# 软件和软件工程

## 什么是软件？

**软件是一系列按照特定顺序组织的计算机数据和指令的集合，包括程序、数据和文档。**其中，程序是按事先设计的功能和性能要求执行的指令序列，数据是使程序能正常操纵信息的数据结构，文档是与程序开发，维护和使用有关的图文材料

## 什么是软件危机，其内容主要是指什么？

**软件危机是计算机软件在它的开发和维护过程中所遇到的一系列严重问题。**概括地说，主要包含两方面的问题：如何开发软件，以满足对软件日益增长的需求；如何维护数量不断膨胀的已有软件。

## 什么是软件工程？

软件工程是：1.将系统化的、严格约束的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件；2.在1中所述方法的研究。该学科的目的是生产出能如期交付的、在预算范围内的、满足用户需求的、质量合格的软件产品。

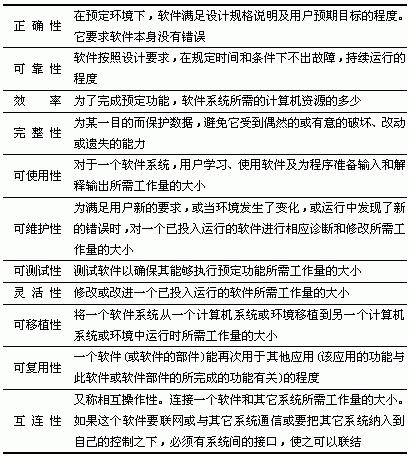
## 软件工程的目标及其组成部分。

软件工程的目标是：在给定成本、进度的前提下，开发出具有适用性、有效性、可修改性、可靠性、可理解性、可维护性、可重用性、可移植性、可追踪性、可互操作性和满足用户需求的软件产品。

组成部分：方法、工具和过程。

## 好的软件的一些主要衡量指标。

产品的质量：（McCall 的质量模型）



## 了解过程成熟度模型CMM。

Level 1 **初始级**：描述一个特别的甚至是混乱的软件开发过程

Level 2 可重复级：标识过程的输入和输出、约束以及用于生产最终产品的资源

Level 3 定义级：管理和工程活动是文档化的、标准化的和集成的

Level 4 管理级：过程的重点是提高产品质量

Level 5 优化级：在过程中融入了量化的反馈以得到持续的过程改进

## 小结

* 软件=程序+数据+文档
* 软件危机: 原因,现象,办法(软件工程学)
* 软件工程(学):**开发、运行和维护**软件**的系统方法**
* 软件工程3个要素：方法、工具和过程。

# 软件过程

## 软件过程的定义

一组有序的任务，包括活动、约束和资源使用的一系列步骤，用于产生想要的输出

## 什么是软件生命周期？主要分为哪些阶段？各个阶段的主要任务及产生的主要制品？

软件生命周期：描述软件产品从概念到实现、交付、使用和维护的整个过程的集合，即软件开发过程。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 阶段 | 任务 | 阶段性制品 |
| 软件定义 | 问题定义 | 确定软件开发过程的总目标 |  |
| 可行性研究 | 了解用户要求和现实环境，从技术、经济、市场等方面研究并论证开发该软件系统的可行性 | 可行性论证报告  初步的项目开发计划 |
| 需求分析 | 为了解决这个问题，目标系统必须做什么 | 软件需求规格说明书SRS(功能，性能和运行环境约束) |
| 软件开发 | 总体设计 | 根据SRS建立目标软件系统总体结构、设计全局数据库和数据结构，规定设计约束，制定集成测试计划等等。 | 概要设计规格说明书  数据库或数据结构设计说明书  集成测试计划。 |
| 详细设计 | 细化概要设计所生成的各个模块, 并详细描述程序模块的内部细节(算法，数据结构等)，形成可编程的程序模块，制订单元测试计划 | 详细设计规格说明书  单元测试计划 |
| 编码和  单元测试 | 写出正确的容易理解、容易维护的程序模块 | 源程序代码 |
| 集成测试 | 根据概要设计规格说明书，将经过单元测试的模块逐步进行集成和测试 | 生成满足概要设计要求、可运行的系统源程序和系统集成测试报告 |
| 确认测试 | 根据软件需求规格说明书，测试软件系统是否满足用户的需求 | 可供用户使用的软件产品(文档，源程序) |
| 软件维护 | | 通过各种必要的维护活动使系统持久地满足用户的需要 | 版本更新的软件产品 |

