软件工程课程复习大纲

第一章 软件工程概论

本章作为全书的概论，主要讲述了软件、软件工程概念和软件开发的目标和本质。

基本要求：

1、“**软件**”概念：

“软件”一词具有三层含义：

（1）一为个体含义，即指计算机系统中的程序及其文档；

（2）二为整体含义，即指在特定计算机系统中所有上述个体含义下的软件的总称，亦指计算机系统中硬件除外的所有成分；

（3）三为学科含义，即指在研究、开发、维护以及使用前述含义下的软件所涉及的理论、方法、技术所构成的学科。

2、“**软件工程**”概念：

（1）一方面，软件工程是一类求解软件的工程。它应用计算机科学、数学及管理科学等原理，借鉴传统工程的原则、方法、创建软件以达到提高质量，降低成本的目的；

（2）另一方面，软件工程也是一门指导计算机软件开发和维护的工程学科。

3、“**模型**”概念：

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。

重点要求：

1、软件开发的本质：实现问题域中的概念和处理逻辑到运行平台的概念和处理逻辑的映射。

第二章 软件过程

本章围绕软件过程这一主题，讲解了三方面的内容：

一：介绍了软件生存周期过程。 按照承担软件开发工作的主体，软件生存周期过程分三类：基本过程、支持过程和组织过程。每类过程又包含一些确定的过程，每一过程又是由一组确定的活动定义的。

二：介绍了几种常用的软件生存模型：瀑布模型、增量模型、演化模型、螺旋模型和喷泉模型等，分析了这些模型的优缺点及它们的适用情况等。

三、讲解了一个软件项目生存周期过程的规划和监控。一个软件项目生存周期过程规划包括三个阶段：第一阶段的目标是选取一个适合该项目特点的软件生存周期模型；第二阶段的目标是确定项目需要的过程、活动和任务，并将它们映射到所选取的软件生存周期模型中，形成软件项目生存周期过程及相应的文档；第三阶段的目标是针对已形成的软件项目生存周期过程，配以适当的组织过程资产，使软件项目生存周期过程成为一个可实施的过程。

基本要求：

1）软件生存周期和软件生存周期模型概念

软件生存周期（Software life cycle）:软件产品或软件系统从产生、投入使用到被淘汰的全过程。通常将软件生存周期分为 5 个阶段，即需求、设计、实现（编码）、测试和维护。

软件生存周期模型（有时称为软件开发模型）：它是整个软件生存周期内的系统开发、运行和维护所实施的全部过程、活动和任务的框架。

软件开发模型表达的是软件生存周期内各种活动如何组织，以及各个阶段应该如何衔接。

重点要求：

1）软件过程：软件生存周期中的一系列相关过程。又称为软件生存周期过程。过程是活动的集合，活动是任务的集合，任务是将输入加工成输出的操作。

2）软件过程的分类：

按照不同人员的工作内容来分，将软件生存周期过程分为三类：基本过程、支持过程和组织过程：

- 基本过程是指那些与软件生产直接相关的过程。

  包括 5 个过程：获取过程、供应过程、开发过程、运行过程、维护过程

- 支持过程是有关各方按他们的支持目标所从事的一系列相关活动集。

  包括 8 个过程：文档过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审计过程、问题解决过程等。

- 组织过程是指那些与软件生产组织有关的过程。

包括 7 个过程：a) 管理过程；　b) 基础设施过程；

c) 改进过程；　d) 人力资源过程；

f) 资产管理过程；

g) 复用程序管理过程

h) 领域软件工程过程。

3）软件过程和软件开发模型的区别？

软件过程：系统化地给出了软件开发所需要的任务；

软件开发模型：如何根据软件项目特点、环境因素等，选择并组织这些开发任务。

4）软件开发模型的概念：

软件开发模型是软件开发全部过程、活动和任务的结构框架。软件开发模型能清晰、直观地表达软件开发全部过程，明确规定要完成的主要活动和任务，它用来作为软件项目工作的基础。

– 外征：软件开发活动的组织

– 内涵：求解软件的计算逻辑

5）各模型之间的差异（这些差异本质上体现了求解软件所采用的不同计算逻辑）。

例如：瀑布模型和喷泉模型的区别？演化模型和增量模型的区别等？

第三章 软件需求与软件需求规约

不论是采用自顶向下的软件开发，还是采用自底向上的软件开发，软件需求是软件开发的工作基础。

基本要求：

1）需求的定义：

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。

2）常用的需求发现技术有哪些？

常用的发现初始需求的技术，包括：

- **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

- **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。

- **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。

- **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。

- **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。

3）软件需求规约的概念：

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。

4）软件需求规约的基本性质：

一般来说，SRS 应必须具有以下 4 个性质：

- **重要性和稳定性程度(Ranked for importance and stability)。**
- **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。**
- **完整的(Complete)。没有被遗漏的需求。**
- **一致的(Consistent)。不存在互斥的需求。**

重点要求：

1）需求分哪几类?

