软件工程

（1） 软件工程知识体系

* **软件需求：**需求工程过程、需求获取、需求分析、需求规格说明、需求验证、需求管理
* **软件设计：**基本概念、关键问题、软件构成与体系结构、质量分析与评价、软件设计符号、策略与方法
* **软件构造：**降低复杂性、预知多样性、结构化验证、使用外部标准
* **软件测试：**基本概念、测试级别、测试技术、测试相关度量、测试过程管理
* **软件维护：**基本概念、维护过程、关键问题、维护技术
* **软件配置管理：**配置过程管理、配置识别、配置控制、配置状态报告、配置审计、软件发布管理与交付**。**
* **软件工程管理：**组织管理、过程/项目管理、软件工程度量**。**
* **软件工程过程：**基本概念、过程基础实施、过程度量、过程定义、定性分析、过程实施与变更。
* **软件工程工具与方法：**软件工程工具、软件开发方法**。**
* **软件质量：**软件质量概念、SQA与V&V的目的与计划、 SQA与V&V的活动、适用于SQA与V&V的度量

# 第一章

# 1.1 背景

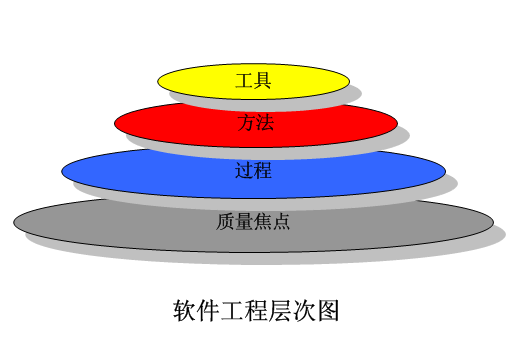
* **软件危机**：软件开发中普遍存在的费用高、开发过程不易控制、工作量估计困难、软件质量低、软件项目失败率高以及软件维护成本和难度大等
* **解决方法**： 1）**进行软件项目管理**：软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

**2）软件研发技术的研究：**软件研发方法学、软件工具和软件工程环境。

# 1.2 软件工程定义

**软件工程：**（1）将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程应用于软件。（2）在（1）中所述方法的研究。

**软件工程层次：**

****

（1）**过程：**建立了一个框架，构成了软件项目管理控制的基础，并建立了一个环境，以便于：

* + **技术方法的采用、**
  + **工程产品（模型、文档、数据）的产生、**
  + **里程碑（milestone，工作产品的目标及条件的结合)的建立、**
  + **质量的保证及正常变更的正确管理。**

（2）**方法：**提供了建造软件在技术上“如何做”。

**方法覆盖面：沟通、需求分析、设计、编程、测试和支持。**

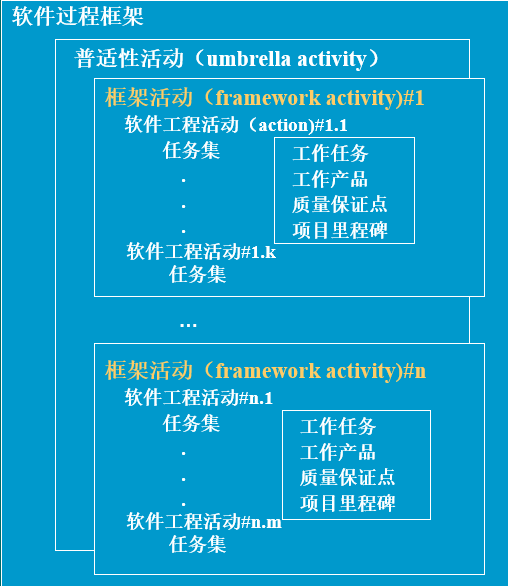
（3）**工具：** 对过程和方法提供了自动或半自动的支持。工具集成起来，建立一个软件开发的支持环境，称为计算机辅助软件工程（CASE）。

（4）**质量焦点：任何工程必须以对质量的承诺为基础。软件工程的根基在于质量关注点（quality focus)。**

# 1.3 过程框架、CMMI、PSP与TSP

**1、过程框架**

过程框架（Process framework）定义了若干框架活动，为完整的软件开发过程建立了基础。这些框架活动可广泛的应用于所有的软件开发项目。



通用框架活动：

* **沟通：包含了与客户之间的交流和协作，还包括需求获取以及其他相关活动。**
* **策划：为后续的软件工程工作制订计划。该计划描述了需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度计划。**
* **建模：包括分析与设计两个action的建模（和/或生成规格说明）。创建模型有助于理解需求、实现需求。**
* **构建：包括编码和测试。**
* **部署：包括交付、安装、评价、反馈。**

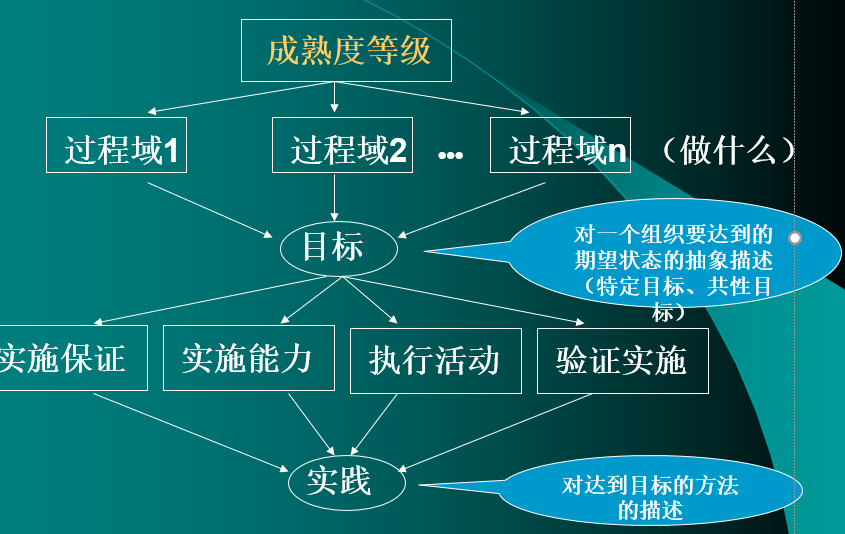
**不同的应用中过程的细节可能不一样，但框架活动是一致的。**

普适性（umbrella）活动一般包括：

* **软件项目跟踪与控制**
* **风险管理**
* **软件质量保证**
* **正式技术评审**
* **度量**
* **软件配置管理**
* **可复用管理**
* **工作产品的准备和生产**

**2、能力成熟度模型集成（CMMI）**

**（1）分级版本的CMMI**



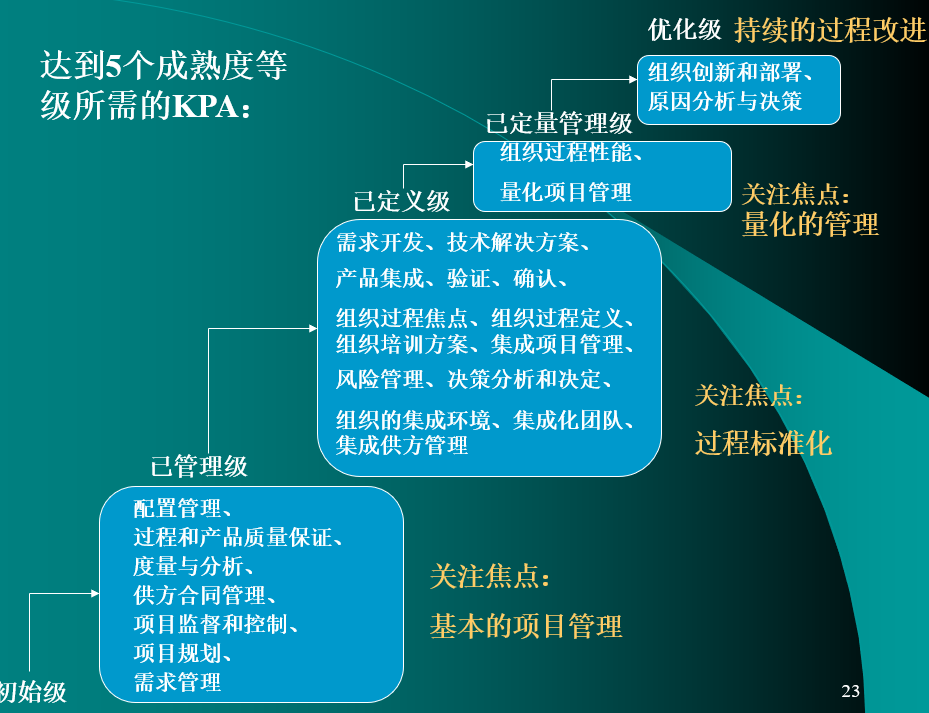
**① 初始级（Initial）：**在初始级，企业一般不具备稳定的软件开发与维护环境。项目成功与否在很大程度上取决于是否有杰出的项目经理和经验丰富的开发团队。此时，项目经常超出预算和不能按期完成，组织的软件过程能力不可预测。

**② 已管理级(Managed)：**在可重复级，组织建立了管理软件项目的方针以及为贯彻执行这些方针的措施。组织基于在类似项目上的经验对新项目进行策划和管理。组织的软件过程能力可描述为有纪律的，并且项目过程处于项目管理系统的有效控制之下。

**③ 已定义级（Defined）：**在该级中，组织形成了管理软件开发和维护活动的标准软件过程，包括软件工程过程和软件管理过程，即建立起文件化的“标准软件过程”。项目依据标准定义自己的软件过程进行管理和控制。达到了这一级，表明组织的软件过程能力“成熟了”。

**④ 已量化管理级（Quantitatively Managed）：**指使用统计和其他量化方法来控制过程。在该级，组织对软件产品和软件过程都设置定量的质量目标，并能对跨项目的重要的过程活动的效率和质量予以度量。组织的软件过程能力可描述为可预测的，软件产品具有可预测的高质量。