## 软件可行性研究的内容。

技术可行性，经济可行性，运行可行性，法律可行性

## 需求分析的定义

确定用户对待开发软件系统的功能、性能、运行环境约束几方面的需求，并用需求规格说明书把客户需求准确的记录下来。

## 典型的软件开发过程模型的特点（优缺点）及要求，特别是原型法、瀑布模型、增量和迭代等。

|  |  |  |  |
| --- | --- | --- | --- |
| 过程模型 | 优点 | 缺点 | 要求 |
| 瀑布模型 | 采用规范的方法；  严格规定每个阶段提交的文档；  要求每个阶段交出的产品必须经过验证。 | 在交付之前客户只能通过文档来了解产品；  开发人员和客户之间缺乏有效沟通，很可能到时产品不满足用户需求 |  |
| 原型开发 | 开发出来的产品通常能满足用户的真实需求；  产品的开发过程基本上是线性过程  允许需求或设计反复调查  减少开发中的风险和不确定性 | 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。  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。  开发过程不便于管理。 |  |
| 螺旋模型 | 有利于已有软件的重用，也有助于把软件质量作为软件开发的一个重要目标  减少了过多测试或测试不足  维护和开发之间并没有本质区别 |  | 风险驱动，需要相当丰富的风险评估经验和专门知识  主要适用于内部开发的大规模软件项目 |
| 增量模型 | 能在较短的时间内向用户提交可完成部分工作的产品  逐步增加产品功能，从而使用户有较充裕的时间学习和适应新产品 |  | 软件工程师必须具有较高的技术水平，能够设计出开放的软件体系结构 |
| 迭代模型 |  |  |  |

## 原型法的分类

抛弃式原型、进化型。

## 敏捷方法的原则是什么？有哪些典型的方法？

原则：相对于过程和工具，更强调个人和交互的价值

更喜欢在生产运行的软件上投入时间 ，而不是在文档的编写上

注重客户的合作，而不是合同谈判

专注于对变化的反应，而不是创建一个计划而后遵循这个计划

过程：极限编程（XP）

自适应软件开发（ASD）

scrum

动态系统开发方法（DSDM）

crystal

特征驱动开发（FDD）

精益软件开发（LSD）

敏捷建模（AM）

敏捷统一过程（AUP）

## 极限编程的特点

交流: 保持客户和开发者的交换看法

简单性: 选择简单设计和实现

勇气: 尽早并经常性交付功能(敢于承诺并信守诺言)

反馈:开发过程中各种活动循环

## 常见的软件开发方法

1）结构化方法

2) 面向数据结构的软件开发方法

3)面向对象软件开发方法

**小结**

* 1）软件开发过程的含义。
* 2）软件生命周期中各个阶段的划分，每个阶段完成一些确定的任务，各个阶段互相衔接完成目标系统的开发和维护；
* 3）各种常见的软件生命周期模型的特点及优缺点。
* 4）常见的软件开发方法。

# 需求分析

## 了解需求的重要性及需求分析阶段的主要产物。

任务:确定用户对待开发软件系统的需求包括：功能,性能,运行环境约束

阶段性产品:软件需求规格说明书SRS(功能，性能和运行环境约束)

