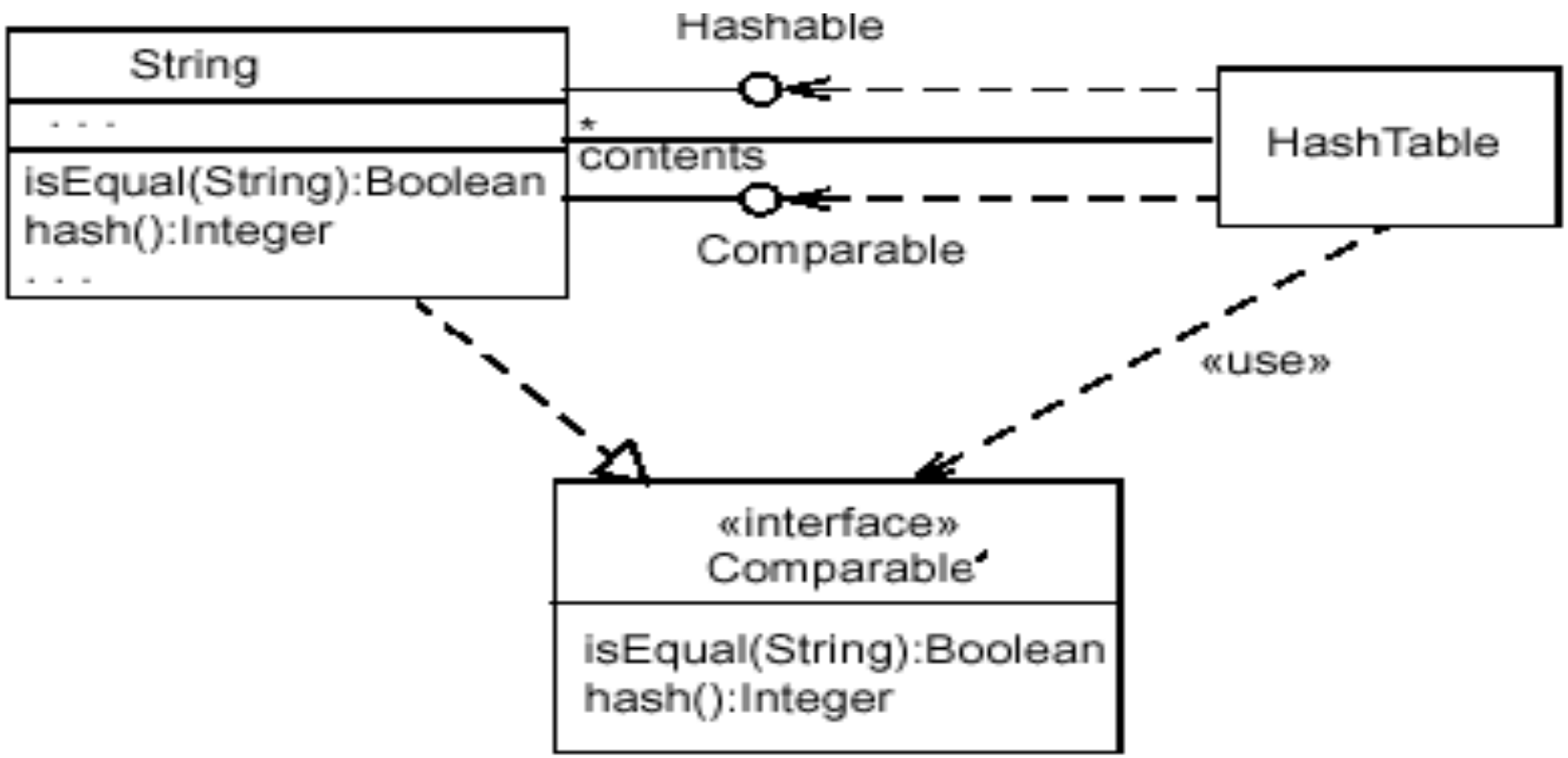
(2) 表示法

可以用带有分栏和关键字<<interface>>的矩形符号表示接口。在操作分栏中给出接口支持的操作列表。接口的属性分栏总是空的。把从类目到它支持的接口的实现关系显示为带有实三角箭头的虚线。用带有《use》标记的虚线箭头表示类目（在箭头尾部）使用或者需要接口提供的操作。

也可以把接口表示成小圆圈。接口名放在圆圈的下面，并用实线把圆圈连接到支持它的类目上。这意味着这个类目要提供在接口中的所有操作，其实类目提供的操作可能要更多。要显示接口的操作列表的话，就不能使用圆圈表示法，而应该使用矩形表示法。可以用指向圆圈的虚线箭头，把使用或者需要接口提供的操作的类目连到圆圈。

上述的虚线箭头还意味着使用接口的类目并不需要使用接口给出的全部操作。

(3) 例子



该图表明，类String支持接口Hashable、Comparable，

而类HashTable使用接口Hashable、Comparable。

以上内容小结

（知识点 和 解决问题的基本思想和途径）

1、知识点

（1）给出了表达客观事物基本成分（对描述客观事物而言不可再分的）的概念：　　对象，类

语义及其表示。

（2）给出了这些成分的基本构造：　属性，操作

语义及其表示

（3）给出了支持功能抽象的机制：　接口

语义及其表示，以及

（4）接口与其他成分之间的关系

①接口：

一组操作（没有实现的），这组操作在一个特定类中实现。

作为一种机制，支持功能抽象。

②操作：一个类对外提供的服务。

可以是抽象的。

其声明可以作为接口的成分。

③方法：是操作的一个实现。

至此可以说：

这五个概念围绕一个问题，即如何描述客观事物-“对象”展开的！

如何抽象对象的“结构”：属性，操作

如何描述一组具有相似性质的对象：类

如何抽象并描述在特定环境中对象的功能：接口。

2、解决问题的基本思想和途径

（1）大千世界是有对象构成的，对象有其自己的属性与运动规律。

（2）基本途径

①数据和操作（功能）的局部化-支持事物语义的表达；

②数据和操作的封装性--支持交互；

③功能模块化（接口）--支持在一个层次上的功能抽象；

④描述的“二分法”（《元数据，数据》）--支持复杂性控制(抽象)

《类，对象》 《接口，操作》 《操作，方法》

提高 模型的稳定性，可维护性

6）关联

对象之间存在的引用关系-元组，称为链。

关联是对一组具有相同结构特性、行为特性和语义的链的描述.

association:any semantic relationship between two or more class or types [Booch,...]

关联不仅适用于类，对接口、用况和子系统也适用.

围绕“关联”，涉及的内容有：

（1）二元关联 及 （2）N元关系；

（3）关联端点 ：用于给出关联语义的描述；

（4）限定符：确定关系的条件；

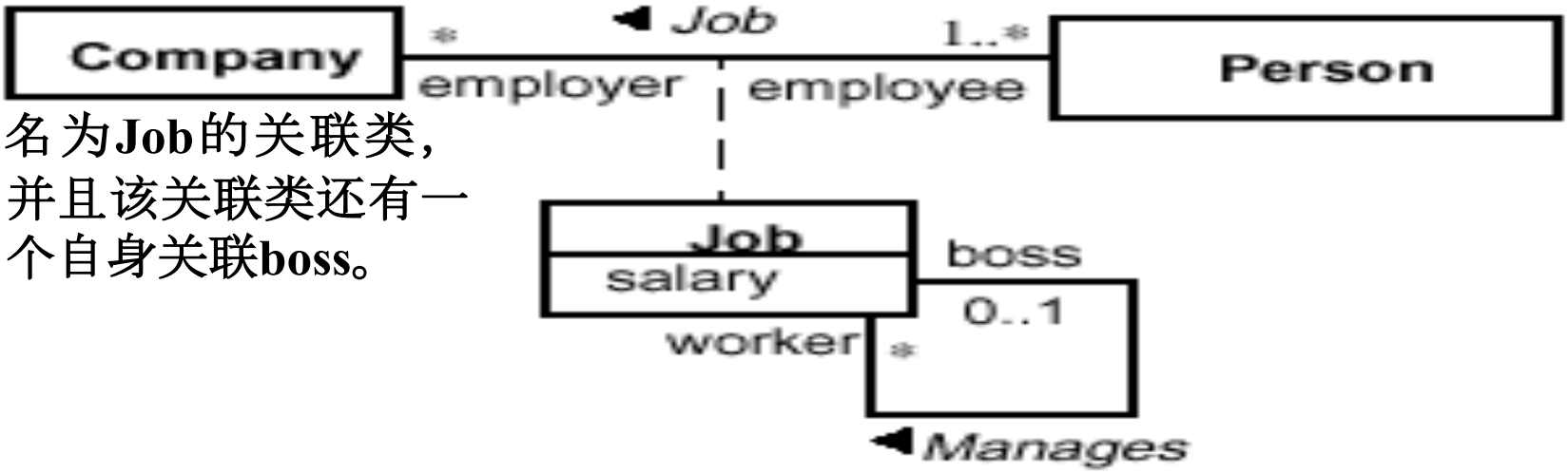
（5）关联类：定义关系的属性及实现；

(1)二元关联(仅以类为例)

①语义

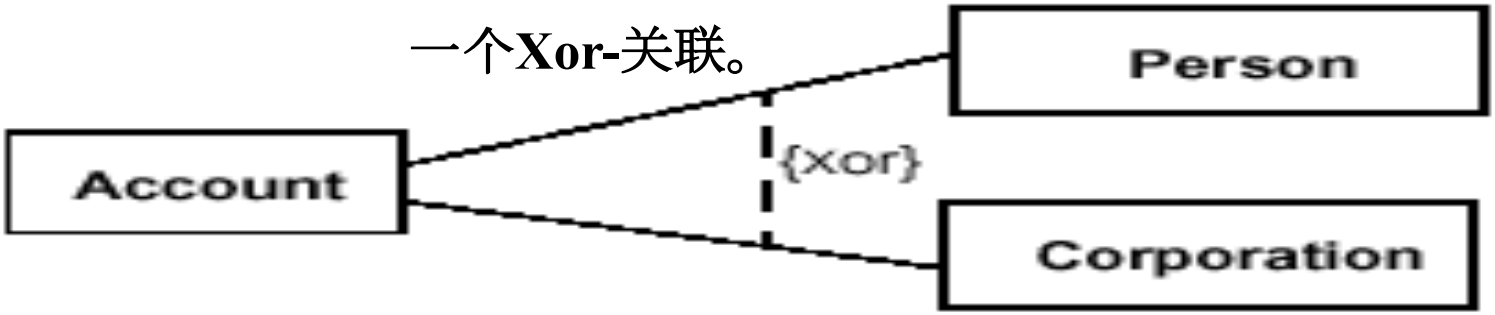
二元关联是两个类之间的关联。其中，相关联的两个类可以是同一个类.

②表示法



名为Job的关联类，并且该关联类还有一个自身关联boss。

一个Xor-关联。



（2）关联端点

①语义

关联端点是关联的组成部分，而不是独立的模型元素。每个关联有两个或多个端点。

②表示法

关联路径通过端点与类符号相连。

在关联路径的每个端点上，都可以附属有该关联的描述信息，以表明该关联的性质。

附属的信息属于关联符号，而不属于类符号。

在关联端点上，可以给出如下类型的信息：

❶

多重性

(A) 语义

多重性是对集合可能采取的基数范围的说明。

可以为关联中的角色、组合中的组成部分、某些元素的重复以及其它目的，给出相应的多重性规约。

多重性规约是非负整数开集的一个子集。

(B) 表示法

多重性表示为：由用逗号分开的整数间隔序列组成的字符串，间隔代表整数的范围（可能无限），其格式为：

下限..上限

其中，下限和上限都是整型值，说明从下限到上限的整数闭区间。星号（*）可以用于上限，表明不限制上限。

❷排序

如果没有指明多重性，相应的元素形成集合就是无序的。

如果多重性大于1，相应的元素集合可以是有序的，也可以是无序的。

在关联端点上，可以通过关键字“{ordered}”来指定对元素的排序。但这并没有指定排序是怎样建立和保持的，仅仅表明元素是有序的，在实现时可以按各种方式排序。

❸导航性

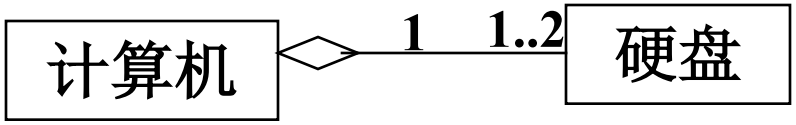
箭头附属到关联路径的端点，表明朝着连接到箭头的类的方向支持导航。

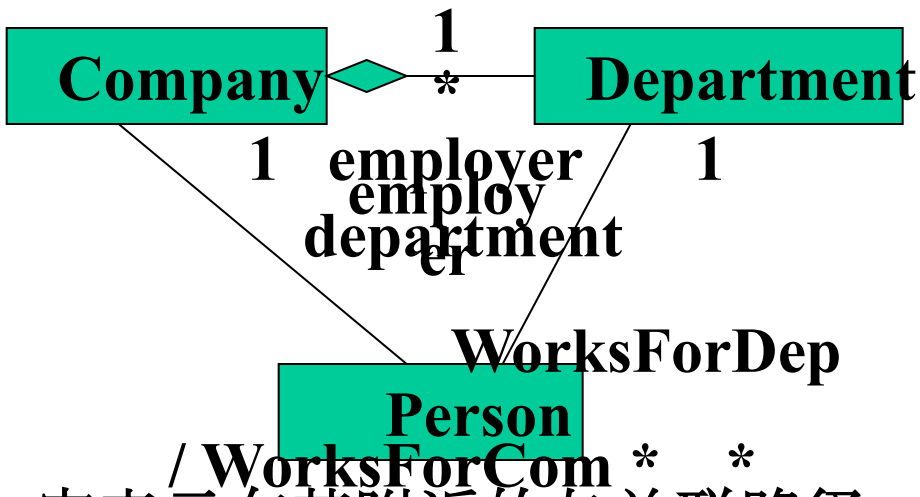
箭头可以附属0个、1个或2个路径端点。

❹聚合符

聚合是一种特殊的关联，表示整体类和部分类之间的“整体-部分”关系，其中的整体类称之为聚集。

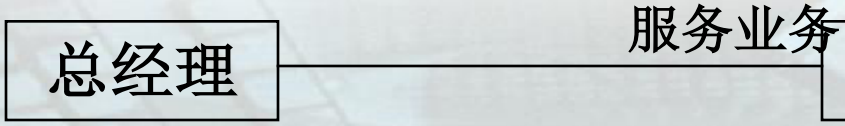
聚合符是一个空心菱形，它附属在关联路径表示聚合的那一端，但不能附属在路径的两端。