- **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。功能需求是整个需求的主体，即没有功能需求，就没有非功能需求，即性能需求、外部接口需求、设计约束和质量属性。

- **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。

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。

- **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。

- **设计约束**

设计约束限制了系统或系统构件的设计方案。

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。

- **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。

2）什么样的陈述可以作为需求？（即需求的基本性质）

IEEE 标准 830-1998 要求单一需求必须具有 5 个基本性质：

- **必要的(Necessary)**。是要求的吗?
- **无歧义的(Unambiguous)**。只能用一种方式解释吗?
- **可测的(testable)**。可以对它进行测试吗?
- **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?
- **可测量的(Measurable)**。可以对它进行测量吗?

3）需求规约的作用?

需求规约的作用可概括为：

第一也是最重要的，作为软件开发组织和用户之间一份事实上的技术合同书；是产品功能及其环境的体现。

第二，对于项目的其余大多数工作，它是一个管理控制点。

第三，对于产品的设计，它是一个正式的、受控的起始点。

第四，是创建产品验收测试计划和用户指南的基础，即基于需求分析规约一般还会产生另外两个文档——初始测试计划和用户系统操作描述。

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。

4）项目的需求和软件需求规约的区别?

项目需求是客户和开发者之间有关技术合同-产品/系统需求的理解，应记录在工作陈述 SOW 中或其他某一项目文档(例如，项目管理计划)中。

即 SRS 应只关注产品需求，即:

产品/系统需求-“交付给客户的产品是什么”

SOW 应关注项目工作与管理，即:

项目需求-“开发组要做的是什么”。

第四章 结构化分析方法

本章主要介绍了通常所说的“需求分析”。

根据软件工程框架，软件工程活动包括“需求、设计、实现、确认和支持”。通常，我们把其中的“需求”看作是软件开发的一个阶段，在这一阶段中，主要包括需求获取、需求分析和需求验证等活动。

需求获取的任务是给出软件系统的需求定义，即“软件需求的完整定义。它是软件开发人员与用户密切合作，了解用户的需求、目的和期望，并进一步表述而成的定义性陈述。”(参见《计算机科学技术百科全书》（第二版），清华大学出版社)。

需求分析的任务是通过形式化或半形式化手段，建立系统模型，该模型正确地、系统地表达了系统的需求，即“建立完整的需求规约(SRS，Software Requirement Specification)”，为此，人们提出了不同范型的软件需求分析方法，例如结构化分析方法、面向对象分析方法以及面向数据结构的系统分析方法等，狭义地说，这些分析方法也可称为“方法学”。

需求验证的任务是保证“需求规约”的正确性(即 SRS 中陈述的每个需求都表达了将要构造的系统的某种要求)、完整性(即·未来系统所做的任何事情均包括在 SRS 的陈述中；·在 SRS 中包括了未来系统在所有可能的情况下对所有可能输入－有效输入和无效输入－的响应；·SRS 中没有任何内容被标为“待定”］、无二义性（即在 SRS 中陈述的每个需求都只有唯一的一种解释）、一致性（即在 SRS 中的陈述，没有与以前的文档发生冲突，SRS 中陈述的各个需求之间也不发生冲突）以及可验证性、可理解性、可修改性等。

重点要求：支持需求分析的结构化方法。

1) 结构化分析方法中引入的基本概念及表示

结构化分析方法是一种基于数据流的方法，为此引入了数据流、变换（加工）、数据存储、数据源和数据潭等概念。

结构化分析的结果是建立了系统模型，并采用数据流图作为工具表示之。结构化分析方法所建立的系统模型包括三个方面：

- DFD（数据流图）
- 数据字典
- 小说明

2) 结构化分析过程

结构化分析过程可概括为：

(1) 确定系统边界，画出系统环境图
(2) 自顶向下，画出各层数据流图
(3) 定义数据字典
(4) 定义小说明
(5) 汇总各步结果

3) **针对一个简单的需求定义（文字叙述的系统需求说明），能够给出该系统的 DFD（三层），并能给出相应的数据字典和加工小说明。**

第五章 结构化设计

系统分析的结果是建立的一个系统的系统模型，以此确定了系统“做什么”。软件设计是在需求分析的基础上来确定“怎么做”，即以软件需求规格说明书为基础，形成软件的具体设计方案，即给出系统的整体模块结构和每一模块过程属性的描述——算法设计。其中，给出系统整体模块结构的过程称为总体设计或概要设计，给出每一模块过程属性描述的过程称为详细设计。

就结构化总体设计而言，其主要任务就是如何将一个系统的 DFD 转化为模块结构图（MSD），或者说把系统的功能需求分配给模块结构图。为此，首先要对诸多系统的 DFD 进行分类，以便控制求解这一问题的复杂性。DFD 可分为两类，一类是事务型数据流图，一类是变换型数据流图。在此基础上，给出了每一类 DFD 转换为模块结构图的方法。

详细设计的主要任务是定义模块，即给出实现模块功能的实现机制，包括算法和数据结构。

基本要求：

1) 变换型、事务型数据流图的概念；

变换型数据流图：具有较明显的输入、变换（或称主加工）、输出界面的数据流图，称为变换型数据流图。

事务型数据流图：数据达到一个处理 T，该处理 T 根据输入数据的类型或数据值，在其后的若干动作序列（称为一个事务）中选出一个来执行，这类数据流图称为事务型数据流图。