## 需求的类型：

功能需求、非功能需求或质量需求、设计约束、过程约束。

## 两种需求文档

需求定义文档和需求规格说明书。

需求定义: 用户想要得到的每一件事情的完整列表，描述打算构建的系统将要安装的环境中的实体

需求规格说明: 将需求重新陈述为关于要构建的系统将如何运转的规格说明

## 需求规格说明书的主要内容。

详细描述输入和输出

根据接口的输入输出重新陈述要求的功能

对用户的质量需求，设计适配标准

## 常用的需求建模表示方法：

ER图、事件跟踪、状态机、Petri网、数据流图、用例图和原型法。

### ER图

一种表示概念模型的流行图形表示法

三个核心结构

实体: 表示为矩形，代表具有共同性质和行为的现实世界对象构成的集合

关系: 表示为两个实体之间的边，边中间有一个菱形，表示关系的类型

属性: 是实体的注释，描述实体相关的数据或性质

统一建模语言UML (Unified Modeling Language) 是用于文档化软件规格说明和设计的一组表示法

用下列术语表示一个系统

对象: 类似于实体，按照具有的继承层次的类进行组织

方法: 在对象的变量上执行动作

类图是“旗舰型”的模型

是与规格说明中的类相关的高级ER图

### 事件跟踪

关于现实世界实体之间交换的时间序列的图形描述

垂直线: 不同实体的时间线，其名字出现在线的顶部

水平线: 两个实体之间的一个事件或交互

时间按从顶到下跟踪进展

每一个图描述一个跟踪，表示只是若干个可能行为中的一个

事件跟踪语义相对简单，易于理解

### 状态机

是一种图形描述，描述了系统与其环境之间的所有对话

点(状态) 表示存在于事件发生之间的一个稳定的条件集合

边(转移) 表示由于一个事件的发生而产生的行为或条件的变化

在表示动态行为方面，以及在描述在响应已经发生的历史事件时行为将如何变化方面很有用

路径: 起始于状态机的初始状态，沿着从状态到状态的转移 , 表示了环境中的一条可观察事件的跟踪

确定的状态机: 对每一个状态和事件都有一个唯一的响应

### Petri网

Petri 网是状态-转移表示法的一种形式，用于建模并发活动以及他们之间的交互。

圆圈：位置条：变迁弧：箭头点：令牌

### 数据流图

ER 图，事件跟踪和状态机描述的仅仅是转移为一组更低层的行为，数据流图 (DFD) 建模功能以及从一个功能到另一个功能数据流

一个泡泡表示：一个加工

箭头表示：数据流

平行线：数据存储: 正式的库或信息库

矩形：表示参与者: 提供输入数据或接受输出的实体

### 用例图

模型元素：

系统边界：一个提供“用例”所需要的功能的“黑盒子”。系统的外部特性由系统的功能来定义；整个系统的功能用一组用例来描述。

执行者：需要使用系统的任何外部实体(例如：人、其它系统或外部设备等)。

用例：用客户或用户的语言和词汇来描述的系统的一个完整功能。

用例图描述系统的边界，和其上的动态行为，图中包括：用例（use

case），系统作用者（actor）及其之间的(关联)关系。用例图是用例视图的

重要组成部分。

## 原型化需求

原型法是指在获取一组基本的需求定义后，利用高级软件工具可视化的开发环境，快速地建立一个目标系统的最初版本，并把它交给用户试用、补充和修改，再进行新的版本开发

（1）优点：符合人们认识事物的规律，系统开发循序渐进，反复修改，确保较好的用户满意度；开发周期短，费用相对少；由于有用户的直接参与，系统更加贴近实际；易学易用，减少用户的培训时间；应变能力强。

（2）缺点：不适合大规模系统的开发；开发过程管理要求高，整个开发过程要经过“修改—评价—再修改”的多次反复；用户过早看到系统原型，误认为系统就是就是这个模样，易使用户失去信心；开发人员易将原型取代系统分析；缺乏规范化的文档资料

# 概要设计

## 好的设计的衡量

高内聚和低耦合。

## 常用的内聚和耦合度类型。

耦合度（两个构件之间依赖关系的多少）

1. 内容耦合（干预内部数据）
2. 公共耦合（共享公共数据）
3. 控制耦合
4. 标记耦合
5. 数据耦合

内聚度（单个构件内元素的必要性与目的性）

* + - 1. 巧合内聚
      2. 逻辑性内聚
      3. 时间性内聚
      4. 过程内聚
      5. 通信内聚
      6. 顺序内聚
      7. 功能内聚

# 详细设计

## OOM中的典型特征，其中特别是封装、继承和多态。

OO（object orientation）面向对象是一种软件开发方法，它将问题和问题的解决方案组织为离散对象的集合，数据结构和行为都包含在对象的表示中，可以通过 OO 的 7 种特性来辨别 OO 表示：标识、抽象、分类、封装、继承、多态、持久性。

1）标识：指将数据组织为离散的、可区别的实体，称为对象。

2）抽象：抽象有助于表示正在开发的系统中的丌同观点。全起来，这些抽象开成一个

层次，以说明不同的系统观点是如何彼此相关的。

3）分类：将具在共同属性的行为的对象分组。

4）封装：类封装对象的行为和属性，隐藏实现细节。

5）继承：根据类之间的相同性戒差异性来组织类层次。

6）多态：同样行为在不同的类或子类中会有不同的表现的性质。

7）持久性：即对象的名称、状态和行为超越时间或空间的能力。

## 类和对象的概念及其构成。

对象：面向对象系统的运行时结构是一系列对象的集合，每个对象都是数据以及用来创建、读取、更改和清除数据的所有操作的内聚的集合。对象的数据称作属性，对象的操作称作方法。

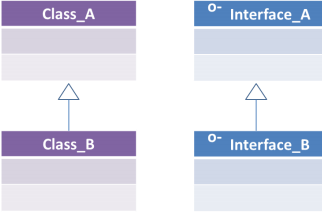
类：类是一部分或完全实现某抽象数据类型的软件模块，它包含了属性数据的定义、操作这些数据的方法的声明，以及部分或所有方法的实现

## 类图的主要作用。

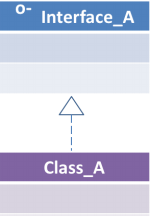
类图是描述类、接口以及它们之间关系的图，它显示了系统中各个类的静态结构，是一种静态模型。

