第一章基本概念

重点是软件的特性、软件工程学的研究范畴，以及学习软件工程的意义。掌握软件的概念、特 点和软件工程的基本特征；理解为什么学习软件工程、如何学习。主要知识点：

1) 软件的概念和特点

软件概念定义：软件=程序+数据+文档

程序：按事先设计的功能和性能需求执行的指令序列

数据：是程序能正常操纵信息的数据结构

文档：与程序开发、维护和使用有关的图文材料

软件的特征/特点

软件是设计开发的或者是工程化的，并不是制造的　软件开发时间和工作量难以估计

软件会多次修改　　软件的开发进度几乎没有客观衡量标准

软件测试非常困难　软件不会磨损和老化　软件维护易产生新的问题

按软件的功能进行划分：

1、系统软件：服务于其他程序的程序。 2、应用软件：解决特定需要的独立应用程序。

2、工程/科学软件：带数值计算的特征。4、嵌入式软件。

5、产品线软件：为不同用户使用提供特定功能。 6、web应用软件。

7、人工智能软件：利用非数值计算解决复杂问题。

按软件规模进行划分：

类别　　参加人员数　　研制期限　源程序行数

微型　　小型　中型　大型　甚大型 极大型

按软件服务对象的范围划分

项目软件　产品软件

2) 软件危机的概念和产生的原因

软件危机的概念

在计算机软件的开发和维护过程中所遇到的一系列严重问题。（效率和质量问题）

项目超出预算　　项目超过计划完成时间　　软件运行效率很低　软件质量差

软件通常不符合要求 项目难以管理并且代码难以维护　　软件不能交付

产生软件危机的原因

客观：软件本身特点：逻辑部件 规模庞大

主观：不正确的开发方法：忽视需求分析，错误认为：软件开发=程序编写，轻视软件维护

消除软件危机的途径

对计算机软件有一个正确的认识：(软件≠程序)

必须认识到软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。

推广使用在实践中总结出来的开发软件的成功技术。开发和使用更好的软件工具。

3) 软件工程的定义、三要素和发展过程

软件工程定义为：

（1）应用系统化的、规范的、可量化的方法，来开发、运行和维护软件，即将工程化方法应用到软件。

（2）对（1）中各种方法的研究。

软件工程的目标是在给定的时间和预算内，按照用户的需求，开发易修改、高效、可靠、可维护、适应力强、可移动、可重用的软件。

软件工程三要素：方法、工具、过程

方法：软件工程方法是构建软件工程的解决方法。软件工程方法分两类：结构化方法和面向对象方法。

工具：为软件工程的过程和方法提供自动化或半自动化的工具支持。

过程：过程贯穿软件开发的各个环节，在各环节之间建立里程碑；

软件工程的7个原则

1、使用阶段性生命周期计划的管理　　2、进行连续的验证3、保证严格的产品控制

4、使用现代编程工具/工程实践　　5、保持清晰的责任分配　　6、用更好更少的人

7、保持过程改进

第二章过程模型

重点是各种实用的软件过程模型，以及不同过程模型的特点比较。掌握几种典型模型的优缺点和能依据项目特征选择使用不同的模型；理解为什么有不同的模型、不同模型的特征。主要知识点：

1）软件生命周期概念、软件过程概念、能力成熟度模型 CMM 概念

软件生命周期定义：软件产品或软件系统从设计、投入使用到被淘汰的全过程。

什么是软件过程：软件生产的一系列活动，这些活动贯穿于软件开发的整个过程。

软件过程都具有以下共同活动：

沟通：该活动包括软件设计者与客户沟通，。

计划：软件开发小组讨论使用何种方法及何种工具来实现客户需求。

建模：论选择何种模型来满足需求。不同的需求需要不同的模型。

构造：编码和测试。

部署：软件交付给客户。客户给出建议和反馈，软件实施小组改进软件。

成熟度模型标准（CMM）

1初始级，工作无序，缺乏健全的管理制度。

2可重复级，管理制度化，建立了基本的管理制度和规程，初步实现标准化。

3已定义级，过程标准化，工作和管理工作，均已实现标准化、文档化。

4量化管理级，产品和过程已建立了定量的质量目标。开发活动中的生产率和质量是可量度的。

5优化级，持续的过程改进，拥有防止出现缺陷、识别薄弱环节以及加以改进的手段。

2) 常见的软件过程模型：瀑布、增量、原型、螺旋、喷泉等，比较各自优缺点

什么是软件过程模型

软件过程模型是软件开发全部过程、活动和任务的结构框架。它能直观表达软件开发全过程，明确规定要完成的主要活动、任务和开发策略。

瀑布模型(Waterfall Model)

·软件开发过程与软件生命周期是一致的，规定了各项软件工程活动，以及它们自上而下，相互衔接的固定次序，如同瀑布流水，逐级下落。以文档为驱动的模型。

传统瀑布模型开发软件的特点

·阶段间具有顺序性和依赖性。

·推迟实现的观点。

·每个阶段必须完成规定的文档;每个阶段结束前完成文档审查,及早改正错误。

瀑布模型缺点

·线性过程太理想化

·各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量；

·由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险；

·早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。

增量过程模型(Incremental Model)

增量模型是一种非整体开发的模型。是一种进化式的开发过程。它允许从部分需求定义出发，先建立一个不完整的系统，通过测试运行这个系统取得经验和反馈，进一步使系统扩充和完善。如此反复进行，直至软件人员和用户对所设计的软件系统满意为止。

增量模型特点

·小而可用的软件

·在前面增量的基础上开发后面的增量

·每个增量的开发可用瀑布或快速原型模型

·迭代的思路

增量模型的优缺点

优点:

•不需要提供完整的需求。只要有一个增量包出现，开发就可以进行。

·在项目的初始阶段不需要投入太多的人力资源。

·增量可以有效地管理技术风险。

缺点:

每个增量必须提供一些系统功能，这使得开发者很难根据客户需求给出大小适合的增量。

快速应用开发模型（RAD）

·快速应用开发模型（RAD）是一个增量过程模型，强调短暂的开发周期。

·RAD模型是瀑布模型的"高速"变体，通过基于组件的构建方法实现快速开发。

RAD模型的缺点

1）对大型项目而言，RAD需要足够的人力资源。

2）开发者和客户都要实现承诺，否则将导致失败。

3）并非所有系统都适合（不能合理模块化的系统、高性能需求并且要调整构件接口的、技术风险很高的系统均不适合）。

演化模型

演化模型的思想是首先实现软件的最核心的、最重要的功能

原型模型

适用情况，客户定义一个总体目标集，但不清楚系统的具体输入输出；或开发者不确定算法的效率、软件与操作系统是否兼容以及客户与计算机交互的方式。此时，原型法是很好的选择。

缺点：1). 设计者在质量和原型间有所折中

2). 客户意识不到一些质量问题

螺旋模型（Spiral Model）

·螺旋模型结合了瀑布模型和原型模型的特点。

·螺旋模型强调风险管理，因此该模型适用于大型系统的开发。

螺旋模型沿着螺线旋转，在笛卡尔坐标的四个象限上分别表达了四个方面的活动:

·**制定计划**。确定软件目标，选定实施方案，弄清项目开发的限制条件。

·**风险分析**。分析所选方案，考虑如何识别和消除风险。

·**实施工程**。实施软件开发。

·**客户评估**。评价开发工作，提出修正建议。

螺旋模型的优点

·支持用户需求的动态变化。

·原型可看作形式的可执行的需求规格说明，易于为用户和开发人员共同理解，还可作为继续开发的基础，并为用户参与所有关键决策提供了方便。

·强调原型的可扩充性和可修改性，原型的进化贯穿整个软件生存周期，有助于目标软件的适应能力。

·螺旋模型为项目管理人员及时调整管理决策提供了方便，进而可降低开发风险。

螺旋模型的缺点和适应场合

缺点

·如果每次迭代的效率不高，致使迭代次数过多，将会增加成本并推迟提交时间；

·需要丰富的风险评估经验和专门知识，要求开发队伍水平较高。

适应场合

支持需求不明确、特别是大型软件系统的开发，并支持面向规格说明、面向过程、面向对象等多种软件开发方法，是一种具有广阔前景的模型。

喷泉模型

喷泉模型是一种以**用户需求**为动力，以**对象**为驱动的模型，主要用于描述**面向对象**的软件开发过程。

喷泉模型的优缺点

优点

·该模型的各个阶段没有明显的界限，开发人员可以同步进行开发。其优点是可以提高软件项目开发效率，节省开发时间，适应于面向对象的软件开发过程。

缺点

·由于喷泉模型在各个开发阶段是重叠的，在开发过程中需要大量的开发人员，因此**不利于项目的管理**。此外这种模型要求严格管理文档，使得**审核的难度加大**，尤其是面对可能随时加入各种信息、需求与资料的情况。

基于构件的模型四个阶段：

需求·与其它模型相同。

组件分析·根据需求规格搜索可满足该需求的组件。没有完全匹配的情况，则组件需要加以修改。

系统设计·该模型是基于重用的。设计者必须考虑到重用的概念，如果没有可重用的组件，还要设计新的软件。

开发和集成·组件集成到系统中。

基于构件的模型优缺点：

优点：组件的重用，降低了成本和风险，节约了时间

缺点：模型复杂，导致需求的折衷，导致系统不能完全符合需求；无法完全控制所开发系统的演化。

敏捷软件过程

·是基本原理和开发准则的结合。基本原理强调客户满意度和较早的软件增量交付；小但有激情的团队；非正式的方法；最小的软件工程产品；简化整体开发。开发准则强调分析和设计的交付，以及开发者和客户之间积极持续的交流。

·目前的敏捷过程模型主要包括极限编程（XP），自适应软件开发（ASD），动态系统开发方法（DSDM）等。

如何选择过程模型？参考原则

1. 在前期需求明确的情况下，尽量采用瀑布模型或改进的瀑布模型。
2. 在用户无系统使用经验，需求分析人员技能不足情况下一定要借助原型。
3. 在不确定因素很多，很多东西前面无法计划的情况下尽量采用增量迭代和螺旋模型。
4. 在需求不稳定的情况下尽量采用增量迭代模型。
5. 在资金和成本无法一次到位的情况下可采用增量模型，软件产品多个版本进行发布。
6. 对于完成多个独立功能开发可以在需求分析阶段就进行功能并行，但每个功能内部都应该遵循瀑布模型。
7. 对于全新系统的开发必须在总体设计完成后再开始增量或并行。
8. 对于编码人员经验较少的情况下建议不要采用敏捷或迭代等生命周期模型。
9. 增量、迭代和原型可以综合使用，但每一次增量或迭代都必须有明确的交付和出口原则。

第三章 需求分析

重点是需求分析的一般步骤、数据流图、用例图、活动图、需求规格说明文档的编制。掌握结　　构化分析模型的导出、数据流图/用例图/活动图的基本画法和需求规格说明文档的编制；理解需求分析的过程、主要步骤。主要知识点：

1）需求分析的概念

为什么需要需求分析?

·需求分析的错误和变更导致的软件开发失败，如缺少用户的输入，不完整的需求和规格说明书，需求和规格说明书的变更

·希望对开发进行指导

·希望开发人员对用户的要求理解

·希望用户理解开发人员

·测试部门有理可依

软件需求管理的过程

需求确认：需求获取-->需求提炼-->需求描述-->需求验证

需求变更：需求变更

需求分析的定义：确定系统必须具有的**功能和性能**，系统要求的**运行环境**，并且预测系统发展的前景。即需求就是以一种**清晰、简洁、一致且无二义性**的方式，对一个待开发系统中各个有意义方面的陈述的一个集合。

需求分析的任务和步骤

·需求分析的任务

建立分析模型：准确地定义未来系统的目标，确定为了满足用户的需求系统必须做什么。

编写需求说明：用《需求规格说明书》规范的形式准确地表达用户的需求。

·需求分析的步骤

需求获取 需求提炼 需求描述（撰写需求规格说明书）需求验证

2）需求分析的过程：需求确认与需求变更

需求变更管理

·变更管理是将个人、团队和组织从**现有状态转移/过渡到期望状态**的结构化方法。它授权雇员**接受并理解**当前业务环境中的变更。在项目管理中，变更管理是指项目变更被引入和接受后的项目管理过程。

·管理和控制需求基线的过程

·需求变更控制系统

一个正式的文档，说明如何控制需求变更

建立变更审批系统

3）需求确认的步骤：需求获取→需求提炼→需求描述→需求验证

第一步：需求获取

软件需求获取指的是软件需求的来源以及软件工程师收集这些软件需求的方法。

需求类型

（1）功能性需求：为用户和其它系统**完成的功能、提供的服务**。

（2）非功能需求：必须遵循的**标准**，外部**界面的细节**，实现的**约束条件**. **质量属性**等等。

非功能需求限定了选择解决问题方案的范围，如运行平台、实现技术、编程语言和工具等。

需求获取技术

－采访－设定情景（用例）－原型－会议－观察商业过程和工作流

需求获取面临的挑战

·客户说不清楚需求·需求易变性·问题的复杂性和对问题理解的不完备性与不一致性

需求诱导十原则

1、倾听；　2、有准备的沟通；　3、需要有人推动；　4、最好当面沟通；　5、记录所有决定；　6、保持通力协作；　7、聚焦并协调话题；　8、采用图形表示；9、继续前进原则（认可某件事情，继续前进；如果不认可某件事情，继续前进；如果某项特性或功能不清晰，当时无法澄清，继续前进）；10、谈判双赢原则。

第二步：需求提炼（需求分析）

对应用问题及环境的理解和分析，为问题涉及的信息、功能及系统行为建立模型。将用户需求精确化、完全化，最终形成需求规格说明书。

·需求分析的核心在于建立分析模型。

·需求分析采用多种形式描述需求，通过建立需求的多种视图，揭示出一些更深的问题。

·需求分析还包括与客户的交流以澄清某些易混淆的问题，并明确哪些需求更为重要，其目的是确保所有风险承担者尽早地对项目达成共识并对将来的产品有个相同而清晰的认识。

需求分析模型

分析建模：

·结构化分析模型（其基本思想是用系统工程的思想和工程化的方法，根据用户至上的原则，自始自终按照结构化、模块化，自顶向下地对系统进行分析与设计。）

·面向对象分析模型（由5个层次（主题层、对象类层、结构层、属性层和服务层）和5个活动（标识对象类、标识结构、定义主题、定义属性和定义服务）组成。）

需求建模图形工具：

	面向过程的需求分析	面向对象的需求分析
数据模型	实体-联系图（ERD） 数据字典（DD）	类图、类关系图
功能模型	数据流图（DFD）	用例图
行为模型	状态变迁图（STD）	活动图、时序图、状态图

第三步：需求规格说明书

软件需求规格说明书（SRS）------软件系统的需求规格说明，是对待开发系统的行为的完整描述。它包含了功能性需求和非功能性需求。

·需求分析工作完成的一个基本标志是形成了一份完整的、规范的需求规格说明书。

·需求规格说明书的编制是为了使用户和软件开发者双方对该软件的初始规定有一个共同的理解，使之成为整个开发工作的基础。

第四步：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文档中的错误，就会导致更大代价的返工。

对需求文档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，需求之间不应该冲突。

（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。

需求验证技术

（1）需求评审

(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。

4）需求分析三类建模：功能模型、数据模型、行为模型。面向过程和面向对象的需分析过程中，三类模型各包含哪些内容?