❺角色名

角色名靠近关联路径的名称串。它表示在其附近的与关联路径端点相连接的类所扮演的角色。



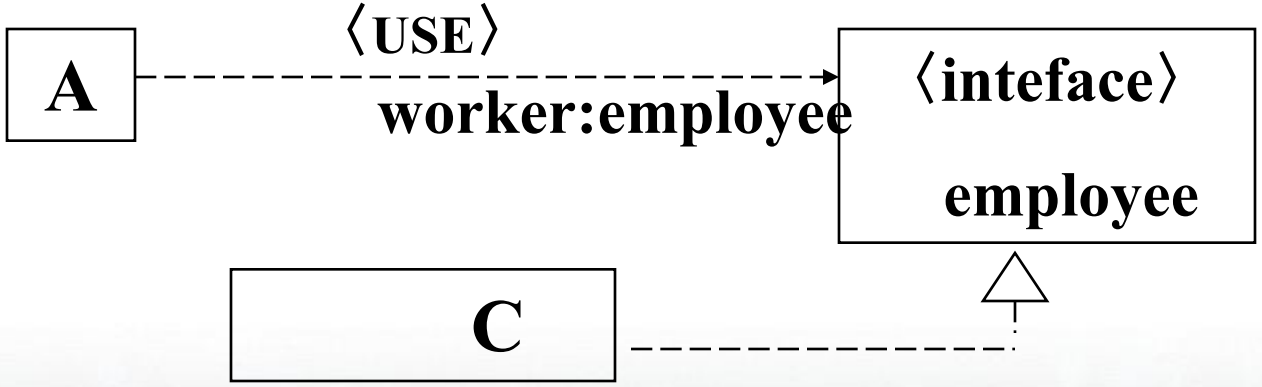
❻接口说明符

接口说明符的语法为：

角色：接口名，…

例如：worker:employee. 表明关联中的一个实例worker期待接口employee提供的操作。

既该关联的实现，需要接口employee为角色worker提供相应的操作。通常实现接口操作的类具有为实现这一特定关联更多的功能（因为实际的类还具有其他的责任）。



注：如果省略一个接口说明符，可用关联获取对被关联的类的完全访问。

❼可变性

性质串{frozen}表明：

在对象被产生和初始化后，不能加入、产生或移走从该对象出发的链。

性质串{addOnly}表明：

可以加入另外的链（设想多重性是可变的），然而不能调整或删除链。如果链是可变的（能加入、删除和移走），就不需要标示信息。

❽可见性

在角色名前面加可见性说明符（‘+’、‘#’、‘-’或{public}、{protected}、{private}）。

指明了朝着给定角色名方向的关联的可见性。

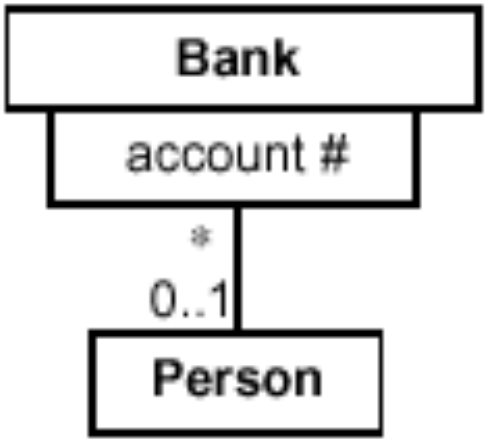
（3）限定符

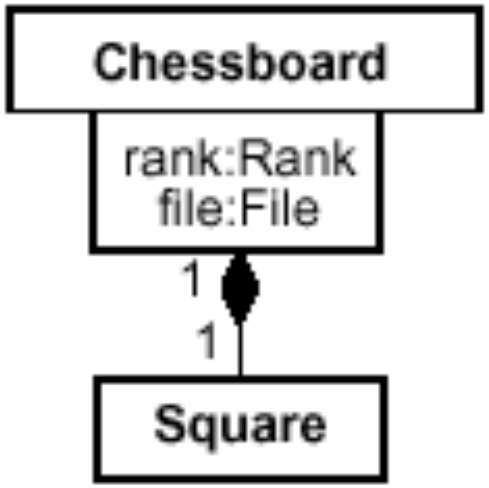
①语义

一个限定符是一个属性或是属性表，这些属性的值将与一个相关联的对象集做了一个划分。

限定符是关联的属性。

②表示法





左图的限定符有一个属性account#，表明：在一个银行中，一个帐户对应一个用户，或没有对应人员。

右图的限定符有两个属性，它们与Chessboard一起确定了Square，且 Square是其组成部分。

（4）N元关联

①语义

一个N元关联是三个或多个类之间的一个关联，其中的一个类可能在一个N元关联中多次出现。

该关联的每一个实例是一个N元组，即它由N个分别来自相关的类的对象组成。

可以规约N元关联的多重性：在一个角色上的多重性是指，当该N元关联中的其它N-1个类的对象被确定时，该关联潜在的实例元组的数目。

对于一个N元关联，在任意角色上都不能有聚合标志。

②表示法

本例描述了一个具有一个特殊守门员的球队在每一个赛季的记录。

假设在本赛季可出售守门员，且该守门员可出现在多个球队。

通过一个大的菱形表示一个N元关联。
如果关联有名字，就显示在菱形附近。
同二元关联一样，角色可以显示在每一条路径上。
可以指出多重性；然而不允许有限定符和聚合符。
可以用虚线把关联类符号与菱形连接起来，表示具有属性、操作或关联的N元关联。

7) 组合

(1) 语义

组合是一种关联，是聚合的一种形式。其部分和整体之间具有很强的“属于”关系，并且它们的生存期是一致的。

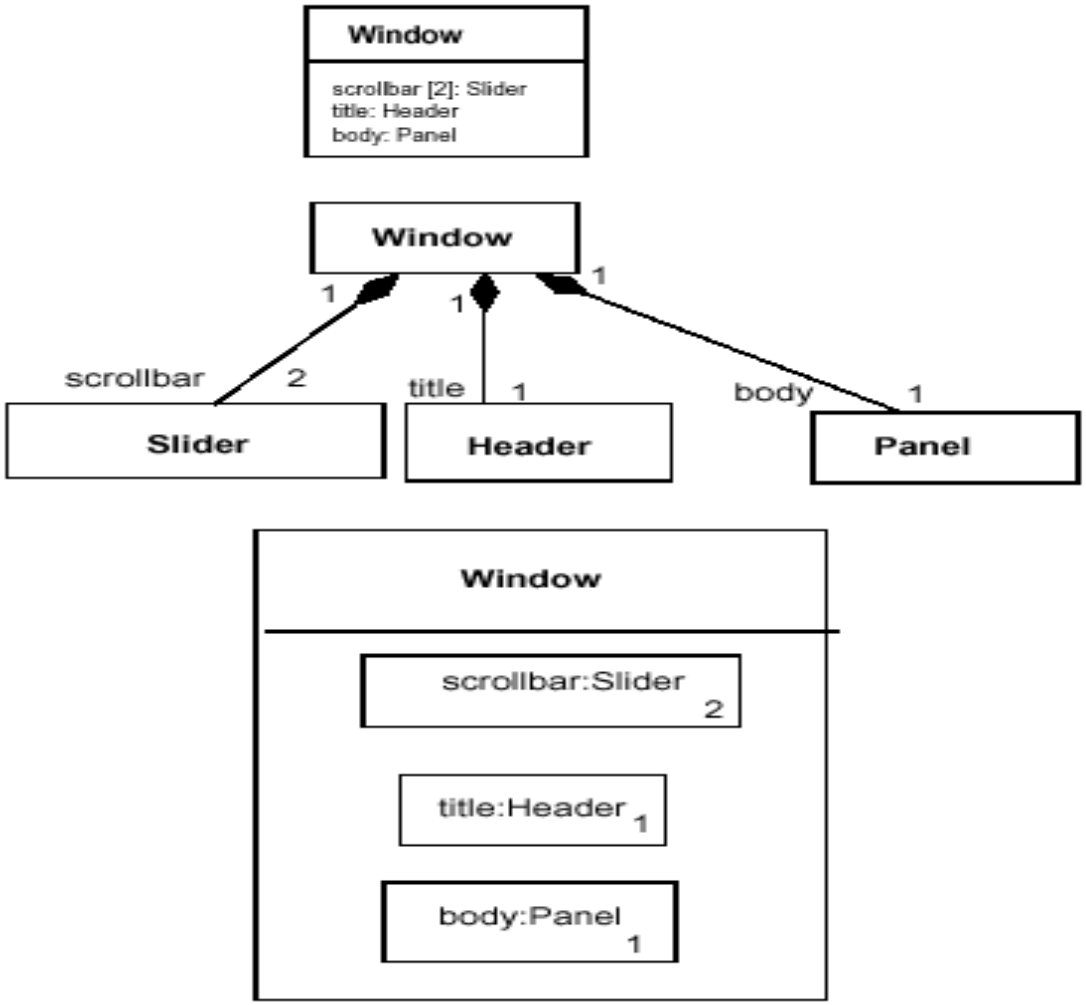
显然，❶这种聚集（也称为组成）末端的多重性不能超过1；

❷在一个组合中，由一个链所连接的对象而构成的任何元组，必须都属于同一个容器对象。

在一个组合中，其部分可以包含一些类和关联；根据需要，也可以把它们规约为关联类。

(2)表示

表示组合的不同方法



该例给出了三种表示组合的方法。

其中，类**Window**由类**Silder**（角色为**Scrollbar**）、**Header**（角色为**title**） 和**Panel**（角色为**body**）组成。

8） 链

(1) 语义

链是对象引用的一个元组。

链是关联的一个实例。

(2) 表示法

二元链表示为两个实例之间的路径。

可以把一个实例与它自身之间的链表示为一个具有单一实例的环。

在链的各端可以表示角色名。

链没有实例名，从与它们相关的关联中得到其标识。

多重性不能显示在链上，因为链是实例。

其它的关联附属物（组合、聚合和导航）可以表示在链端点上。

限定符可以表示在链上，限定符的值也表示在它的框中。

用带有连向各个参与实例的路径的菱形表示N元链。

9 ）泛化

(1)语义

泛化是一般元素（父亲）和特殊元素（儿子）之间的一种分类关系，其中特殊元素的结构完全与一般元素一致，并附加了一些信息。

泛化可用于类、包、用况、关联和其它元素.

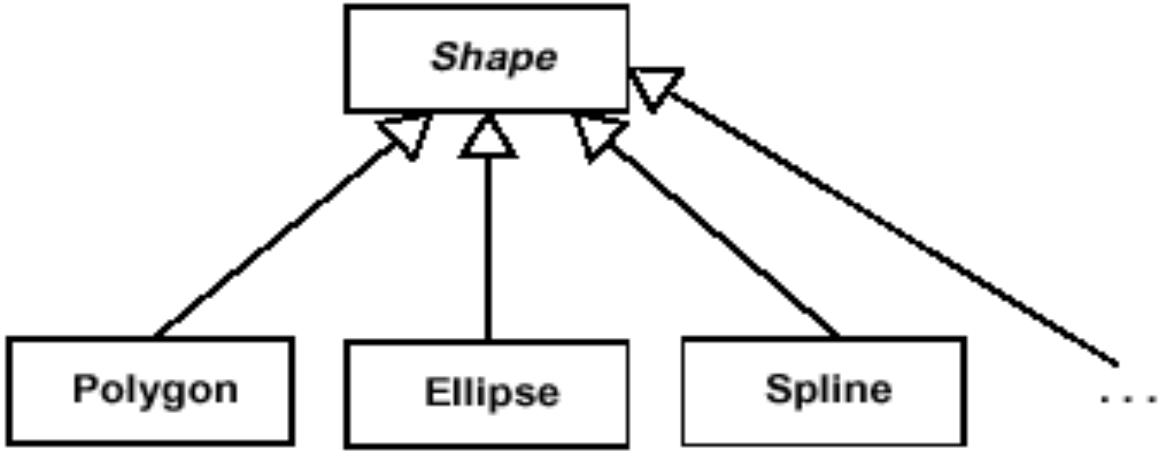
泛化支持继承 (inheritance)：复用机制

the incremental construction of a new definition in term of existing definitions without disturbing the original constructions and their client .[Ada95,firesmith]