**⑤ 优化级（Optimizing）：**该级，组织通过预防缺陷、技术创新和调整过程等方式，不断提高软件过程能力以适应业务需要。组织的软件过程能力可描述为持续改善的。



CMMI分级模型中的

**实施保证：**制定过程改进的策略。

**实施能力：**提供项目和（或）组织具有实现过程改进所需要的资源。

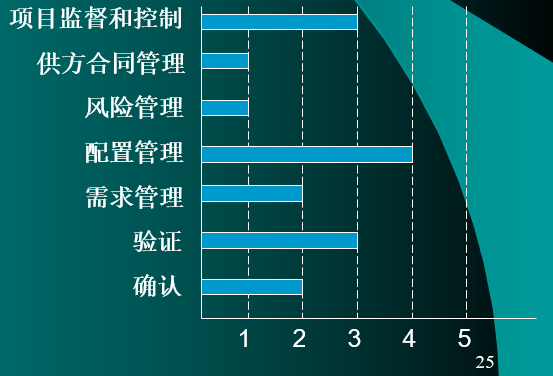
**执行活动：**包括收集、度量和分析与过程相关的数据。

**验证实施：**验证过程活动和产品是否遵循规程。

随着某个成熟度等级的一组过程域的目标的实现，则提高了组织的过程成熟度并得到过程改进。

**（2）连续性版本的CMMI结构**

该模型不根据离散级来分类，它为每个过程域赋予一个1-6级的能力评价。将过程域为4类：**过程管理、项目管理、工程、支持**。对不同的过程域，组织机构会在不同的成熟度级别上运作，成熟度评估的结果不是一个值，而是一组值，是一个能力概图。



**3、个体软件过程（PSP）**

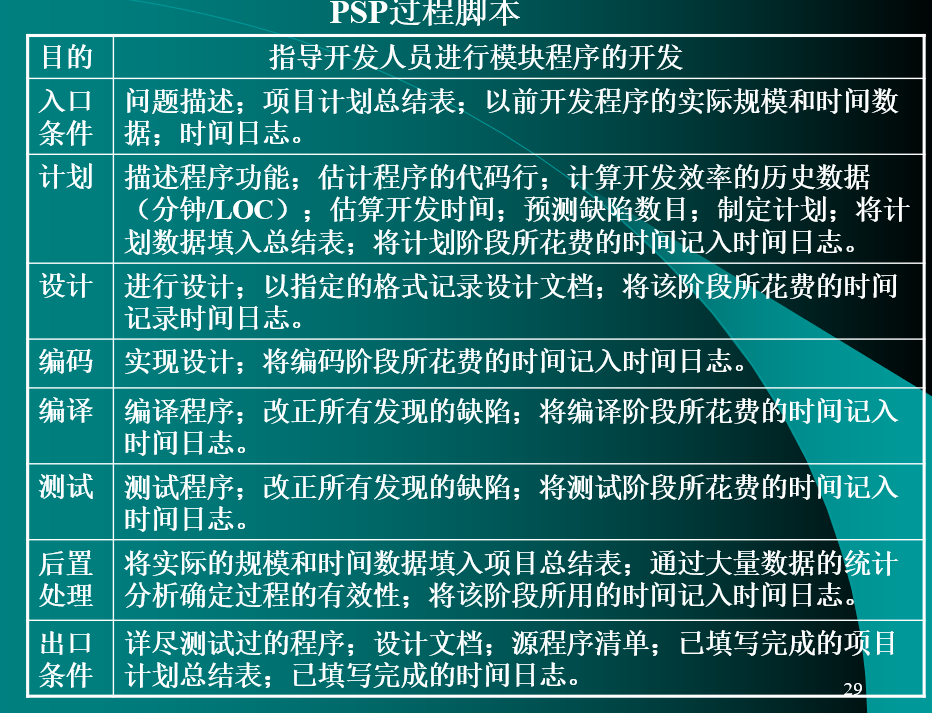
**个体软件过程（Personal Software Process）**是一种用以指导软件开发人员控制、管理、改进个人工作方式的自我完善过程，是一个包括软件开发表格、指南和规程的结构化框架。

**作用：**PSP能够指导软件工程师如何保证自己的工作质量，估算和规划自身的工作，度量和追踪个人的表现，管理自身的软件过程和产品质量。PSP的实施有助于组织CMMI目标的实现。PSP着眼点在于软件缺陷的预防。

**个体软件过程框架**

**PSP过程流程：**





**PSP进化框架共有4级：**

**• 个体度量过程（Personal Measurement Process）：** PSP0和PSP0.1

**• 个体规划过程（Personal Planning Process）：** PSP1和PSP1.1

**• 个体质量管理过程（Personal Quality Management Process） ：** PSP2和PSP2.1

**• 个体循环过程（Cycle Personal Process）：**PSP3

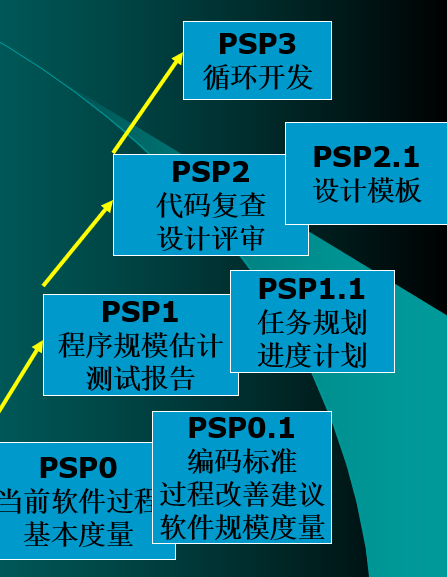
在PSP的前面三级进化框架中，后一个版本都是前一个版本的增强版。

**PSP0**目的是建立**个体过程基线**，包括计划，开发（设计、编码、编译和测试）及后置处理，并对过程中主要任务和工作结果进行度量。

**PSP1**重点是**个体计划**。

**PSP2**重点是**个体质量管理**。

**PSP3**目标是把个体开发小程序所能达到的生产率和生产质量，延伸到大型程序，其方法是采用螺旋式上升过程，即**迭代增量式开发方法**。



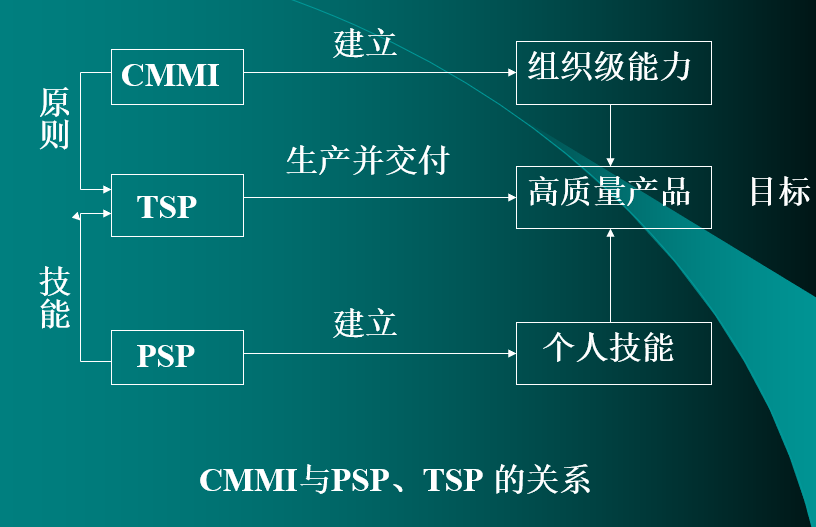
**PSP规范为软件工程师提供了发展个人技能的一个阶梯式的进化框架，以循序渐进的方法逐步达到开发无缺陷产品的目标。**

**评价：**PSP代表的是一种严格有序的、基于度量的软件工程方法，对第一线人员产生文化冲击。未被广泛采用的主要原因在于个人和软件开发组织的惯性。

**4、团队软件过程（TSP）**

**TSP （Team Software Process）的目标**是建立一个能“**自我管理**”的项目团队。它定义了每个团队成员的角色和责任；跟踪定量项目数据（包括生产率和质量）；确定适合该项目的团队过程和执行此过程的具体策略；定义适合团队工作的本地标准；持续评估风险并采取风险避免措施；跟踪、管理和报告项目状态。

**框架活动：**项目启动、高层设计、实现、集成、测试、后验。



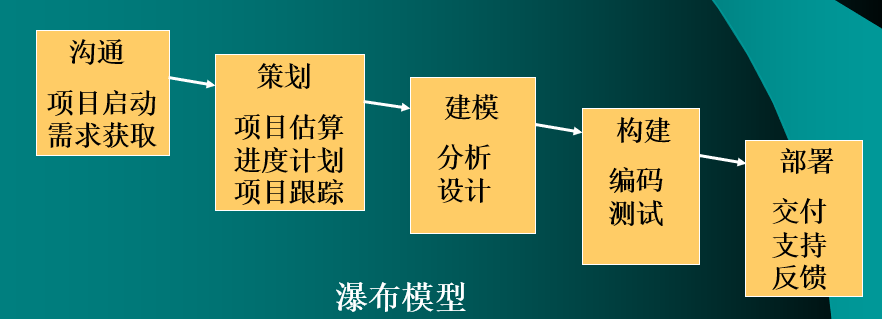
# 1.4软件过程模型

**框架活动**：沟通、策划、建模、构建、部署。

这些活动如何衔接，开发过程中采用什么样的策略，应遵守什么样的规定和制约，将这些活动及其关联（忽略不必要的细节）用一种模型表示出来，称为**软件过程模型**（或**软件开发模型**）。

应用过程模型的**目的**是提高软件质量、项目的可管理性以及对于交付时间和项目费用的可预测性，这样把规模大、结构复杂、管理复杂的软件开发变得容易控制和管理**。**

**⑴ 瀑布模型（线性顺序模型）**



**特点**：

• 提供了软件过程模型的基本框架（模板）。

• 强调了每一阶段活动的严格顺序。

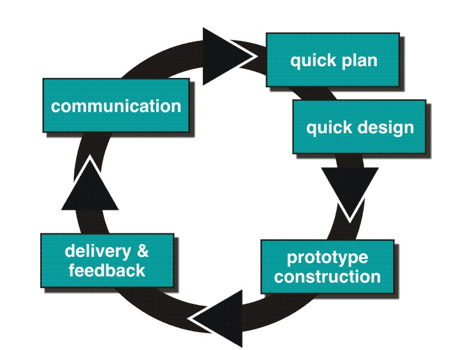
• 质量保证观点：以经过评审确认了的阶段工作产品（文档）驱动下一阶段的工作，便于管理。

• 是一种整体开发模型，程序的物理实现集中在开发阶段的后期，用户在最后才能看到自己的产品。

• 适合于用户需求明确、完整、无重大变化的软件项目开发。

**（2） 原型模型**

在用户不能给出完整、准确的需求说明，或者开发者不能确定算法的有效性、操作系统的适应性或人机交互的形式等许多情况下，可以根据用户的一组基本需求，快速建造一个原型（可运行的软件），然后提交、评估，进一步精化（迭代）、调整原型，使其满足用户的要求，也使开发者对将要做的事情有更好的理解。