面向过程的分析方法

结构化分析方法

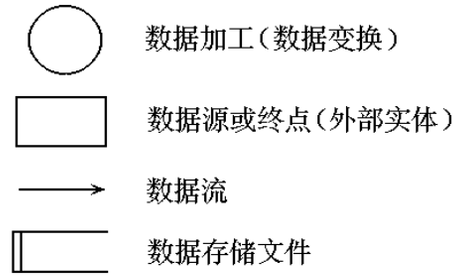
·面向**数据流**进行需求分析的方法

·结构化分析方法适合于**数据处理类型软件**的需求分析

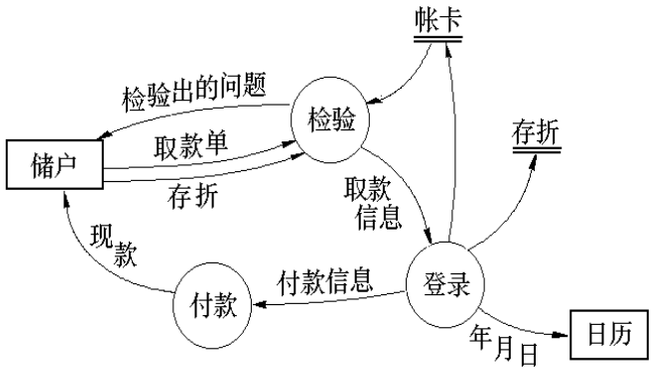
·结构化分析方法按照软件内部**数据传递、变换**的关系，**自顶向下**逐层分解，直到找到满足功能要求的所有可实现的软件为止

功能模型——数据流图

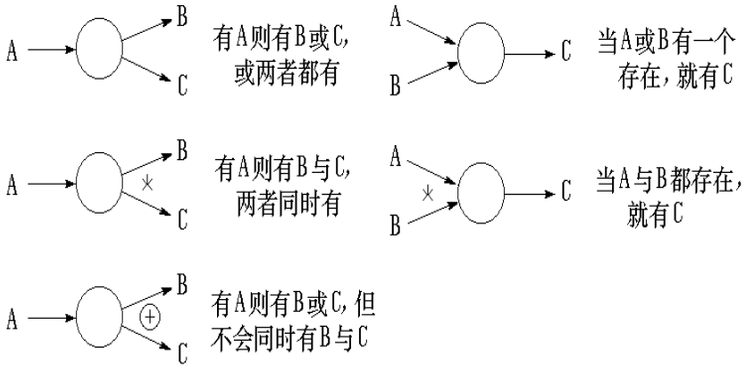
数据流图中的主要图形元素:



描述银行取款过程的数据流图

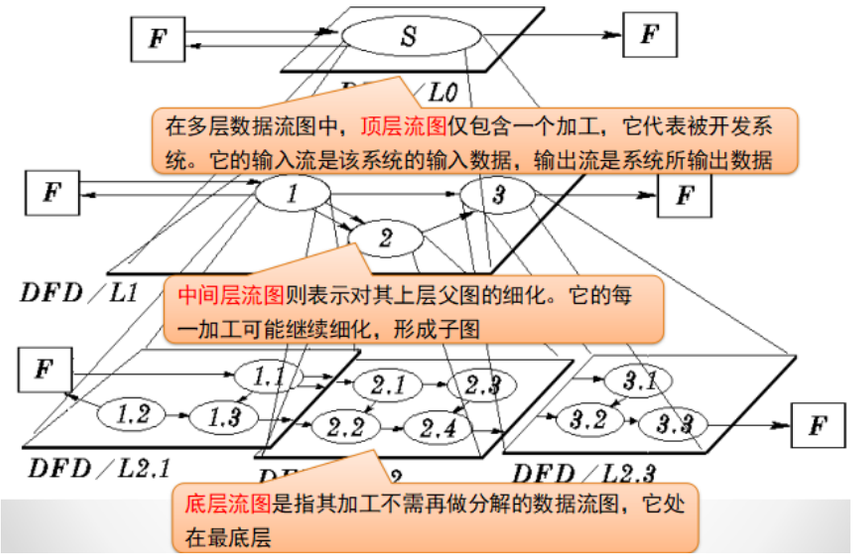


数据流与数据加工之间的关系



数据流图的层次结构

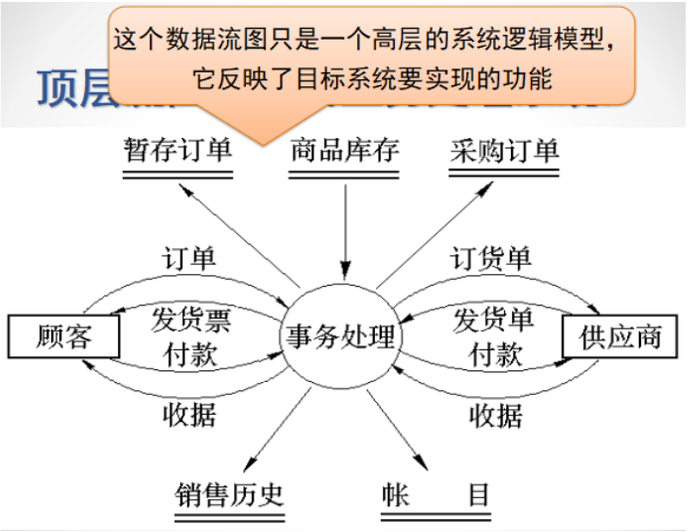
为了表达数据处理过程的数据加工情况，需要采用层次结构的数据流图。按照系统的层次结构进行**逐步分解**，并以分层的数据流图反映这种结构关系，能清楚地表达和容易理解整个系统



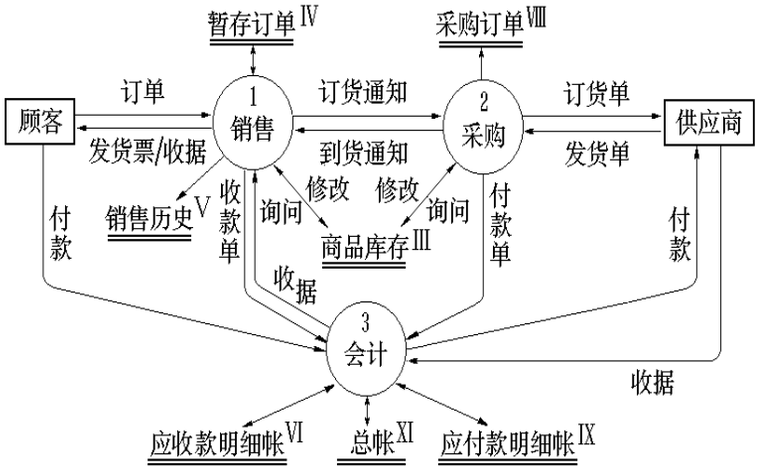
商店业务系统数据流图绘制步骤

·首先确定系统的输入和输出，画出顶层数据流图，以反映最主要业务处理流程

顶层数据流图

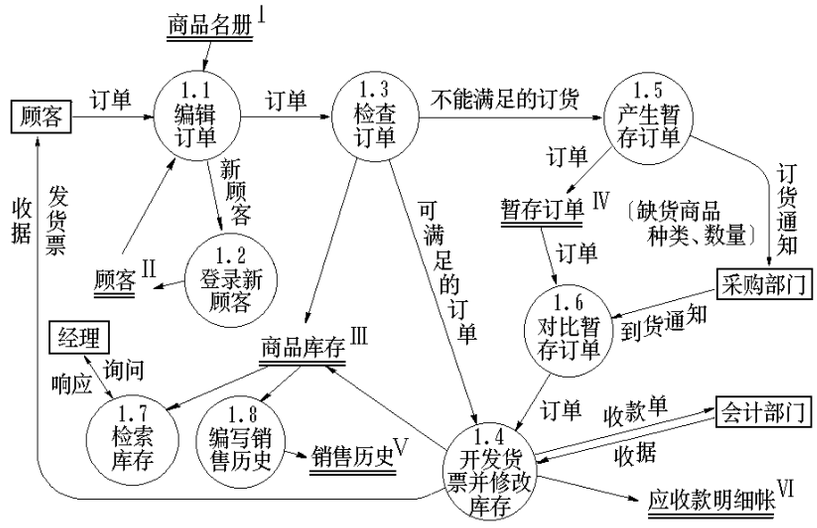


第一层数据流图



第二层：加细每个加工框

1.销售细化



数据模型——ER图

实体-联系图(ERD)