2) 如何将变换型 DFD 转换为初始的模块结构图；
3) 如何将事务型 DFD 转换为初始的模块结构图；
4) 如何根据低耦合、高内聚的模块独立性等原则，将初始的模块结构图转换为最终可供详细设计使用的模块结构图。
5) 详细设计工具：
   - 程序框图
   - N – S 图
   - PAD 图
   - 伪码

熟练掌握以上设计工具的表示。

重点要求：

1）如何将变换型 DFD 转换为初始的模块结构图；

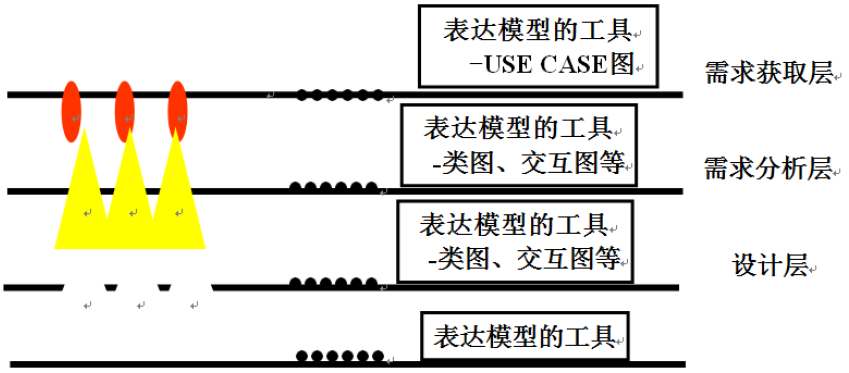
2）如何将事务型 DFD 转换为初始的模块结构图；

3）详细设计工具的各种表示以及详细设计不同表示形式的转换。

第七章　面向对象方法- UML 及 OOA

本章复习请认真阅读课程讲义胶片。

就软件开发方法学而言，UML 作为一种半形式化语言，给出了方法学中可用于不同抽象层次的术语表,给出了表达各种模型的表达格式。



基于面向对象方法的世界观,即“大千世界是由对象组成的，对象有其自己的属性和运动规律，对象之间的相互作用构成了客观世界各种各样的系统。”，为了支持软件开发，面向对象方法主要提供了两类术语:

一类是结构化地表达客观事物的术语；

一类是表达客观事物之间关系(相互作用\相互影响)的术语。

除了这两类术语之外，

为了控制信息组织和文档组织的复杂性，还引入了用于组织特定对象结构的包。包是模型元素的一个分组。一个包本身可以嵌套在其他包中，并且可以具有子包和其他种类的模型元素。

为了使建造的系统模型容易理解，引入了术语-注解，用于对模型增加一些辅助性说明。

基本要求：

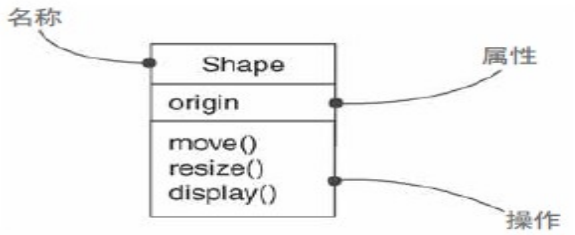
1、结构化地表达客观事物的术语：

（1）类与对象-体现数据抽象

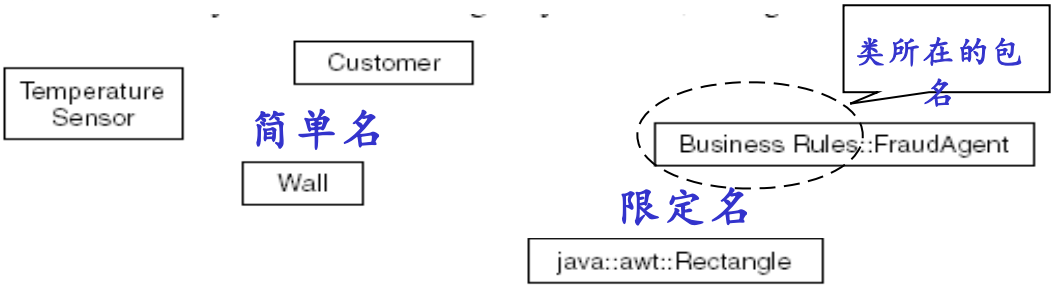
1）类和对象的定义与表示：

- 类(Class):是一组具有相同属性、操作、关系和语义的对象的描述。
  对象(object)：对象是类的一个实例。

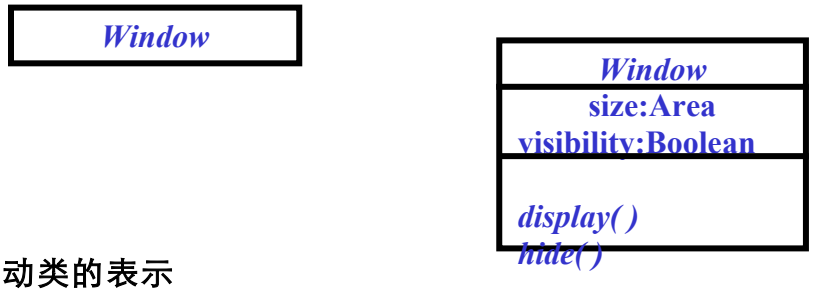
a) 类表示为具有三个栏目的矩形，如下所示:



b)依据类出现的场景,可以给出如下简化的表示:



c) 类可以是抽象类，即没有实例的类,此时类名采用斜体字:



d）主动类的表示

e）对象的表示

2) 属性

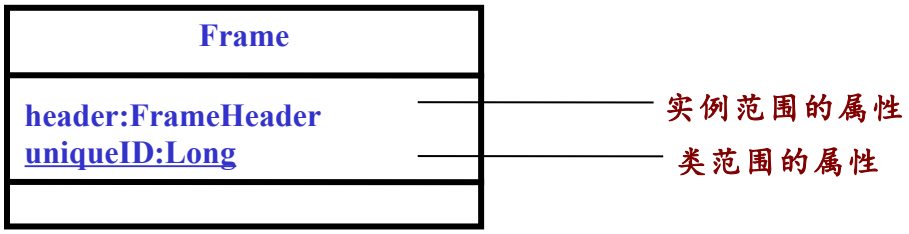
属性是类的一个命名特性，由该类的所有对象所共享，用于表达对象状态的数据。

属性是用来描述对象静态特征的一个数据项。

α) 属性的作用范围:

- **实例范围的属性**：一个类的所有对象具有相同的属性即属性的个数、名称、数据类型相同，但属性值可不同，并随程序的执行而变化．实例范围的属性是默认的，不需要附加的符号．
- **类范围的属性**：描述类的所有对象共同特征的一个数据项，对于任何对象实例，它的属性值都是相同的，通常对属性加下划线来表示该属性为实例范围的属性．

注：如 C++中冠以 static 的成员变量和 smalltalk 中的 class attribute 都是类属性。



b) 定义属性的格式为：

[可见性]属性名[:类型]［多重性］［=初始值][{特性串}]

❖ 可见性

表明该属性是否可以被其它类所使用。

其可见性的值可以为：

+ 公有的：可供其它类使用之；

受保护的：其子类可以使用之；

- 私有的：只有本类的操作才能使用之；

·包内的：只有在同一包中声名的类才能使用之。

也可以使用关键字 public、protected、private和package，分别表示公有的、受保护的、私有的和包内的。

引入“可见性”的目的，是为了支持信息隐蔽这一软件设计原则。所谓信息隐蔽是指在每个模块中所包含的信息（包括表达信息的数据以及表达信息处理的过程）不允许其它不需要这些信息的模块访问。信息隐蔽是实现模块低耦合的一种有效途径。

❖ 属性名

属性名是一个表示属性名字的标识串。通常以小写字母开头，左对齐。

❖ 类型

类型是对属性实现类型的规约，与具体实现语言有关。

❖ 多重性

多重性用于表达属性值的数目。即该类实例的这一特性可以具有的值的范围。例如：points[2..*]：Point

❖ 初始值

初始值是与语言相关的表达式，用于为新建立的对象赋予初始值。例如：origin:Point=(0,0)

❖ 性质串

如果说“类型”、“多重性”以及“初始值”都是围绕一个属性的可取值而给出的，那么“性质串”是为了表达该属性所具有的性质而给出的。例如：

a:integer=1{frozen}

3) 操作

操作是对一个类中所有对象要做的事情的抽象.

操作是用来描述对象动态特征（行为）的一个动作序列。

a) 操作的作用范围

b) 表达操作的完整语法格式为：

[可见性] 操作名[（参数表）][:返回类型][{性质串}]

其中：

- **可见性** 如同属性的可见性一样，其值可以为：

  + 公有的 可供其它类访问之；

  # 受保护的 其子类能访问之；

  - 私有的 只有本类的操作才能访问之；

  ~ 包内的 只有在同一包中声名的类才能访问之。

  当把可见性作为操作分栏中的性质串时，使用关键字 public、protected、private 和 package，分别表示公有的、受保护的、私有的和包内的。

- **操作名** 操作名一般是一动词或动词短语，通常以小写字母开头，左对齐。
- **参数表** 给出该操作的参数。
- **返回类型** 返回类型是对操作的实现类型或操作的返回值类型的规约，它与具体的实现语言有关。
- **性质串** 给出应用于该操作的性质值。

（2）接口 --体现功能抽象

接口（interface）是操作的一个集合，其中每个操作描述了类或构件的一个服务。

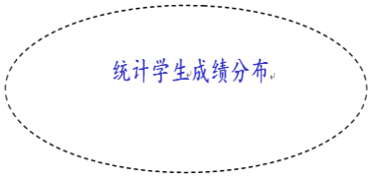
可以用带有分栏和关键字<<interface>>的矩形符号来表示接口。

其中：•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

（3） 协作（collaboration）--体现行为结构抽象

协作是一个交互，涉及交互三要素：交互各方、交互方式以及交互内容。交互各方的共同工作提供了某种协作行为。



（4）用况（use case）--体现功能抽象

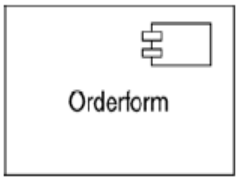
是对一组动作序列的描述，系统执行这些动作应产生对特定的参与者有值的、可观察的结果。

