OO的基本思想：

1、从现实世界客观存在的事物出发来构造软件系统，并在系统构造中尽可能运用人类的思维方式。

2、范畴已超出了目前对OO的定义。

OO的基本原理和三大要素：

OO直接反映了人们对客观世界的认知模式——人类认识客观世界有两个基本过程，一个是从特殊到一般的归纳过程，另一个是从一般到特殊的演绎过程。

面向对象=对象+对象类+类继承

对象———数据抽象

对象类———抽象数据类型

类继承———类型继承

对象包括：

problem domain object：现实世界中个体的数据抽象模型；是事物状态和行为的数据抽象，既是事物状态的集合，也是为改变状态而施加的操作方法或算法程序的集合。对象将两者封装在一起，既可以存放状态也可以因受外部作用而改变状态。

resolve domain object：是系统中的基本运行实体，是具有特殊属性（数据）和行为方式（方法或称服务）的实体。从动态的观点看，对象的行为即发生在对象上的操作。从静态的观点看，对象的属性反映了对象的状态。

对象涉及基本的分类规则：

分类（聚类）技术目前广泛应用，采用的方法有：爬山法、截面法、C5、遗传算法、神经网络、模糊数学、决策树、时间序列、滑动窗口等。

OOA的主要原则

抽象

过程抽象和数据抽象

分类

封装

继承

聚合

关联

消息通信

粒度控制

行为分析

封装(encapsulation): 在面向对象的系统中，我们将一段信息与对该信息进行操作的特定行为相结合。然后，我们将这些打包到一个对象中，这称为封装。

另一种查看封装的方法是，我们将应用程序划分为相关功能的一小部分。

封装的一个好处是，它限制了系统更改的影响。

In object-oriented systems, we combine a piece of information with the specific behavior that acts upon that information. Then we package these into an object. This is referred to as encapsulation.

Another way to look at encapsulation is that we divide the application into small parts of related functionality.

Another benefit of encapsulation is that it limits the effects of changes to the system.

继承(inheritance):

在面向对象的系统中，继承是一种允许基于旧对象创建新对象的机制：子对象继承父对象的品质。

继承的好处是易于维护。

In object-oriented systems, inheritance is a mechanism that lets you create new objects based on old ones: The child object inherits the qualities of a parent object.

多态(polymorphism):

字典定义了多态性作为不同形式、阶段或类型的发生。多态性意味着具有特定功能的许多形式或实现。

就面向对象的系统而言，这意味着我们可以具有特定功能的许多实现。

多态性的好处之一是易于维护。

The dictionary defines polymorphism. as the occurrence of different forms, stages, or types. Polymorphism means having many forms or implementations of a particular functionality.

In terms of an object-oriented system, this means that we can have many implementations of a particular functionality.

One of the benefits of polymorphism, as with the other principles of object orientation, is ease of maintenance.

消息(message): 是用来请求对象执行某一处理或回答某些信息的要求。是对象的相互作用和相互联系。消息统一了数据流和控制流。程序的执行是通过对象间传递消息来实现的。它非常类似于传统的程序设计语言中子程序之间通过参数的方式调用。

消息的形式用消息模式来描述。

消息是从发送者到接受者的一条执行服务的请求。

UML 的主要设计目标如下：

• 为用户提供一种随时可用、富有表现力的视觉建模语言，以开发和交换有意义的模型。

• 提供可扩展性和专业化机制，以扩展核心概念。

• 独立于特定的编程语言和开发过程。

• 为理解建模语言提供正式基础。

* 鼓励对象工具市场的增长。

• 支持更高级别的发展概念，如协作、框架、模式和组件。

• 整合最佳实践。

可视化建模(visual modeling): 可视化建模是利用一组标准图形元素从模型中收集信息并以图形方式显示信息的过程。

标准对于实现可视化建模的好处之一至关重要：通信。用户、开发人员、分析师、测试人员、经理和参与项目的其他人之间的通信是可视化建模的主要目的。可视化模型为向所有相关方展示建议的系统提供了强大的工具。

对象管理技术(object management technology):

OMT包括三个模型：对象模型、动态模型和功能模型。

对象模型描述系统的静态结构，包括：

对象、类、

属性

行为

协会

关系

聚集

继承。

动态模型:描述系统的变化与时间。

状态;

事件;

状态图;

功能模型描述系统内的计算、功能，包括：

过程;