·ER图 ---- 是用来建立数据模型的工具。

·数据模型 ---- 是一种面向问题的数据模型，是按照用户的观点对数据建立的模型。

·数据模型中包含3种相互关联的信息：数据对象（**实体**）、数据对象的**属性**及数据对象**联系**。

(1) 数据对象

·数据对象：对软件必须理解的复合信息的抽象。复合信息：是指具有一系列不同性质或属性的事物。

(2) 属性

·属性定义了数据对象的性质。应该根据对所要解决的问题的理解，来确定特定数据对象的一组合适的属性。如：学生具有学号、姓名、性别、年龄、专业（其它略）等属性

(3) 联系

·数据对象彼此之间相互连接的方式称为联系，也称为关系。

a. 一对一联系(1：1)，b. 一对多联系(1：N)，c. 多对多联系(M：N)·联系也可能有属性。

(4) 实体-联系图的符号

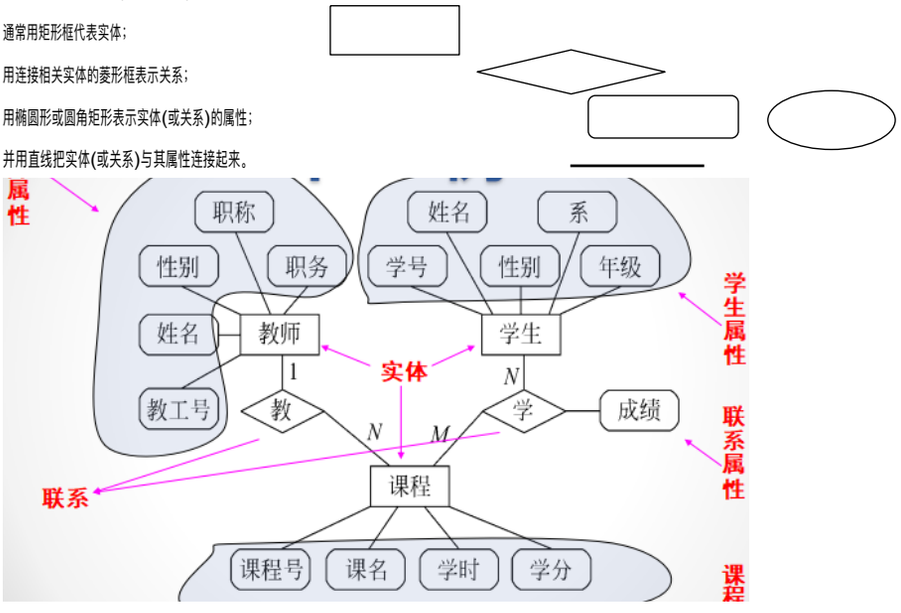
ER图中包含了实体(即数据对象)、关系和属性等3种基本成分。

通常用矩形框代表实体；

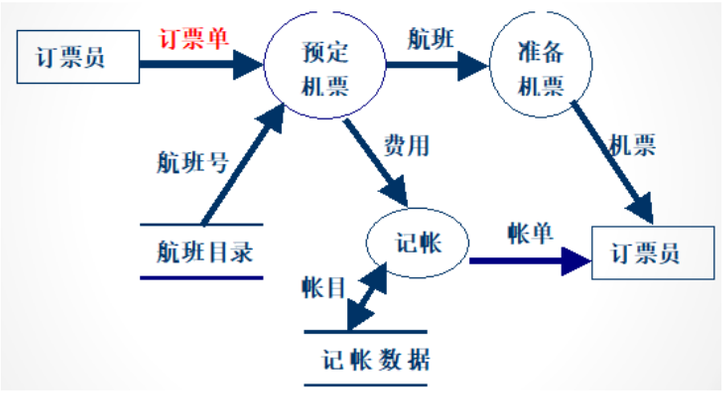
用连接相关实体的菱形框表示关系；

用椭圆形或圆角矩形表示实体(或关系)的属性；

并用直线把实体(或关系)与其属性连接起来。



数据字典



订票单

名字: 订票单

数据类型: 航班日期＋目的地＋出发地＋航班号

使用说明: 必须给出各个数据项

解释性说明: 无

缺省值: 出发地＝填写本地

作为输出流的转换列表: 无

作为输入流的转换列表: 预定机票

行为模型——状态变迁图

状态变迁图

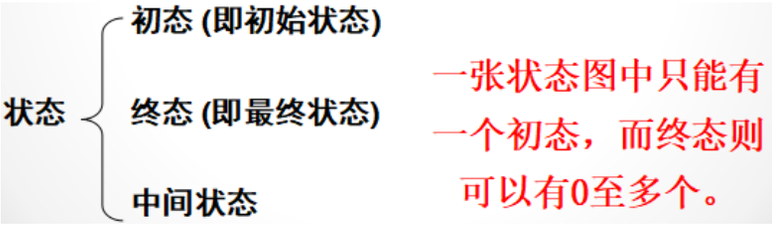
通过描绘**系统的状态**及引起系统状态转换的**事件**，来表示系统的行为。此外，状态图还指明了作为特定事件的结果系统将做哪些动作(例如，处理数据)。

1)状 态

·状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。

·状态规定了系统对事件的响应方式。

·系统对事件的响应，既可以是做一个(或一系列)动作，也可以是仅仅改变系统本身的状态，还可以是既改变状态又做动作。



2)事 件

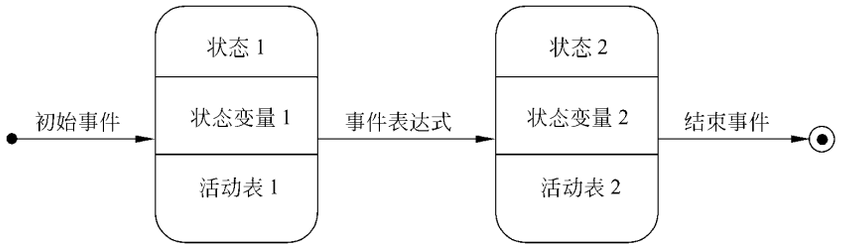
·事件是在某个特定时刻发生的事情，它是对引起系统做动作或(和)从一个状态转换到另一个状态的外界事件的抽象。

3)符号

·**初态**用实心圆表示，**终态**用一对同心圆(内圆为实心圆)表示。

·**中间状态**

用圆角矩形表示，可以用两条水平横线把它分成上、中、下3个部分。上面部分为状态的名称，这部分是必须有的；中间部分为状态变量的名字和值，这部分是可选的；下面部分是活动表，这部分也是可选的。



·状态图中两个状态之间带箭头的连线称为状态转换，箭头指明了转换方向。

•状态变迁通常是由事件触发的，在这种情况下应在表示状态转换的箭头线上标出触发转换的事件表达式；如果在箭头线上未标明事件，则表示在源状态的内部活动执行完之后自动触发转换。

•事件表达式的语法：事件说明［守卫条件］/ 动作表达式

•事件说明的语法为：事件名(参数表)

•

守卫条件是一个布尔表达式。如果同时使用事件说明和守卫条件，则当且仅当事件发生且布尔表达式为真时，状态转换才发生。如果只有守卫条件没有事件说明，则只要守卫条件为真状态转换就发生。

