软件工程

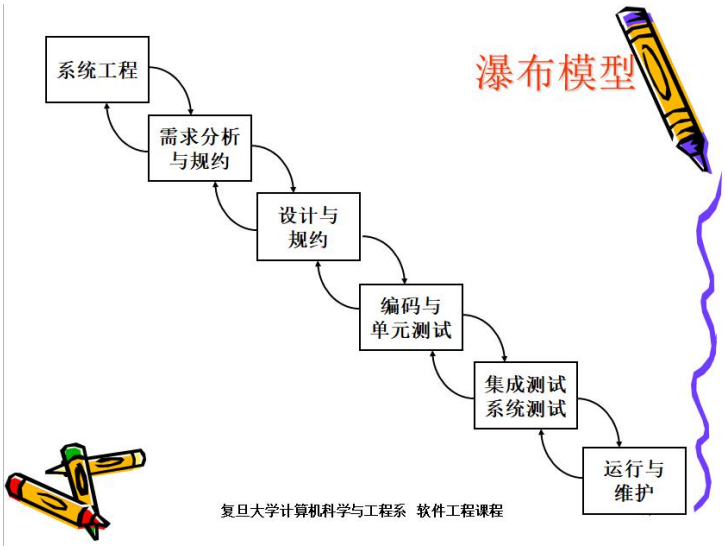
一、软件过程

软件过程的概念

软件过程又称软件生存周期过程，定义了软件组织和人员在软件产品的定义、开发和维护等阶段所实施的一系列活动和任务。同时软件过程也描述了活动及任务的时序关系以及预期目标的途径。
软件生存周期大体可分为如下几个活动：**计算机系统工程、需求分析、设计、编码、测试、运行和维护。**

经典软件过程模型的特点（瀑布模型、增量模型、演化模型、统一过程模型）

瀑布模型

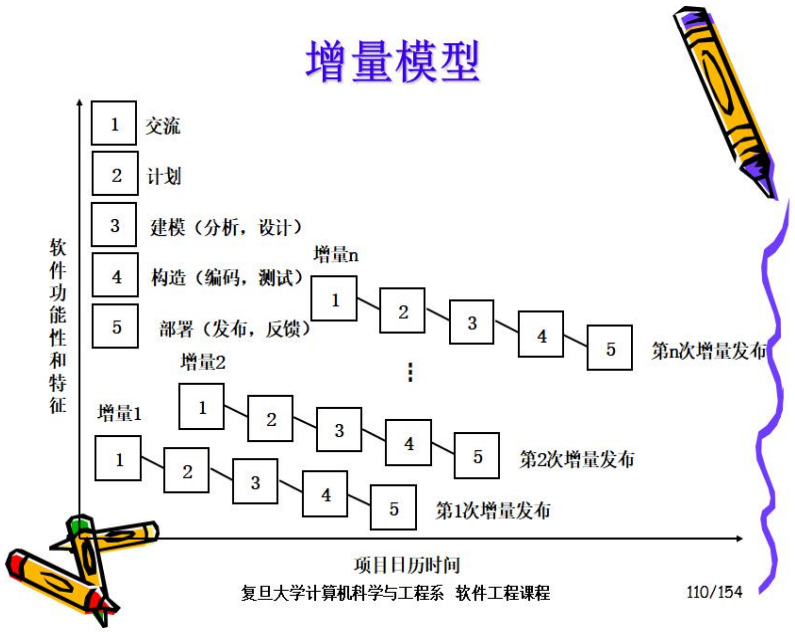


一个阶段完成后再开始下一个阶段。
特点：过于理想，假设每个环节都可以一次通过。

演化模型

演化模型适用于对软件需求缺乏准确认识的情况。
许多软件项目在开发早期对软件需求的认识是模糊的、不确定的，因此软件很难一次开发成功。
可以在获取了一组基本的需求后，通过快速分析构造出该软件的一个初始可运行版本，称之谓原型（prototype），然后根据用户在试用原型的过程中提出的意见和建议、或者增加新的需求，对原型进行改造，获得原型的新版本，重复这一过程，最终得到令客户满意的软件产品。
特点：演化模型采用迭代的思想，渐进地开发，逐步完整软件产品。
典型的演化模型有：增量模型、原型模型、螺旋模型

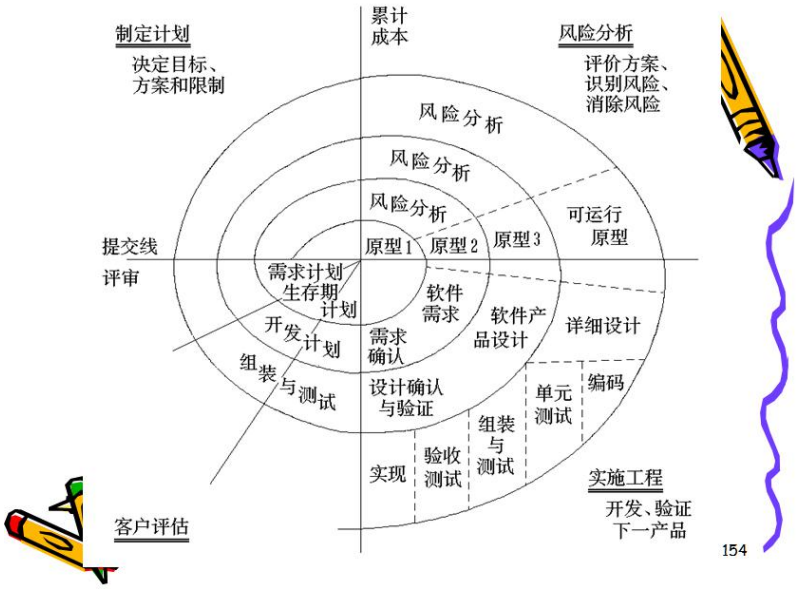
增量模型



增量模型将产品的开发分成若干个增量进行，每个增量都执行一系列活动，且生产出一个可执行的中间产品，后一个版本是对前一版本的修改和补充，重复增量发布的过程，直至产生最终的完善产品。