（5）主动类（active class）--体现并发行为抽象

是一种至少具有一个进程或线程的类，因此它能够启动控制活动。

（6）构件（component）

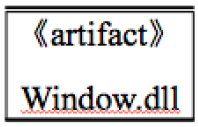
是系统设计的模块化部件，通过外部接口隐藏了它的内部实现．构件遵循并提供了一组接口的实现．



（7）制品（artifact）

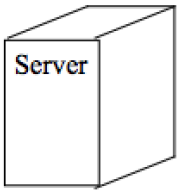
是系统中物理的、可替代的部件，其中包含物理信息(比特)。

表示：



（8）节点（node）

是在运行时存在的物理元素，通常它表示一种具有记忆能力和处理能力的计算机资源。



小结

1、抽象客观世界中任何实体的基本术语

UML 给出了以上八个术语(模型化概念)

--类、接口、协作、用况、主动类、构件、制品、节点，

它们是可包含在一个 UML 模型中的基本模型化元素。

它们存在一些变体，例如：

类的变体-参与者、信号、实用程序；

主动类的变体-进程和线程；

制品的变体-应用、文档、库、页和表等。

•在 UML 中，把以上结构化概念统称为类目（classifier）

2、表达关系的术语

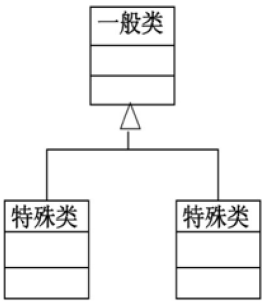
在 UML 中，提供了以下 4 种关系,作为 UML 模型中的基本关系构造块，表达类目之间的关系，以构造一个结构良好的 UML 模型.

- 关联(association)
- 泛化（generalization）
- 实现（realization）
- 依赖(dependency)

（1） 泛化

泛化是一般性事物（称为超类或父类）和它的较为特殊种类（称为子类）之间的一种关系，有时称为“is-a-kind-of”关系。

如果类 A 具有类 B 的全部属性和全部操作，而且具有自己特有的某些属性或操作，则 A 叫做 B 的特殊类，B 叫做 A 的一般类。特殊类的对象拥有其一般类的全部属性和操作，称作特殊类对一般类的继承。继承关系又称为一般－特殊关系，在 UML 中把继承关系称为泛化关系。



（2） 聚合

一种特殊形式的关联，表达一种“整体 /部分”关系。即一个类表示了一个大的事物，它是由一些小的事物（部分）组成的。

一个类的对象，以另一个类的对象作为其组成部分，这样的对象之间具有“a part of”或“has a”语义。

聚合是对象实例之间的关系。

组合是聚合的一种形式，其部分和整体之间具有很强的“属于”关系，整体类的对象管理部分类的对象，决定部分类的对象何时属于它，何时不属于它。

具有聚合关系的对象之间的关系，有两种实现方式：

- 第一种方式是用部分对象的类作为一种广义的数据类型来定义整体对象的一个属性，构成一个嵌套对象；

采用嵌套对象方式时，一个部分对象称为整体对象不可分割的一部分，其数据空间包含在整体对象之中，从生存时间看它与整体对象同生同灭。这种方

式适合于表达紧密的、固定不变的聚合关系，如飞机和发动机、人体和消化系统等。

- 第二种方式是独立地定义和创建整体对象和部分对象，并在整体对象中设立一个属性，它的值是部分对象的对象标识，或是一个指向部分对象的指针。 采用对象指针或对象标识方式时，整体对象和部分对象是各自独立创建的。这种方式适合于表达松散的、动态变化的聚合关系，并可表达一个部分对象能同时属于多个整体的情况。如一个企业与它的法律顾问就是这种松散的聚合关系。

（3） 关联

关联是类目之间的结构关系，描述了一组具有相同结构、相同语义的链（links）．链是对象之间的连接（connection）。

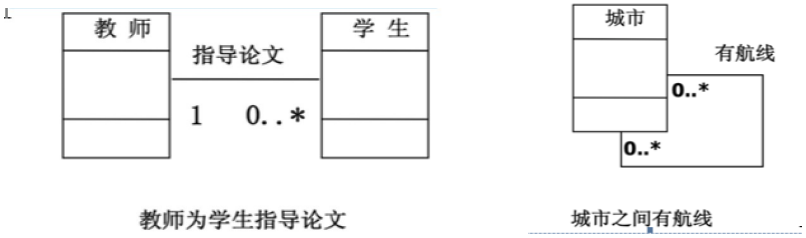
它指明一个类的对象与另一个类的对象间的联系。如果类的对象之间通过属性有连接关系，那么这些类之间的语义关系就是关联。

两个类之间可以有多个关联。

链是关联的实例，是对象间的语义连接，是对象引用的元组（列表）。在最常见的情况下，它是一对对象引用。

关联表示对象之间的静态联系（即通过对象（实例）属性体现的联系），如教师和学生之间的任课关系，在实现中可以通过对象（实例）的属性表达出来。

关联与整体—部分结构很相似，但是它没有明显的整体与部分语义。

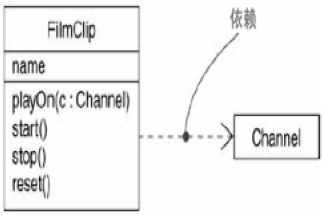


教师为学生指导论文　　城市之间有航线

（4） 依赖

依赖是一种使用关系，说明一个事物（如类 windows)使用另一个事物（如类 event)的信息和服务。显然在这种情况下，如果被使用的类发生变化，那么另一个类的操作也会受到一定影响。