•动作表达式是一个过程表达式，当状态转换开始时执行该表达式。

面向对象的分析方法

功能模型——用例图

用例（Use case）需求分析

•用例需求分析方法采用一种面向对象的情景分析方法

•用例是系统向用户提供一个有价值的结果的某项功能

•从用户角度出发考虑的功能需求

•所有的用例结合起来就构成了用例模型

用例视图

•用例建模用于描述系统需求，把系统当作黑盒，从用户的角度，描述系统的场景。主要元素有以下几个：参与者、用例、执行关联

参与者（Actor）

参与者：是指外部用户或外部实体在系统中扮演的角色

定义：是直接与系统相互作用的系统、子系统或类的外部实体的抽象。它是用户所扮演的角色，是系统的用户。每个参与者定义了一个角色集合。通常，一个参与者可以代表一个人、一个计算机子系统、硬件设备或者时间等角色。典型的参与者如销售部经理、销售员和结帐系统。

图形表示

用例（Use Case）

定义

•对一组动作序列描述，系统通过执行这一组动作序列为参与者产生一个可观察的结果

图形表示

执行关联

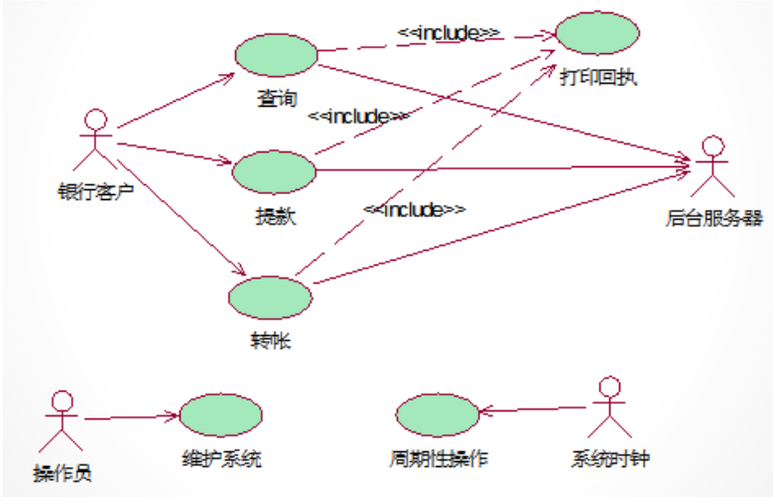
执行关联：Actor执行Use Case的关系。

泛化：用例之间的is a kind of关系，表示用例之间的场景共享；Actor之间的 is a kind of关系，一般描述职责共享。

实现：用例与用例实现之间的实现关系。

扩展：由一个用例的扩展点可以扩展出另外一个用例。

包含：一个用例可以包含另外一个用例。



用例建模的过程

1确定谁会直接使用该系统。这些都是参与者(Actor)。

2选取其中一个参与者。

3定义该参与者希望系统做什么，参与者希望系统做的每件事成为一个用例。

4对每件事来说，何时参与者会使用系统，通常会发生什么，这就是用例的基本过程。

5描述该用例的基本过程。

6考虑一些可变情况，把他们创建为扩展用例。

7复审不同用例的描述，找出其中的相同点，抽出相同点作为共同的用例。

8重复步骤2~7找出每一个用例。

数据模型——类图

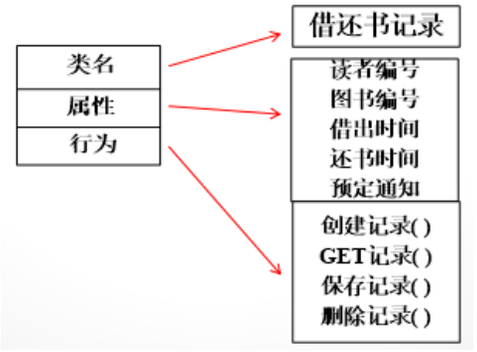
类图不仅能够表现信息的结构，还能够反映系统的行为。

•软件开发不同时期的类图反映了不同层次上的抽象。在需求分析阶段，类图用于研究领域的概念，主要反映**实体类和界面类**；在设计阶段，类图描述类与类之间的**接口和控制**；在实现阶段，类图描述系统中类的**具体实现**。

类

•类是包含信息和影响信息行为的逻辑元素。

•包含类的名字，类的属性，类的操作行为。



•寻找类有两种办法：一种是从**用例的描述**开始，检查用例描述中的每个名词。另一种是检查**时序图中的对象**，研究对象具有的共同属性和操作来发现类。

属性

•属性是与类相关联的信息，描述该类对象的共同特点。例如，“客户”类有“客户名”、“地址”。

•常见的属性可见性有Public、Private和Protected三种，分别表示为“+”、“-”和“#”

•属性的类型可以是基本数据类型，例如整数、实数、布尔型、字符串型等，也可以是用户自定义的类型。一般它由所使用的程序设计语言确定。

操作

•操作是与类相关的行为，用于修改、检索类的属性或执行某些动作。

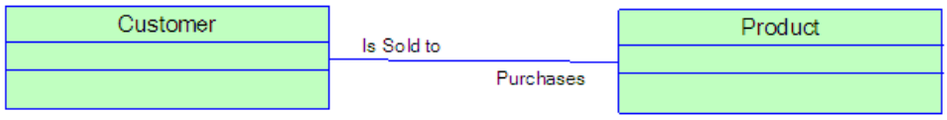
•在类图中，描述类的操作分三个部分：操作名、返回类型和参数表。

类间关系

类图中的基本关系包括：关联关系，聚合关系，组合关系，依赖关系，泛化关系等。

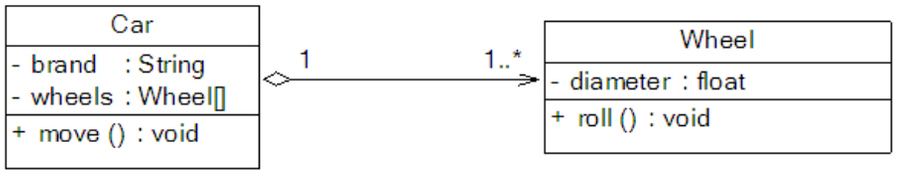
关联关系

关联是一种结构化的关系，指一种对象和另一种对象有联系。给定有关联的两个类，可以从一个类的对象得到另一个类的对象，关联有两元关系和多元关系。



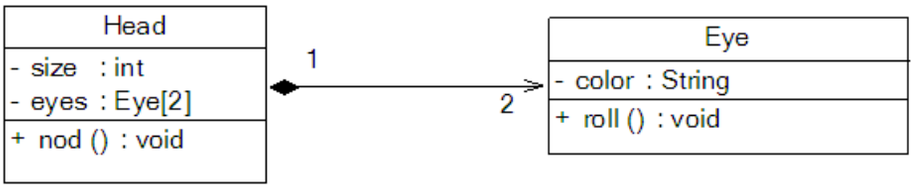
聚合关系

聚合关系指的是整体与部分的关系。**在聚合关系中，类A是类B的一部分，但是类A可以独立存在，**聚合关系用带空心菱形的直线表示。



组合关系

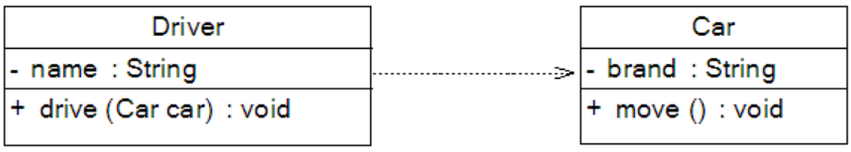
组合关系也表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期。一旦整体对象不存在，部分对象也将不存在，部分对象与整体对象之间具有共生死的关系。在组合关系中，类A包含类B，而且可以控制类B的生命周期。在UML中，组合关系用带实心菱形的直线表示。