**原型的分类：**

**• 抛弃型：**主要用于需求分析阶段，针对开发目标模糊、用户与开发者对项目都缺乏经验的情况。建立原型的目的是为了搞清用户的需求，确定所期望的特性，探索各种方案的可行性。产生完整、一致、准确的需求说明。

**• 实验型：**主要用于设计阶段，通过原型验证设计方案的可行性。原型或成为设计结果的一部分或抛弃。

**• 演化型：**用于整个开发阶段。原型经过不断扩充，演化为最终的软件系统。

**存在的问题**：

⑴ 为了使原型尽快的工作，没有考虑软件的总体质量和长期的可维护性。

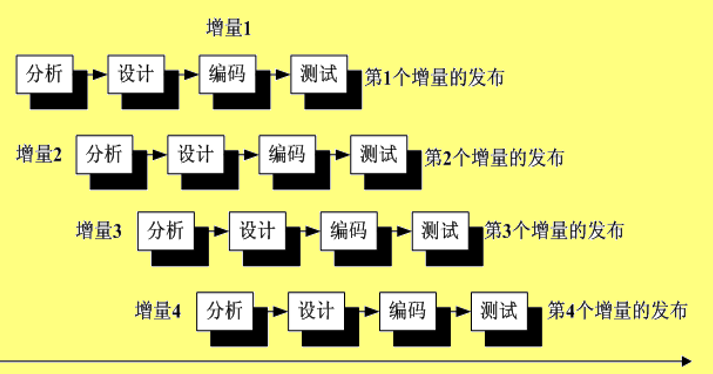
⑵ 为了演示，可能采用不合适的操作系统、编程语言、效率低的算法，这些不理想的选择成了系统的组成部分。

⑶ 开发过程不便于管理。

**有效的使用原型模式是：**建造原型仅是为了定义需求，之后就被抛弃（或被部分抛弃），实际的软件在充分考虑了质量和可维护性之后才被开发。

**（3） 增量模型**

是一种渐进地开发逐步完善的软件版本的模型



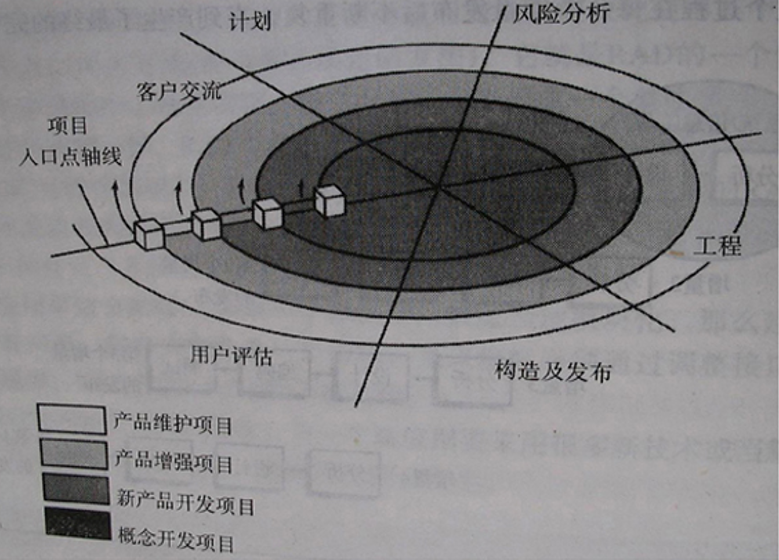
**特点：**

• 反复的应用瀑布模型的基本成分和原型模型的迭代特征，每一个线型过程产生一个“增量”的发布或提交，该增量均是一个可运行的产品。

• 第一个增量往往是核心产品，即满足用户的基本需求，提供给用户评估的平台，以便制定下一个增量计划。

**（4） 螺旋模型**

对于复杂的大型软件，开发一个原型往往达不到要求。螺旋模型将瀑布模型和增量模型结合起来，加入了风险分析。在该模型中，软件开发使一系列的增量发布，早期的迭代中，发布的增量可能是一个理论模型或原型，在以后的迭代中，逐步产生更加完善的系统版本。每个演进过程，除了考虑风险，还要标记里程碑。



螺旋的第一圈可能产生产品的规格说明，再下面的螺旋可能用于开发一个原型系统，随后可能是软件的更完善的版本。每一圈还要根据用户评估的反馈对项目计划（包括进度、费用）进行调整。

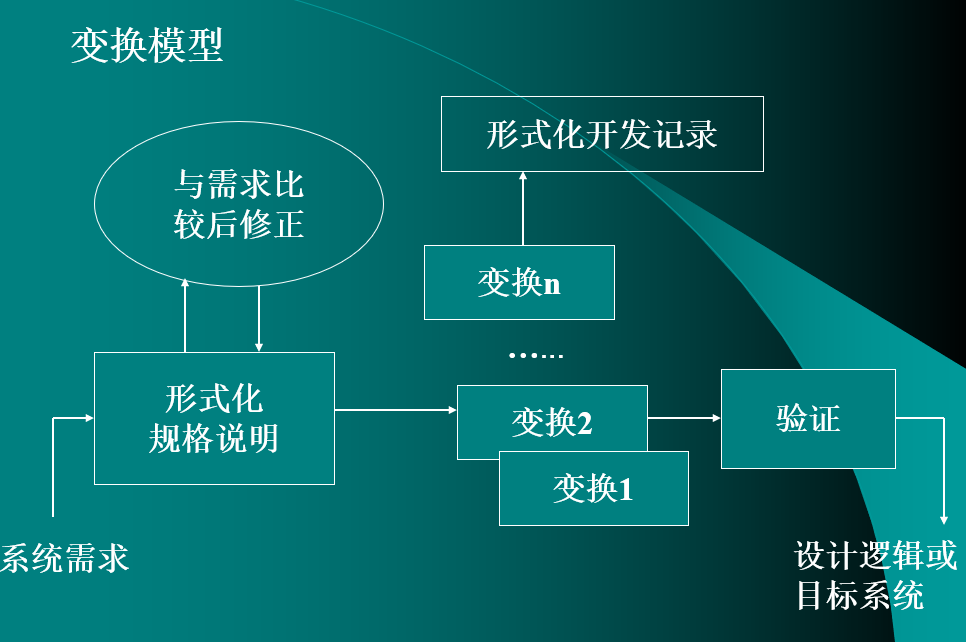
**特点：**

• 适合于大型系统的软件开发，随着过程的进展演化，开发者和用户能够更好的识别和对待每一个演化级别上的风险。

• 演进的方法不一定可控，依赖于相当丰富的风险评估经验来保证成功。

**（5） 形式化方法模型**

是基于形式化语言和程序变换的模型，因此，也称变换模型。从软件需求形式化说明开始，经过一系列的数学变换和正确性证明，最终得到系统高层的设计逻辑或低层的目标程序。形式化方法使开发者应用一个严格的数学符号体系来表示、构造和验证系统，从而大大提高软件的可靠性。



• **基于模型的方法**

基于模型的方法使用数学上的概念如集合、关系、函数、结构等概念为系统建模。它们能展现系统的状态以简化对某些行为的描述，即将系统操作表达成系统状态模型的改变。基于模型的描述语言及方法如Z、VDM(Vienna Definition Method)、B、Petri Nets等。

**• 代数方法**

系统用类型操作和它们之间的联系来描述。代数方法适合于对接口的描述。这里接口被定义为一组对象类或抽象数据类型的集合。

**特点**：

• 该模型迫使对系统需求的分析在软件开发的早期阶段完成。在这个阶段改正错误比在系统被交付之后修改错误要经济得多。

• 形式化描述是对非形式化描述技术的补充，可以用来精化非形式化的详细的系统需求描述。描述是精确的和无二义的，避免了由于语言误解而产生的一些问题。形式化描述可用来描述标准。

• 最适合用于安全性、可靠性和保密性等性能要求极高的系统。

• 开发成本较高，往往不能快速开发。

• 需要严格的数学理论和开发环境的支持。

• 难以与用户进行沟通。

*形式化过程模型的一个扩展，称为净室软件工程（cleanroom software engineering)或净室模型，它除了强调分析和设计上的严格性，以及使用基于数学的正确性证明来对设计模型的每个元素进行形式化验证外，还强调了统计质量控制技术。*

***基本思想：***

*力求在分析和设计阶段就消除错误，确保正确，然后在无缺陷或“洁净”的状态下实现软件的制作。*

***关键技术：***

*• 基于统计过程控制之下的增量开发。*

*“盒”（封装系统或某些方面）结构设计入手→*

*→正确性验证→统计使用测试*

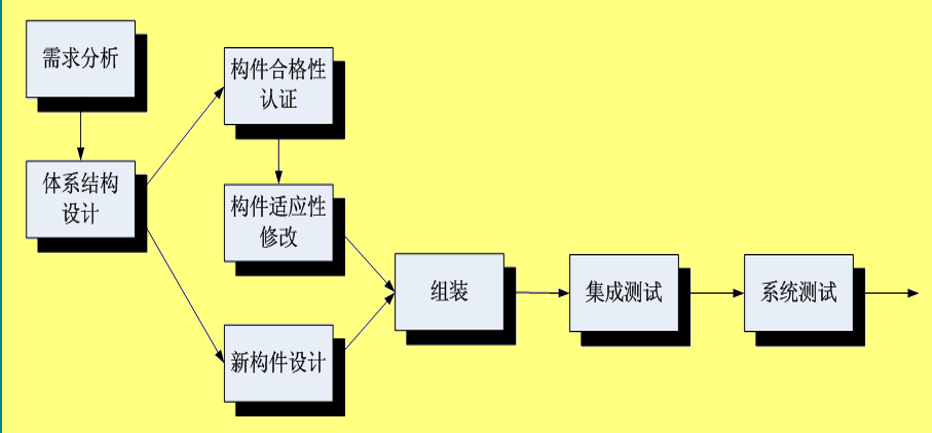
*• 基于函数的规范、设计、验证。*

*• 统计使用测试（按照对用例使用的概率分布来确定进一步测试）和软件可靠性认证。*

**（6）构件组装模型**

构件（component)也称为组件，是一段实现一系列有确定接口的程序体，具有自己的功能和逻辑，能同其他构件组装起来协调工作。

该模型支持软件重用，对缩短软件开发周期、降低项目成本有重要的现实意义。同时，建造符合某应用领域体系结构标准的构件，可以用来搭建分布式的、跨越不同操作平台的软件，扩展了软件的应用前景，促进了软件标准化、商品化的发展。