特点：增量模型融合了瀑布模型的基本成分（重复地应用）和演化模型的迭代特征。

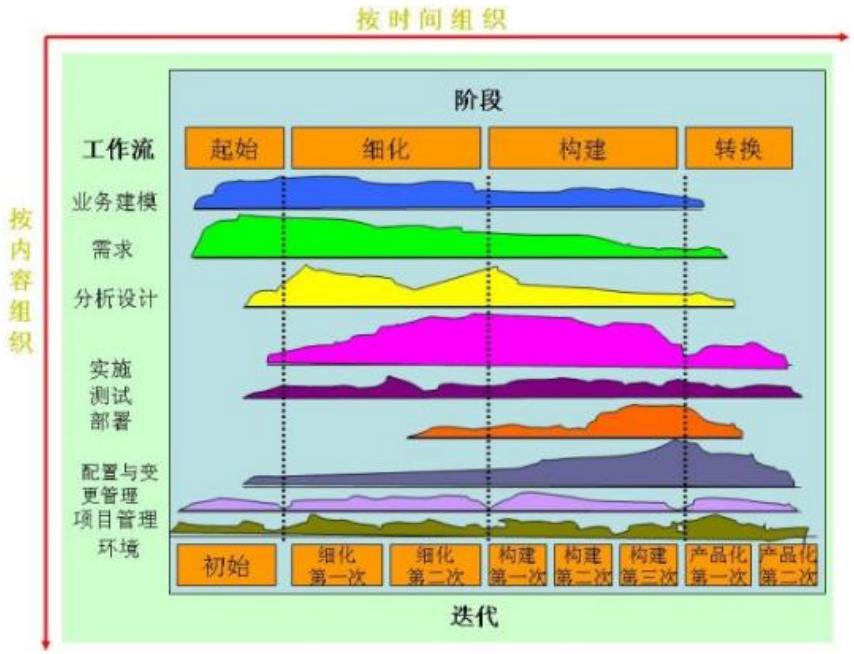
螺旋模型



螺旋模型是瀑布模型和演化模型的结合，并增加了**风险分析**。

特点：首先是对风险的强调。在每个迭代中都包含一个显式的风险分析过程，使得开发人员和用户对本次迭代可能出现的风险有所了解，继而可以采取相应的行为减少或者消除风险的损害。其次，螺旋模型通过把每次迭代映射到 4 个象限，清晰地定义了里程碑，从而有助于不同角色之间的沟通和协作。然而，过多的迭代次数会增加开发成本，延迟提交时间。

统一过程模型



特点：统一过程是一种以用例驱动、以体系结构为核心、迭代及增量的软件过程模型。

用例驱动

用例获取系统的功能需求，它们“驱动”需求分析之后的所有阶段的开发。

以体系结构为核心

首先定义一个基础的体系结构，然后将它原型化并加以评估，最后进行精化。

迭代

不要试图一次就定义模型或图的所有细节，开发是逐步进行的，每次迭代增加一些新的信息和细节。每次迭代都要对前次的结果评价，并用于下一次迭代的输入。迭代的过程是提供连续的反馈，这些反馈不仅改善了最终的产品，而是改善了过程本身。

在决定每次迭代应做什么时，要考虑这次迭代对系统的最大影响或最高风险。每个迭代周期都是一个小的瀑布模型。

增量

增量开发是在多次迭代的过程中每次增加一些功能(或用例)的开发，每次迭代都包含分析、设计、实现、测试等阶段。

RUP（Rational Unified Process）的四个阶段

初始阶段：大体上的构想，业务案例，范围，和模糊评估。定义系统的业务模型，确定系统的范围。完成后建立目标里程碑。

细化阶段：已精化的构想、核心架构的迭代实现、高风险的解决、确定大多数需求和范围以及进行更为实际的评估。建立结构里程碑。

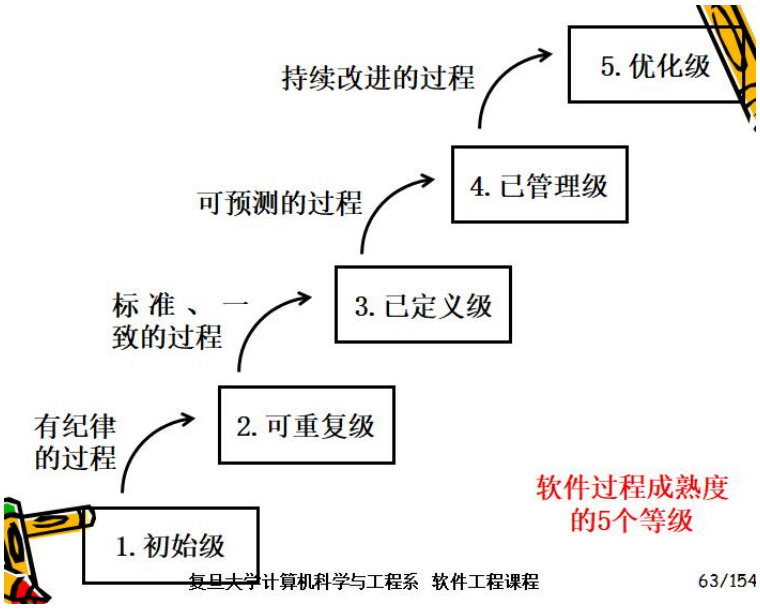
构造阶段：对遗留下的风险较低和比较简单的元素进行迭代实现，准备部署。构造产品，并继续演进需求、体系结构和计划，直到产品完成。