在 UML 中，把依赖表示为一条有向虚线段，如下图所示。



（5）细化（realization）

定义：细化是类目之间的一种语义关系，其中一个类目规约了保证另一个类目执行的契约。

说明：在以下 2 个地方会使用细化关系：

- 接口与实现它们的类和构件之间；
- 用况与实现它们的协作之间。

多态性

对象的多态性是指在一般类中定义的属性或操作被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为。这使得一个属性或操作名在一般类及其各个特殊类中具有不同的语义。

重点要求：

1、熟练掌握 UML 的各种概念及表示方法：

- 可用于抽象客观世界中任何实体的基本术语：

类、接口、协作、用况、主动类、构件、制品、节点，以及相关的变体。

在 UML 中，把以上结构化概念统称为类目（classifier）

- 可用于组织信息的术语—包,以及可用于解释信息的术语—注解
- 可用于抽象客观世界中任何实体关系的基本术语：

关联，泛化，细化，依赖，以及相关的特殊形式。

其中为了增强关系语义的表达，还给出了一些基本概念，例如

角色名，多重性，限定符，关联类等。

2、

针对一个简单的问题，能够建立用况图、类图以及包图、状态图。并以一个交互为例，给出其顺序图。

3、OOA 的过程：

（1）发现对象、定义它们的类；

（2）识别对象的内部特征；

- 定义属性
- 定义操作

（3）识别对象的外部关系

- 分类关系、继承——泛化（一般-特殊）

泛化、一般类、特殊类

- 构成关系——聚合（整体-部分）

聚合、聚集、成分、组合

- 静态联系——关联(实例连接)

关联、链、多重性、角色、多元关联、关联类

- 使用关系（行为依赖）——依赖

依赖

（4） 给出系统的相关顺序图、通讯图和活动图等，以建立系统的动态模型。

（5） 划分包，建立系统的包图

（6）建立系统的详细说明

4、

面向对象分析方法与结构化分析方法的比较

（1） 面向对象方法在建造系统模型时，捕获的是比较稳定的“原子”－对象，而结构化方法捕获的是相对不太稳定的“加工”和“数据”，因此，用面向对象方法建造的模型是比较稳定的。

（2） 面向对象方法在分析阶段和设计阶段采用了统一的符号体系，而结构化方法不然，因此，从分析到设计需要一个映射，即分析和设计之间存在一个所谓的“鸿沟”，但面向对象方法几乎没有这一问题。

（3） 采用面向对象方法建造的系统模型由于使用接近客观的对象和关系，作为模型的基本元素，因此使模型结构几乎与客观世界保持一致，这对系统的维护提供了很大的方便。而结构化方法则不然，由于使用过程抽象和数据抽象，使建造的模型结构与客观世界的结构具有很大的差异。

（4） 面向对象方法由于使用了继承和多态性等概念，因此可以支持复用，而结构化方法在复用方面，也不及面向对象方法。

5、 面向对象分析方法中复杂度控制的机制

a) 信息组织的复杂性：

- 抽象:从许多事物中舍弃个别的、非本质的特征，抽取共同的、本质性的特征：

  系统中的对象是对现实世界中事物的抽象；

  类是对象的抽象；

  一般类是对特殊类的抽象；

  属性是事物静态特征的抽象；

  操作是事物动态特征的抽象。

- 分类机制:把具有相同属性和操作的对象划分为一类，用类作为这些对象的抽象描述。
- 继承:特殊类的对象拥有其一般类的全部属性和服务(一般-特殊结构)；
- 聚合:把一个复杂的事物看成若干比较简单的事物的组装体，从而简化对复杂事物的描述。(整体-部分结构)
- 消息通讯：要求对象之间只能通过消息进行通讯，而不允许在对象之外直接地存取对象内部的属性。
- 多个视图：从多个角度认识系统