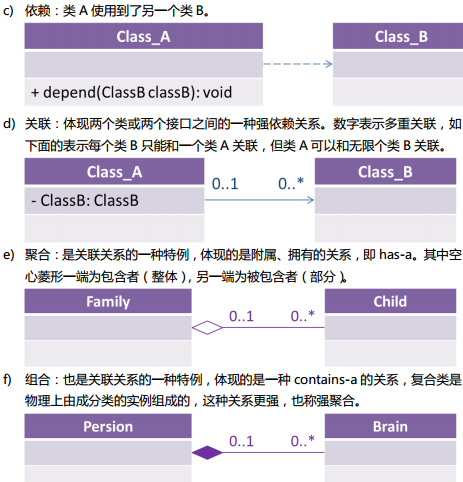
## 了解类之间的各种关系：关联、依赖、继承或泛化、组合/聚合等。

a) 继承：一个类继承另一个类的功能。



b) 实现：一个类实现接口的功能。





## 了解类图的基本建模步骤。

（1）主要分析类

（2）辅助类

（3）确定属性或类

（4）定义属性

（5）定义操作

（6）画出类图

（7）画出类之间的关系

## 接口和抽象类的定义及各自的特点。

**抽象类**是指那些不具有任何对象的类，其作用是为其他的类描述它们的公共属性和行为。（例如Vehicle）通常，抽象类具有一组抽象操作。一个拥有至少一个抽象操作的类必定是一个抽象类。由abstract修饰的方法叫抽象方法；由abstract修饰的类叫抽象类，抽象方法必须声明在抽象类中

**接口**是一组没有实现的操作的集合。接口只提供操作的声明，不提供任何相应的功能代码。具体的功能代码由使用该接口的类实现，这叫作实现关系。

**一个类和一个接口不同**：一个类可以有它形态的真实实例，然而一个接口必须至少有一个类来实现它。抽象类允许增加一些方法的实现。

## 交互图的分类：顺序图和协作图。这两种图形各自的优缺点。注意UML 2.0中协作图改称通信图。

* + 序列图（顺序图）主要用来描述对象之间信息交换时的时间顺序.
  + 而协作图则用来描述系统对象之间如何协作共同完成系统功能的要求。
  + 序列图强调消息的时间顺序，协作图强调参与交互的对象的组织关系
  + 序列图和协作图在语义上是等价的，两者可以相互转换。

**优缺点：**

序列图突出对象的执行时序，协作图能更清楚地表示对象之间的静态连接关系。

交互图擅长显示对象之间的合作关系，尽管它并不对这些对象的行为进行精确定义。如要描述一个用例中几个对象协同工作的行为时，交互图是一种有力的工具。

虽然交互图能清楚地显示消息机制，但当消息中有太多的条件或循环时，交互图就失去其简明性。交互图仅适用于条件判断和循环不太多的时序过程。

## 状态图和活动图各自的作用。注意活动图中泳道的作用。

**状态图**描述了一个对象或交互过程在它的生命周期中对一系列外界激励的所呈现出的不同状态以及它相应的响应和活动。

**活动图**是一种特殊形式的状态机，用于对计算流程和工作流程建模. 描述对象（或模型元素）的活动的顺序关系所遵循的规则，着重表现的是系统的行为，而不是系统的处理过程

活动图和状态机图的不同之处:

* + 状态机图强调的是在外部事件的驱动下，软件对象的控制在不同的状态之间的流动；
  + 而活动图强调的是在完成变迁引导下，对象的控制在活动之间的流动。

**泳道**是活动图里对其中的活动按照其职责上的关联进行的划分，区分了活动图中的活动的不同职责。泳道在活动图内是一系列的垂直的隔断，这也是泳道这个名字的由来。

在活动图里，泳道区分了其中的活动

## 组件图的作用以及组件与接口间的关系。

组件是系统的一个物理的和可替代的组成部分，该组成部分遵循并实现了一组给定的接口。

组件图是反映代码的物理结构

**组件和接口之间的联系的描述：**

**实现关系**

一个组件实现了一个接口

输出接口：被一个组件实现的接口是该组件的实现接口

**依赖关系**

一个组件使用了另一个组件通过接口提供的服务

输入接口：被一个组件调用的接口是该组件的输入接口

## 部署图的作用以及节点的分类。

**部署视图**(deployment view)用来描述软件产品在计算机硬件系统和网络上的安装、分发（delivery ）、分布（distribution ）

**结点的分类：**处理器与设备

**处理器**是具有数据处理能力的节点，意味着它能执行组件内包含的程序

**设备**是在当前所处的抽象级别上不具备数据处理能力的节点

## 主要的面向对象设计原则及各自的原理：OCP, LSP, DIP, ISP, CARP, LoD.

* **LSP：Liskov替换原则**

任何父类可以出现的地方,子类都可以出现**.**

* **OCP：开放-封闭原则**

软件实体（类、模块、函数等）应该是可扩展的，但是不可修改的，关键在于抽象

* **SRP：单一职责原则**

就一个类而言，应该仅有一个引起它变化的原因

* **ISP：接口隔离原则**

客户不应该依赖他们不用到的方法，只给每个客户它所需要的接口

* **DIP：依赖倒置原则**

高层模块不应该依赖于低层模块。二者都应该依赖于抽象

抽象不应该依赖于细节。细节应该依赖于抽象

针对接口编程，不要针对实现编程

* **FCOI(或CRP)：组合复用原则**

优先使用（对象）组合，而非（类）继承

* **LoD：迪米特原则**

一个软件实体应该尽量少与其他实体发生相互作用。

## LSP中的子类型与继承的关系及区别。

子类型是一种继承，但简单继承父类的方法和成员不改写，子类不能添加任何父类没有的附加约束

## 开闭原则的思想及关键。

OCP原则思想：试图设计出永远也不需要改变的模块

关键：抽象化

## 设计模式的分类。

* **创建型模式**

抽象的实例化过程

* **结构型模式**

如何组合类和对象以获得更大的结构

* **行为型模式**

涉及到算法和对象间职责的分配

模式的另一种分类：体系结构模式、设计模式、代码模式（习语）

## 设计模式与面向对象设计原则之间的关系，特别是OCP原则。

复用是目标，两种重要的复用手段：继承，组合/集成

接口和现实分离，接口保持不变，分离带来灵活性，多态性

解耦合，降低复杂性

## 掌握各种工厂模式的设计思想及其原理，了解如何从OCP的角度进行分析。

**简单工厂：又称静态工厂方法模式**

**简单工厂模式是有一个工厂类根据传入的参量决定创建出哪一种产品类的实例**

功能的扩展体现在引进新的产品上。“开–闭”原则要求系统允许当新的产品加入系统中，而无需对现有代码进行修改。这一点对于产品的消费角色是成立的，而对于工厂角色是不成立的。

**工厂方法：又称多态性工厂模式**

**工厂方法模式的用意是定义一个创建产品对象的工厂接口，将实际创建工作推迟到子类中。是简单工厂模式的进一步抽象和推广。**

可以允许系统在不修改具体工厂角色的情况下引进新的产品，这个系统完全支持“开-闭”原则。

**抽象工厂模式：又称工具箱模式（Toolkit)**

**是所有形态的工厂模式中最为抽象和最具一般性的一种形态。**

**抽象工厂模式可以向客户端提供一个接口，使得客户端在不必指定产品的具体类型的情况下，创建多个产品族中的产品对象。这就是抽象工厂模式的用意。**

工厂方法模式针对的是一个产品等级结构；而抽象工厂模式则需要面对多个产品等级结构

# 测试

## 技术评审（在建模编码阶段）

找出错误和发现可能对将要部署的软件产生负面影响的问题

## 验证与确认

验证：确保软件正确地实现某一特定功能的一系列活动

确认：确保开发的软件课追溯到客户需求的另一系列活动

## 测试的目标和衡量标准。

测试目标：发现错误。

衡量标准：只有当发现了错误时，测试才被认为是成功的。

## 测试的分类（或组织）。各种类型的测试的主要任务及所依赖的文档。

单元测试：将每个程序构件与系统中其他构件隔离，对其本身进行测试。文档：构件代码。

集成测试：验证系统构件是否能够按照系统和程序设计规格说明中描的那样共同工作。文档：设计规格说明。

功能测试：对系统进行评估，以确定集成的系统是否确实执行了需求规格说明书中描述的功能。文档：系统功能需求。

性能测试：将系统与这些软件和硬件需求的剩余部分进行比较。当测试在客户的实际工作环境中成功时，它会产生一个确认的系统。文档：其他软件需求。

验收测试：与客户一起执行验收测试，其中根据客户的需求描述对系统进行检查。文档：客户需求规格说明书。

安装测试：确保系统将按照它应该的方式来运行。文档：用户环境。

## 黑盒测试和白盒测试的思想，了解白盒测试中的基本路径测试等方法。

闭盒或黑盒: 测试对象的功能。如果从外部观察测试对象，将其看作一个不了解其内容的黑盒，那么，我们的测试就是向闭盒提供输入数据，并记录产生的输出。在这种情况下，测试的目标是确保每一种输入都被提交，并且观察到的输出与预期的输出相匹配。（程序控制结构）

开盒或白盒: 测试对象的结构。对于某些测试对象，测试小组不可能生成一组证明所有情况下功能正确的、有代表性的测试用例，为了克服这个难题，我们将测试对象看做一个开盒，然后可以根据测试对象的结构用不同的方式来进行调试。（确认功能需求）