移交阶段：进行系统部署，系统测试，最终移交给用户。建立发布里程碑。

过程评估与 CMM/CMMI 的基本概念

CMM 的基本概念

CMM（Capability Maturity Model）即能力成熟度模型，是美国卡耐基梅隆大学软件工程研究所（SEI）在美国国防部资助下于二十世纪八十年代末建立的，用于评价软件机构的软件过程能力成熟度的模型。此模型在建立和发展之初，主要目的在于提供一种评价软件承接方能力的方法，为大型软件项目的招投标活动提供一种全面而客观的评审依据。而发展到后来，又同时被软件组织用于改进其软件过程。



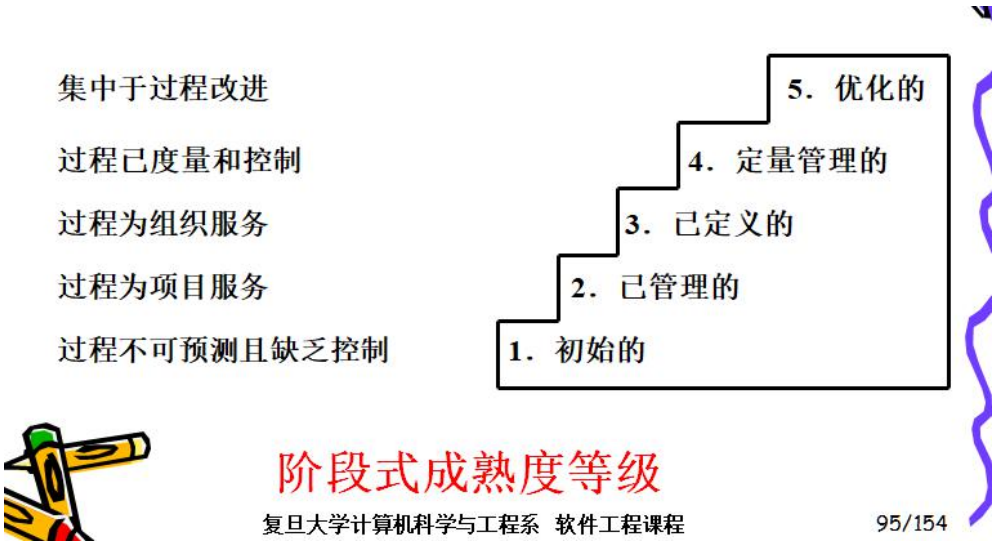
CMMI 的基本概念

美国国防部、美国国防工业委员会和 SEI/CMU 于 1998 年启动 CMMI 项目，希望 CMMI 是若干过程模型的综合和改进，是支持多个工程学科和领域的系统的、一致的过程改进框架，能适应现代工程的特点和需要，能提高过程的质量和工作效率。

CMMI 模型为每个学科的组合都提供两种表示法：阶段式模型和连续式模型

阶段式模型

阶段式模型的结构类同于软件 CMM，它关注组织的成熟度，其成熟度等级如下图所示



敏捷宣言与敏捷过程的特点

敏捷宣言

1) 个人和交互高于过程和工具
2) 可运行软件高于详尽的文档
3) 与客户协作高于合同（契约）谈判
4) 对变更及时做出反应高于遵循计划

个人和交互高于过程和工具

不是否定过程和工具的重要性，而是更强调软件开发中人的作用和交流的作用。

可运行软件高于详尽的文档

通过执行一个可运行的软件来了解软件做了什么，远比阅读厚厚的文档要容易得多。

与客户协作高于合同（契约）谈判

要想通过合同谈判的方式，将需求固定下来常常是困难的。敏捷软件开发强调与客户的协作，通过与客户的交流和紧密合作来发现用户的需求。

对变更及时做出反应高于遵循计划

随着项目的进展，需求、业务环境、技术等都可能变化，任务的优先顺序和起止日期也可能因种种原因会改变。因此，项目计划应具有可塑性，有变动的余地。当出现变化时及时做出反应，修订计划以适应变化。

二、软件需求

软件需求的概念

软件需求是对软件产品或服务所需要具备的外部属性的一种刻画，这些属性应当保证所提供的解决方案能满足用户所需要解决的显示世界问题的需要。

软件需求一般包括三类：

功能性需求： 实现功能；

质量需求： 要求质量属性，如系统性能、可靠性、稳定性等；

约束： 开发语言，系统运行上下文环境等；

需求工程的基本过程

需求获取：调查研究

需求提炼：分析建模

需求描述：编写软件需求说明书

需求验证

分层数据流模型

它是将提供给用户的业务流程图(“物理模型”)进行功能建模，转化成开发人员能够理解的一系列“逻辑模型”图，即以图形化的方法描绘数据在系统中的流动和处理的过程，这些图都应该用规范的 DFD 描述。