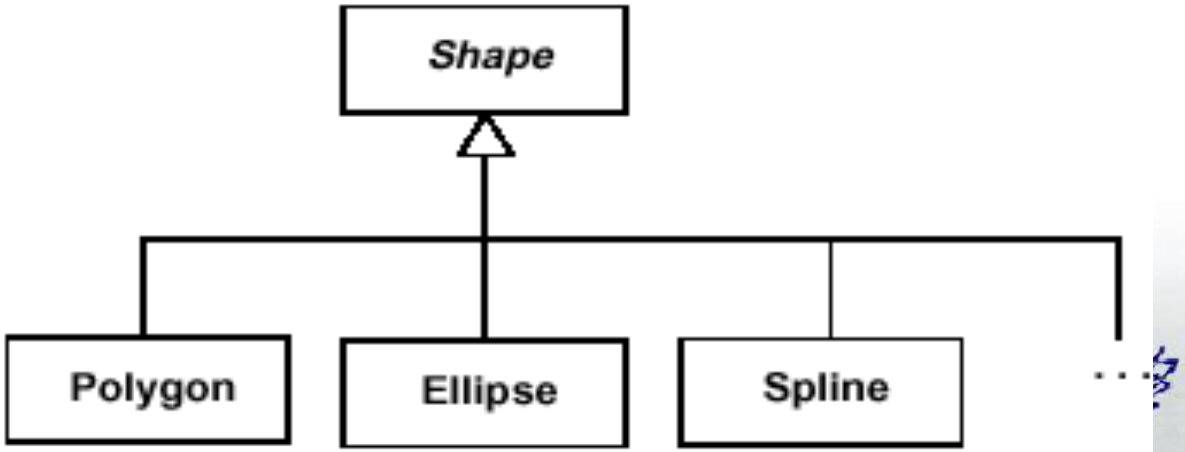
(2) 表示法

把泛化表示成从子类（特殊类）到父类（一般类）的实线路径，在较一般的元素的路径端有一个大的空心三角。

分离表示法



共享表示法

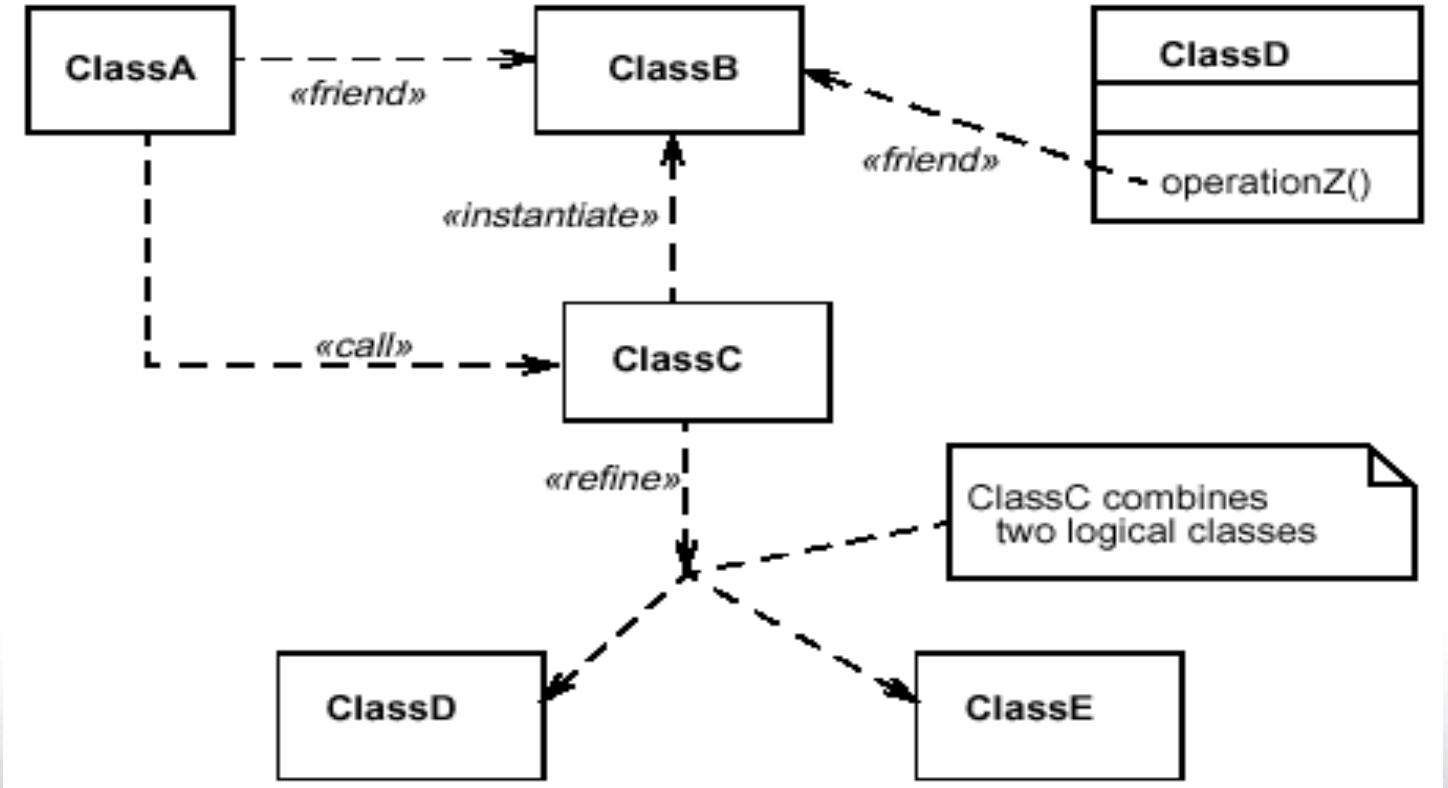


10） 依赖

（1）语义

一个依赖规约了两个模型元素（或两个模型元素集合）之间的一种语义关系，即：对目标元素的改变可能需要改变该依赖中的源元素。

(2) 表示法。



❶ 把依赖表示为两个模型元素之间的虚线箭头。

箭头尾部的模型元素（客户）依赖箭头头部的模型元素（提供者）。

❷依赖的特定情况：一组元素作为客户或提供者

在这种情况下，可以自客户端引出一条带箭头的虚线，并把该虚线连接到多个提供者端。

如果需要的话，可以在连接处放置一个小的圆点，作为连接点。此时，依赖的注释应该依附在连接点上。

注意：在一些面向对象的建模语言中，预定义了一些特定的依赖。其关键字要用书名号括起，并表示在虚线箭头附近。

关于“关系”描述的小结

1、知识点

（1）给出了表达客观事物之间关系的基本概念：链。

（2）给出了关联的语义和表示，并且，

（3）还给出了一些特定的关联（分类）。主要包括：二元关联与N元关联，聚合与组合，泛化，依赖。

语义及其表示。

（4）给出了关联语义的描述：

①关联端点-多重性，排序，导航，角色名，接口说明符，可变性，可见性等；

②限定符-关联的属性；

③关联类；

④对泛化给出了子类划分的语义约束。

2、解决问题的基本思想和途径

紧紧围绕“关系”的表达（其中包括“形”与“义”），给出了一组概念。

①采用“二分法”：《关联，链》-控制信息组织的复杂性；

②通过对关系的分类，可以有效地支持关系的建模；

③为了祢补“形”的表达能力，给出了一些概念（多重性，可变性等），支持语义的表达；特别地，

④为了支持关系属性和操作的表示，引入了“关联类”和“限定符”。

11）包

(1) 语义

包是模型元素的一个分组。一个包本身可以被嵌套在其它包中，并且可以含有子包和其它种类的模型元素。

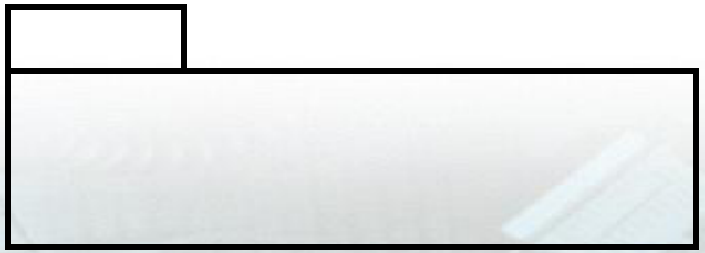
包间只有依赖。

标准依赖：访问依赖和引入依赖。作用：使一个包可以访问和引入其它包。

其它依赖：通常隐含了元素间存在着的一个或多个依赖。

(2) 表示法

包表示成一个大矩形，并且在这一矩形的左上角还有一个小矩形（作为一个“标签”）。



关于“包”的小结

（1）包是控制文档组织复杂性的机制。

（2）包也可以作为“模块化”、“构件化”机制。

（3）包之间（在包的层次上）存在两种依赖：访问和引入

访问依赖表明：目标包的内容可以被客户包引用，或被递归嵌套在客户包中的其它包引用。

- 不修改客户的名字空间；
- 不以任何其它方式自动创建一些引用。

引入依赖表明：将目标包中那些具有合适可见性的名字引入到客户包（即对它们的引用可不需要一个路径名）。

2 顺序图

目的：为了表示实例之间按时间顺序组织的交互。支持实时系统和复杂场景的详细建模。

途径：❶引入实例生命线，表示参与交互的实例；

❷引入消息，并将消息按时间顺序排列，表示实例之间的交互。

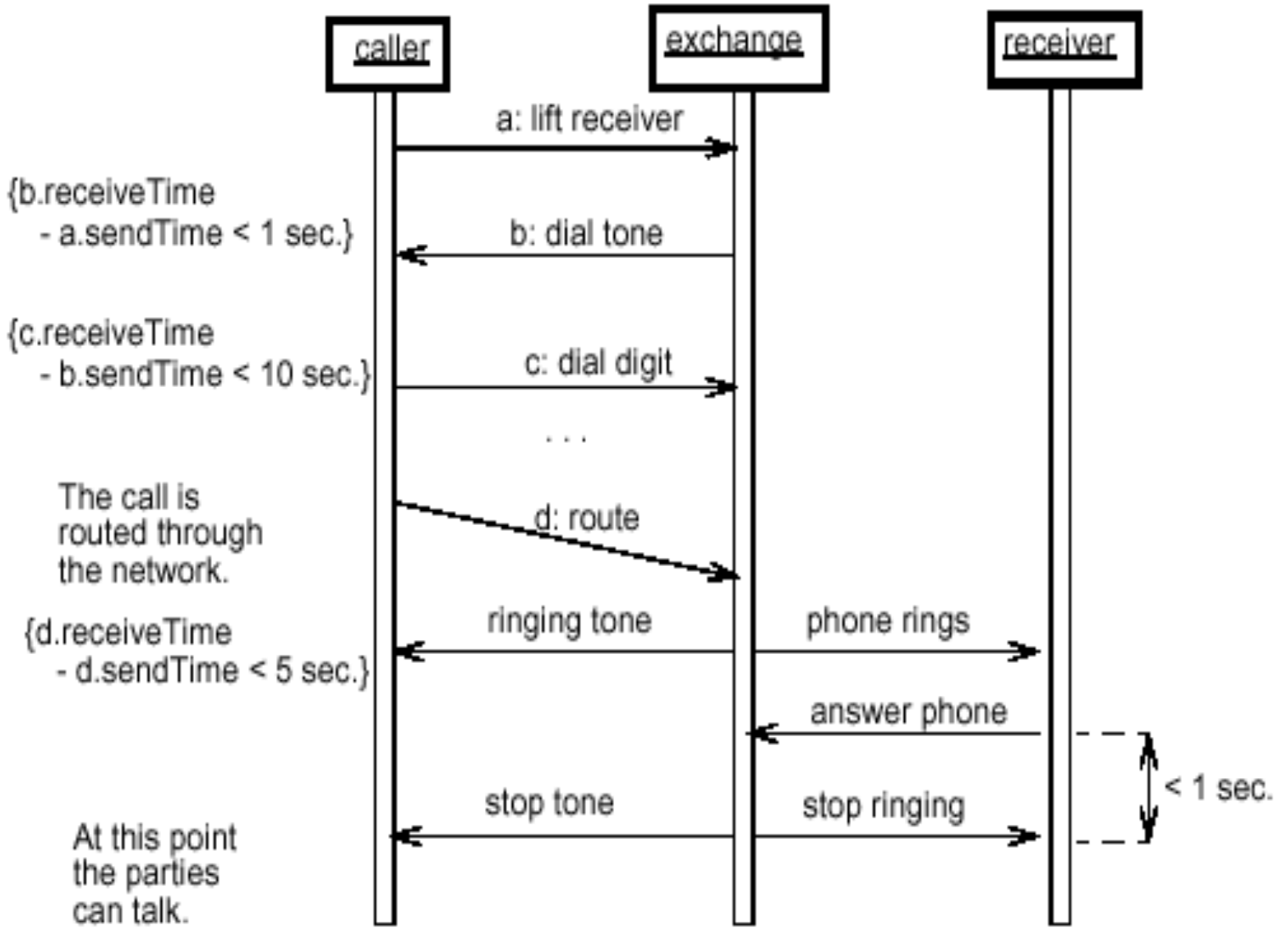
局限性：只是对一组实例及之间的交互进行描述。

1） 顺序图

(1) 语义

顺序图是一种表达对象间交互的图，由一组对象以及其间顺序发送的消息组成。

(2) 表示法


caller
exchange
receiver
a: lift receiver
{b.receiveTime
- a.sendTime < 1 sec.}
b: dial tone
{c.receiveTime
- b.sendTime < 10 sec.}
c: dial digit
. . .
The call is
routed through
the network.
d: route
{d.receiveTime
- d.sendTime < 5 sec.}
ringing tone
phone rings
answer phone
< 1 sec.
stop tone
stop ringing
At this point
the parties
can talk.

可见：

顺序图是二维的。其中：垂直方向表示时间，水平方向表示不同的对象。

(3) 表示选项

- 生命线之间的顺序是任意的。
- 可以交换轴；
- 可以在消息名上使用时间表达式，表示计时约束。

其中：可以把函数**sendTime**（发送消息时间）和函数**receiveTime**（接收消息时间）应用于消息名，以计时间。

可以使用一些构造标记，用于指示时间间隔。

类似地，可以用消息名表示在计时表达式中的发送或接收消息的时间。

2）对象生命线

(1) 语义

在顺序图中，对象生命线表示扮演特定角色的对象。类的角色规定了对象的角色，它描述了扮演该角色的对象的性质。

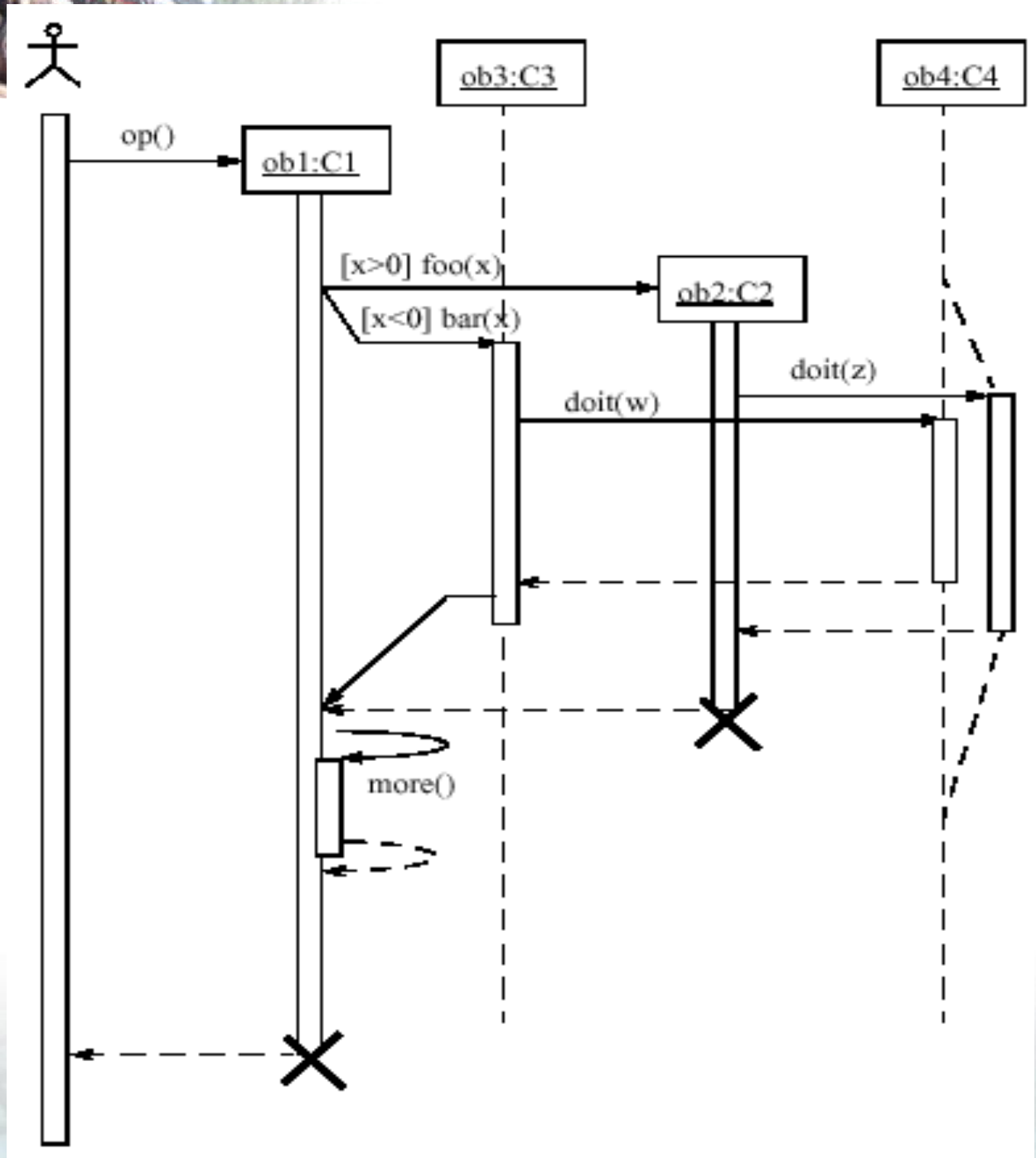
与对象生命线相关的有：

◆生命线之间的箭头（消息）：表示扮演这些角色的对象之间的通讯。

◆窄长的矩形（活化）：表示处于一个角色状态的对象的存在和持续。

(2) 表示法

“对象生命线”表示为垂直虚线。生命线代表一个对象在特定时间内的存在。例如：



其中：

---：生命线

激活（活化）

[X>0]：转换时间

：消息

Foo(x):消息名

：撤消

3） 激活

(1) 语义

激活（控制焦点）表示一个对象直接或者通过从属例程执行一个行为的时期。它既表示了行为执行的持续时间，也表示了活动和它的调用者之间的控制关系。

(2) 表示法

用一个窄长的矩形表示激活（活化）。

❶活化的始末：矩形顶端为活化的开始时刻，末端为结束时刻。

❷激活的动作：可以用文本标注被执行的动作。

❸并发：在每个拥有自己的控制线程的对象并发的情况下，一个激活说明了一个对象执行一个操作的持续时间，与其他对象的操作并不相关。

❹嵌套：可以在一个特定的时间看到所有活动着的嵌套过程激活。在递归调用一个已激活的对象的情况下，第二个激活符号画在第一个符号稍微靠右的位置，并有重叠。（可以按任意的深度嵌套递归调用）

❺如果直接计算和间接计算（由嵌套过程执行）的区别并不重要，那么可以用整条生命线表示一个激活。

4） 消息

(1) 语义

消息是两个对象间的通讯，这样的通讯用于传输将产生的动作所需要的信息。一个消息会引起一个被调用的操作，产生一个信号，或者引起一个对象被创建或者被消除。

(2) 表示法

在顺序图中，把消息表示为从一个对象生命线到另一个对象生命线的一个水平实线箭头。对于到自身的消息，箭头就从同一个对象符号开始和结束。用消息（操作或信号）的名字及其参数值或者参数表达式标示箭头。

箭头也可以用一个序列数标示，以表示消息在整个交互中的顺序。序列数对于标示并发控制线程很有用处。另外，还可以用监护条件标示消息。

5） 转换时间

(1) 语义

消息可以指定几个不同的时间（例如，发出时间和接受时间）。这些时间可以用在约束表达式中。

用户可以按需要为特定的目的给出时间表达，如**elapsedTime**（占用时间）和**startExecutionTime**（开始执行时间）。可以在约束中使用这些表达式，以给消息指派有效的具体时间约束。

(2) 表示法

可以赋予消息一个名字。把时间约束写成为一个基于消息名字的表达式。例如，如果消息的名字是**stim**，用**stim.sendTime()**表示发送时间，用**stim.receiveTime()**表达接收时间。可以把时间约束表示在与箭头对齐的图的左边上，也可以通过把布尔表达式（可能包括时间表达式）放在括号中表示约束。

3 状态图

状态图用于描述模型元素（如对象）的行为。特别是，用它描述元素之状态的可能序列和动作的可能序列。因对特定事件（如信号和操作调用）的响应，元素在其生命期中要经历这样的状态，并执行相应的动作。