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。

！计算环复杂度（课本P348）

## 单元测试的主要内容。

边界测试、错误处理测试、路径测试、局部数据结构测试、模块接口测试。

**检查代码**

* 代码走查
* 代码审查

**证明代码的正确性**

* 形式化证明技术
* 符号执行
* 自动定理证明

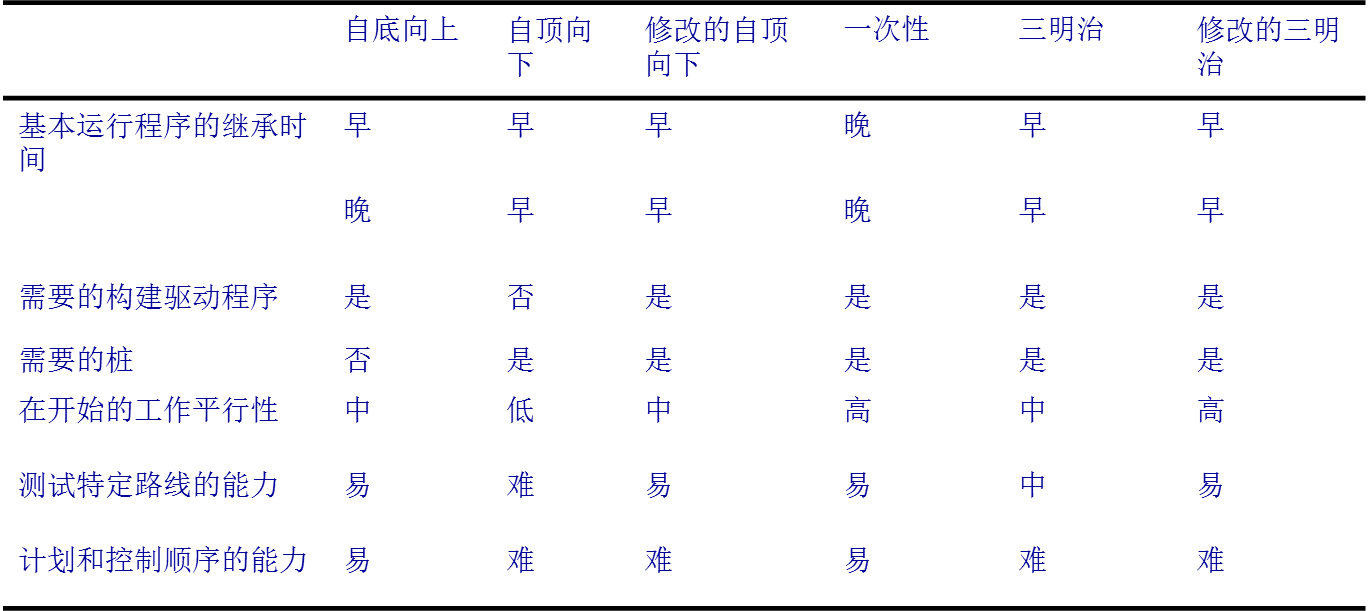
**测试程序构件**

* 测试与证明
* 选择测试用例
* 测试的完全性

**技术比较**

## 集成测试的类型及主要的测试策略。

* 自底向上的测试：主控模块—原子模块
* 自顶向下测试：原子—主控
* 一次性测试
* 三明治测试
* 改进的自顶向下测试
* 改进的三明治测试
* **策略比较**



## 确认测试的内容。

确认测试的目的是向未来的用户表明系统能够像预定要求那样工作。经[集成测试](http://baike.baidu.com/view/106652.htm)后，已经按照设计把所有的模块组装成一个完整的[软件系统](http://baike.baidu.com/view/8343.htm)，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是确认测试的任务，即软件的功能和性能如同用户所合理期待的那样。

α测试：软件开发公司组织内部人员模拟各类用户对即将面市软件产品（称为α版本）进行测试，试图发现错误并修正。在受控环境下进行

β测试：软件的多个用户在实际使用环境下进行的测试，这些用户返回有关错误信息给开发者。（又称客户验收测试）

## 了解测试计划的主要内容。

* 构建测试目标
* 设计测试用例
* 编写测试用例
* 测试测试用例
* 执行测试
* 评估测试结果

## 测试系统中的测试过程：功能测试、性能测试、验收（或确认）测试、安装测试，及它们的内容。

* 功能测试: 集成系统是否按照需求规格说明执行它的功能?
* 性能测试: 是否满足非功能需求? 用于检查响应速度、结果的精确性、数据的可访问性
* 验收测试: 系统是客户期望的吗？让客户和用户能够确定我们构建的系统满足了他们的期望
* 安装测试: 系统能在客户端运行吗 ? 配置系统，将正确的数量和种类的设备连接到主处理器上，与其他系统建立通信