对于一个软件系统，其数据流图可能有许多层，每一层又有许多张图。为了区分不同的加工和不同的 DFD 子图，应该对每张图进行编号，以便于管理。

- 顶层图只有一张，图中的加工也只有一个，所以不必为其编号。
- 0 层图只有一张，图中的加工号分别是 0.1、0.2、…，或者 1， 2 。
- 子图就是父图中被分解的加工号。
- 子图中的加工号是由图号、圆点和序号组成，如：1.12，1.3 等等。

图3-3 飞机机票预定系统顶层图

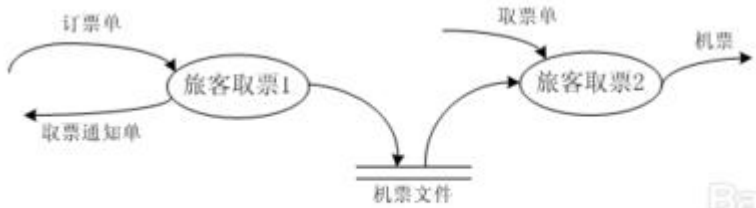


图3-4 飞机机票预定系统0层图

数据流图有四种基本图形符号：

：箭头，表示数据流；

〇：圆或椭圆，表示加工；

= ：双杠，表示数据存储；

□：方框，表示数据的源点或终点。

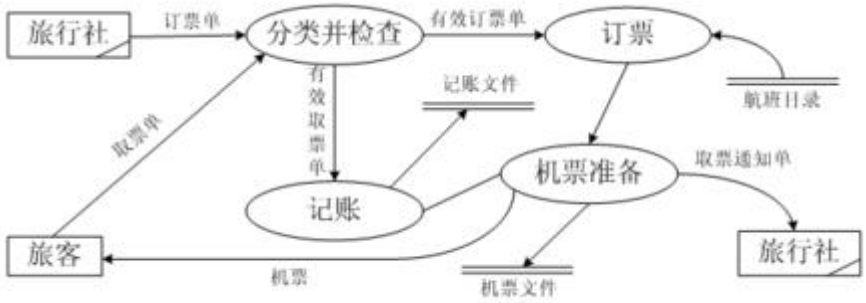
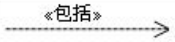
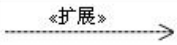


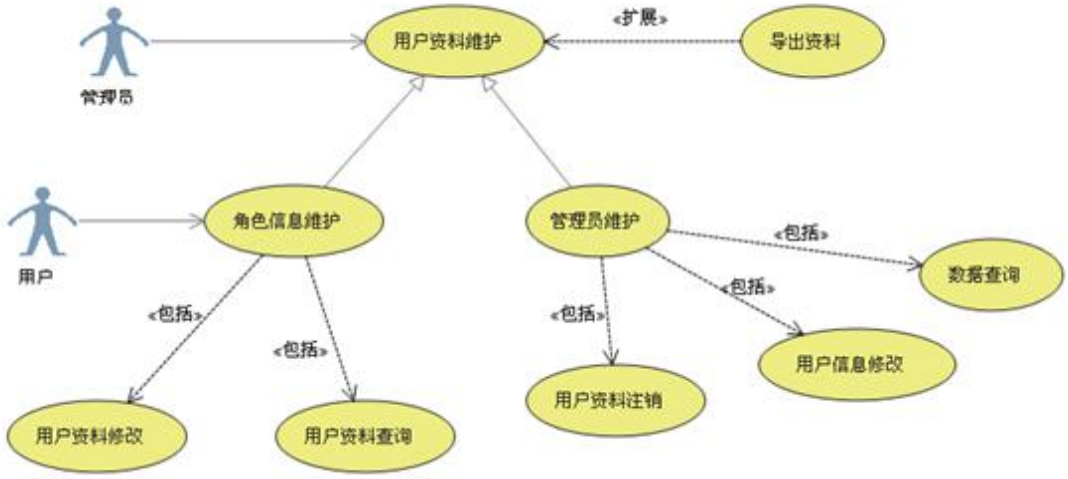
图3-2 飞机机票预订系统

用例和场景建模及其 UML 表达（用例图、活动图、泳道图、顺序图）

用例图

也被称为用户模型图，是从软件的需求分析的到最终实现的第一步，它是从客户角度来描述系统功能的。
它包含三个基本组件：参与者(使用系统的人或事物)、用例(代表系统的某项完整的功能，在图形中使用椭圆型表示)、关系(关联、泛化、包含、扩展)。
扩展关系：如果一个功能在完成的时候，偶尔会执行另外一个功能，使用扩展关系表示。
泛化关系：表示同一个业务的不同技术实现。其实就是继承关系的一种。
包含关系：是指一个用例可以含有其他用例具有的行为。

关系类型	说明	表示符号
关联	参与者与用例间的关系	
泛化	参与者之间或用例之间的关系	
包含	用例之间的关系	
扩展	用例之间的关系	



活动图

它用于描述系统的活动，判定点和分支等。活动中的动作状态，原子的、不可已中断的动作。并在此动作完成后向另一个动作转变。
分支与合并。分支在软件系统中很常见：用于表示对象类具有的条件行为。用一个布尔型的表达式真假来判定动作的流向，合并有两个如转换一个出转换。分支有一个如转换两个出转换。
分叉与汇合：分叉又来描述并发线程。每个分叉可以有一个输入的转换和两个或多个输出转换。汇合代表两个或多个并发控制流的同步发生。当所有流都到达汇合点后，程序才能继续前进。