1) 状态图

(1) 语义

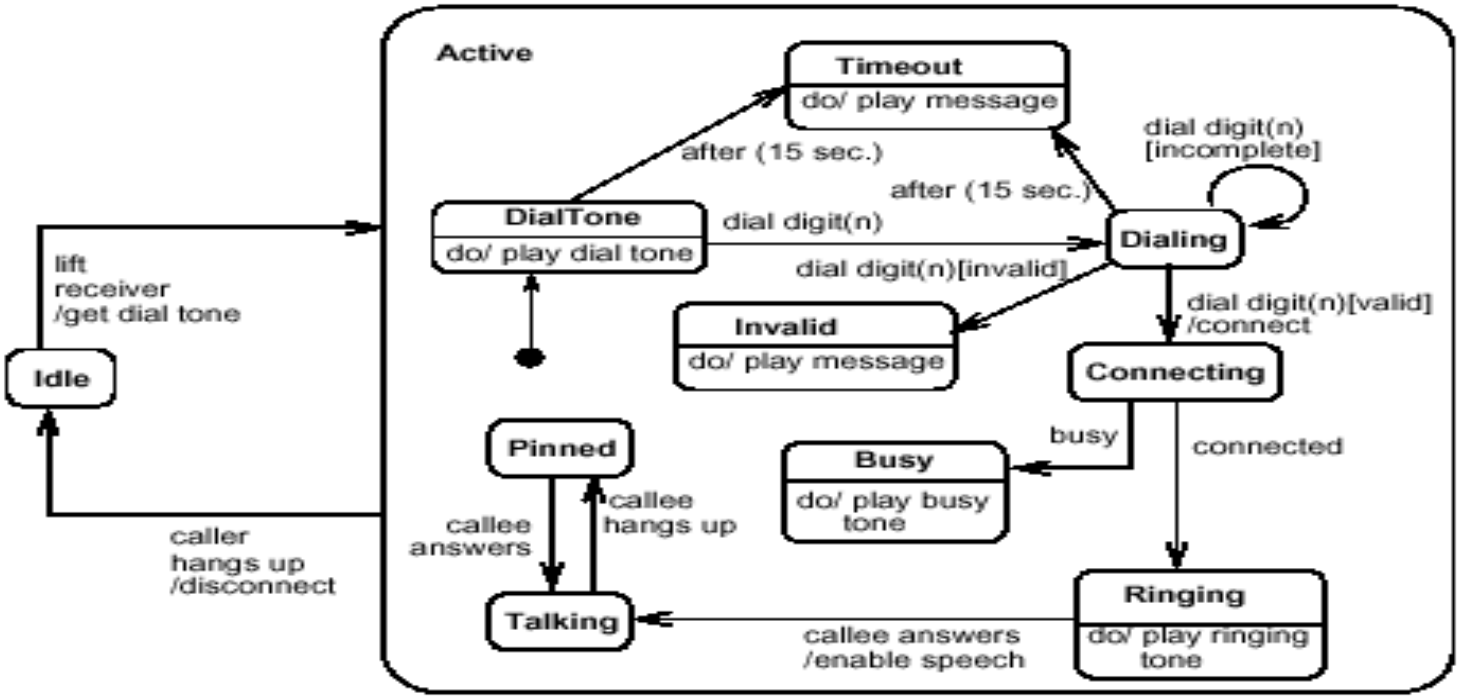
通过描述对事件实例接收的响应，状态图描述了具有动态行为能力的实体之行为。

通常用状态图描述类的行为，也可以用它描述其它模型实体（如用况、参与者、子系统、操作或方法）的行为。

(2) 表示法

状态图是表示状态机的图。用适当的状态表示状态机图中的状态；一般地，用连接状态的有向弧表示转换。

(3) 实例　简单的电话状态的转换



其中：

状态

转换

Ringing 事件

2) 状态

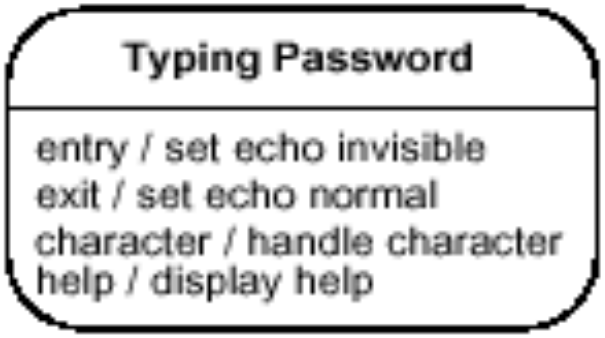
(1) 语义

一个状态是对象在其生命期内的一个条件，或在对象满足某个条件、进行某个动作或等待某个事件的期间内的一个交互。在概念上，对象要在一个状态内维持一段时间。

在这一语义下，可以对瞬时状态建模，以及对非瞬时的交互建模。

(2) 表示法

把一个状态表示成一个四角均为圆角的矩形。



根据需要，可以把状态划分成由水平线相互分隔的多个分栏：

❶名称分栏

给出状态名。在同一张状态图里不应该出现具有相同名称的状态。

如果没有状态名称，那么该状态就是匿名的。同一张图中的匿名状态是各不相同的。

❷内部转换分栏

给出在这个状态中对象所执行的内部动作或活动的列表

其一般格式为：

动作标号’/’ 动作表达式

其中：动作标号标识在该环境下要调用由动作表达式指定的动作。

动作表达式可以使用对象范围内的任何属性和链。若动作表达式为空，则可省略斜线分隔符。

下面给出专用的动作标号（注：它们不能用作事件名）：

- **entry**

该标号标识在进入状态时，执行由相应的动作表达式规定的动作（进入动作）。

- **exit**

该标号标识在退出状态时，执行由相应的动作表达式规定的动作（退出动作）。

- **do**

该标号标识正在进行的活动（“do 活动”）（只要被建模的元素是在状态中，没有完成由动作表达式指定的计算，就执行这个活动；当动作表达式指定的计算完成时，可能产生一个完成事件）。

3） 事件

(1) 语义

事件是值得注意的所发生的事情。按照状态图的具体用意，事件是指可以引发状态转换的所发生的事情。

事件可以分为：

a)条件（用布尔表达式描述）变为真。不论何时，只要条件变为真，事件都发生。（注意：这不同于监护条件。无论什么时候激发具有监护条件的事件，都对监护条件进行求值。如果求值的结果为假，转换就不发生，并且事件丢失。）

b)一个对象对另一个对象的显式信号的接收，导致一个信号事件。把这样的事件的特征标记放由它所触发的转换上。

c)对操作的调用的接收，导致一个调用事件。

d)在指定事件（经常是当前状态的入口）后，经过了一定的时间或到了指定日期/时间，导致一个时间事件。

(2) 表示法

●可以按如下的格式定义信号事件或调用事件：

事件名　　‘(‘用逗号分隔的参数列表‘)’

参数的格式如下：

参数名 ‘:’　　类型表达式

在类图中，在类符号上用关键字<<signal>>声明信号。把该关键字放在信号名的上面，把参数说明为信号的属性。注意，信号是实例之间异步传送的消息的规格说明。

●可以用关键词“after”和计算时间量的表达式表示时间事件，比如“after (5 秒)”或者”after (从状态A退出后经历了10秒)”。如果没指明时间起始点，那么从进入当前状态开始计时。可把其它的时间事件指定为条件，比如“when (date= 2000年1月1日)”。

●用关键词“when”和布尔表达式表示变为真的事件。可以把其看作是连续测试条件，直到它为真。

4） 转换

(1) 语义

转换是两个状态之间的关系，表示当一个特定事件出现时，如果满足一定的条件，对象就从第一个状态进入第二个状态，并执行一定的动作。

--对于这样的状态的改变，称为“触发”转换。

转换的触发器就是标注在转换上的事件。事件可能有参数，这样的参数可由转换指定的动作访问，也可由与源和目标相联系的退出和进入动作分别访问。

在状态图中，每次处理一个事件。如果事件没有触发任何转换，就丢弃它。

如果在同一个简单状态图中触发了多个转换，就只对优先级最高的那个转换点火。如果这些相冲突的转换具有相同的优先级，就随机地选择一个转换，进行触发。

(2) 表示法

把转换表示成从源状态出发并在目标状态上终止的带箭头的实线。它可以由转换串标记。转换串的格式为：

事件特征标记 ‘[‘监护条件’]’ ‘/’动作表达式

其中：事件特征标记描述带参数的事件：

事件名 ‘(‘由逗号分隔的参数表‘)’

监护条件：是布尔表达式，根据触发事件的参数和拥有这个状态机的对象的属性和链来书写这样的布尔表达式。也可用监护条件显式地指定某个可达对象的状态（例如，”in State1”或”not in State2”）。

如果触发了转换，就执行动作表达式。可以根据对象的属性、操作和链以及触发事件的参数，或在其范围内的其它特征书写动作表达式。

动作表达式可以是由一些有区别的动作组成的动作序列，其中包括显式地产生事件的动作，如发送信号或调用操作。表达式的细节与为模型选择的动作语言有关。

面向对象方法—RUP

RUP

（Rational Unified Process,RUP）

统一软件过程是由统一建模语言（Unified Modeling Languang，简称UML）的开发者们提出来的，并为对象管理组织(Object Management Group，简称OMG)所推荐。

统一软件过程是在权衡了三十年的软件开发实践的基础上形成的。例如，它吸取了数百个用户多年的现场经验以及Ratioanal公司多年的工作成果。

RUP对于如何运用UML的概念进行软件开发提供了详细指导。即：①指导开发队伍安排其开发活动的次序；②为各开发者和整个开发组指定任务；③明确地规定需要开发的制品；④提供对项目中的制品和活动进行监控与度量的准则。

1、概述

RUP是以用况（use case）为驱动、以体系结构(architecture)为中心的迭代的(iterative)、增量的(incremental)过程。具体地说：

❶ 以用况为驱动

用况驱动的含义是：以用况为单位，制定计划、分配任务、监控执行和进行测试等。这样，可以实现：

在产品开发的各个阶段中都可以回溯到用户的实际需求

由此可见，用况将实现软件开发的核心工作有机地组合为一体。

在用况驱动的含义下，通过用况，可以获得体系结构和其他制品。即：

（A）首先选择在体系结构上具有重要意义的用况，接着实现那些关键的用况，构造出初步的体系结构，逐步得到稳定的体系结构。

（B）通过枚举用况的不同执行路径，可导出测试案例和测试规程。

（C）通过用况，还可估算系统性能、硬件需求和可用性，并能进行用户界面设计，也能作为编写用户手册的基础。

（D）对用况的细化，在一定的程度上涉及系统的内部功能，于是可以对各用况进行完整的功能描述。

细化时，要区分用况中的三类事物：

◆反映系统与参与者(actor)之间的接口；

◆那些在用况中包含实现接口（功能）的活动和属性的实体，以及

◆对前两者进行协调和控制的机制。

用况驱动有助于模型之间的追踪和系统演化

❷以体系结构为中心

以系统需求（用况模型）为驱动，紧紧围绕系统体系结构，从不同角度描述要构造的系统的静态和动态结构。其中，主要涉及：子系统、构件、接口、协作、关系和节点等重要的模型元素的描述。并且，实现的描述应：

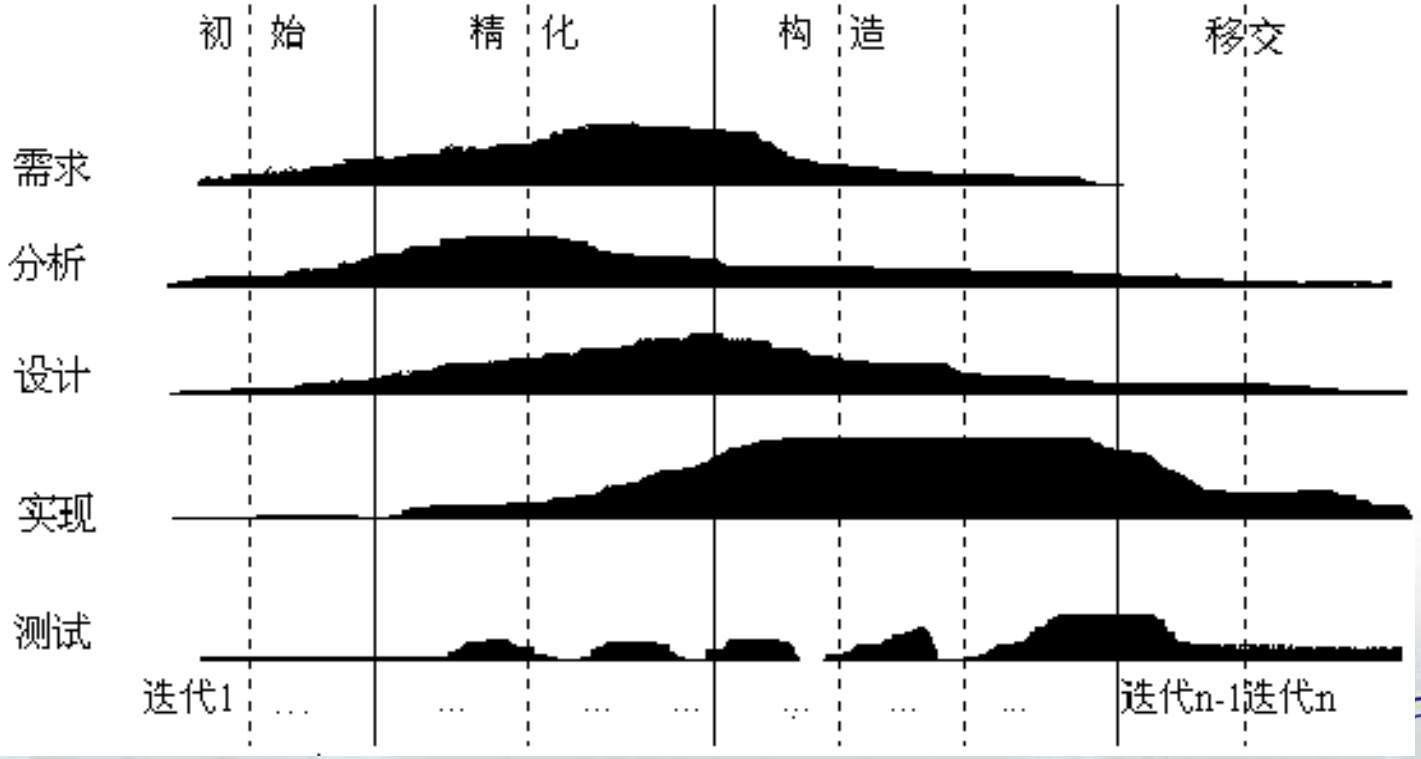
- 便于用户和其他关注者的理解，并达到共识；
- 便于对系统概貌的语义表述；
- 便于控制系统的开发、复用和演化。

如何获得系统体系结构？

❸迭代与增量

简单地讲，开发过程是将整个项目划分为一些“小”的项目

- 按核心工作流，即需求规约、分析、设计、实现和测试这五个活动，对每个小项目都进行迭代；
- 迭代被组织在4个阶段中：初始阶段、精化阶段、构造阶段和移交阶段。



2、核心工作流

❶ 捕获需求

• 依据客户、用户、计划者、开发者的想法（需求或用况），搜取特征（feature），形成特征表。

对特征表中的每个特征给出简洁的定义。 并

◇ 描述其状态（提议、批准、合并和验证等）；

◇ 实施的代价及风险；

◇ 重要的程度以及对其它特征的影响等。

- 用领域模型和业务模型捕获需求

◇用领域模型捕获系统环境中最重要的对象, 即:

业务对象（如帐目和合同等）、

现实世界中的对象和概念以及事件。

--领域模型用**UML**的类图描述,

此时，类图中包含领域类（**UML**类的构造型）及它们间的关系。

◇用业务模型捕获业务及业务过程,

--由业务用况模型和业务对象模型支持。

其中，

业务用况模型：描述具体的业务过程；

业务对象模型：描述一个业务用况的实现，即该业务用况是如何通过工作者和一组业务实体（如支票）和工作单元（一些业务实体形成的有机体）实现的。这一实现可以用交互图和活动图表示。 --可见，业务对象模型是一个业务的内部模型。

综上，这一阶段主要是产生表达需求的用况模型，即通过特征表、领域模型和业务模型，产生如下制品：

参与者、用况、用况模型和体系结构描述。

其中，详述的用况模型由：

整体描述、一组图和详述的用况组成

❷分析

分析是对需求的精化和构造。此阶段产生的制品有：

◆ 分析类

特点：多为概念性的，粒度比类大，很少有操作和特征标记，要用责任定义其行为，在高层有概念性属性和关系。分析类包括：

◇用于对系统和参与者之间的接口建模的边界类；

◇用于对一些概念和现象的信息和相关行为建模的实体类；

◇用于协调、顺序化、处理和控制其它类的控制类。

- 用况$_{\text{实现-分析}}$

特点：注重于功能需求。涉及：

①类图（使用分析类）；

②交互图（用协作图描述分析类之间的交互）；

③事件流$_{\text{分析}}$（用文本解释图），以及

④补充需求（用文本描述：持久、分布、并发、安全特征、容错等方面的非功能需求）等。

◆分析包

由分析类、用况$_{实现\text{-}分析}$以及其它分析包组成。

一般地，把支持一个特定的业务过程或参与者的一些用况组织在一个包中，或把具有泛化或扩展关系的用况组织在一个包中。

◆系统体系结构描述$_{分析模型角度}$

主要包括由分析模型分解成的分析包和它们之间的依赖、关键分析类、实现重要及关键功能的用况$_{实现\text{-}分析}$。

❸设计

产生的制品有：

◆设计模型

描述用况物理实现的对象模型，注重于功能需求和实现环境对系统的影响。

◆设计类

考虑与实现有关的因素，具体描述操作的参数、属性和类型等。

◆用况$_{\text{实现—设计}}$

是设计模型中的一个协作，按设计类及其对象的术语，描述一个具体的用况如何实现和执行。由类图、交互图、事件流$_{\text{设计}}$（按对象或子系统的术语进行的文本描述）和与实现相关的需求组成。