# 项目管理

## 了解项目计划和管理的主要内容和常用的方法。

跟踪项目进展

项目人员和组织

工作量和进度估计

风险管理

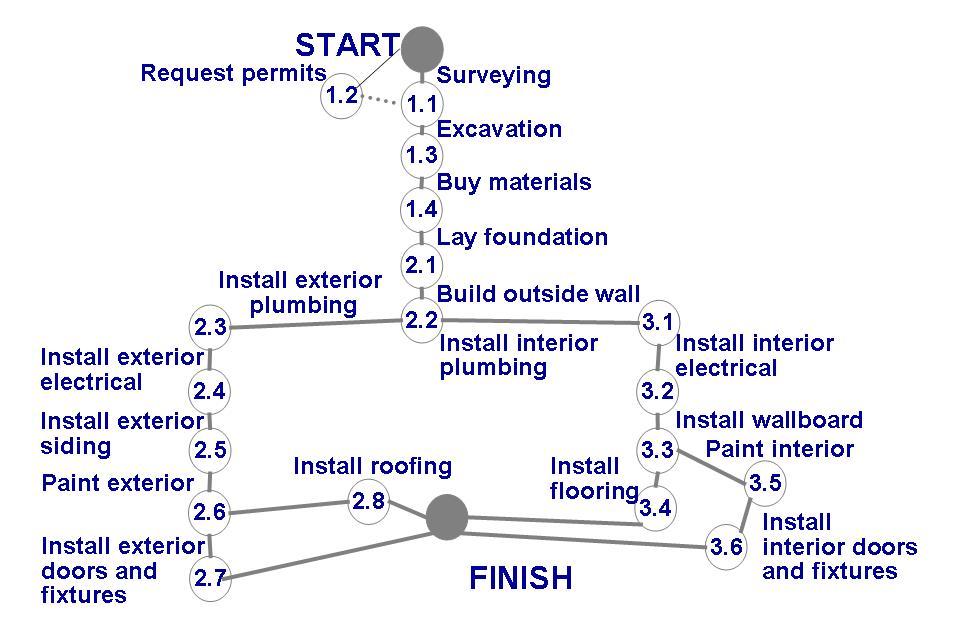
项目计划与过程建模

### WBS工作分解结构和活动图

工作分解结构把项目描述为由若干离散部分构成的集合，活动图描述了活动的独立性

*点 (圈)* : 项目里程碑

*线（框）*: 包含的活动



### 关键路径法 (CPM)

将描述完成项目花费的最短时间

揭示出按时完成这个项目最为关键的那些活动

**真实时间或实际时间:估算的完成此活动的必需时间**

**可用时间:** 完成活动可用的时间量

**时差:** 可用时间和实际时间的差

**关键路径**: 每个节点的时差都是零

可以有多条

**时差** = 可用时间 – 实际时间= 最晚开始时间 – 最早开始时间

两个活动（前置（前驱）任务）之间的关系：

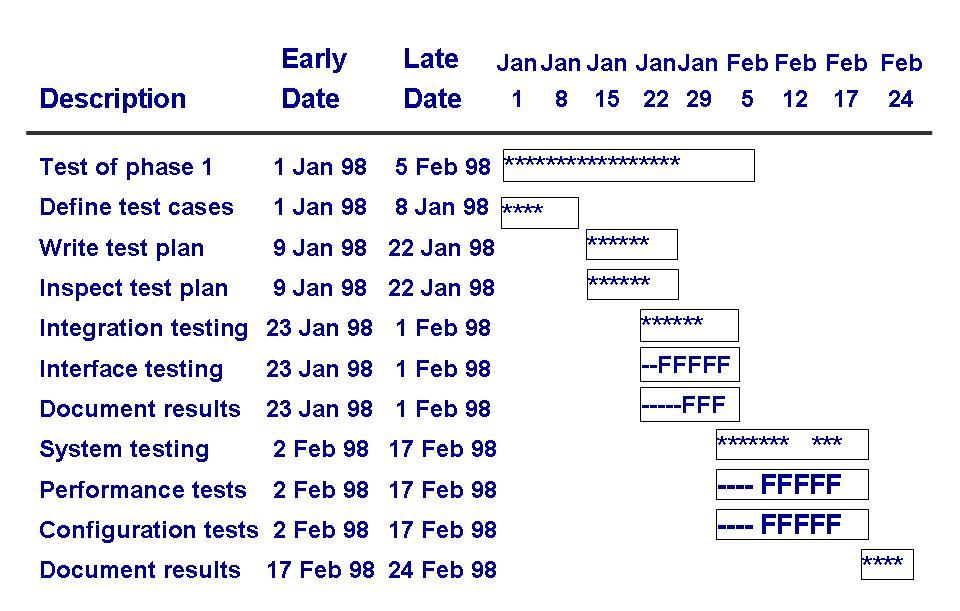
开始-完成 SF 开始-开始 SS

完成-开始 FS 完成-完成 FF



### CPM 条状图（甘特图）

包括早的开始日期和迟的开始日期的信息，表明关键路径



## 估算工作量的主要方法：代码行、任务分解技术、自动估算成本技术。

1）代码行技术

软件成本 = 每行代码的平均成本×估计的源代码总行数

估算方法：每个人都估计程序的最小规模(*a*)、最大规模(*b*)和最可能的规模(*m*)，分别算出这3种规模的平均值、和之后，再用下式计算程序规模的估计值：

单位：LOC或KLOC。

2）任务分解技术

软件开发项目分解为若干个相对独立的任务，分别估计每个单独任务的成本：

单独任务成本 = 任务所需人力估计值×每人每月平均工资；

软件开发项目总成本估计 = 各个单独任务成本估计值之和。

3）自动估计成本技术

采用自动估计成本的软件工具估计。

# UML图

## UML的作用

是为软件系统的制品进行描述（specifying）、可视化（visualizing）、构造（constructing）、文档化（documenting）的一种语言。

**UML成员 = UML 基本模型元素+ 关系+ 模型图**

* **基本模型元素可分为四类， 它们是：**
  + **结构模型元素（structural things）**
  + **行为模型元素（behavioral things）**
  + **分组模型元素（grouping things）**

**注解元素（annotational things）**

* **UML中共有4种关系，它们是：**
  + **关联关系（association）**
  + **依赖关系(dependency)**
  + **泛化关系(generalization)**
  + **实现关系(realization)**
* **9种UML模型图。它们是:**
  + **类图**
  + **对象图**