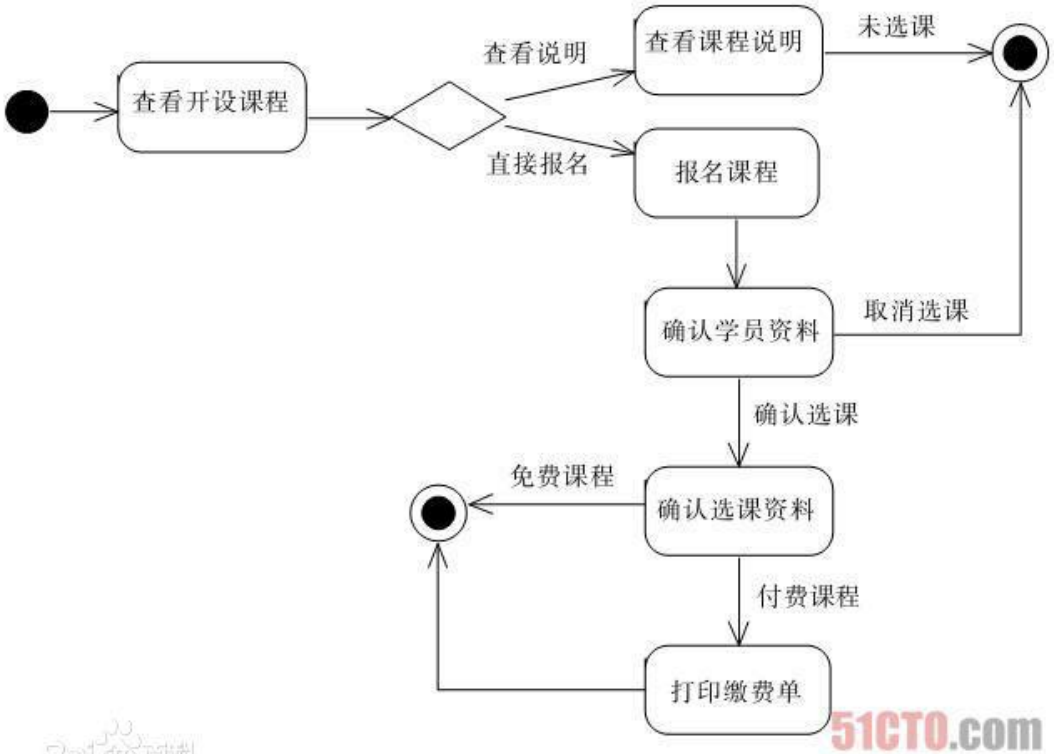
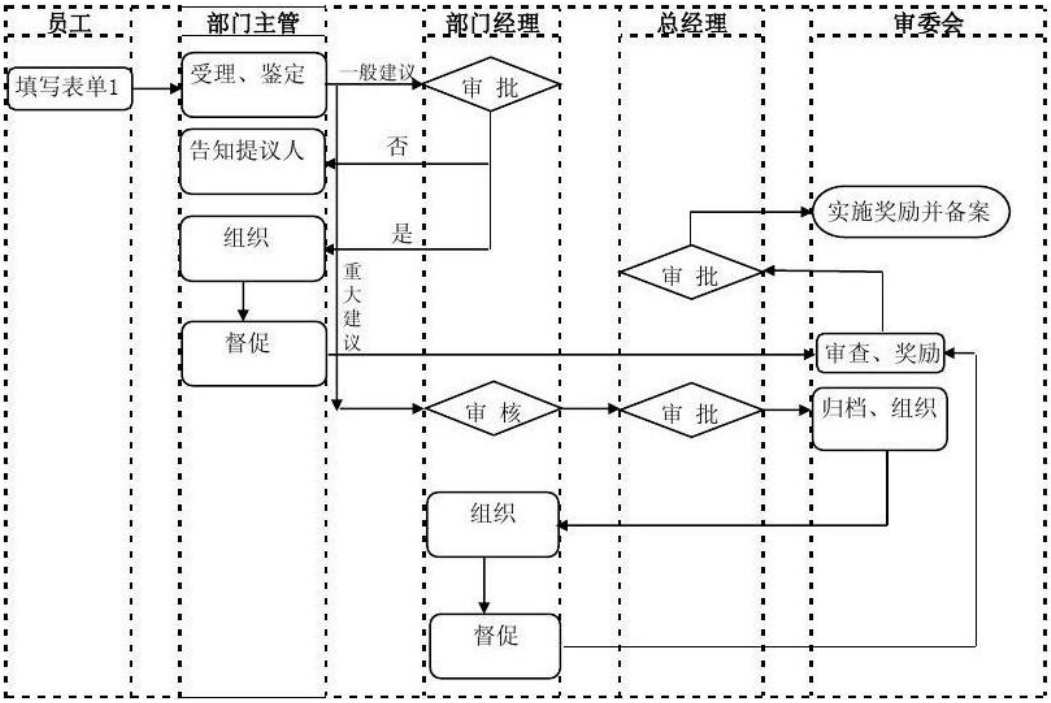