数据流;

角色;

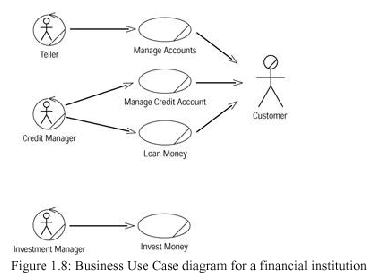
数据仓库;

控制流;

数据流图。

业务用例图(business use case diagrams):

业务用例图用于表示整个组织提供的功能。它们在业务建模活动期间被广泛使用，用于为系统设置上下文，并形成创建用例的基础。



用例图(use case diagrams):

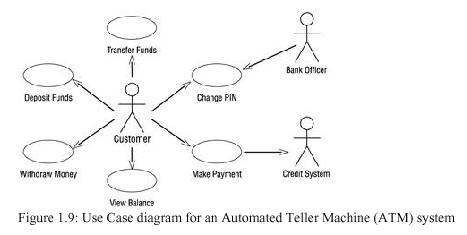
用例图显示用例和参与者之间的交互。

用例表示系统功能，从用户的角度表示系统的要求。

参与者代表提供或接收系统信息的人或系统;他们是系统的利益相关者之一。

因此，用例图显示哪些参与者启动用例;它们还说明参与者从用例中接收信息。

从本质上讲，用例图可以说明系统的要求。



虽然业务用例图与自动化(automated processes)关系不相关，但用例图只关注自动化流程。业务用例和用例之间没有一对一的关系

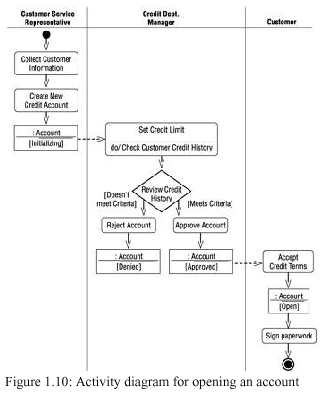
活动图(activity diagrams):

活动图说明了系统中的功能流。

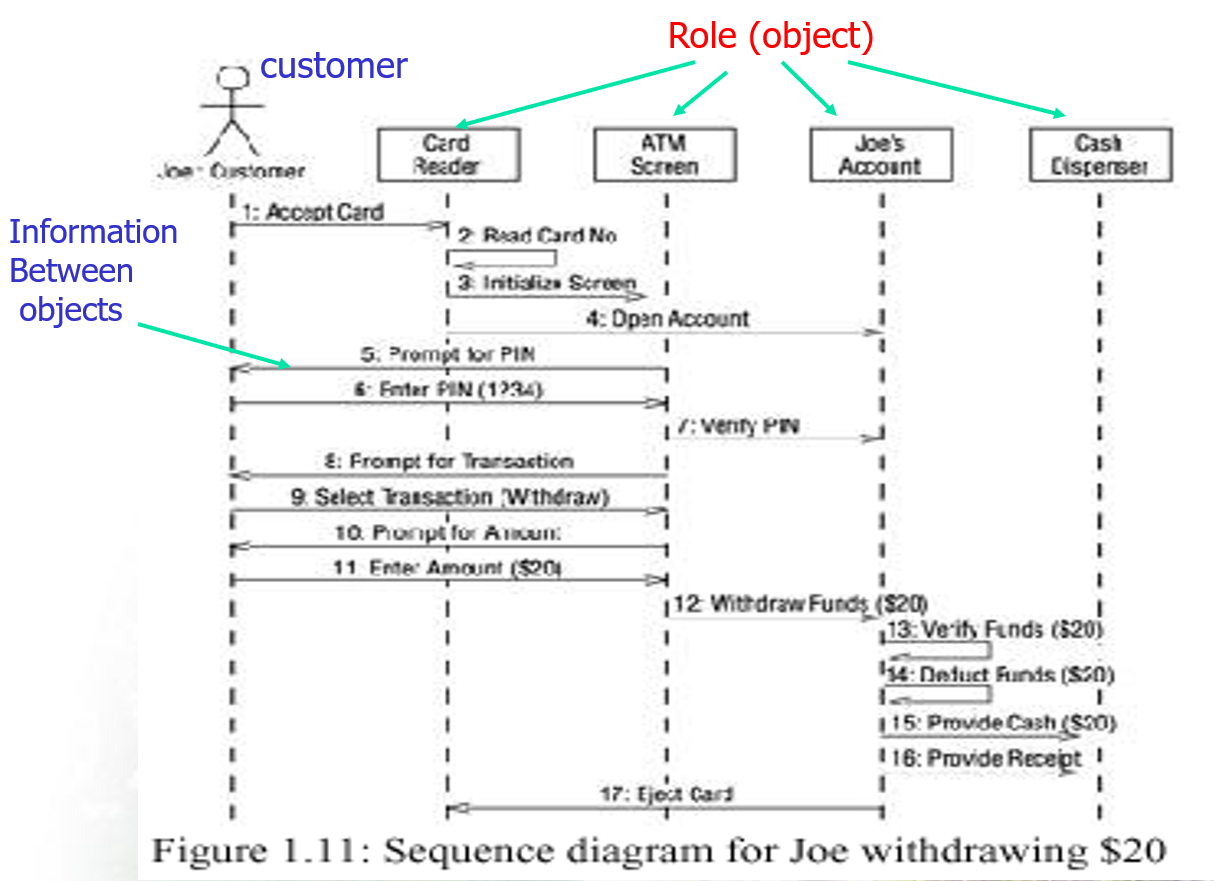
它们可用于业务建模以显示业务工作流。它们可用于需求收集，以说明事件通过用例的流。