◆设计子系统

是组织设计模型制品的一种手段，用以描述大粒度的构件，由设计类、用况$_{\text{实现}}$、接口和其它子系统组成。

◆接口

表示由设计类和子系统提供的操作。

◆体系结构描述设计模型角度

主要包括由设计模型分解的子系统、接口、依赖、关键设计类和用况实现—设计。

◆ 部署图

是一个对象模型，按在计算节点上的功能分布描述系统的物理分布。

◆体系结构描述部署图的角度

包括部署模型的体系结构方面的视图。

❹实现

任务：

- 实现设计类和子系统；
- 由设计类生成构件；对构件进行单元测试；对构件进行集成和连接；
- 把可执行的构件映射到部署模型。

此阶段产生的制品有：

◆实现模型：

描述如何用构件实现设计模型中的元素；描述如何按实现环境组织构件；描述构件间的依赖关系。

构件：是对模型元素（如设计模型中的设计类）的物理封装。

◆实现子系统：由构件、接口和其它子系统组成。

◆　接口：用于表示由构件和实现子系统所实现的操作。在这一阶段可以使用设计时的接口。

◆ 体系结构描述实现模型的角度：

包括由实现模型分解的子系统、子系统间的接口、子系统间的依赖以及关键构件。

◆ 集成建造计划：

在增量开发中，每一步的结果即为一个建造（**Bulid**），即系统的一个可执行的版本。在一个迭代中，可能创建一个建造序列，该序列即集成建造计划。

❺ 测试

测试包括内部测试、中间测试和最终测试。

特别是在精化阶段中体系结构基线变为可执行时，在构造阶段中系统变为可执行时，以及在移交阶段中检测到缺陷时，要进行测试。

测试阶段产生如下制品：

◆测试模型：主要描述2点：

◇集成测试者和系统测试者怎样测试已实现的可执行构件；

◇怎样测试系统的特殊方面 （如用户接口、可用性、一致性、用户手册是否达到目的等）。

测试模型是测试用例、测试过程和测试构件的集合体。

◆测试用况：描述测试系统的方式。测试用况的一般描述：包括输入、输出和条件。

◆测试过程：描述怎样执行一个或几个测试用况，也可以描述其中的片段。

◆测试构件：用于测试模型实现中的构件。测试时，要提供测试输入，并控制和监视被测构件的执行。

用脚本语言描述或编程语言开发测试构件，也可以用一个测试自动工具进行记录，以对一个或多个测试过程或它们片段进行自动化。

◆测试计划：

测试计划描述测试策略、资源和时间表。测试策略包括对各迭代进行测试的种类、目的、级别、代码覆盖率以及成功的准则。

◆缺陷：系统的异常现象。

◆评价测试：

对一次迭代，对测试用况覆盖率、代码覆盖率和缺陷情况（可绘制缺陷趋势图）进行评价。把评价结果与目标比较，准备度量。

3、RUP的四个阶段

❶初始阶段

本阶段确定所设立的项目是否可行，具体要做如下工作：

- 对需求有一个大概的了解，确定系统中的大多数角色和用况，但此时的用况是简要的，这些用况构成了一个简化的用况模型。对给出的系统体系结构的概貌，细化到主要子系统即可。
- 识别影响项目可行性的风险。
- 考虑时间、经费、技术、项目规模和效益等因素。
- 关注于业务情况，制订出开发计划。

❷ 精化阶段

- 识别出剩余的大多数用况

对当前迭代的每个用况进行细化，分析用况的处理流程、状态细节以及可能发生的状态改变。细化流程时，可以使用程序框图和协作图，还可以使用活动图、类图分析用况。

- 对风险的处理

考虑需求风险、技术风险、技能风险和政策风险。

- 进行高层的分析和设计，并作出结构性决策

所产生的基线体系结构包括：用况列表、领域概念模型和技术平台。

以后的阶段对精化阶段建立的体系结构不能进行过大的变动。

体系结构基线的稳定是精化阶段结束的准则。

- 为构造阶段制订计划

精化阶段完成，意味着已经完成了如下的任务：

①用况完全细化并被用户接受；
②完成概念验证；
③完成类图；
④开发人员能给出项目估算（可分为精确、人月和无法估算）；
⑤基于用况考虑了所有风险（可分为高风险、可能的风险和不可能的风险）；
⑥并制订了相应的对策和计划；
⑦对用况标出优先级（可分为必须先实现、短期内实现和长期实现）。

❸ 构造阶段

识别出剩余的用况。此阶段每一次迭代开发都针对用况进行分析、设计、编码（如对类、属性、范围、函数原型和继承的声明等）、测试和集成，得到一个满足项目需求子集的产品。（由于在精化阶段已经完成软件设计，此时各项目组可以并行开发。）

在代码完成后，要保证其符合某些标准和设计规则，并要进行质量检查。

对于新出现的变化，要通过逆向工具把代码转换为模型，对模型进行修改，再重新产生代码，以保证软件与模型同步。

此阶段也要建立一些类图、交互图和配置图：

如果一个类具有复杂的生命周期，可绘制状态图；如果一个算法特别复杂，可绘制活动图。

❹ 移交阶段

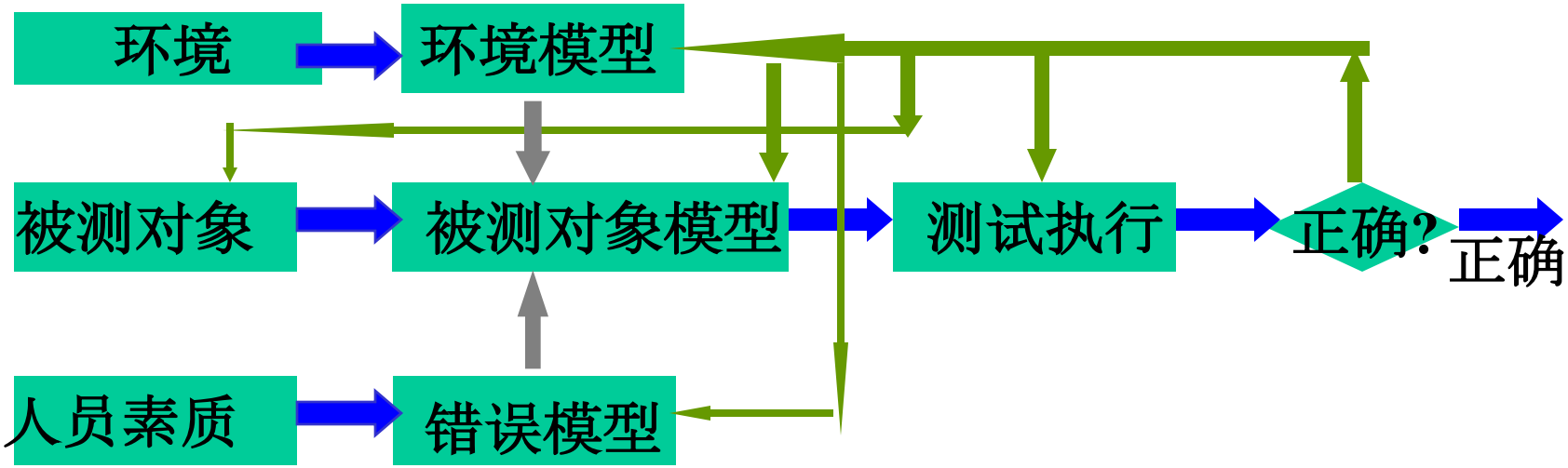
完成最后的软件产品和最后的验收测试，并完成用户文档以及准备对用户培训等。

附：面向对象方法与结构化方法的比较

·模型稳定
·保持结构
·平滑过渡
·支持复用
·易于维护

第六章 软件测试技术

1) 测试过程模型

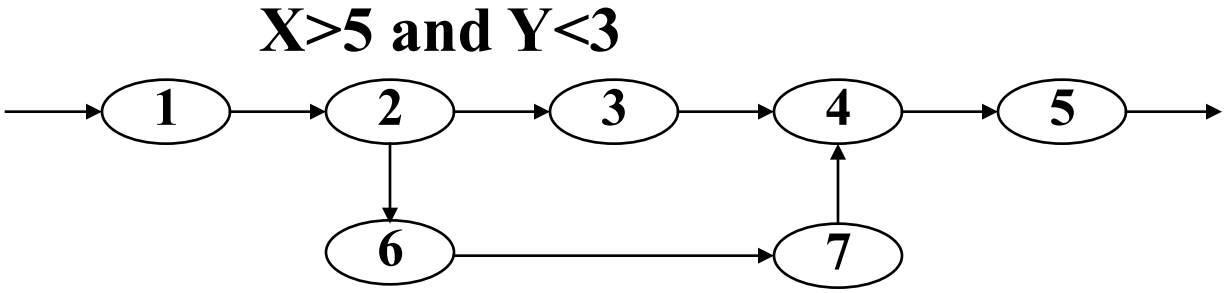


- 软件测试过程所涉及的要素,以及
- 这些要素之间的关系

2）依据程序逻辑结构-白盒测试技术

（1）关于建立被测对象模型

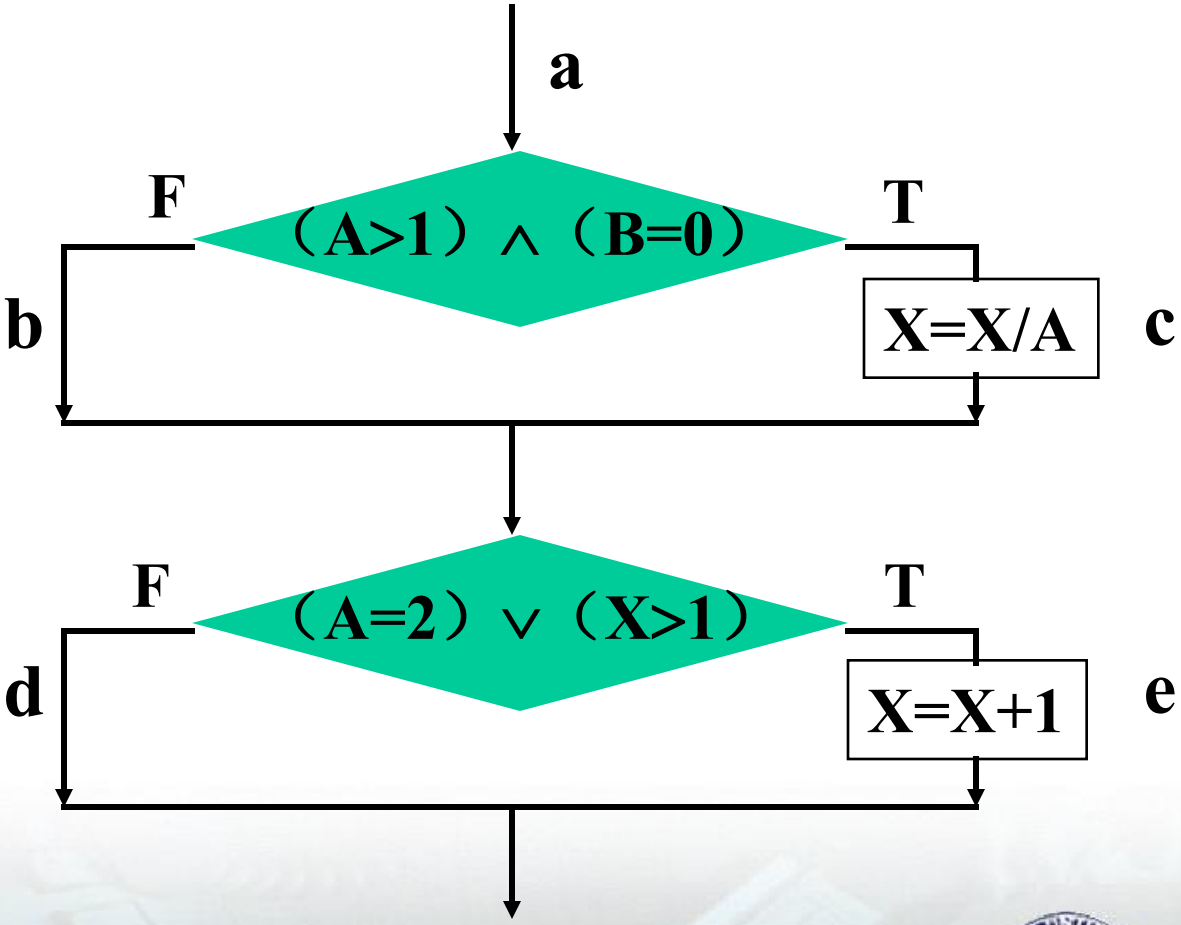
控制流程图:结点/分支/过程块/链路径



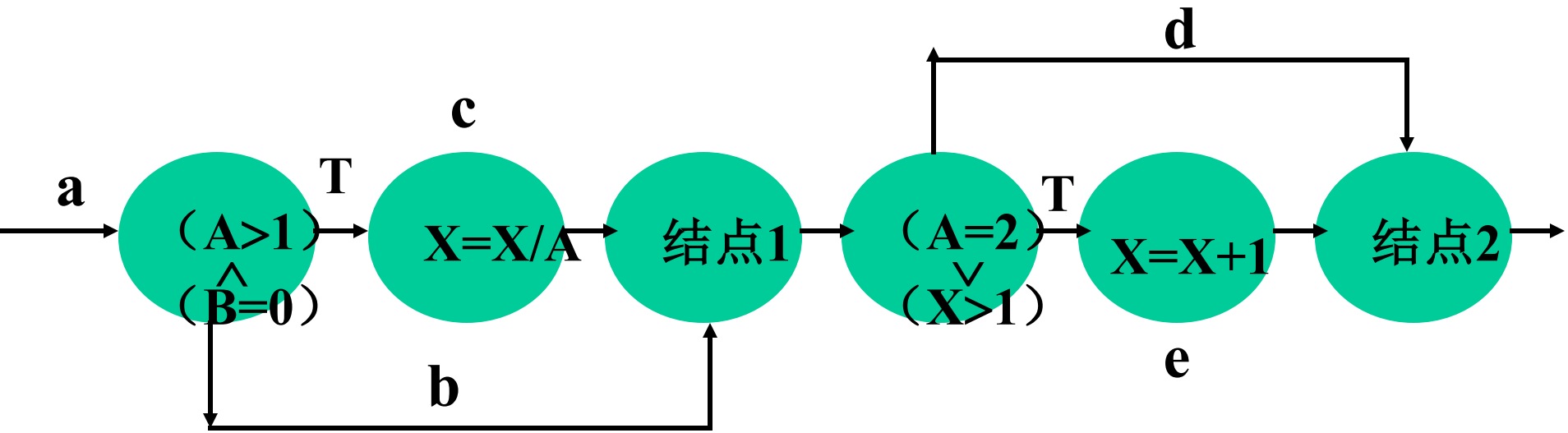
其中：节点1、节点3、节点5、节点6、节点7为过程块

节点2为分支，节点4为结点

例如：以下为一个程序流程图，其中该例子中有两个判断，每个判断都包含复合条件的逻辑表达式。



其控制流程图为：



（2）“完整测试”策略

该控制流程图有4条不同的路径。4条路径可表示为：

L1（a→c→e）简写ace 、L2（a→b→d）简写abd

L3（a→b→e）简写abe 、L4（a→c→d）简写acd

- 路径测试（PX）：执行所有可能的穿过程序的控制流程路径。

一般来说，这一测试严格地限制为所有可能的入口/出口路径。如果遵循这一规定，则我们说达到了100%路径覆盖率。在路径测试中，该策略是最强的，但一般是不可实现的。

针对该例子，要想实现路径覆盖，可选择以下一组测试用例（规定测试用例的设计格式为：【输入的（A，B，X），输出的（A，B，X）】）。

测 试 用 例	覆盖路径
【（2，0，4），（2，0，3）】	L1
【（1，1，1），（1，1，1）】	L2
【（1，1，2），（1，1，3）】	L3
【（3，0，3），（3，0，1）】	L4

● 语句测试（P1）：至少执行程序中所有语句一次。如果遵循这一规定，则我们说达到了100%语句覆盖率（用C1表达）。

在该例子中，只要设计一种能通过路径ace的测试用例，就覆盖了所有的语句。所以可选择测试用例如下：

【（2，0，4），（2，0，3）】 覆盖L1

语句覆盖是最弱的逻辑覆盖准则。

问题：就该程序而言，如果两个判断的逻辑运算写错，例如，第一个判断中的逻辑运算符“∧”错写成了“∨”，或者第二个判断中的逻辑运算符“∨”错写成了“∧”，利用上面的测试用例，仍可覆盖所有4个可执行路径，而发现不了判断中逻辑运算符出现的错误。