依赖关系

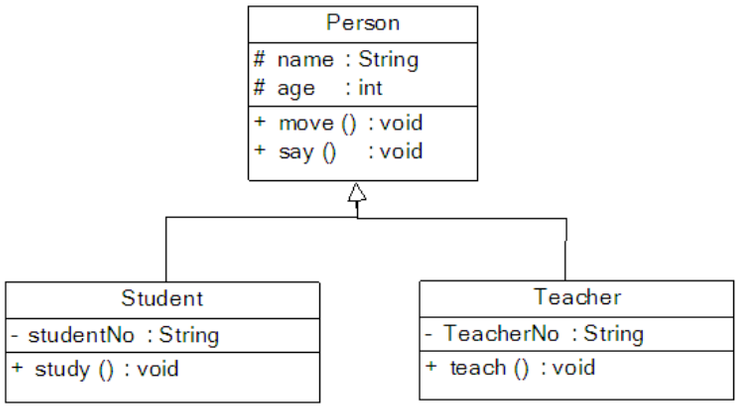
依赖关系是一种使用关系，特定事物的改变有可能会影响到使用该事物的事物。通常情况下，**依赖关系体现在某个类的方法使用另一个类作为参数。**

在UML中也可以在其它的事物之间使用依赖关系，如节点之间的关系。依赖关系用带箭头的虚线表示，由依赖的一方指向被依赖的一方



泛化关系

泛化也就是继承关系，也称为“is-a-kind-of”关系，泛化关系描述了超类与子类之间的关系，**超类又叫做基类，子类又叫做派生类。**在UML中，泛化关系用带空心三角形的直线来表示。