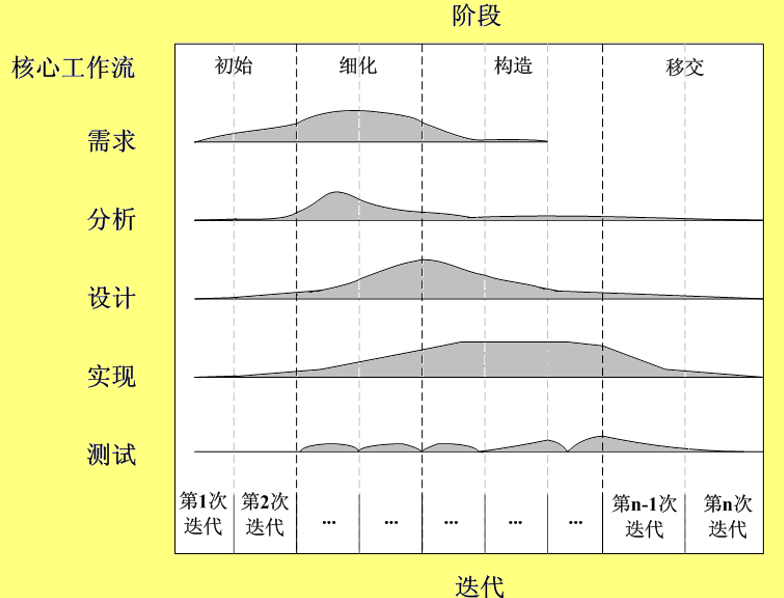


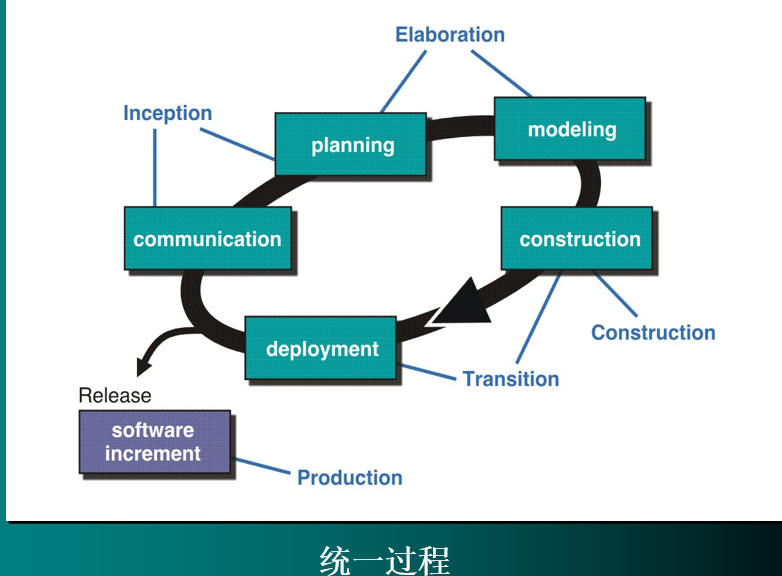
# 1.5统一过程-RUP

**RUP是一个二维的迭代结构:**

**横轴**为生命周期的四个阶段——起始、细化、构建和产品化，各阶段结束于一个里程碑。

**纵轴**为九个工作流（RUP2000版）——业务建模、需求、分析设计、实施、测试、部署、配置(称为核心过程工作流）与变更管理、项目管理、环境（称为核心支持工作流）。





**起始（inception）：**定义业务需求、提出系统大致的架构，识别各种资源，评估主要风险，制定开发计划。

**细化（elaboration）：**扩展起始阶段定义的用例，扩展体系结构以包括各类视图—用例模型、分析模型、设计模型、实现模型和部署模型，修改项目计划。

**构建（construction）：**以体系结构模型作为输入，开发或获取软件构件，确保最终用户能够操作用例。

**转换（transition）：**软件被提交给用户进行Beta测试和验收测试，根据缺陷进行必要的变更，创建系统发布所必要的支持信息。

**生产（production）：**持续的监控软件的运行并提供运行环境的技术支持。

在构建、转换和生产阶段的同时，下一个增量的工作已开始。



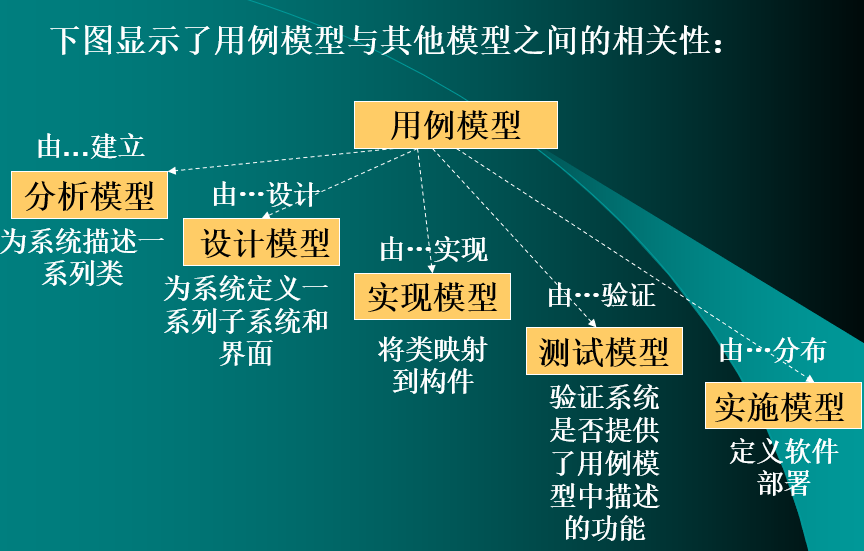
**2**、**RUP的主要特点：**

**（1）用例驱动**

用例作为系统分析、设计、实现和测试的基本输入。即用例不只是一种确定系统需求的工具，它还能驱动系统的设计、实现和测试的进行。

基于用例模型，开发人员可以创建一系列实现这些用例的设计模型和实现模型。开发人员可以审查每个后续建立的模型是否与用例模型一致。

测试人员测试实现以确定实现模型的构件是否实现了用例。所以用例启动了开发过程，还使开发过程结合为一体。开发过程是沿着一系列从用例得到的工作流前进的。



**（2）以构架（Architecture)为中心**

软件系统的构架从不同角度描述了即将构造的系统，它刻画了系统的整体设计，去掉了细节部分，突出了系统的重要特征，包含了系统中最重要的静态结构和动态行为。

构架是根据应用领域的需要逐渐发展起来的，并在用例中得到反映。每种产品都具有功能和表现形式，功能与用例对应，表现形式与构架对应。用例与构架是相互影响的，用例在实现时必须符合于构架，构架必须预留空间以实现现在或将来所有需要的用例。

**（3）迭代与增量的过程**

迭代指工作流中的步骤，是反复求精的概念；增量指产品中增加的部分,是逐块建造的概念。

迭代过程要处理一组用例，这组用例合起来能扩展所开发产品的可用性，后续的迭代过程建立在前一次迭代过程末期所开发的产品上。

构架提供了一种结构来指导迭代过程中的工作，用例则确定了目标并驱动每次迭代的工作。

**（4）基于构件**

统一过程所构造的软件系统，是由软件构件通过明确定义的接口相互连接所建造起来的。

**5）使用UML**

统一过程使用UML来制定软件系统的所有蓝图，UML是整个统一过程的一个完整部分，他们是共同发展起来的，它强调创建和维护模型。

**6）过程可剪裁**

RUP是一个过程模式，或过程定义（PD）而非过程（P）。可以根据系统的规模、项目的性质、开发组织的经验对过程加以裁剪。

另外，在**人员方面**，RUP定义了角色的概念，从角色视角对不同人员从事的不同活动进行了规范。

在**方法与产品方面**，RUP提供了使用UML进行可视化建模、基于用例驱动且以架构为中心的分析设计方法，并有相应的工具支持，如Rational Solutions，包括建模工具Rose、文档自动生成工具SoDA、测试工具Purify和Quality、配置管理工具ClearCase、变更管理工具ClearQuest等。RUP强调了开发方法的统一、建模语言的统一、研究成果的统一。

**影响软件实践的三个基本要素是：人、技术和过程。**

**RUP 主要强调的是过程，认为具有好的过程才能生产出稳定质量的软件。**

**新提出的敏捷（Agile）方法（第8章）强调的是人，认为人才是创造的源泉，只有人与人之间的充分交流（包括与用户的交流和开发者之间的交流）才是解决许多问题的关键。**

# 1.6软件工程方法

是软件生产的组织方式，包括软件过程指南、使用的标记法、软件描述和设计的规则等。

**1、结构化方法**

它强调结构的合理性，基于模块化的思想，用抽象模型的概念，按照软件内部数据传递、变换等关系，自顶向下逐层分解，直到找到满足功能需要的可实现的软件元素为止。

# 1.7工具

**【**例】假设你被任命为一家软件公司的项目负责人，你的工作是管理该公司已被广泛应用的字处理软件的新版本开发。由于市场竞争激烈，公司规定了严格的完成期限并且已对外公布。你打算采用哪种软件生命周期模型？为什么？

**解**： 对这个项目的一个重要要求是，严格按照已对外公布了的日期完成产品开发工作，因此，选择生命周期模型时、应该着重考虑哪种模型有助于加快产品开发的进度。使用增量模型开发软件时可以并行完成开发工作，因此能够加快开发进度。

这个项目是开发该公司已被广泛应用的字处理软件的新版本，从上述事实至少可以得出3点结论：第一，旧版本相当于一个原型，通过收集用户对旧版本的反映，较容易确定对新版本的需求，没必要再专门建立一个原型系统来分析用户的需求；第二，该公司的软件工程师对字处理软件很熟悉，有开发字处理软件的丰富经验，具有采用增量模型开发新版字处理软件所需要的技术水平；第三，该软件受到广大用户的喜爱，今后很可能还要开发更新的版本，因此，应该把该软件的体系结构设计成开放式的，以利于今后的改进和扩充。