• 分支测试（P2）：至少执行程序中每一分支一次。如果遵循这一规定，则我们说达到了100%分支覆盖率（用C2表示）。

分支覆盖是一种比语句覆盖稍强的逻辑覆盖。但若程序中分支的判定是由几个条件联合构成时，它未必能发现每个条件的错误。

例如对于以上例子，如果选择路径L1和L2，就可得到实现分支覆盖的测试用例：

【（2，0，4），（2，0，3）】 覆盖L1

【（1，1，1），（1，1，1）】 覆盖L2

如果选择路径L3和L4，还可得另一组可用的测试用例：

【（2，1，1），（2，1，2）】覆盖L3

【（3，0，3），（3，1，1）】覆盖L4

问题：分支覆盖还不能保证一定能查出在判断的条件中存在的错误。例如，在该例子中，若第二个分支X>1错写成X<1，利用上述两组测试用例进行测试，无法查出这一错误。因此，需要更强的逻辑覆盖准则去检验判定的内部条件。

- 条件组合测试

条件组合测试是一种具有更强逻辑覆盖的测试。

条件组合测试，就是设计足够的测试用例，使每个判定中的所有可能的条件取值组合至少执行一次。如果遵循这一规定，则我们说就实现了条件组合覆盖。只要满足了条件组合覆盖，就一定满足分支覆盖。

在条件组合覆盖技术发展过程中，最初，在设计测试用例时，人们只考虑使分支中各个条件的所有可能结果至少出现一次。但发现该测试技术未必能覆盖全部分支。例如，在上图的例子中，程序段中有四个条件：A>1，B=0，A=2，X>1。

条件A＞1 取真值标记为T1，取假值标记为F1

条件B=0 取真值标记为T2，取假值标记为F2

条件A=2 取真值标记为T3，取假值标记为F3

条件X＞1 取真值标记为T4，取假值标记为F4

在设计测试用例时，要考虑如何选择测试用例实现T1、F1、T2、F2、T3、F3、T4、F4的全部覆盖：

例如，可设计如下测试用例实现条件覆盖：

测试用例	通过路径	条件取值	覆盖分支
【（1，0，3），（1，0，4）】	L3	F1 T2 F3 T4	b,e
【（2，1，1），（2，1，2）】	L3	T1 F2 T3 F4	b,e

从上面的测试用例，可以看到该组测试用例虽然实现了判定中各条件的覆盖，但没有实现分支覆盖，因为该组测试用例只覆盖了第一个判断的取假分支和第二个判断的取真分支。为此，人们又进一步提出了条件组合覆盖技术。

例如，在该例子中，前一个判定有4种条件组合：

（1）(A＞1)，(B=0)，标记为 T1 、T2 ；

（2）(A＞1)，(B≠0)，标记为 T1 、F2, ；

（3）(A≤1)，(B=0)，标记为 F1 、T2 ；

（4）(A≤1)，(B≠0)，标记为 F1 、F2 ；

后一个判定又有4种条件组合：

（5）(A=2)，(X＞1)，标记为 T3、T4 ；

（6）(A=2)，(X≤1)，标记为 T3、F4 ；

（7）(A≠2)，(X＞1)，标记为 F3、T4 ；

（8）(A≠2)，(X≤1)，标记为 F3、F4。

因此，要满足条件组合覆盖，设计的测试用例必须满足以下16种条件组合：

（1）(A＞1)，(B=0)，(A=2)，(X＞1)，可标记为 T1 、T2、T3、T4；
（1）(A＞1)，(B=0)，(A=2)，(X≤1)，可标记为 T1 、T2、T3、F4；
（1）(A＞1)，(B=0)，(A≠2)，(X＞1)，可标记为 T1 、T2、F3、T4
（1）(A＞1)，(B=0)，(A≠2)，(X≤1)，可标记为 T1 、T2、F3、F4。
（2）(A＞1)，(B≠0)，(A=2)，(X＞1)，可标记为 T1 、F2, T3、T4；
（2）(A＞1)，(B≠0)，(A=2)，(X≤1)，可标记为 T1 、F2、T3、F4；
（2）(A＞1)，(B≠0)，(A≠2)，(X＞1)，可标记为 T1 、F2、F3、T4；
（2）(A＞1)，(B≠0)，(A≠2)，(X≤1)，可标记为 T1 、F2、F3、F4。
（3）(A≤1)，(B=0)，(A=2)，(X＞1)，可标记为 F1 、T2、T3、T4；
（3）(A≤1)，(B=0)，(A=2)，(X≤1)，可标记为 F1 、T2、T3、F4；
（3）(A≤1)，(B=0)，(A≠2)，(X＞1)，可标记为 F1 、T2、F3、T4；
（3）(A≤1)，(B=0)，(A≠2)，(X≤1)，可标记为 F1 、T2、F3、F4。
（4）(A≤1)，(B≠0)，(A=2)，(X＞1)，可标记为 F1 、F2、T3、T4；
（4）(A≤1)，(B≠0)，(A=2)，(X≤1)，可标记为 F1 、F2、T3、F4；
（4）(A≤1)，(B≠0)，(A≠2)，(X＞1)，可标记为 F1 、F2、F3、T4；
（4）(A≤1)，(B≠0)，(A≠2)，(X≤1)，可标记为 F1 、F2、F3、F4。

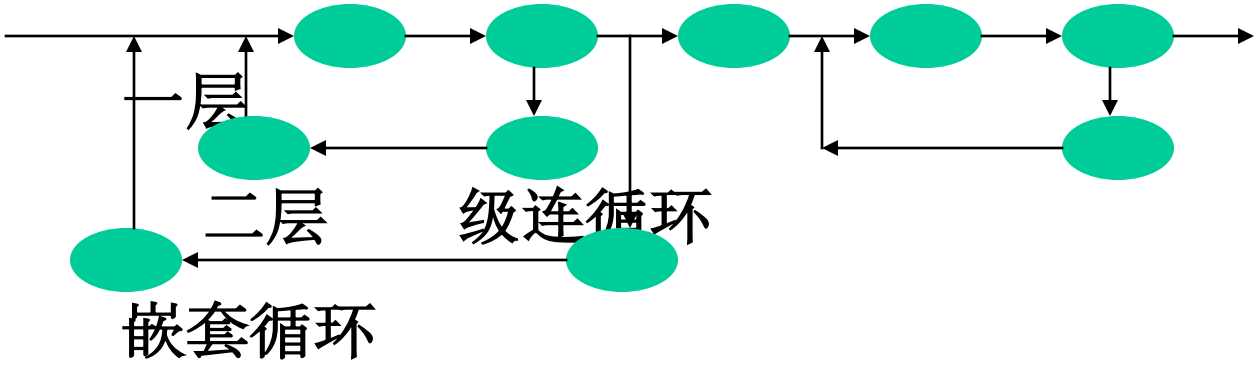
可以采用以下四组测试数据，从而实现条件组合覆盖。

测 试 用 例	覆盖条件	覆盖组合号	通过路径
【（2，0，4），（2，0，3）】	T1 T2 T3 T4	1、5	L1
【（2，1，1），（2，1，2）】	T1 F2 T3 F4	2、6	L3
【（1，0，3），（1，0，4）】	F1 T2 F3 T4	3、7	L3
【（1，1，1），（1，1，1）】	F1 F2 F3 F4	4、8	L2
【（3，0，3），（3，0，1）】	T1 T2 F3 F4	1、8	L4

●●●

这组测试用例实现了分支覆盖，也实现了条件的所有可能取值的组合的覆盖。

(3) 循环情况的路径选取



还要考虑循环变量的具体情况

关键路径的选取

主要功能路径
没有功能的路径
最短路径

…

3） 功能测试-基于规格说明的测试

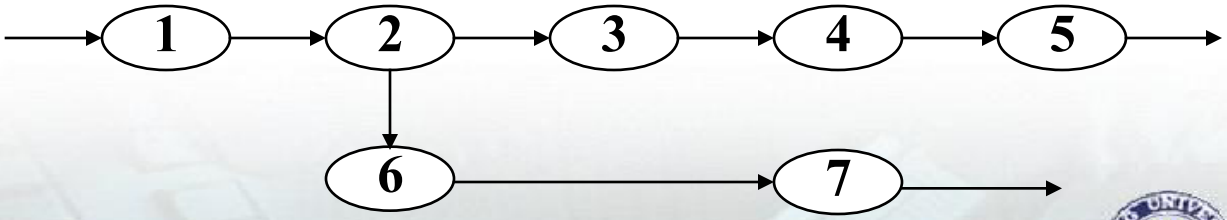
3.1 事务流测试技术

（1）基本概念：

①事务：以用户的角度所见的一个工作单元。

一个事务由一系列操作组成。其中某些操作可含有系统执行成分，或含有设备执行成分。

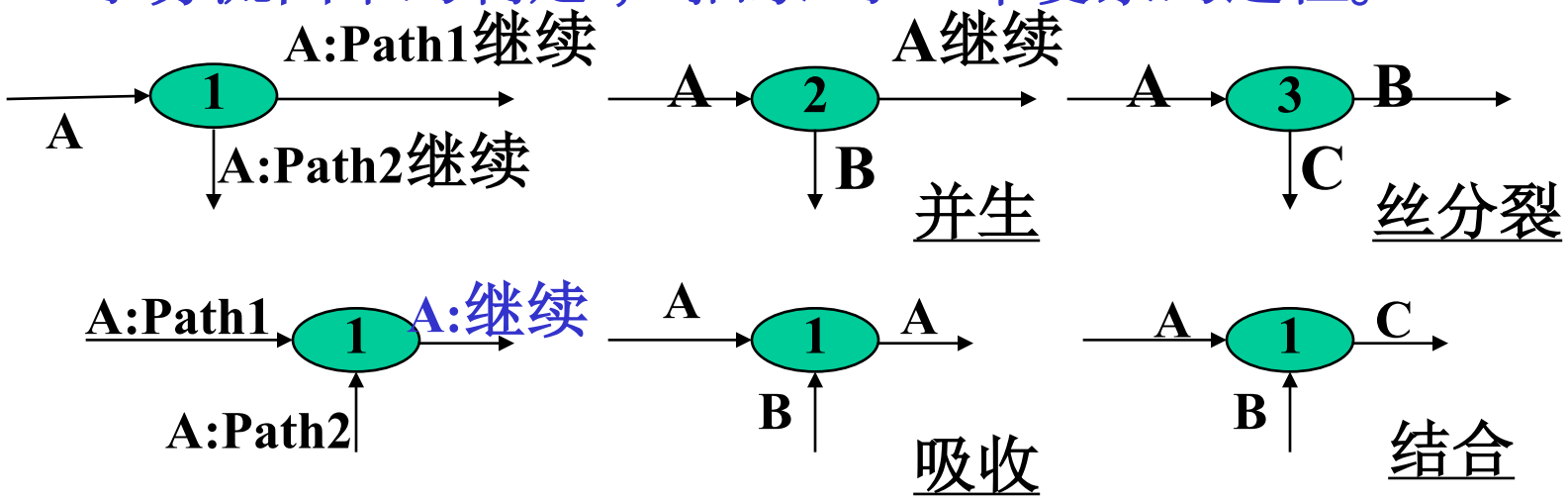
②事务处理流程（图）：系统行为的一种表示方法，为功 能测试建立了软件动作模式。其中使用了白盒测试中的一些概念，例如：分支，结点，链等。



（2） 与程序控制流程图的比较：

①事务流图是一种数据流图，即从操作应用的历史，观察数据对象。

②事务流图中的判定；“抽象”了一个复杂的过程。



③事务流图存在“中断”，把一个过程等价地变换为具有繁多出口的链支。

测试设备:路径分析器,测试用例数据库,

测试执行调度器, 路径敏化问题...

（3）测试步骤

第一步：获取事务流程图，即建立被测对象模型；

第二步：浏览与复审

主要对事务进行分类，为设计用例奠 定基础；

第三步：用例设计

涉及：覆盖策略，事务选取，路径敏化等；

第四步：测试设备开发:

路径分析器,测试用例数据库,

测试执行调度器, ...

第五步：测试执行；

第六步：测试结果比较。

3.2 等价类划分技术

(1) 基本概念

①等价类：输入域的一个子集，在该子集中，各个输入数据对于揭示程序中的错误都是等效的。即：以等价类中的某代表值进行的测试，等价于对该类中其他取值的测试。

②有效等价类：指那些对于软件的规格说明书而言，是合理的、有意义的输入数据所构成的集合。

-用于实现功能和性能的测试。

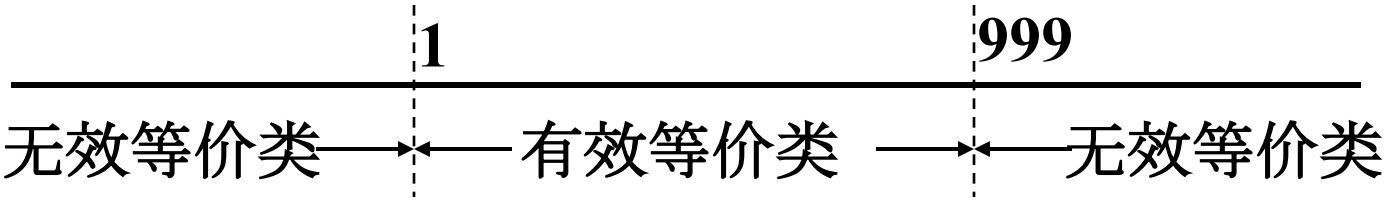
③无效等价类：指那些对于软件的规格说明书而言，是不合理的、无意义的输入数据所构成的集合。

-用于测试那些所实现的功能和性能不符合规格说明书的要求。

（2）等价类划分原则（指南）

① 如果输入条件规定了输入数据的取值范围或值的个数，则可以确定一个有效等价类和二个无效等价类。例如：

输入条件："...项数可以是1到999…"



② 如果输入条件规定了输入值的集合，或规定了"必须如何"的条件，则可以确定一个有效等价类和一个无效等价类。例如："标识符是一字母打头的…串。"则字母打头的--为一个有效等价类，而其余的—为一个无效等价类

③ 如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。

④ 如果输入条件规定了输入数据的一组值，而且软件要对每个输入值进行处理，则可以为每一个输入值确定一个有效等价类，为所有不允许的输入值确定一个无效等价类。

⑤ 如果输入条件规定了输入数据必须遵循的规则，则可以确定一个有效等价类（符合规则），和若干个无效等价类。

例如：“语句必须以；号结束”

注意：如果在已确定的等价类中各元素在软件中的处理方式不同，则应根据需要对等价类进一步进行划分。

（3） 测试用例设计

在确定了等价类之后，建立等价类表：

输入条件	有效等价类	无效等价类
……	……	……
……	……	……

（4） 实例研究

某一8位计算机，其十六进制常数的定义为：以0x或0X开头的数是十六进制整数，其值的范围是-7f至7f（大小写字母不加区别），如0x13，0X6A，-0x3c

第一步：建立等价类表

输入条件	有效等价类	无效等价类
十六进制整数	1、 0x或0X开头 1-2位数字串	4、非0x或非-开头的串 5、含有非数字且（a,b,c,d,e,f）以外字符 6、多于5个字符
	2、以- 0x开头的 1-2位数字串	7、-后跟非0的多位串 8、-0后跟数字串 9、-后多于3个数字
	3、在-7f至7f之间	10、小于-7f 11、大于 7f

第二步：为有效等价类设计测试用例

测试用例	期望结果	覆盖范围
0x23	显示有效输入	1，3
-0x15	显示有效输入	2，3

第三步：为无效等价类至少设计一个测试用例

测试用例	期望结果	覆盖范围
2	显示无效输入	4
G12	显示无效输入	5
123311	显示无效输入	6
-1012	显示无效输入	7
-011	显示无效输入	8
-0134	显示无效输入	9
-0x777	显示无效输入	10
0x87	显示无效输入	11

3.3 软件测试步骤

(1) 单元测试

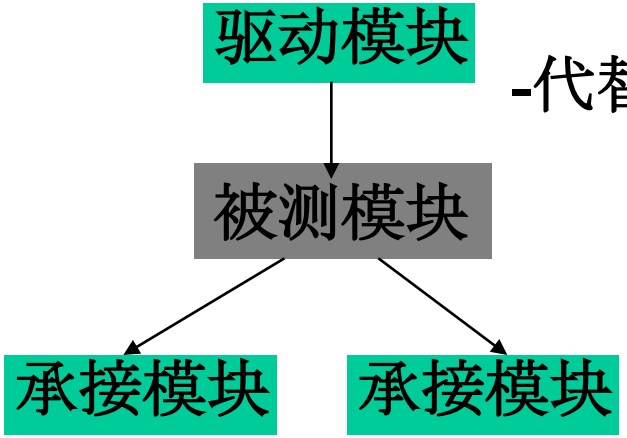
(2) 集成测试

集成测试是一种软件集成化技术

- 方式:自顶向下或自底向上
- 设计测试设备

  驱动模块

  承接模型



-代替原来的控制模块

-代替原来的被控模块

(3) 有效性测试

第七章

软件过程改善

能力成熟度模型（CMM）简介（*注：为了符合考纲，这里介绍的是CMM，基本思想类似于CMMI*）

1） 问题的提出

计算机软件的开发一直是广泛应用计算机的瓶颈。

解决这一问题，初期着重于研究一些新的开发方法和技术，

--对提高计算机软件的生产率和质量起到了很大的作用，但问题并没得到很好解决。

在80年代中期，美国工业界和政府部门开始认识到：在软件开发中，关键的问题在于软件开发组织不能很好地定义和控制其软件过程。

--从而使一些好的开发方法和技术都起不到所期望的作用。

在无纪律的、混乱的软件项目开发状态中，开发组织不可能从软件工程的研究成果中获益。尽管仍有一些软件开发组织能够开发出个别优秀软件，但其成功往往归功于软件开发组的一些杰出个人或小组的努力。