这些关系图定义工作流的开始位置、结束位置、工作流期间发生的活动以及活动发生的顺序。

活动是在工作流期间执行的任务。



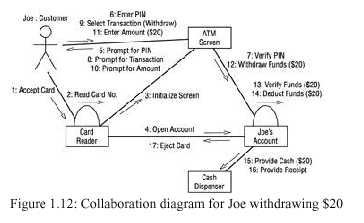
序列图(sequence diagrams): 序列图通过用例显示功能流。



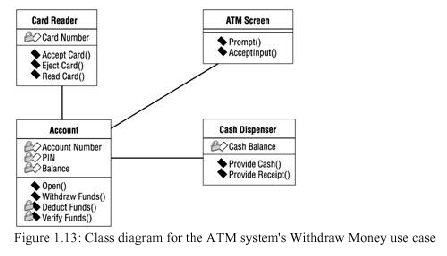
协作图(collaboration diagram):

协作图显示的信息与序列图完全相同。但是，协作图以不同的方式和不同的用途显示此信息。

序列图说明了对象和参与者在一段时间中的交互，而协作图则显示对象和参与者交互，而不引用时间。



类图(class diagrams): 类图显示系统中类之间的交互，类可视为对象的蓝图。

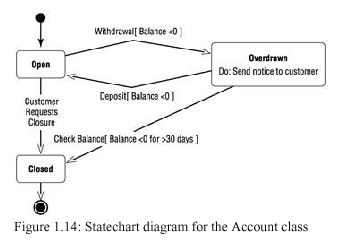


状态图(statechart diagrams):

状态图提供了一种对对象可能存在的各种状态建模的方法。虽然类图显示类及其关系的静态图片，但状态图用于对系统的更动态的行为进行建模。

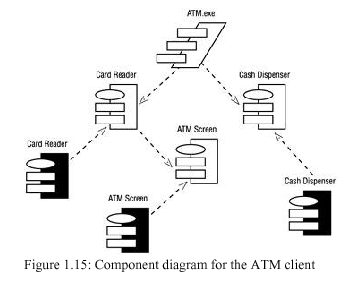
这些类型的图表在构建实时系统时被广泛使用。Rose 甚至可以从状态图生成实时系统的完整代码。