b) 文档组织的复杂性—控制机制

包：使模型具有大小不同的粒度层次，以利于控制复杂性

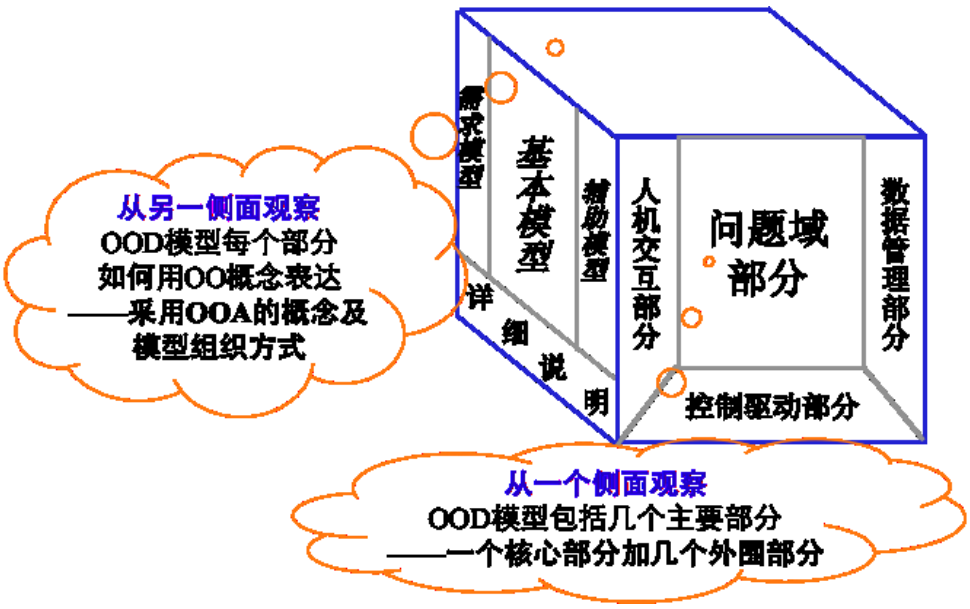
第七章　面向对象设计

本章复习请认真阅读课程讲义胶片

主要介绍了面向对象设计（OOD）。根据 OOD 模型，比较详细地讲述了“**问题域部分**”，“**人机交互部分**”，“**控制驱动部分**”和“**数据管理部分**”的设计。并针对每一部分设计，给出了主要任务及实施策略。

OOD模型

——从两个侧面来描述



基本要求：

1、　　OOD 模型的四个组成部分：

- 问题域部分：将OOA结果搬到OOD，并根据实现条件（例如编程语言、可复用构件、机器性能、存储方案等）做必要的补充与调整，其结果就是OOD的问题域部分。
- 人机交互部分：根据选用的图形用户界面和特定用户对人机界面的要求而设计的系统人机界面。它是由新定义的关于人机界面的类及对象构成的。
  - **OOA：通过人机界面反映需求（原型开发）**
  - **OOD：设计人机交互的细节**
- 控制驱动部分：用于定义系统中需要并发执行的各个任务。该部分由

系统中全部主动类构成。这些主动类描述了整个系统中所有的主动对象，每个主动对象是系统中的一个控制流的驱动者。

- 数据管理部分：按选定的数据管理系统而设计的负责对象存储及检索的系统组成部分

2、 OOD 过程：针对四个部分，进行四个相应的设计活动：

（1）问题域部分的设计
（2）人机交互部分的设计
（3）控制驱动部分的设计
（4）数据管理部分的设计

3、OOD 每一部分设计的主要任务。

第八章 软件测试

本章主要针对程序测试，介绍了两种常有的测试技术——基于“白盒”的路径测试技术和基于“黑盒”的事务处理流程测试技术。

基本要求：

1) **软件测试与调试的差异；**
2) **测试过程模型；**
3) **路径测试技术，其中要掌握该技术的基本概念；**

- 控制流程图
- 路径

以及路径测试策略，并能针对一个特定的控制流程图，设计最少的测试用例，实现语句覆盖、分支覆盖和条件组合覆盖；

4) **事务流测试技术**，其中要理解事务的概念，并清楚这一技术与路径测试技术的“相同点”和不同点；

5) **软件测试步骤；**

- 单元测试（往往采用白盒测试技术）：集中于每个独立的模块。该测试以详细设计文档为指导，测试模块内的重要控制路径。
- 集成测试：集中于模块的组装。其目标是发现与接口有关的错误，将经过单元测试的模块构成一个满足设计要求的软件结构。
- 有效性测试：目标是发现软件实现的功能与需求规格说明书不一致的错误。（通常采用黑盒测试技术）
- 系统测试：集中检验系统所有元素（包括硬件、软件）之间协作是否合适，整个系统的性能、功能是否达到。