2、过程成熟度的基本概念

①软件过程能力：描述(开发组织或项目组)通过遵循其软件过程能够实现预期结果的程度。

用途：一个组织的软件过程能力，提供了一种预测该组织承担下一个软件项目可能结果的方法。

②软件过程性能：表示(开发组织或项目组)遵循其软件过程所得到的实际结果。

注意：软件过程能力与软件过程性能之间的关系：

❶一个是能够实现预期结果的程度，一个是得到的实际结果

❷一个项目的实际过程性能，可能并不充分反映其所在组织的整个过程能力。

（由于该项目的具体属性和执行该项目的环境所限）

③软件过程成熟度：

一个特定软件过程被明确和有效地定义、管理、测量和控制的程度。

指明：

- 一个软件开发组织软件过程能力的增长潜力；

--能力提高的基础性

- 表明一个开发组织软件过程的丰富多样性，

--能力提高的可能性

- 在各开发项目中运用软件过程的一致性。

--能力提高的持续性

这意味着：由于开发组织通过运用软件过程，使各项目执行软件过程的纪律性一致地增强，导致软件生产率和质量可以得到不断地的改进。

作用：随着一个软件开发组织的软件过程成熟度的提高，并通过该组织的方针、标准和组织机构等将其软件过程规范化和具体化，从而使得开发组织对有关管理和工程的方法、实践和规程等有明确定义。

④软件能力成熟度等级：

软件开发组织在走向成熟的过程中，几个具有明确定义的、可以表征其软件过程能力成熟程度的“平台”。

该平台（每一等级）包含一组过程目标。当一个软件开发组织达到其中一个目标时，则表明软件过程的一个(或几个)重要成分得到了实现，从而导致该组织软件过程能力的增长。

作用：每一个成熟度等级为达到下一个等级提供了一个基础。

⑤关键过程域：

过程域：互相关联的若干个软件实践活动和有关基础设施的集合，即《软件实践活动，基础设施》。

关键过程域：对某一成熟度等级将起到至关重要的过程域即它们的实施将对达到该成熟度等级的目标起保证作用，这些过程域被称为关键过程域。

每一软件过程成熟度等级均包含一组特定的关键过程域。

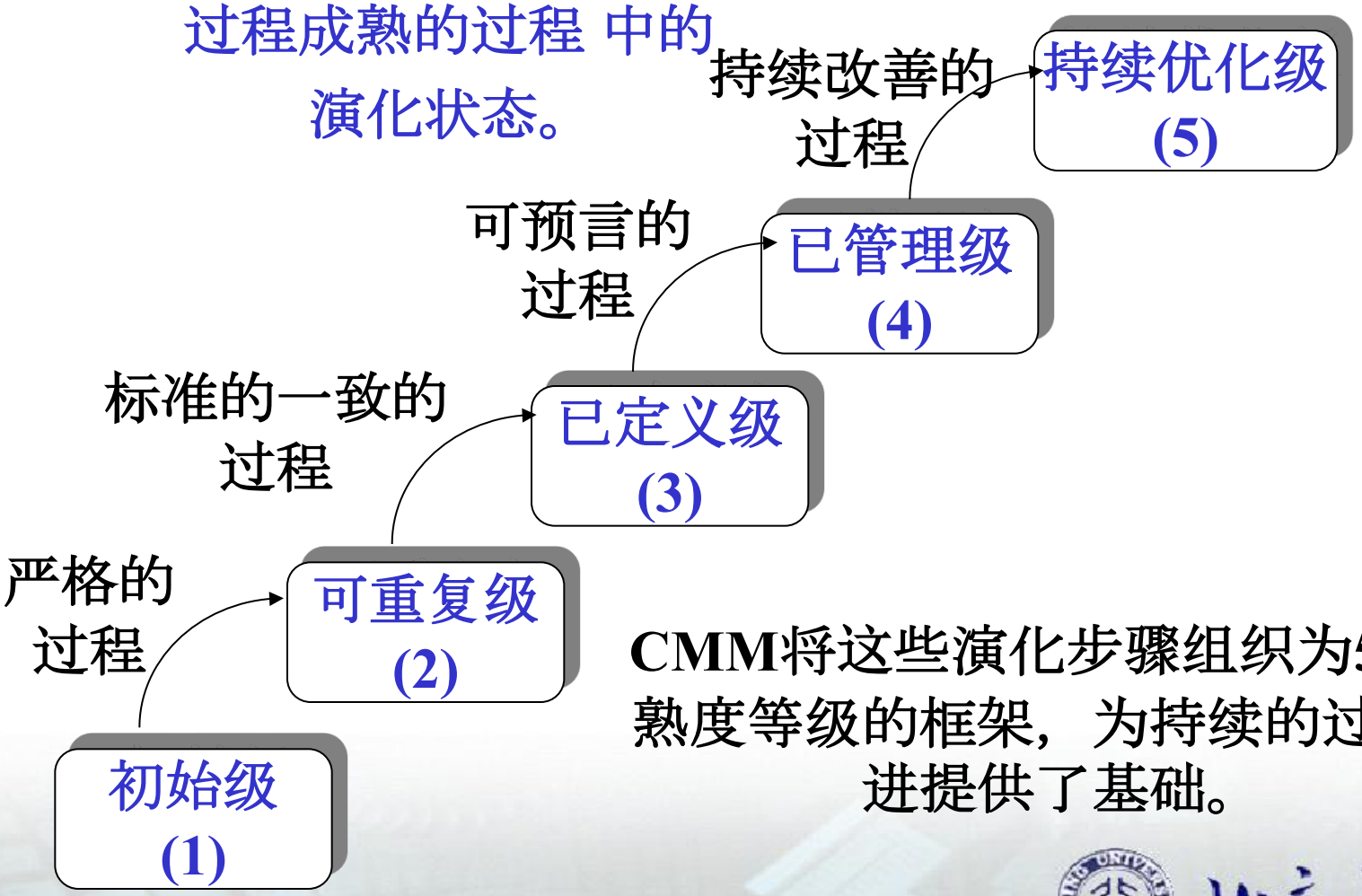
⑥关键实践：对关键过程域的实施起关键作用的方针、规程、措施、活动以及相关的基础设施的建立。

“关键实践”的表述，一般只给出“做什么”。

作用：通过实施一个关键过程域中所包含的关键实践（“小”的进化），才可以达到该关键过程域中的目标。

3、CMM的软件过程成熟度框架

通过成熟度级别，定义了在使软件过程成熟的过程 中的演化状态。



CMM将这些演化步骤组织为5个成熟度等级的框架，为持续的过程改进提供了基础。

可见，过程成熟度框架：

描述什么：一条从无序的、混乱的过程达到成熟的、有纪律的软件过程的进化途径。

怎么描述：集软件过程、软件过程能力、软件过程性能和软件过程成熟度等概念为一体。

用途：以软件过程成熟度框架，可以导出过程改进策略，为软件过程的不断改进的历程提供了一份导引图。

--指导软件开发组织不断识别出其软件过程的缺陷

--引导开发组织在各个平台上“做什么”改进（但它并不提供“如仍做”的具体措施。

基础：软件过程成熟度框架的基础是软件能力成熟度模型。

4、软件能力成熟度模型

--涉及组织，项目，过程能力等要素

模型作用：

指导软件开发组织

❶ 通过确定当前过程的成熟度，并识别出执行软件过程的薄弱环节，通过解决对软件质量和过程改进至关重要的若个问题

形成对其过程的改进策略

❷ 通过关注并认真实施一组有限的关键实践活动

稳步地改善其全组织的软件过程

使全组织的软件过程能力持续增长

软件能力成熟度等级

- **初始级**　　主要特征：
  - 组织：组织通常没有提供开发和维护软件的稳定的环境。
  - 项目：当发生危机时，项目通常放弃计划的过程，回复到编码和测试。
  - 过程能力：不可预测。**(unpredictable)**　由于：
    - ① 软件开发无规范；
    - ② 软件过程不确定、无计划、无秩序；
    - ③ 过程执行不“透明”；
    - ④ 需求和进度失控。

结果：项目的成败完全取决于个人的能力和努力；

软件性能随个人具有的技能、知识和动机的不同而变化，并只能通过个人的能力进行预测。

- 可重复级

实现了关键过程域：

软件配置管理、软件质量保证、软件子合同管理、
软件项目跟踪和监督、软件项目规划、需求管理。

主要特征：

–组织：将软件项目的有效管理过程制度化，这使得组织能够重复以前项目中的成功实践。

–项目：配备了基本的软件管理控制。

–过程能力：

① 可重复的：即对当前项目的需求分析后制定的，能重复以前的成功实践，尽管在具体过程中可能有所不同。

这是该级的一个显著特征

②基本可控的：即对软件项目的管理过程是制度化的。

- 对软件需求和为实现需求所开发的软件产品建立了基 线；
- 为管理、跟踪软件项目的成本、进度和功能提供了规范；
- 在项目的策划和服踪过程中规定并设置了监测点；
- 提供了当不满足约定时的识别方法和纠偏措施。

软件项目过程基本上是可视的

③过程是有效的：即对项目可建立实用的、已文档化的、已实施的、已培训的、已测量的和能改进的过程。

项目的过程基本是可特征化的

④项目是稳定的：即对新项目的策划和管理，有明确的管理方针和确定的标准（包括对分承制方），可使项目的进展稳定。

新项目的策划和管理是基于成功项目经验的

⑤是有纪律的：即对所建立和实施的方针、规程，对软件项目 过程而言，已进化为组织的行为。从而使软件开发组织能够保证准确地执行给定的软件过程。

- 已定义级

实现了可重复级（2级）的关键过程域：

软件配置管理、软件质量保证、软件子合同管理、软件项目跟踪和监督、软件项目规划以及需求管理，

实现了关键过程域：

组织过程焦点、组织过程定义、培训大纲、集成软件管理、软件产品工程、组间协调以及同行评审

主要特征：

组织：在组织范围内开发和维护软件的标准过程被文档化，其中包括软件工程过程和管理过程，它们集成为一个一致的整体。

项目：对组织的标准软件过程进行裁剪，来开发它们自己的定义软件过程。

过程能力：标准的和一致的。**(standard and consistent)**

由于：

①建立了“组织的标准软件过程”：即

- 关注的焦点转向组 织的体系和管理；
- 全组织建立了软件开发和维护的标准过程；
- 软件工程过程和软件管理过程，被综合为一个有机的整体，并且已经文档化。

② 建立了负责组织的软件过程活动的机构：即

- 在软件组织中存在负责软件过程活动的机构，并具体实施全组织的过程制定、维护和改进。
  其中包括全组织的人员培训，使之具备必须的技能和知识，能高效地履行其职责。

③ 项目定义的软件过程：即

- 项目能够依据其环境和需求等，通过剪裁组织的标准过程，使用组织的过程财富，自定义项目的软件过程。
  其中，允许有一定的自由度，但任务间的不匹配情况，应在过程策划阶段得到识别，进行组间协调和控制。

④组织可视项目的进展：即

- 项目定义的软件过程将开发活动和管理活功综合为一个协调的、妥善定义的软件过程；
- 明确规定了每一活动的输入、输出、标准、规程和验证判据。

管理者或软件项目负责人能够洞察

所有项目的技术进展、费用和进度

❺组织的软件能力均衡、一致：即

- 整个组织范围内的软件开发和维护过程已经标准化，
- 软件工程技术活动和软件管理活动都实现文档化的规范管理，
- 组织和项目的软件过程都是稳定和可重复的。
- 这种过程能力是建立在整个组织范围内对已定义过程中的活动、作用和职责的共同理解基础之上。

在整个组织范围内软件能力是均衡、一致的

- 定量管理级

实现了关键过程域：定量过程管理和软件质量管理。

主要特征：

–项目：项目减小过程性能的变化性，使其进入可接收的量化边界，从而达到对产品和过程的控制。

–组织：为软件产品和过程都设定了量化的质量目标。

–过程能力：可预言的。**(predictable)**

由于：

① 设置了定量的质量目标：即

- 组织对软件产品和过程设置了定量的质量目标；
- 软件过程具有明确定义和一致的测量方法与手段；

可以定量地评价项目的软件过程和产品质量

②项目产品质量和过程是受控和稳定的：即
可以将项目的过程性能变化
限制在一个定量的、可接受的范围之内。
产品质量和过程是受控和稳定的

③ 开发新领域软件的风险是可定量估计的：即
由于组织的软件过程能力是已知的，从而可以利用全组织的软件过程数据库，分析并定量地估计出开发新领域软件的风险。

④ 组织的软件过程能力是可定量预测的：即
过程是经测量的并能在可预测的范围内运行，一旦发现过程和产品质量偏离所限制的范围时，能够立即采取措施予以纠正。

- 持续优化级

实现了关键过程域：

缺陷预防、技术变化管理、过程变化管理

主要特征：

- 组织：关注于持续的过程改进。
- 项目：软件过程被评价，以防止过失重复发生，从中获得的教训散布给其它项目。
- 过程能力：持续的改善。**(continuously improving)**

(1)过程不断改进，即组织注重不断地进行过程改进。

- 组织有办法识别出过程的弱点，并及时地予以克服；
- 能够利用关于软件过程有效性的数据，识别最佳软件工程实践的技术创新，并推广到整个组织。

(2)缺陷能有效预防：即

软件项目组能分析并确定缺陷的发生原因，认真评价软件过程，以防止同类缺陷再现，并且能将经验告知其他项目组。

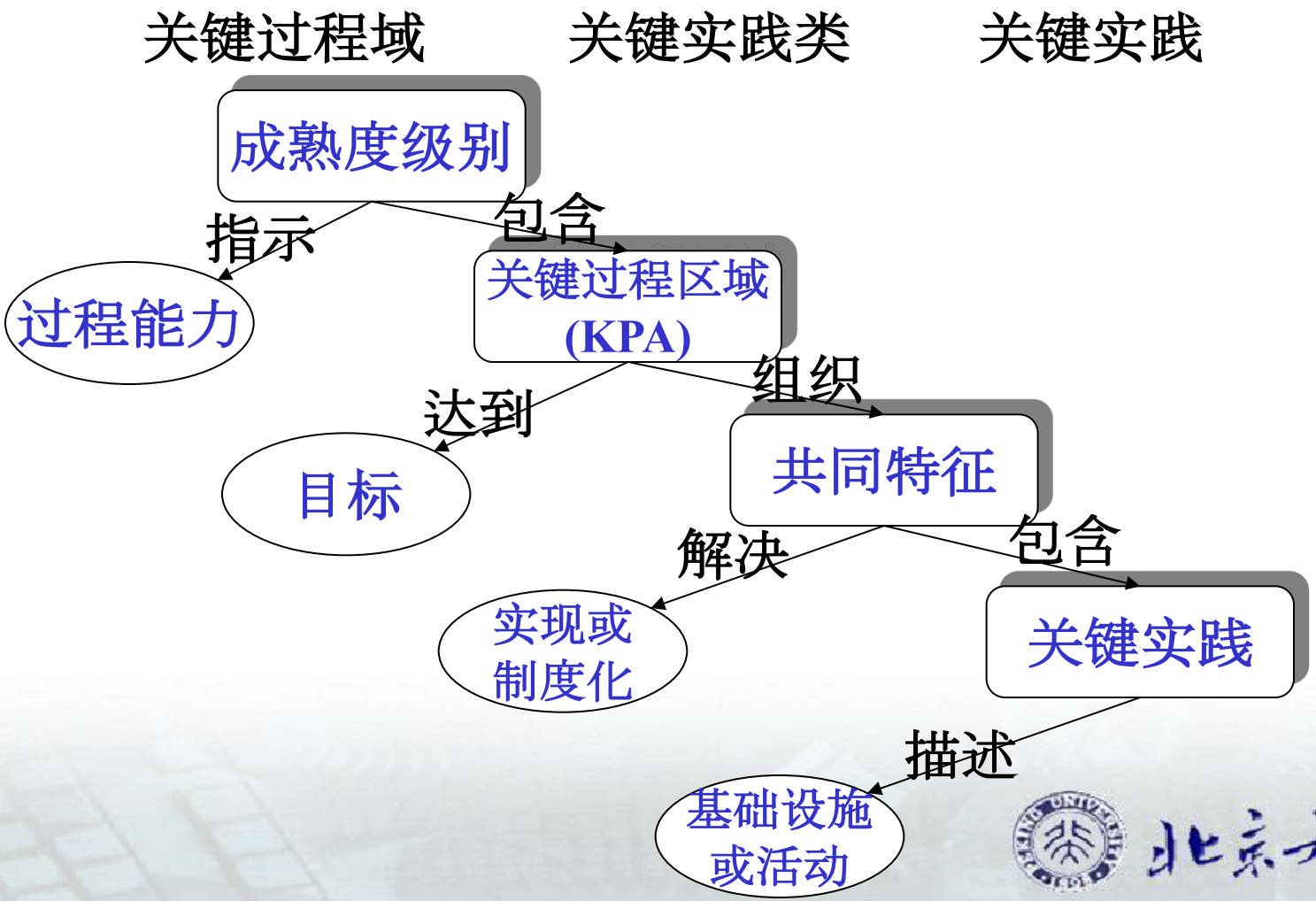
(3)组织的过程能力不断提高：即

组织既能在现有过程的基础上以渐进的方式，又能以技术创新等手段，不断努力地改善过程性能。

特别值得注意的是，在CMM定义的五个成熟度级别中，每一等级形成了一个必要的基础，从此基础出发才能达到下一个等级。因此，软件能力成熟度等级的提高是一个循序渐进的过程，即使一个软件开发组织具有实施较高成熟度等级的能力，也不能表明该组织可以跳越成熟度等级。