# 第二章 需求工程

# 2.1 领域分析

**1、领域分析的概念**

软件工程要处理两类工程：

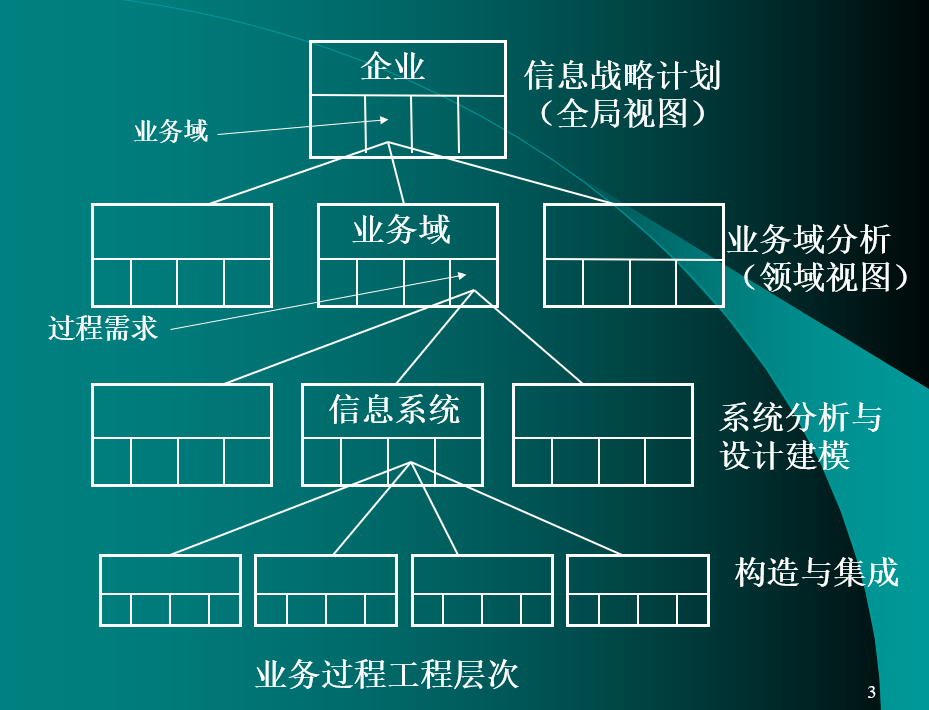
（1）面向用户的业务过程工程

（2）面向市场的产品工程 （见下图）

**领域(domain)**，就是指解决问题的范围，从最高层的角度（业务域）描述系统。

系统分析可以发生在许多不同的抽象层次：在业务或企业级层次，可定义描述模拟整个业务的功能、结构和行为的模型；

在应用层次，建模着重于特定的用户需求。





**2、领域分析过程的活动**

（1）定义被调查的领域中感兴趣的项

从业务域、系统类型或产品范畴中分离出感兴趣的“项”。

感兴趣的项包括：现存的应用软件的规约、支持软件（如GUI或数据库访问构件）以及和领域相关的构件库以及测试案例。

（2）对从领域中抽取出来的项进行分类并建立分类层次。

（3）收集领域中应用系统的代表性样本。

（4）分析样本中的每个应用

• 标识候选的每个可复用对象。

• 指明对象被标识为可复用的理由。

• 定义对象的适应性。

• 估算在领域中复用这些对象的应用的百分率。

• 使用配置管理技术控制这些对象。

（5）为对象开发分析模型。

**3、领域分析的价值**

**• 快速开发。**有助于集中精力关注最重要的问题，更有效地与相关人员进行交流，可以更快的确定需求。

**• 优化系统。**了解领域的细节有助于保证所采纳的解决方案更有效地解决用户的问题。会少犯错误，知道应该遵循那些规程和标准。领域分析给出一个应用领域的总体视图，会引导出更好的抽象从而改进设计。

**• 有了领域知识，就可以洞察新兴趋势及进一步开发的机会，有助于创建适应性更强的系统。**

**• 了解通用性和特殊性，有助于创建出具有更好的可重用性和更宽的销售市场的软件。**

# 2.2 需求获取

**1、需求（requirements）的概念**

**（1）需求的定义**

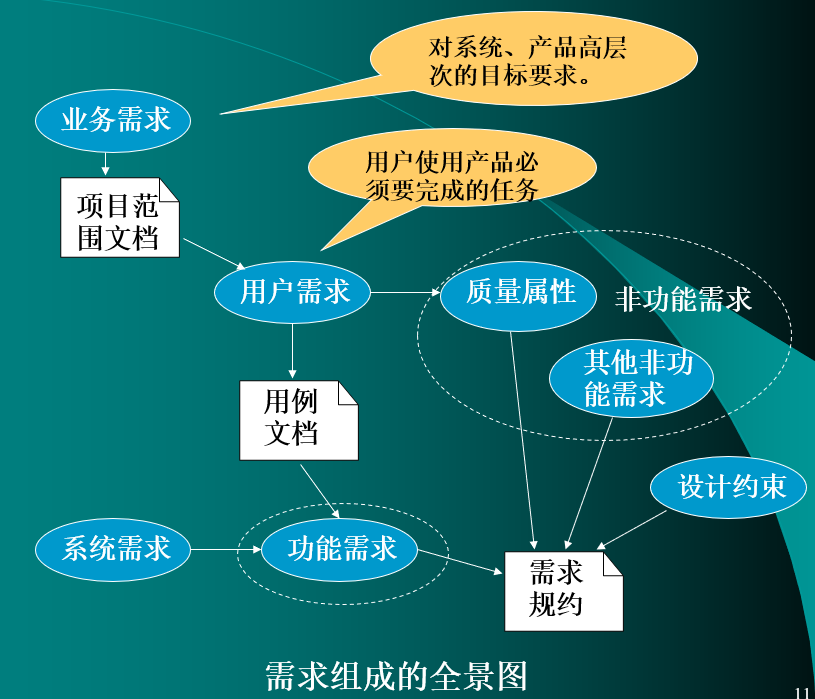
**• 需求是一段…描述：**意思是每个需求是相对短小简明的一段信息，表现为一个事实。它可以是一段话或用各种图表示。一组需求的集合成为需求文档。

**•…关于系统将要完成什么工作…：**需求描述了系统应当完成的任务，不描述系统将如何实现。

**•…必须经过所有相关人员的认可…：**意指需求必须经过评审，才能成为正式的需求。

**•…其目的是彻底地解决用户的问题。有助于解决用户的问题，该需求才有存在的价值。**

**（2）需求的组成**



**业务需求：**反映组织机构和客户对系统、产品高层次的目标要求。

**用户需求：**从用户使用的角度给出需求的描述。

**系统需求：**从系统或技术的角度描述为实现业务需求和用户需求而提供的服务以及所受到的约束。

**• 功能性需求：**描述系统应该做什么，即为用户和其它系统完成的功能。

**• 非功能性需求：**产品必须具备的属性或品质。

**• 设计约束：**设计与实现必须遵循的标准、约束条件。如运行平台、协议、选择的技术、编程语言和工具等。

**（3）需求的描述**

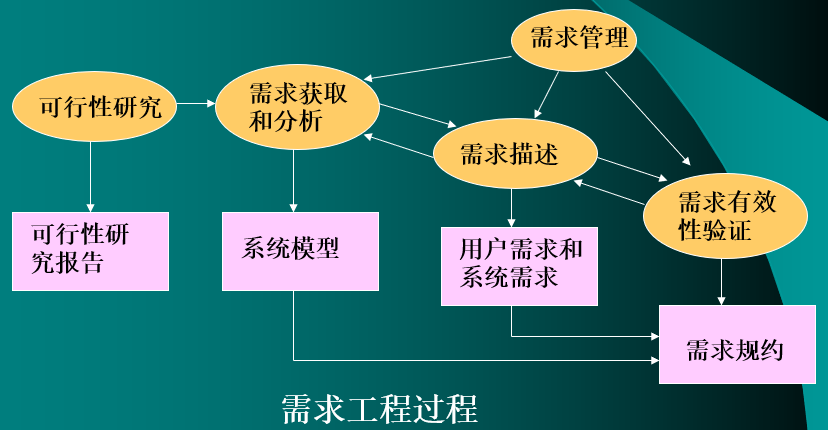
• 自然语言、结构化语言、PDL

• 图形化表示（使用Rational Rose、Microsoft Visio、Enterprise Architect等工具）

• 数学描述（形式化语言描述）

**2、需求工程过程**

需求工程是一个包括创建和维持系统需求文档所必需的一切活动的过程。它包含了如下活动：系统可行性研究、需求获取和分析、需求描述和文档编写、需求有效性验证、需求管理（管理需求工程的变更）**。**



**可行性研究：**

（1）系统是否符合组织机构的总体目标。系统不支持这些目标，就没有价值。

（2）系统是否可能在现有的技术条件、预算和时间限制内完成。

（3）系统能否与已经存在的系统集成。

**需求获取与需求分析的区别**

• 需求获取是开发人员与客户或用户一起对应用领域进行调查研究，收集系统需求的过程。

• 需求分析是将获取到的需求准确的理解、求精，并将其转化为完整的需求定义（包括建模），进而生成需求规约的过程。

**需求获取技术**

**获取的需求的表达方式：**

1）需求列表 需求与系统的特殊视角或环境的关系

2）业务流程图（状态/活动图）

3）用例（Use-Case）/场景（Scenario） /用户故事（user story)/ 视点(viewpoint)的描述

4）数据流图

5）实体关系图

6）用例图、类图、活动图、时序图、状态图等。

**视点 可以理解为：**

1、数据源或数据池（SA、SD方法）

2、一个表示框架（系统模型）

3、服务的接收者（系统的外部成分）

**用例**

“用例帮助定义系统之外存在什么（参与者）以及系统应该完成什么”。把系统分成一组逻辑的、互相联系较少的部分，每一部分都描述了系统与外部角色交互所提供的服务，即用例的集合代表了所有将会在系统需求中出现的交互。因此容易从使用的角度理解系统应达到的功能。而“用户故事”是某种形式的简化版用例。

场景

**场景是用例的具体描述。**一个用例封装了一组场景，每个场景就是一个单个线程。

# 2.3 需求分析与建模

**需求分析活动具有以下任务：**

1）分析需求的可行性：允许的成本、性能；与其他需求的冲突；外界因素的依赖和技术障碍等。

2）对于渐增式开发要确定需求的优先级别，以便确立产品版本。

3）建模：模型能突出或强调某些关键的系统特征。使用文本 和图表形式的组合，以相对容易理解和能直接评审正确性、完整性和一致性的方式来描述数据（信息）、功能和行为的需求。图形化的表示分析模型可以增强对软件需求的理解，也为软件设计奠定了基础。