单元测试在实现阶段进行，它所依据的模块功能描述和内部细节以及测试方案应在 详细设计 阶段完成，目的是发现编程错误。

集成测试 所依据的模块说明书和测试方案应在 概要设计 阶段完成，它能发现设计错误。

有效性测试 应在模拟的环境中进行强度测试的基础上，测试计划应在软件需求分析阶段完成。

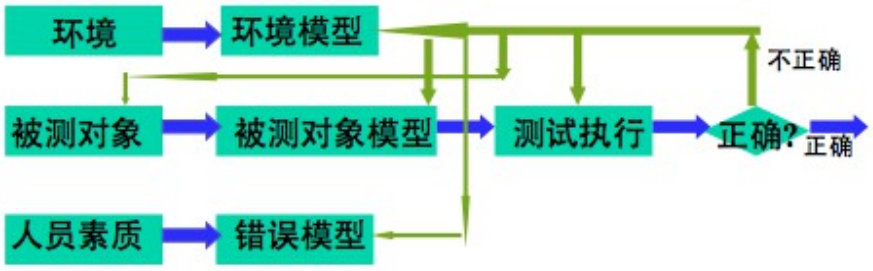
6) 测试技术与测试模型的关系。

不同测试技术，对同一被测对象—程序，可产生不同的测试程序模型。这一简化或着重于程序的控制结构，或着重于处理过程，于是形成了所谓的“白盒”测试和“黑盒”测试。

重点内容：

上述第 2、3 和 5 点：

（1）测试过程模型



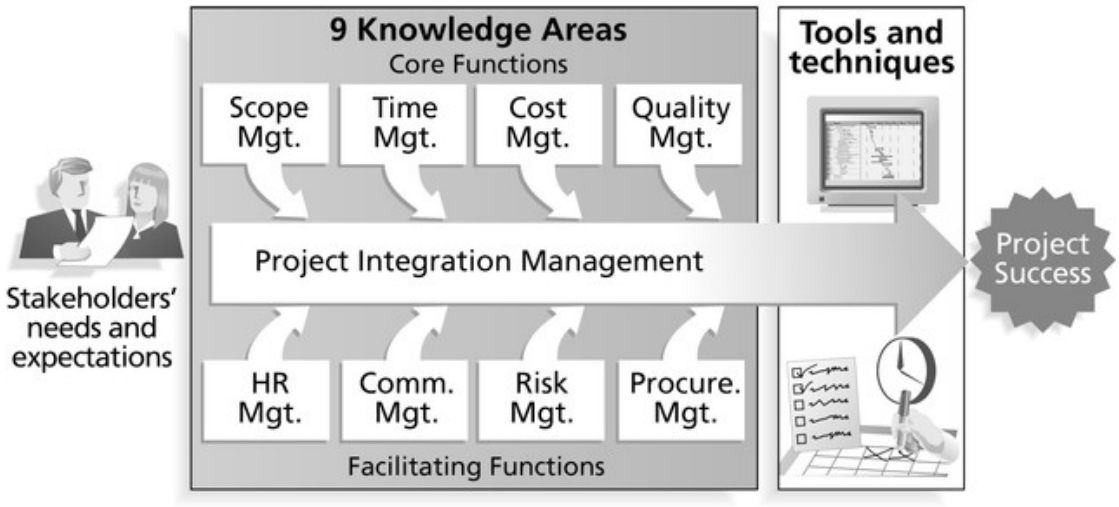
- 软件测试过程所涉及的要素,以及
- 这些要素之间的关系

（2）路径测试技术，要求熟练掌握白盒测试中各种逻辑覆盖，并能针对一个特定的控制流程图，设计最少的测试用例，实现语句覆盖、分支覆盖和条件组合覆盖；

（3）软件测试步骤

第九章 软件工程项目管理概述

基本要求：



项目管理框架

1、项目管理九大知识领域

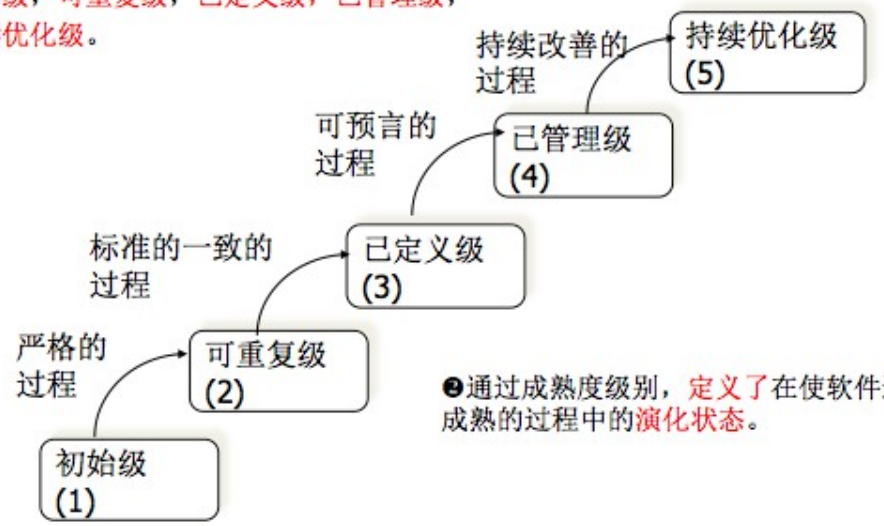
是指项目经理必须具备的一些重要的知识和能力。

- 项目管理的四大核心知识领域是指范围、时间、成本和质量。这四个方面会形成具体项目的项目目标
- 四大项目管理辅助知识领域包括人力资源管理、风险管理、沟通管理和采购管理。之所以称其为辅助知识领域，是因为项目目标是通过他们来实现的。
- 项目整体管理包括在项目生命周期中协调所有其他项目管理知识领域所涉及的过程。

2、CMM五级模型

(1)成熟度框架

❶在这一框架中，将过程能力成熟度分为五级：初始级，可重复级，已定义级，已管理级，持续优化级。



❷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。

第十章　计算辅助软件工程 CASE

本章主要对 CASE 概念作了一般性介绍，并对 CASE 系统进行了分类。在此基础上，针对工作台和软件开发环境，给出了相应的概念框架。最后，简要介绍了我国自行研制的大型软件开发环境青鸟系统。

本章的基本要求：

1) CASE 概念；
2) CASE 系统分类；
3) 软件工程环境的定义；
4) 集成化软件工程环境的五级模型、SEE 模型；
5) 程序设计工作台、分析和设计工作台、测试工作台的概念模型；
6) 青鸟系统总体结构，并能简单扼要地说明其中的主要成分以及各成分之间的关系。

重点要求：

以上 3、4、5 点即：

1）软件工程环境的定义；

2）wasserman 提出的集成化软件工程环境的五级模型：

- 平台集成
- 数据集成
- 表示集成

- 控制集成
- 过程集成

3) SEE 模型

4）程序设计工作台、分析和设计工作台、测试工作台的概念模型。