类的分类

•类的版型可以将类进行分类，并且有助于理解每个类的责任

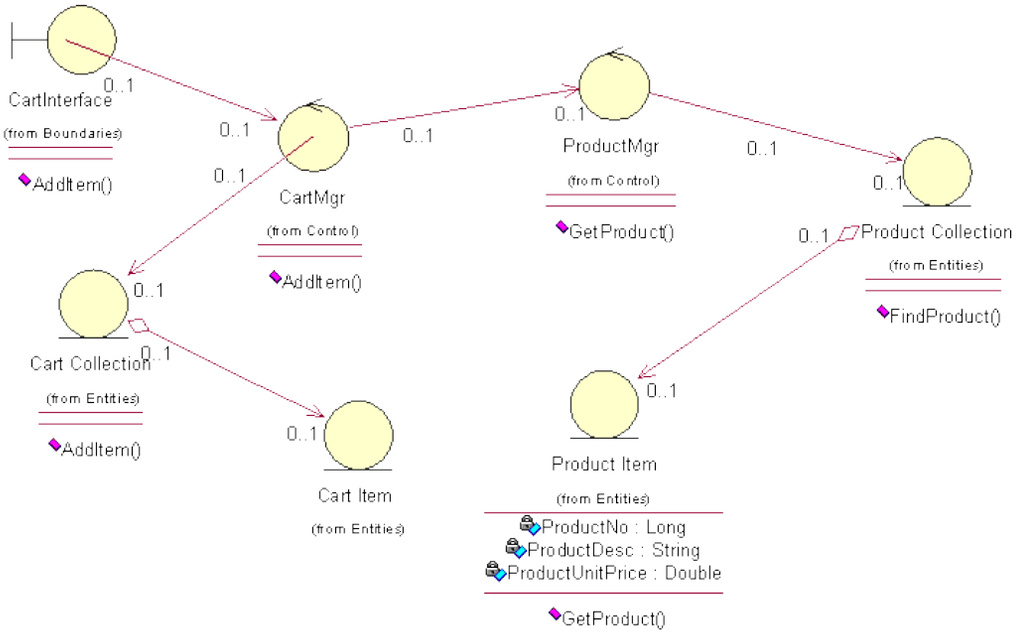
•分析过程中，可以根据功能将类分为实体类、边界类和控制类。

•边界类——位于**系统与外界的交界处**，包括所有的窗体、报表、系统硬件接口、与其它系统的接口。

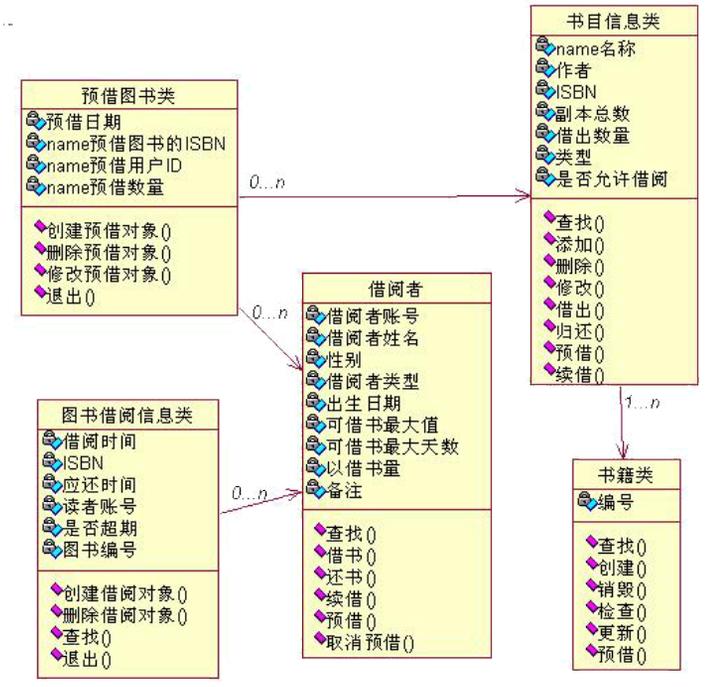
•实体类——实体类**保存要存入永久存储体的信息。**

•控制类——控制类**负责协调其它类的工作。**

分析类图



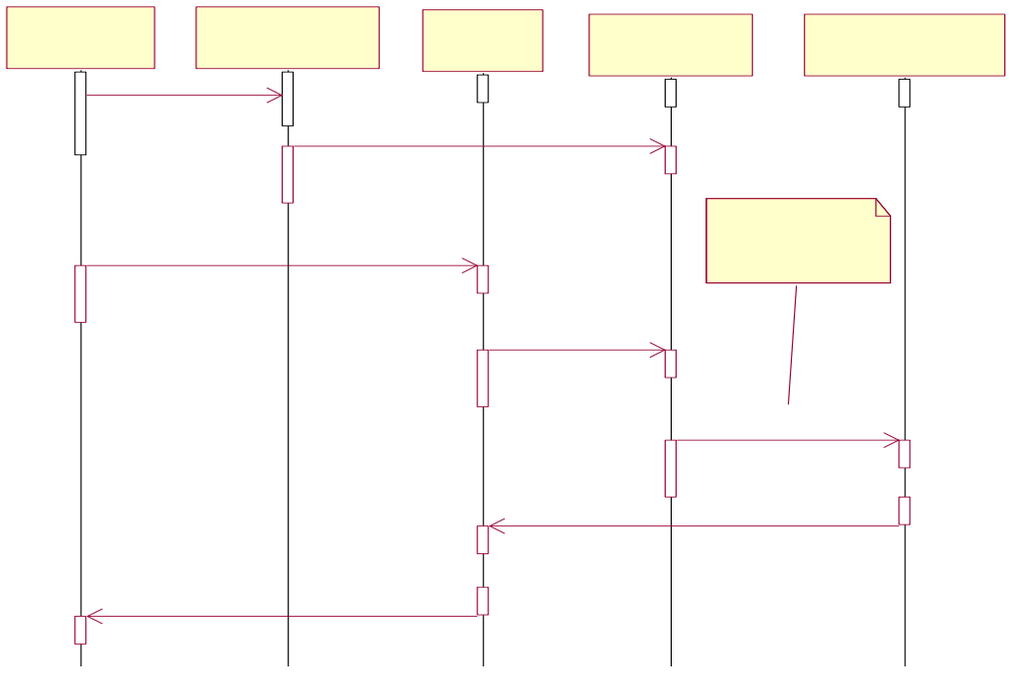
设计类图



行为模型——活动图、时序图、状态图等

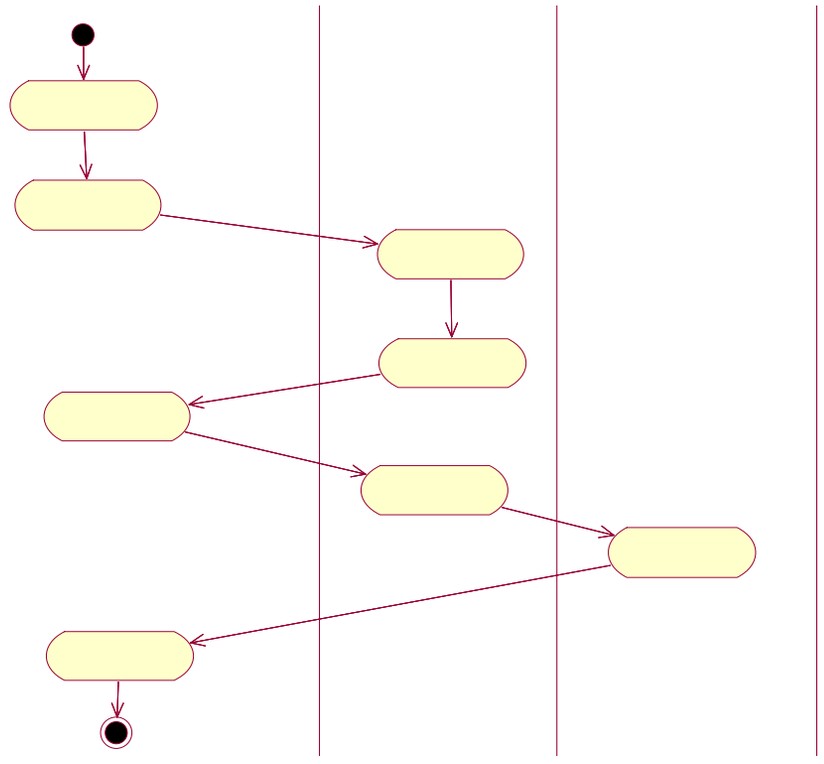
时序图

时序图展示了几个对象之间的动态协作关系，主要用来显示对象之间发送消息的顺序，还显示对象之间的交互，即系统执行某一特定时间点所发生的事。



活动图

活动图用来描述执行工作流程中涉及的活动，展示了连续的活动流



状态图

•状态图是对类描述的补充，它说明该类的对象所有可能的状态以及哪些事件将导致状态的改变。

•它是一个类对象所可能经历的所有历程的模型图

5) 掌握数据流图和用例图作法。

第四章系统设计

重点是面向数据流的设计方法、面向对象的设计方法、过程设计的常用工具。掌握软件设计的　　主要技术、主要内容和主要方法，能根据具体项目进行模块划分和软件架构设计；理解软件设计和需求分析之间的相互关系。主要知识点：

1）系统设计分为概要设计和详细设计

软件设计定义：软件系统或组件的架构、构件、接口和其他特性的定义过程及该过程的结果。

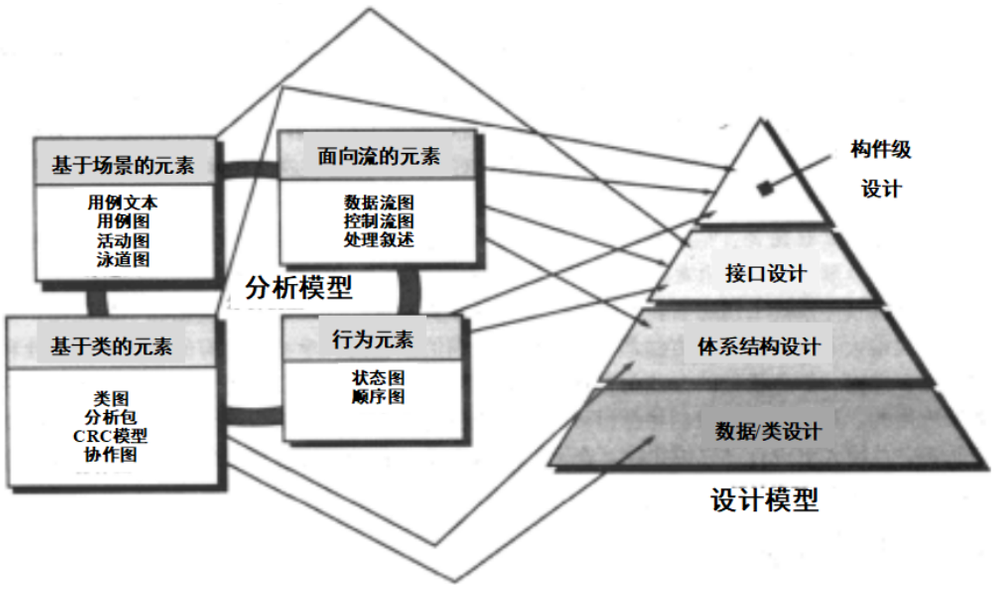
设计指导原则

设计应该是一种架构，设计应该是模块化的，设计应该包含数据、体系结构、接口和组件各个方面，应该设计出系统所用的数据结构，该设计出展现独立功能特性的各组件

设计质量属性

功能性，易用性，可靠性，性能，可支持性包含三个属性：扩展性、适应性、可维护性

设计模型分类：数据设计，架构设计，接口设计，组件级设计



2）设计相关的8个概念，着重考察体系结构、模块化、信息隐藏、功能独立。

体系结构定义：软件的整体结构和这种结构为系统提供概念上完整性的方式

体系结构设计可以使用大量的一种或多种模型来表达

结构模型、框架模型、动态模型、过程模型、功能模型

模块化定义：软件被划分为命名和功能相对独立的多个组件（模块），通过这些组件的集成来满足问题的需求

模块化设计标准

模块化的分解性：可分解为子问题

模块化的组合性：组装可重用的组件

模块化的可理解性：可作为独立单元理解

模块化的连续性：需求小变化只影响单个模块

模块化的保护：模块内异常只影响自身

信息隐藏

模块化基本问题：分解软件系统以达最佳的模块划分

信息隐藏原则：模块应该具有彼此相互隐藏的特性，模块定义和设计时应当保证模块内的信息（过程和数据）不可以被不需要这些信息的其他模块访问

特点

抽象有助于定义构成软件的过程（或信息）实体。

信息隐藏原则定义和隐藏了模块内的过程细节和模块内的本地数据结构。

功能独立定义：每个模块只解决了需求中特定的子功能，并从程序结构的其他部分看该模块具有简单的接口

优点：**易于开发**：功能被划分，接口被简化，**易于维护**（和测试）：错误传递减少，模块重用

定性衡量标准

内聚性：模块的功能相对强度

耦合性：模块之间的相互依赖程度

模块独立性强＝高内聚低耦合

设计模式定义：在给定上下文环境中一类共同问题的共同解决方案

重构定义：不改变组件功能和行为条件下，简化组件设计（或代码）的一种重组技术

方法：检查现有设计的冗余情况、未使用的设计元素、无效或不必要的算法、较差的构建方式或不恰当的数据结构，或任何其他可更改并导致更好设计的错误

抽象定义：是“忽略具体的信息将不同事物看成相同事物的过程”