4）生成需求规格说明。

**具体的建模方法有:**

上下文模型（ ERD、包图）

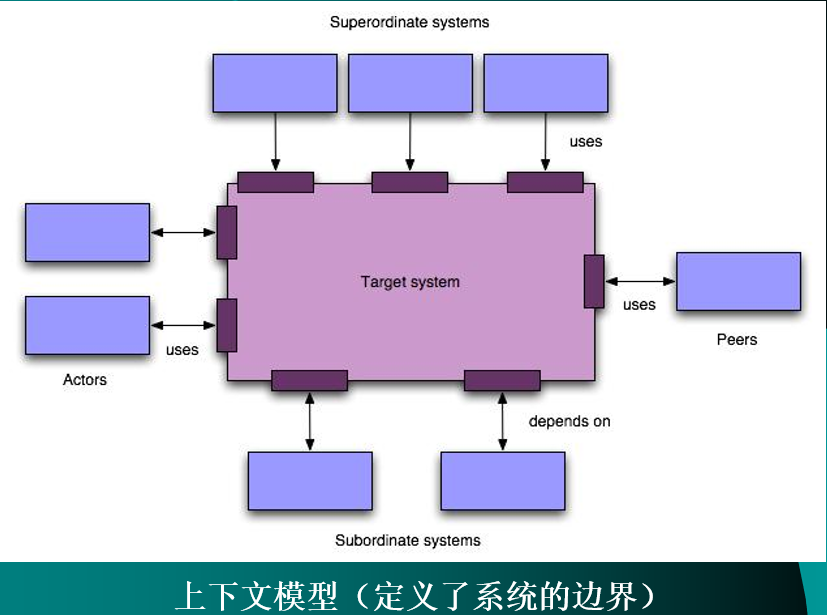
面向流的建模：数据流图（DFD/CFD）

数据建模：实体关系图（ERD）

基于场景的建模：用例图、顺序图、活动图

基于类的建模：类图、包

基于行为的建模： Petri网、状态图、顺序图、协作图、活动图



**1、数据处理模型** 如数据流图，说明系统的不同阶段数据如何被处理。

**2、组成模型** 实体关系图，说明一个实体与另外实体间的关系或如何由其他实体组成。

**3、体系结构模型** 说明构成整个系统的那些主要的子系统。

**4、分类模型** 如对象模型，说明对象间怎样具有共同特性。

**5、激励-响应模型** 说明系统对来自内部和外部的事件的响应。

# 2.4 需求规约与验证

**1、软件需求规约**

**SRS是需求分析阶段的产品，是所有其他开发和管理活动的基础。对系统开发过程中其他活动的影响：**

• 项目经理根据它制定或修改开发计划。

• 设计人员根据它进行系统设计。

• 测试人员根据它编写测试计划，设计测试用例。

• 产品发布人员根据它编写产品介绍和用户文档。

• 培训人员根据它编写培训教程。

**2、需求验证**

**需求验证的重要性：**如果在后续的开发或当系统投入使用时才发现需求规约中的错误，就会导致更大代价的返工。由需求问题而对系统做变更的成本比修改设计或代码错误的成本要大的多。

**对需求规约需执行以下类型的检查：**

**（1）有效性检查** 检查不同用户使用不同功能的有效性。

**（2）一致性检查** 在文档中，需求不应该冲突。

**（3）完备性检查** 需求文档应该包括所有用户想要的功能和约束。

**（4）现实性检查** 检查保证能利用现有技术实现需求。

**验证技术：**

（1）需求评审：

由分析员、设计员、测试员、用户参与的正式或非正式的会议评审。正式会议要有严格的评审程序，要有会议记录，开发组根据缺陷建议修改需求说明并重审。

（2）利用原型检验系统是否符合用户的真正需要。

（3）对每个需求编写概念性的测试用例。

（4）编写用户手册。用浅显易懂的语言描述用户可见的功能。

（5）自动的一致性分析。可用CASE工具检验需求模型的一致性。

# 2.5 需求管理

需求管理是对系统需求控制的过程。完成需求规约并不代表需求工程过程的结束，不可避免的遇到需求的变更。

**需要维护的3类可跟踪性信息是：**

**• 源可追踪性信息** 连接到提出该需求的相关人员或基本原理。

**• 需求可追踪性信息** 连接文档中彼此依赖的需求，用来评估一个变更会对多少需求产生影响以及引发的需求变更的范围和程度。

**• 设计可追踪性信息** 连接需求到其实现的设计模块。用来评估变更对设计与实现的影响。

**一个变更管理过程有3个基本阶段：**

**1）问题分析和变更描述**  对识别出的需求问题或变更提议进行分析并检查它的有效性，以产生更明确的变更提议。

**2）变更分析和成本计算**  使用可追踪信息和系统需求的一般知识评估被提议的变更所产生的影响。变更的成本计算不仅估计对需求文档的修改，在适当的时候还要估计设计和实现的成本，并产生对此变更是否执行的决策。

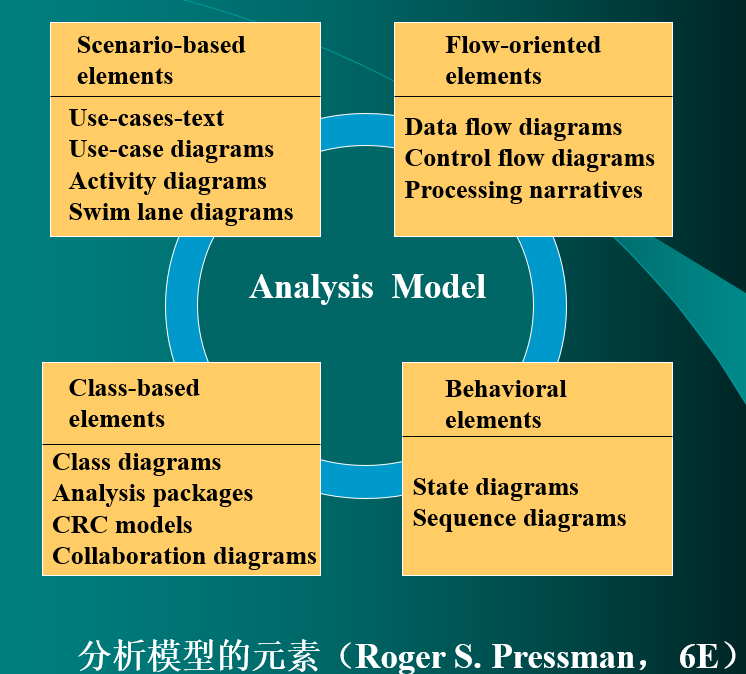
**3）变更实现** 需求规约以及系统设计和实现在必要时都要作修改。建立一个好的需求文档的格式，使得变更不会带来大量文字的修改。

# 第三章 构建分析模型

# 3.1 分析建模的目标与原则

分析建模的目标是创建各种表现形式，以描述软件信息、功能和行为的需求。

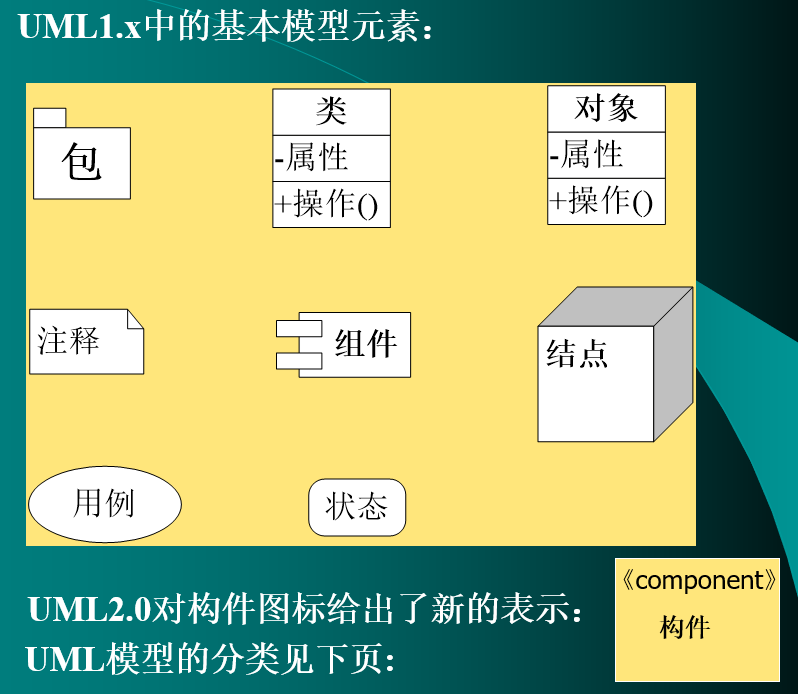
* **结构化分析**把软件看作信息转换器，辅助开发人员识别数据、数据 之间的关联以及这些数据流经软件处理功能时转换的方式。
* **面向对象分析**检查定义为一组用例的问题域，提取该域的类，每个类都有自己的属性和操作，类和类之间有多种关联方式，并使用UML建模。



# 3.2 统一建模语言UML

**1、UML概述**

UML是一个通用的标准建模语言，允许任何一种开发过程和面向对象方法使用它，对任何具有静态结构和动态行为的系统进行可视化建模。UML不是一个开发过程，也不是一种方法，它没有提及如何生成模型。