5、成熟度等级的内部结构

CMM的每个等级是通过三个层次加以定义的：



每个等级由几个关键过程域组成，它们共同形成一定的过程能力。

每个关键过程域都有一些特定的目标，为实现这些目标，将实现目标的关键实践组织为四个关键实践类。

每个关键实践类规定了相应部门或有关责任者应实施的一些关键实践。当关键过程域的这些关键实践都得到实施时，就能够实现该关键过程域的目标。

关键实践按类组织，描述了为有效实施并规范化关键过程域，应具备的基础设施和从事的活动。

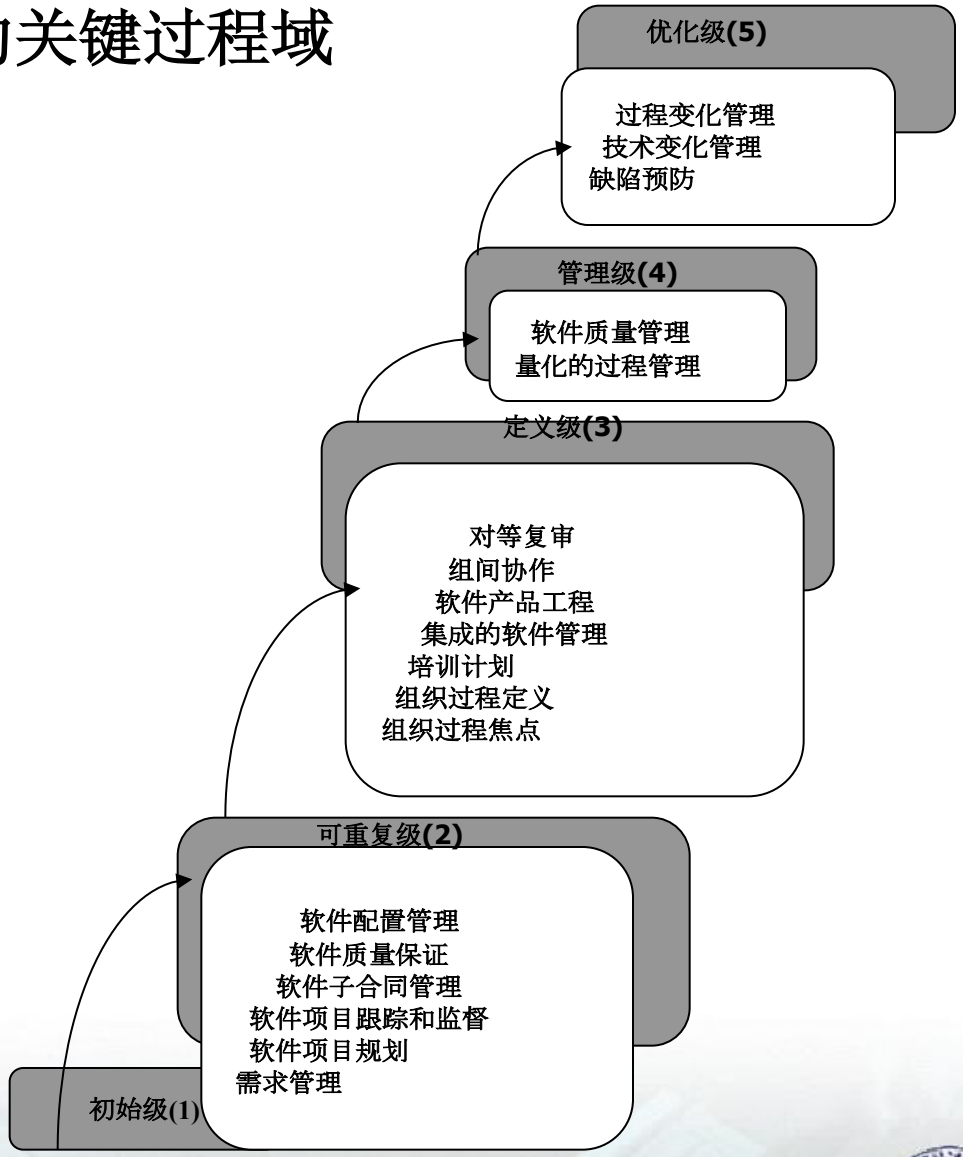
目标概括了一个关键过程域的关键实践，表明该关键过程域的范围、边界和意图。

可用来确定某组织或某项目是否已有效地实施该关键过程域。

一个成熟度等级的关键过程域表示了这些关键过程域的完全实现是达到该成熟度等级的*必要条件*；

一个关键过程域的关键实践表示了实施这些关键实践是实现该关键过程域目标的*必要条件*。

（1）各等级的关键过程域



例如：软件项目规划

❶所属等级：是2级的一个关键过程域。

❷目的：制定进行软件工程和管理软件项目的合理计划。这些计划是管理软件项目的必要基础，促进按选定的软件生存周期模型分阶段分工地进行软件开发。按阶段组织检查，实施控制。

❸目标：

(1) 软件生存周期已选定，并经评审确认；

(2) 对计划中的软件规模、工作量、成本、风险等已经进行估计;

(3) 软件项目的活动和约定是有计划的；

(4) 影响计划进度的关键路径是已标识的、且受控的；

(5) 影响计划进度的关键资源需求是已标识的；

(6) 文档化的软件开发计划已经正式评审，并确认；

(7) 在软件生存周期的里程碑处，对计划的执行有检查、有记录，问题有报告;

(8) 对于介入软件开发计划的软件负责人、软件工程师和有关人员进行了软件估计和计划方面的培训。

❹实现目标的关键实践

例如：(9) 选定有可管理规模的、预定阶段的软件生存周期模型 如： 1)瀑布型; 2)增量型；

3)渐进型；4)螺旋型：

5)逆向工程型。

(10) 标识为控制软件项目所必需的软件工作产品。

注：其他关键过程域的简单描述，可见P217-P218。

由此可见：

❶每个关键过程域只与特定的成熟度等级直接相关；

❷每个关键过程域指明一组相关的活动，当全部完成时，就能实现对增强过程能力至关重要的目标。

❸仅当一个关键过程域的全部目标均已达到时，该关键过程域才能实现。

❹ 对于一个组织来说，仅当其所有项目均已达到一个关键过程域的目标时，才可以说，该组织已使以该关键过程域为特征的软件过程(软件能力)规范化了。

（2）关键实践类

关键实践类：●指出各主要负责人(或部门)对该关键过程域的实施和规范化应起的作用和应负的责任。●决定关键过程域的实施和规范化是否有效、可重复、能持久。

每个关键过程域包含：四个关键实践类：

制定方针政策、确保必备条件、

实施软件过程、检查实施情况

❶制定方针政策

描述组织的高层管理者(决策者)应起的作用：一般说，应保证过程得以建立并持续有效。为此，应制定组织的方针或策略，规定高层管理者的支持或保障活动。

由组织的最高领导及其指定代理负责。

❷确保必备条件

描述项目负责人应起的作用：一般说，应保证解决实施软件过程所必需的先决条件。为此，应建立适当的资源和组织结构，组织必要的培训。

通常由项目负责人和计划、人事、等部门负责。

❸实施软件过程

描述软件开发的具体实施者应起的作用：一般说，应制定实施计划和规程，进行实践，跟踪实施，必要时采取纠正措施。

通常由软件项目人员和计划、质量等部门负责。

❹检查实施情况

描述管理者和软件质量保证部门应起的作用：一般说，应保证各项活动按照已建立的过程进行，还应确定过程实施的状态和有效性。为此，应组织适当的测量和分析、评审和审计。

由质量部门负责人、软件课题负责人和主管领导负责

四个关键实践类之间的关系是：

❶实施软件过程这一关键实践类中的关键实践，描述了为建立过程能力，过程实施者必须作些什么；

❷其他关键实践类中的实践，作为一个整体使实施软件过程中的实践规范化。

（3）关键实践

每个关键实践的描述由两部分组成：

❶前一部分：说明关键过程域的基本方针、规程和活动

--称为顶层关键实践

❷后一部分：通常是详细描述，可以包括例子

--称为子实践

注意：

❶关键实践描述应该做“什么”，而不强制要求应该“如何”实现目标。

❷其它替代的实践也可能实现该关键过程域的目标。

❸要合理地解释关键实践，以便判断关键过程域的目标是否已被有效地实现。

总结：--知识点

❶软件能力成熟度等级框架：

通过成熟度级别，定义了在 使软件过程成熟的过程 中的 演化状态。

在这一框架中，将过程能力成熟度分为5级：初始级，可重复级，已定义级，已管理级，持续优化级。

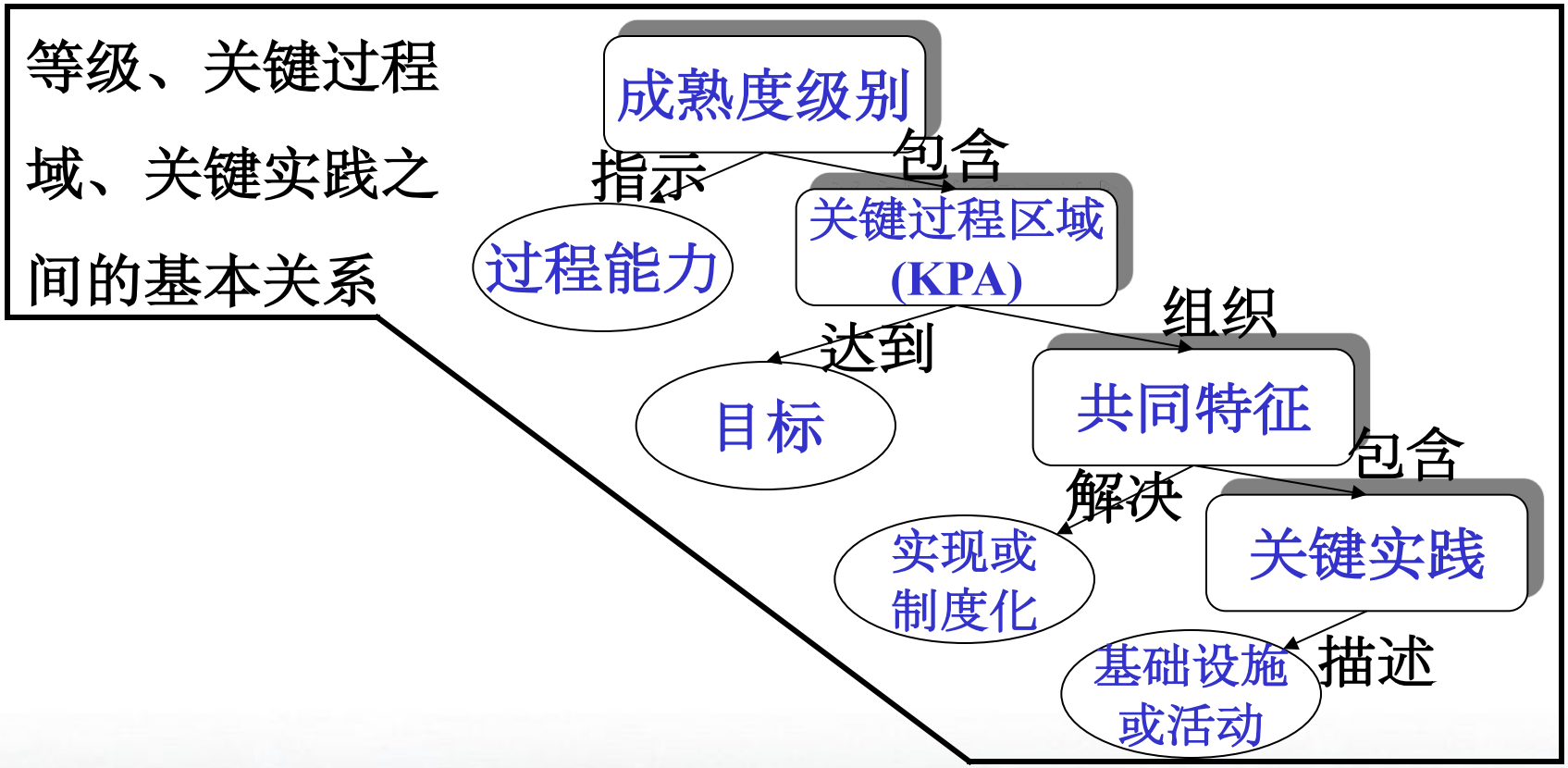
基本关系：

①通过过程改善，即实现了有关关键过程域的目标，才能演化为更高的一级，其中不可能“飞跃”；

②集软件过程、软件过程能力、软件过程性能和软件过程成熟度等概念为一体

即软件成熟度框架基础是软件能力成熟度模型。

❷软件能力成熟度模型：在每一等级的内部结构中，包括：关键过程域，关键实践类， 关键实践。



其中：

❸关键过程域：在框架的某一“平台”上，其实施将对达到下一成熟度等级的目标起保证作用的过程域，被称为关键过程域。

基本关系：

每一关键过程域包含一组关键实践。并按“共同特征”组织为每一级的关键实践类：制定方针政策，确保必备条件，实施软件过程，检查实施情况。

❹公共特征

实施承诺(Commitment to Perform)

描述为了保证软件过程的建立和持续执行软件开发组织所需采取的活动，包括：方针政策、高层管理者的保障、其它责任等

实施能力

描述为了很好地实施软件过程所需的先决条件。包括：资源、组织结构、培训等

实施活动

描述为了实现关键过程域所需的角色及其进行的活动或过程。

度量与分析

描述为了度量软件过程和分析度量结果所需的活动

验证实现

描述为了保证过程的实施情况和所定义的过程完全一致所需的验证步骤。

❺关键实践：描述了对关键过程域的实施起关键作用的方针规程、措施、活动以及相关的基础设施。在表述中，一般只给出“做什么”，而不规定“如何做”

通过实施一个关键过程域中所包含的关键实践，才可以达到该关键过程域中的目标。

❻各级包含的关键过程域：

可重复级：软件配置管理，软件质量管理，子产品工程项目跟踪和监督，软件项目规划，需求管理

已定义级：对等复审，组间协作，软件产品工程，集成的软件管理，培训计划，组织过程定义，组织过程焦点

已管理级：软件质量管理，量化的过程管理

持续优化级：过程变化管理，技术变化管理，缺陷预防

总结：--基本思想

❶ 随项目的规模和复杂度增长，在解决软件质量和生产率问题中，目前

更加关注组织和管理。

其突破点为过程改善，增强过程能力。

主要理论基础为费根堡姆的质量体系：

“在制造及传递某种合乎特定质量标准的产品时，必须配合适当的管理及技术作业程序，这些程序所组成的结构，称之为质量体系”。

❷CMM是（依据软件过程特点）质量体系概念的一个实现