图 1-56　选课流程的活动图

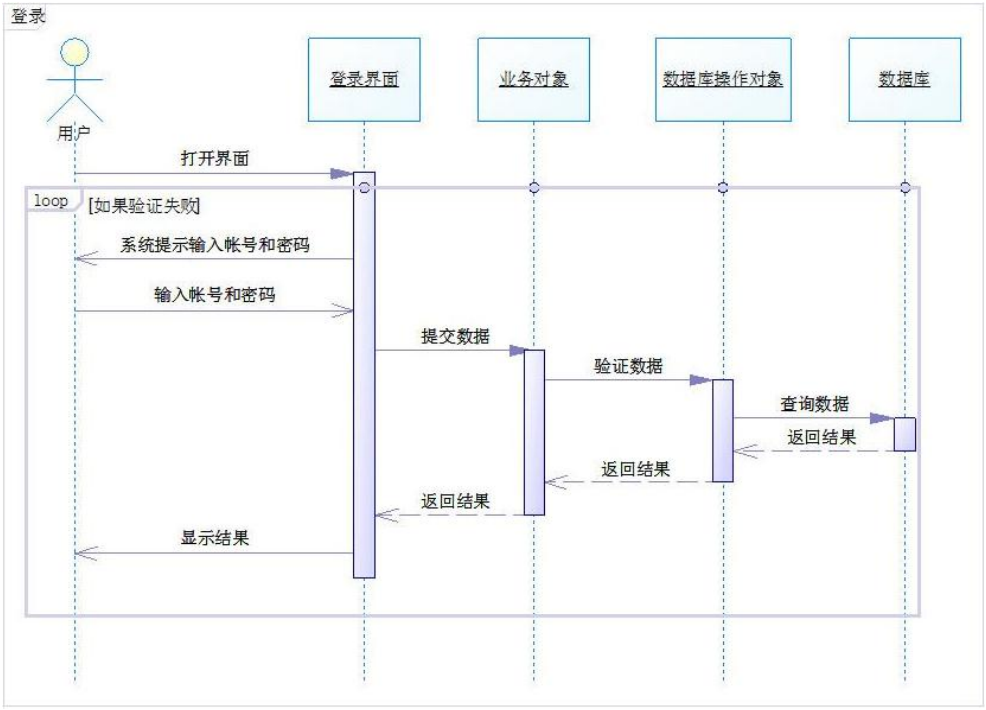
泳道图

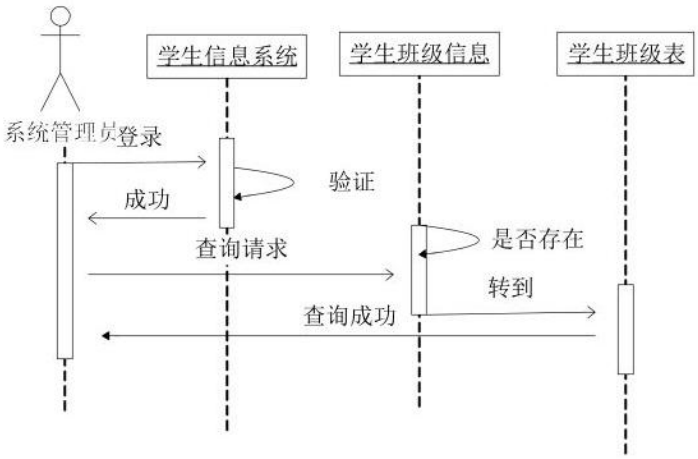
泳道图是将模型中的活动按照职责组织起来。这种分配可以通过将活动组织成用线分开的不同区域来表示。由于它们的外观的缘故，这些区域被称作泳道。它可以方便的描述企业的各种业务流程，能够直观地描述系统的各活动之间的逻辑关系，利于用户理解业务逻辑。



顺序图

时序图用于描述对象之间的传递信息的时间顺序。即用例中的行为顺序。当执行一个用例时，时序图中的每一条消息对应了一个类中操作或者引起转换的触发事件。纵轴表示时间时间轴向下延伸。横轴代表协作中的各个独立对象。对象存在时。消息用从一个对象的生命线到另个对象的生命线的箭头表示。箭头以时间的顺序在图中上下排列。



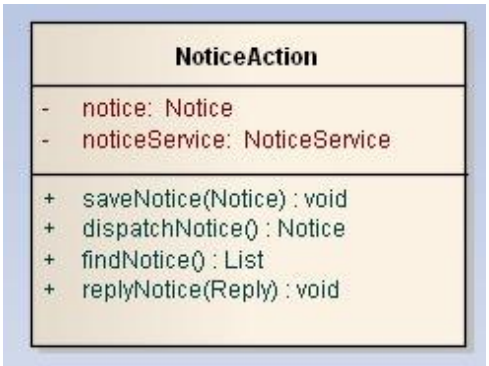


数据模型建模及其 UML 表达（类图）

类图（Class Diagram）：类图是面向对象系统建模中最常用和最重要的图，是定义其它图的基础。类图主要是用来显示系统中的类、接口以及它们之间的静态结构和关系的一种静态模型。

类图的 3 个基本组件：类名、属性、方法。

（+：表示 public； -：表示 private； #：表示 protected（friendly 也归入这类））



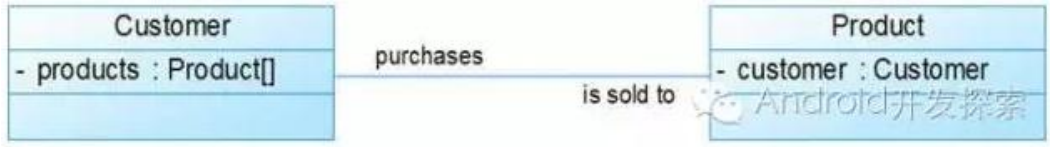
类与类之间关系的表示方式

1、 关联关系（单向关联、双向关联和自关联）

（1）单向关联

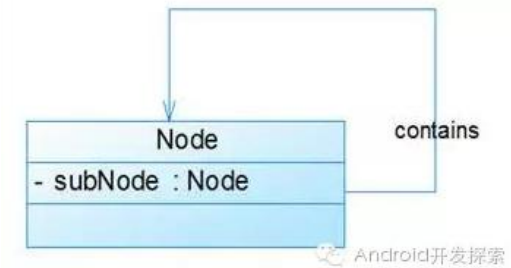
我们可以看到，在 UML 类图中单向关联用一个带箭头的直线表示。上图表示每个顾客都有一个地址，这通过让 Customer 类持有一个类型为 Address 的成员变量类实现。