**• 用户模型视图**

**用例图：** 从用户的角度描述系统能提供哪些功能以及功能的使用者。

**• 结构模型视图：**

**类图：描述系统的静态结构；**

**对象图：描述系统在某个时刻的静态结构；**

**包图：将类分组成更高层次的静态结构。**

**• 行为模型视图**

**顺序图：按时间顺序描述系统元素之间的交互；**

**协作图：着重描述系统元素之间的协作；**

**状态图：描述系统元素对事件的响应引起的状态转换；**

**活动图：描述系统元素的活动。**

**• 实施模型视图**

**构件图：描述实现系统的元素（构件）的组织结构；**

**配置图：描述环境元素的配置，并把实现系统的元素映射到配置上。**

**1、用例图**

用例规定了一个动作序列，系统执行这些动作并产生出对于特定参与者可见的有价值的结果。参与者（Actor） 可以是人、组织、硬件设备或外部系统。

**用例有以下特点：**

• 捕获某些用户可见的需求，实现一个具体的用户目标。

• 由参与者激活，并返回确切的值给参与者。

• 有大小，但必须是对一个具体的用户目标实现的完整描述。

用例图用于展示参与者与系统提供的用例之间的连接（通信）关系。仅仅从参与者使用系统的角度描述系统，不反映内部的处理方式。因此用例图定义的是系统可见的功能需求。

* **用例之间可以相互关联：**

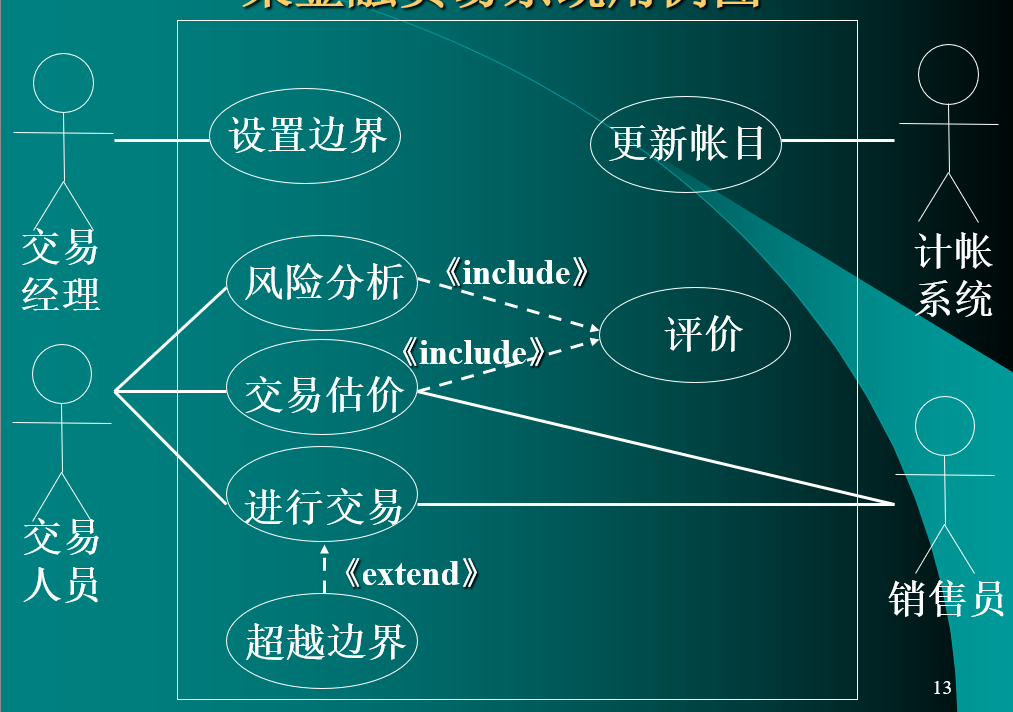
**包含（inclusion）：一个用例复用另一个用例中的步骤。**

**扩展（extension）：通过对已有用例增加步骤创建另一个新用例。**

**泛化（generalization)：一个用例可以继承另一个用例的行为和含义。**

**分组（grouping）：一组用例的简单组织方式（如相关的用例放在一个包中）。**

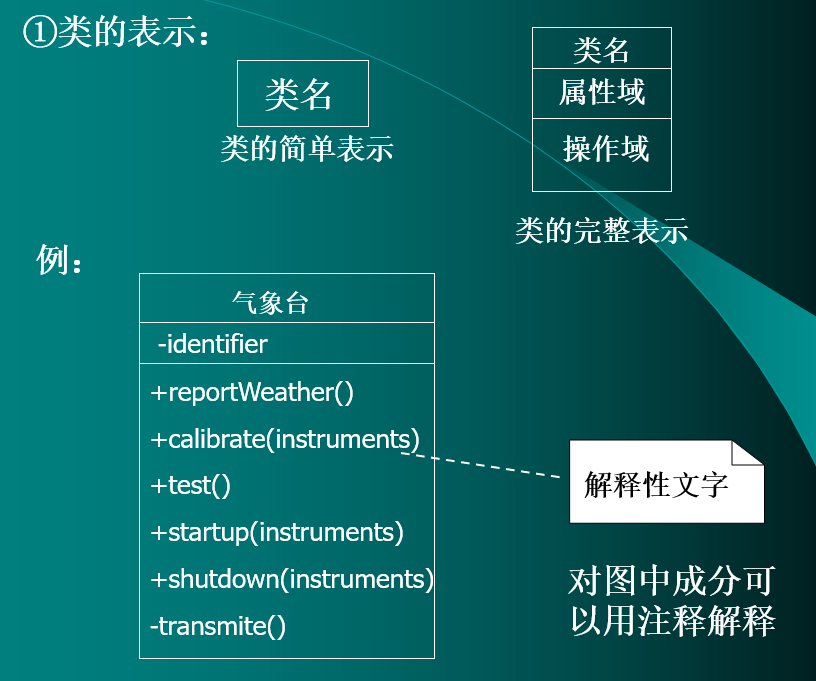
**用例之间的关联用“类图”中表示依赖关系的虚线箭头表示。看下图例：**

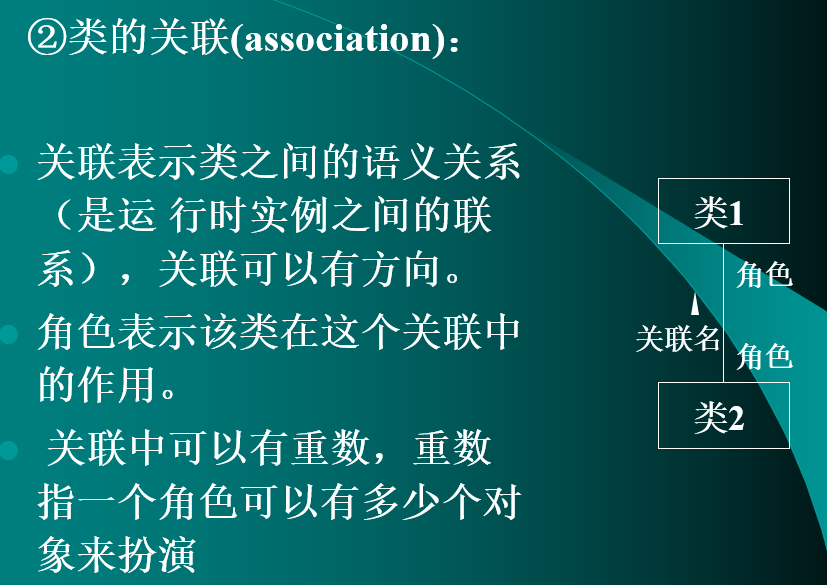


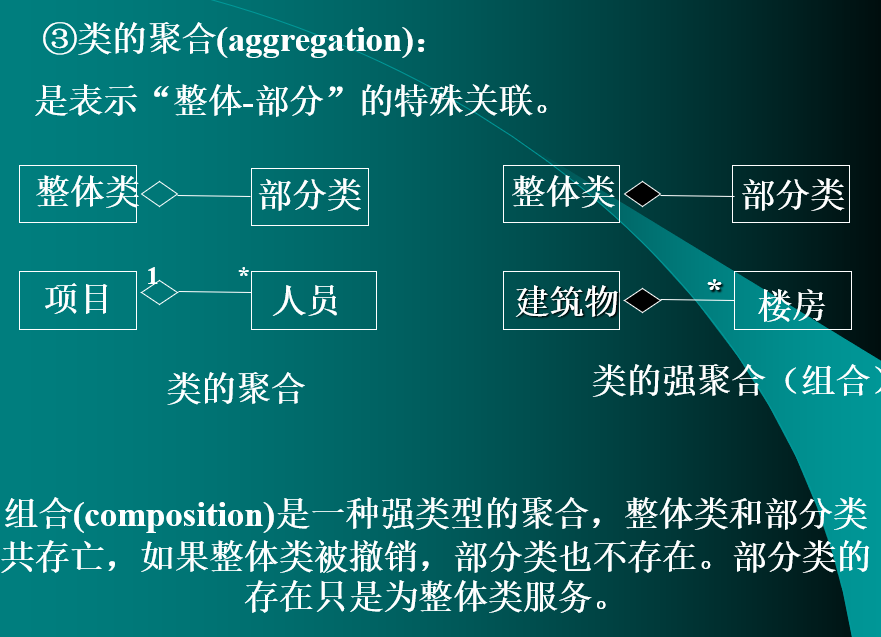
* **双尖括号括起来的元素**为**构造型（stereotype)元素。**构造型使设计者能够在现有的UML元素的基础上创建新的元素，被括起来的名称称为关键字。
* **当在两个或多个用例中出现重复描述（有共用行为）时，可采用《include》避免重复。**
* **当描述一般行为时有例外、任选或异常处理时，采用《extend》。**扩展关系就像是从反向来看的包含关系。

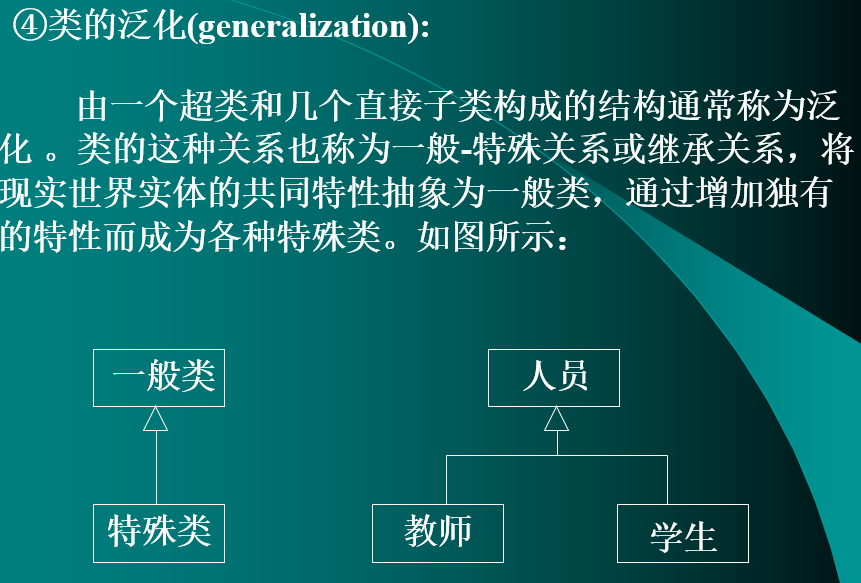
**2、类图**

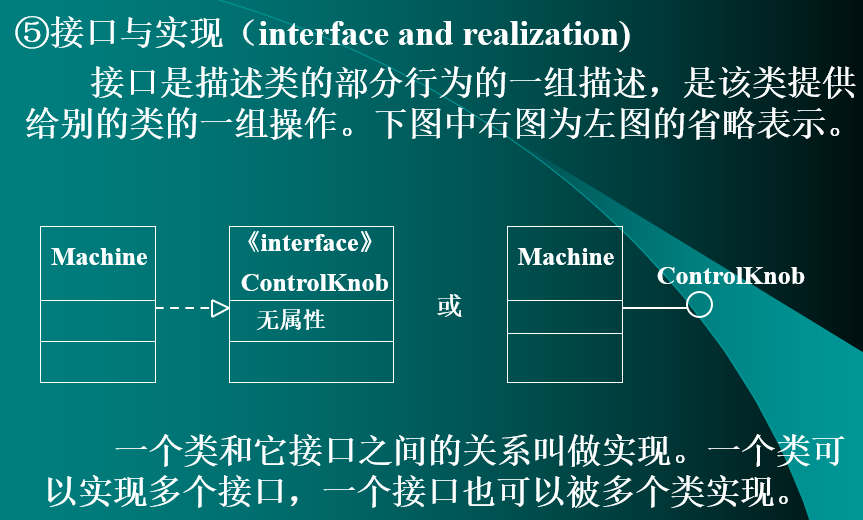
用来描述系统中类和类之间的关系，是系统的静态结构。类图中还描述了类的属性、操作、以及对模型中各种成分的约束。类图中包含有类、关联和关系等模型元素。类图是定义其他图的基础**。**