数据抽象：描述数据对象的冠名数据集合

过程抽象：具有明确和有限功能的指令序列

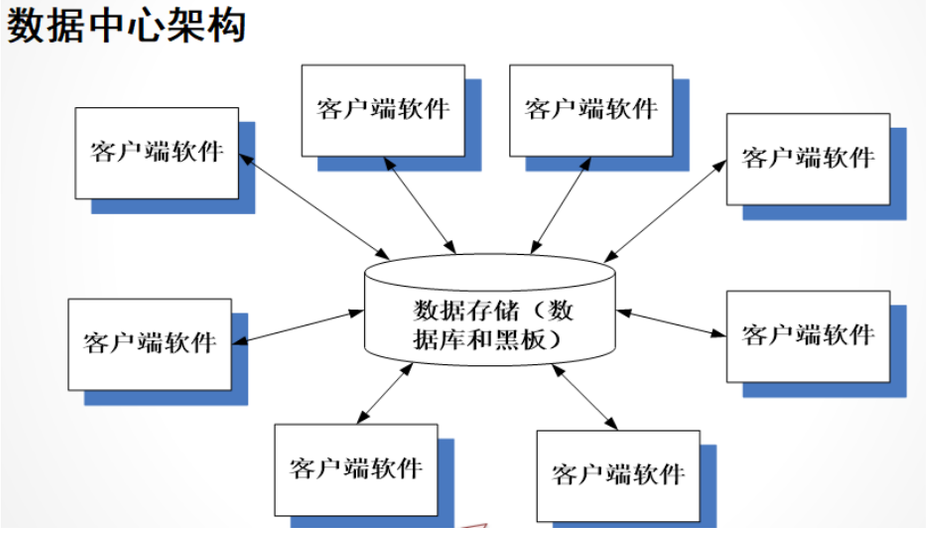
细化：逐步求精的过程

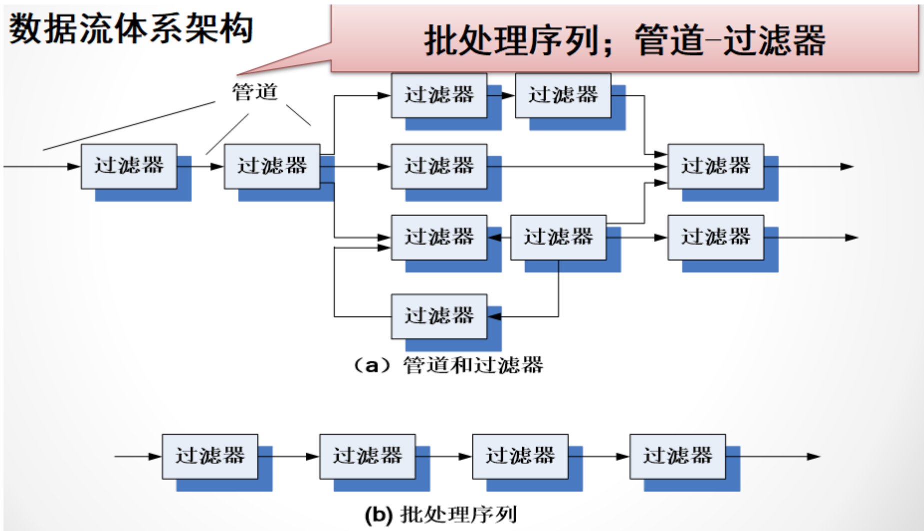
与抽象的关系：抽象使设计师确定过程和数据，但不局限于底层细节；细化有助于设计者在设计过程中揭示底层细节

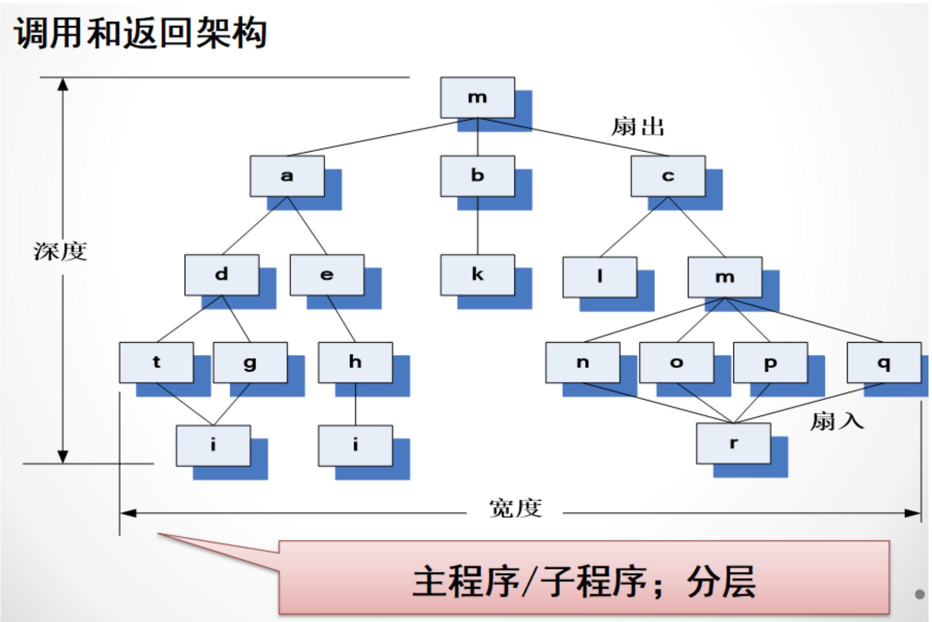
3）系统设计从体系结构、数据、接口和组件四方面进行设计。面向过程和面向对象的系统设计，各自包含哪些设计内容?

数据设计：数据设计（有时也被称为数据架构）构建高层抽象（客户/用户的数据视图）的数据模型、信息模型

体系结构设计：系统需要执行的函数功能组件集（如数据库、计算模块），组件之间通信、协同和合作的连接器，组件集成构成系统的约束，设计人员通过分析其组成部分的已知特性理解系统整体特性的语义模型分析







界面设计原则

允许用户操作控制（用户为中心）

减少用户记忆负担

保持界面一致

4) 掌握流程图和顺序图作法。

质量保证 重点是软件测试策略和技术。掌握质量保证的概念、软件测试的概念及常用方法；理解质量保 证活动在软件工程中的重要作用和意义。主要知识点： 1) 质量保证的概念 2) 测试策略 V 模型概念，测试与开发的各阶段对应关系。 3) 单元测试的内容、集成测试的分类、系统测试的分类、验收测试的分类。 4) 回归测试的概念 5) 测试技术常见术语的概念：软件缺陷、验证和确认、测试与质量保证、质量与可靠性、调 试与测试、测试用例 6) 白盒测试、黑盒测试、静态分析各有哪些方法？ 7) 掌握逻辑覆盖与等价类划分测试方法。

软件维护 重点是软件维护的分类、软件的可维护性的决定因素和软件维护技术。掌握软件维护的基本类 型、软件的可维护性的决定因素、IEEE 软件维护的过程模型；理解软件维护的基本概念、软件维 护在技术上的要点、维护费用的估算、软件逆向工程的概念及主要内容。主要知识点： 1) 软件维护的基本概念 2) 理解软件维护的四个基本类型：纠错性、适应性、完善性、预防性维护。哪种占比重最大？哪种最小？ 3) 可维护性的决定因素 4) 软件维护过程模型、软件再工程、逆向工程的概念

7. 项目管理 重点是项目计划和估计的方法。掌握软件项目管理的主要内容和主要方法，能根据具体的项目进行项目计划和项目估计；理解软件项目管理四个基本要素：人、产品、过程和项目。主要知识点：

1) 项目管理四要素：人员、产品、项目、过程（概念） 2) 软件度量有哪些方法：生产率估计（基于规模（KLOC）、基于功能点（FP））、工作量度 量（算法成本模型、COCOMO模型）。掌握直接测量（基于规模）方法。3)项目计划与风险管理的概念