（2）双向关联



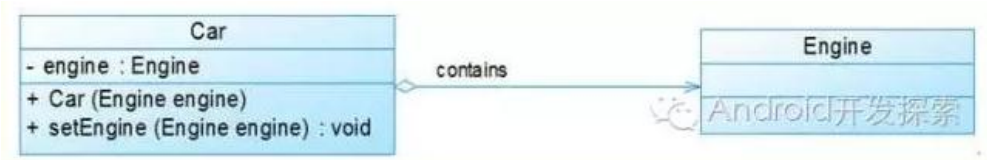
从上图中我们很容易看出，所谓的双向关联就是双方各自持有对方类型的成员变量。在 UML 类图中，双向关联用一个不带箭头的直线表示。上图中在 Customer 类中维护一个 Product[]数组，表示一个顾客购买了那些产品；在 Product 类中维护一个 Customer 类型的成员变量表示这个产品被哪个顾客所购买。

（3）自关联



自关联在 UML 类图中用一个带有箭头且指向自身的直线表示。上图的意思就是 Node 类包含类型为 Node 的成员变量，也就是“自己包含自己”。

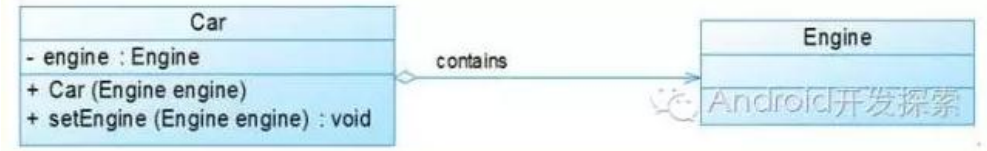
2、聚合关系



上图中的 Car 类与 Engine 类就是聚合关系（Car 类中包含一个 Engine 类型的成员变量）。由上图我们可以看到，UML 中聚合关系用带空心菱形和箭头的直线表示。聚合关系强调是“整体”包含“部分”，但是“部分”可以脱离“整体”而单独存在。比如上图中汽车包含了发动机，而发动机脱离了汽车也能单独存在。

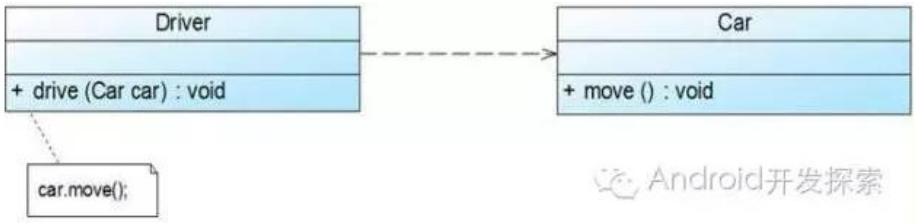
3、组合关系

组合关系与聚合关系见得最大不同在于：这里的“部分”脱离了“整体”便不复存在。比如下图：



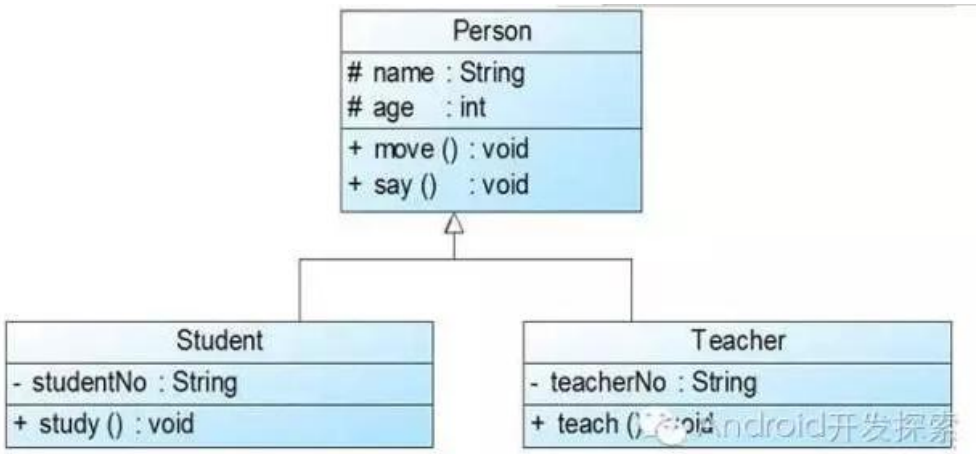
显然，嘴是头的一部分且不能脱离了头而单独存在。在 UML 类图中，组合关系用一个带实心菱形和箭头的直线表示。

4、依赖关系



从上图我们可以看到，Driver 的 drive 方法只有传入了一个 Car 对象才能发挥作用，因此我们说 Driver 类依赖于 Car 类。在 UML 类图中，依赖关系用一条带有箭头的虚线表示。