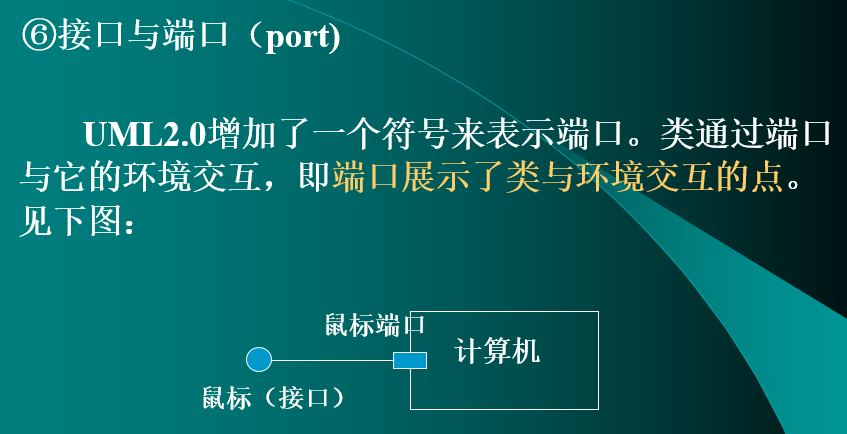






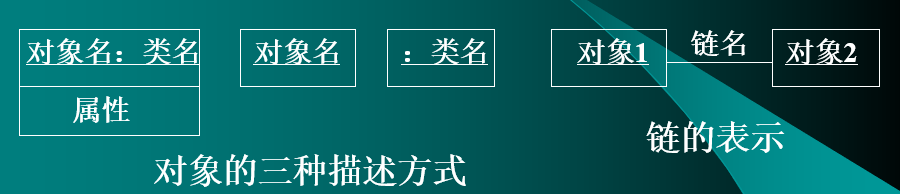






**3、对象图**

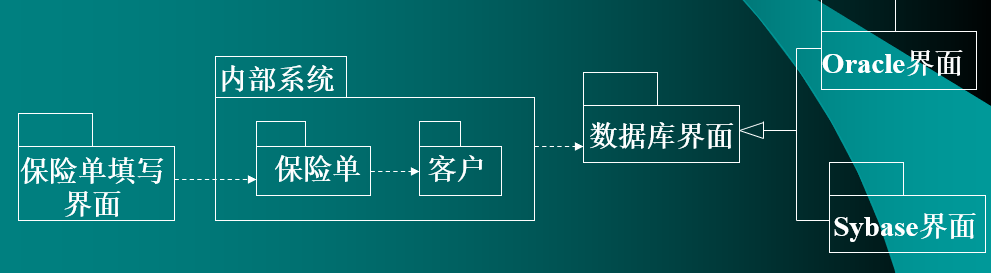
对象图是类的实例图，给出了存在于程序执行过程中某个特定点的对象与链接的配置示例。



链是类图中关联的实例，类图中关联表示两个类之间可能存在的所有链接，而对象图中的链上不附加重数。对象图中也不可能包含泛化。对象图不像类图那样具有重要地位，通常用来实例化一个复杂的类图以分析特定的场景。对象图也常用在协作图中，反映一组对象之间的动态协作关系。

**4、包**

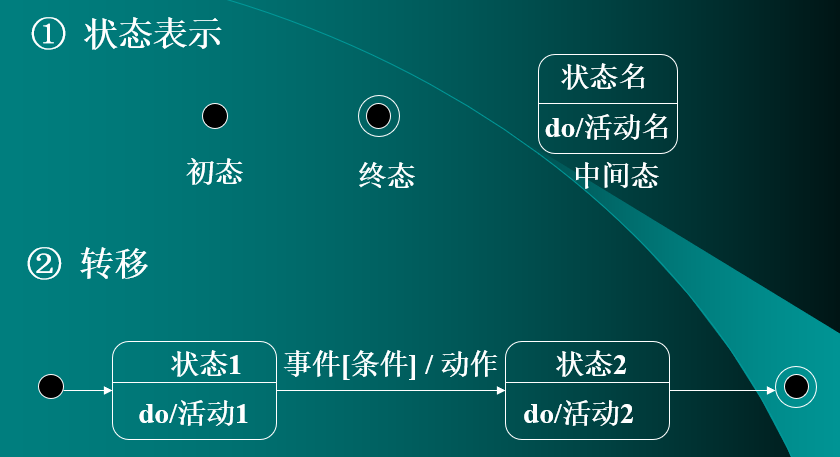
包是将类分组成更高层次的单位。包图用以描述包中存在的类以及包与包之间的依赖关系和泛化关系。包的内容可以是类的列表、类图、或另一个包图。依赖关系用虚线箭头表示（如对一个元素的修改可能会引起对另一个元素的修改）。



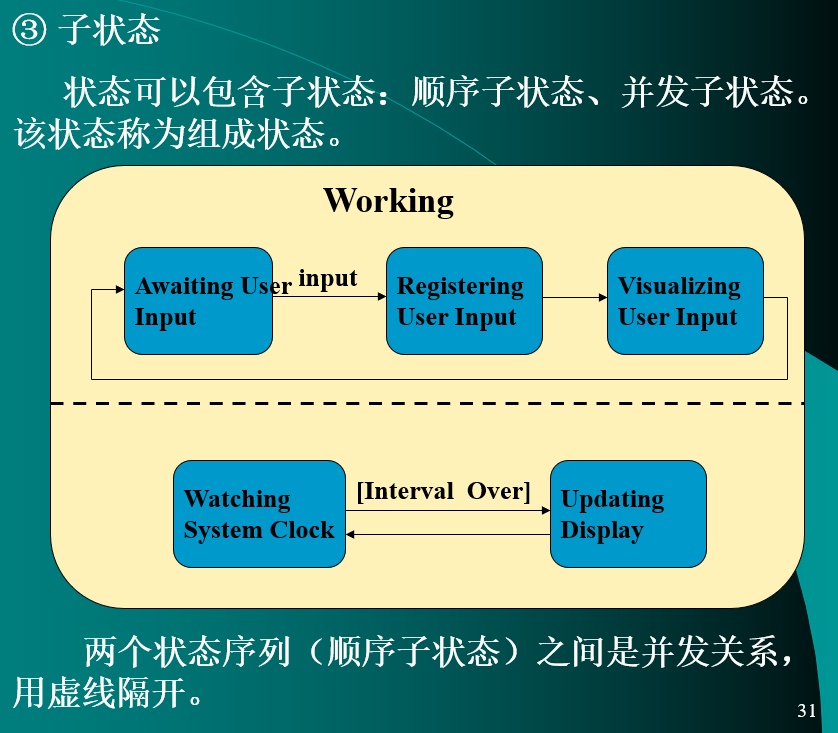
**5、状态图**

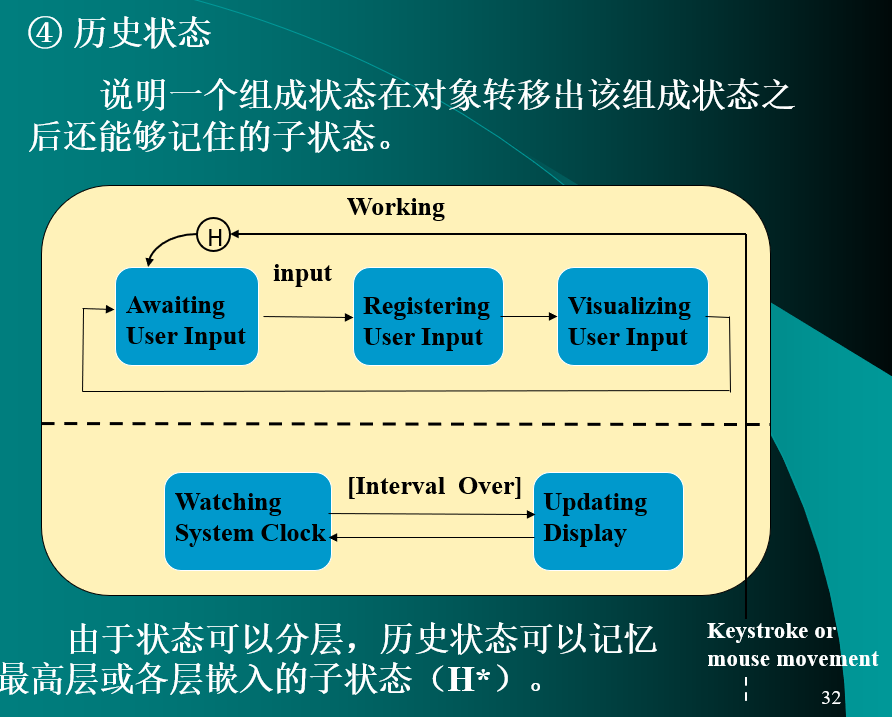
状态是对象进行一些活动的结果。当某个事件发生后，对象的状态会发生变化。状态图描述了类内对象在生命周期内所有可能的状态以及事件发生时状态的转移(transition)情况（类内对象的公共行为）。一个转移可以有一个与之相连的动作(action)，这个动作指明了对象状态转移时应该作些什么。

状态图是类图的补充。 不需要为所有的类画状态图，仅为那些有多个状态其行为受外界环境的影响并且发生改变（有重要的时序行为）的类画状态图。



“动作”是为响应事件而执行的行为。do活动指在某个状态内发生的持续一段时间的活动，执行中可被外部的某事件中断。





**6、时序图（顺序图）**

用来描述对象之间的动态协作关系以及协作过程中的先后次序。它常用来描述一个用例的行为，显示该用例中所涉及到的对象以及对象之间交互的时间顺序。

每个对象图符下面的垂直虚线表示对象的生命线，生命线上的细长矩形称为激活（activation)，表示该对象正在执行某个操作，矩形的长度表示执行操作的持续时间。