状态图显示对象的行为。



状态图不是为每个类创建的，而仅用于非常复杂的类。如果类的对象可以存在于多个状态中，并且在每个状态中的行为非常不同，则可能需要为它创建状态图。

组件图(component diagrams): 组件图显示模型的物理视图，以及系统中的软件组件及其之间的关系。组件图上有两种类型的组件：可执行组件和代码库。

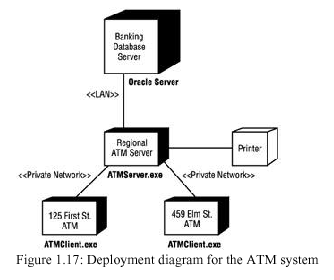


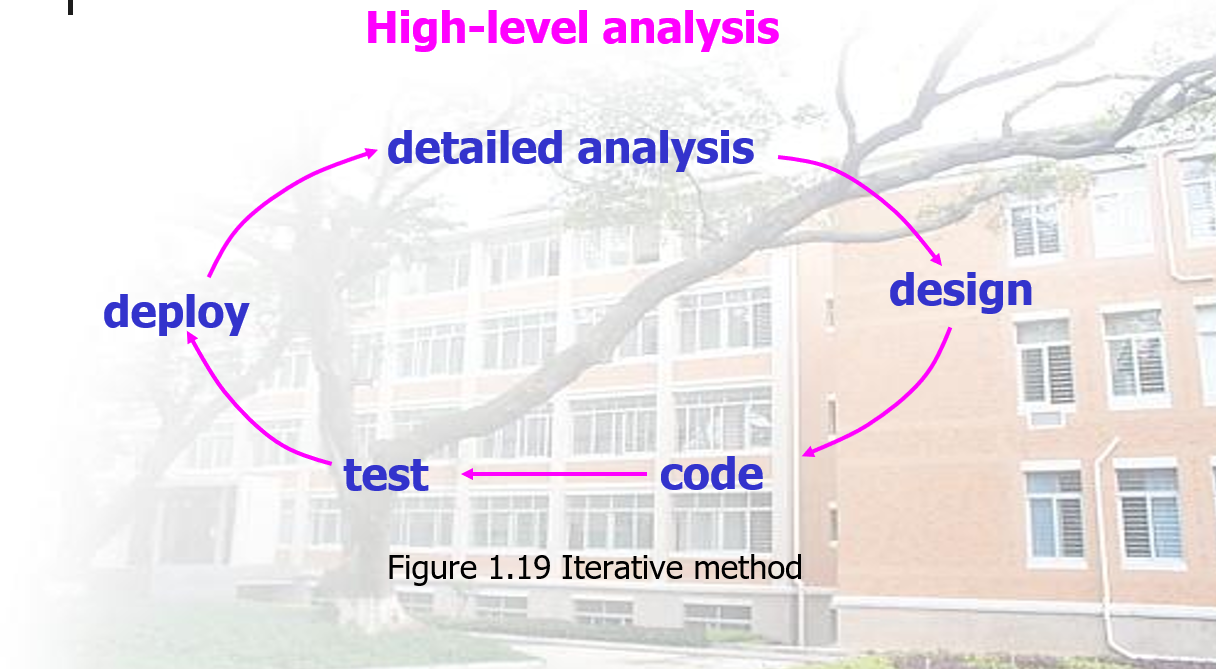
每个类都有自己的标头和正文文件，因此每个类都映射到图中自己的组件。

着色组件称为包体。无阴影组件称为包规范。

组件通过显示它们之间的依赖关系的虚线进行连接。

部署图(deployment diagrams): 部署图显示网络的物理布局以及各种组件将驻留的位置。





RUP(Rational Unified Process)统一软件开发过程实现了迭代生命周期，分为四个阶段：

* 1. 初始阶段（inception）-定义项目范围（scope）
  2. 细化阶段（elaboration）计划项目，指定（specify）特征和基线架构
  3. 建造阶段（construction）建造产品
  4. 过渡阶段（transition）将产品过渡到终端用户社区

定义 UML 的动机（我们建模的原因）

开发模型与拥有大型建筑蓝图一样重要。良好的模型对于项目团队之间的沟通以及确保建筑的健全性至关重要。

我们构建复杂系统的模型，因为我们无法理解任何此类系统的全部。随着系统复杂性的增加，良好的建模技术的重要性也日益增加。

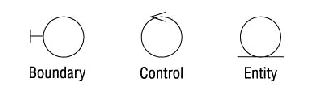
逻辑视图(logical view): 逻辑视图侧重于系统如何在用例中实现行为。它提供了系统部件的详细图片，并描述了这些部件的相互关系。逻辑视图包括所需的特定类、类关系图和状态图。

有了这些详细的元素，开发人员可以构建系统的详细设计。

通常，团队对逻辑视图采用两种方法。

在第一种方法中，他们识别分析类。分析类是独立于语言的类。通过首先专注于分析类，团队可以开始查看系统的结构，而不会陷入具体实现的细节。

分析类(analysis class):用以下符号表示



但是，当我们将其转换为设计类(design class)时，我们会查看特定于语言的详细信息。