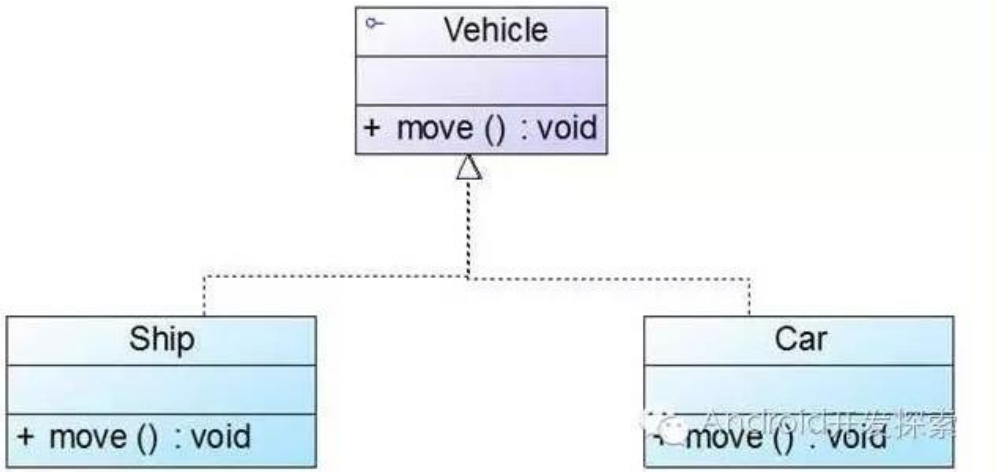
5、继承关系



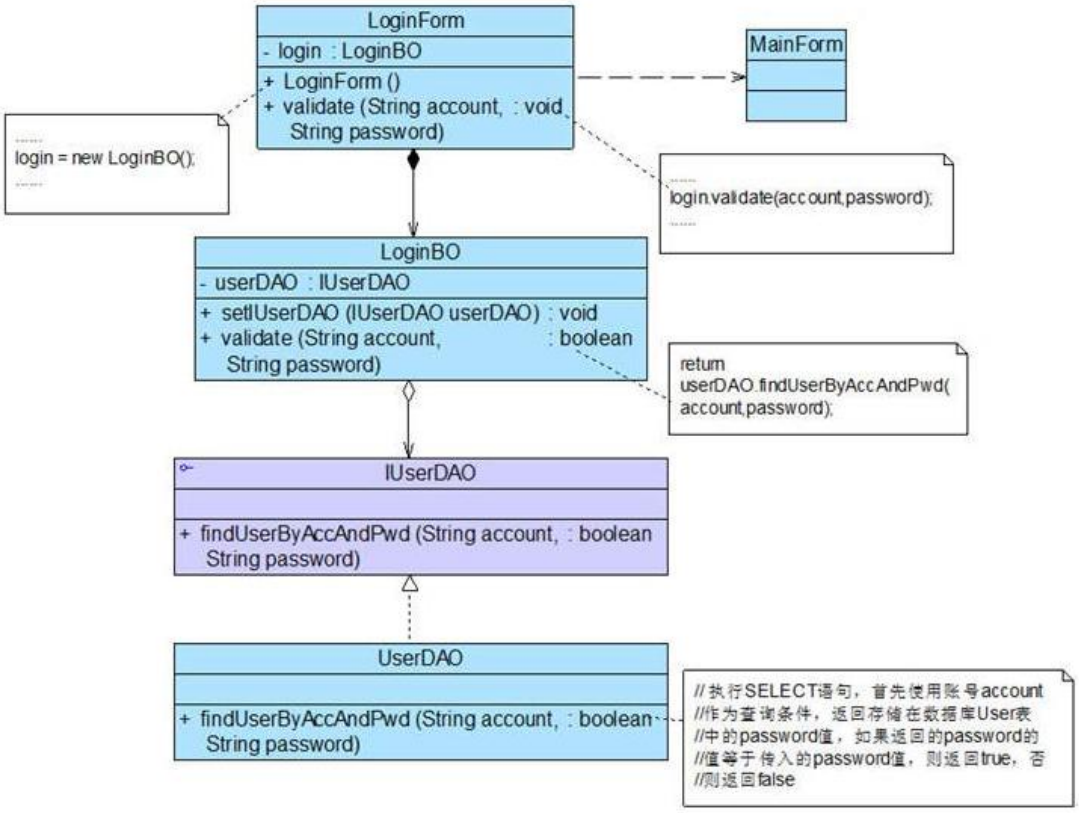
继承关系对应的是 extend 关键字，在 UML 类图中用带空心三角形的直线表示，如下图所示中，Student 类与 Teacher 类继承了 Person 类。

6、接口实现关系

这种关系对应 implement 关键字，在 UML 类图中用带空心三角形的虚线表示。如下图中，Car 类与 Ship 类都实现了 Vehicle 接口。

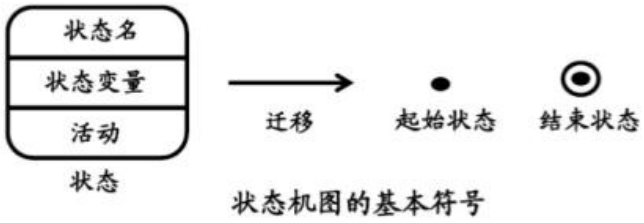


【例】用户通过登录界面(LoginForm)输入账号和密码，系统将输入的账号和密码与存储在数据库(User)表中的用户信息进行比较，验证用户输入是否正确，如果输入正确则进入主界面(MainForm)，否则提示“输入错误”。



行为模型建模及其 UML 表达（状态机图）

利用状态机可以精确地描述对象的行为。从对象的初始状态起，开始响应事件并执行某些动作，这些事件引起状态的转换；对象在新状态下又开始响应事件和执行动作，如此连续进行直到终结状态。



状态机图的基本符号

【例】请假条

UMLonline