  + **用例图**
  + **序列图**
  + **协同图**
  + **状态图**
  + **活动图**
  + **组件图**

**分布图**

## UML中的4＋1视图：

用例视图，设计视图，进程视图，实现视图，分布视图。

**设计视图**

**实现视图**

**进程视图**

**部署视图**

**动态行为**

**设计词汇、功能描述**

**系统组装、配置管理**

**性能、稳定性、吞吐率**

**系统拓扑、分布、分发、安装**

**逻辑**

**视图**

### 用例视图

* 用例视图用来支持软件系统的需求分析，它定义系统的边界，关注的是系统的外部功能的描述。
* 它从系统的使用者的角度，描述系统的外部的
  + 静态的功能
  + 动态行为

P.S:

* 系统的动态功能由UML以下模型图描述：
  + 交互图(interaction diagram)，包括顺序图和协作图。
  + 状态图(state-chart diagram)
  + 活动图(activity diagram)

### 逻辑视图

* 逻辑视图定义系统的实现逻辑, 描述为实现用例图描述的功能，在对软件系统进行设计时, 所产生的设计概念，设计概念又称为软件系统的设计词汇 (vocabulary)。
* 逻辑视图定义了:
  + 设计词汇的逻辑结构
  + 存在于它们之间的语义联系
  + 设计词汇包括系统的类/协同/接口及其关系

### 实现视图

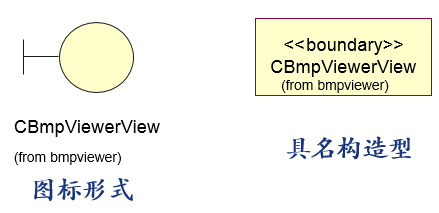
* 实现视图描述组成一个软件系统的各个物理部件，这些部件以各种方式组合起来，构成了一个可实际运行的系统。（如： 不同的源代码经过编译，构成一个可执行系统； 或者不同的软件组件配置成为一个可执行系统；以及不同的网页文件，以特定的目录结构，组成一个网站，等等）
* 当系统的逻辑结构在逻辑视图里被定义之后， 需要定义逻辑结构的物理实现。这包括：
  + 设计元素对应的源代码文件
  + 各物理文件之间的关系、存放路径，等等
* 实现视图就是定义这些内容的地方，它当于电子产品的印刷电路板的布线图
* 实现视图包含的模型图有:
  + 部件图（Component diagram）
  + 交互图（Interaction Diagram）
  + 状态图（state-chart diagram）
  + 活动图（activity diagram）

### 部署视图

* 软件产品将运行在计算机硬件系统上, 如果软件产品是面向网络的应用系统，则有可能同时运行在多个计算机上。
* 分布视图用来描述软件产品在计算机硬件系统和网络上的安装、分发（delivery）、分布（distribution）
* 在分布视图中，系统的静态特性用分布图（deployment diagram）描述
* 动态特性的描述用
  + 交互图（interaction diagram）
  + 状态图（state-chart diagram）
  + 活动图（activity diagram）

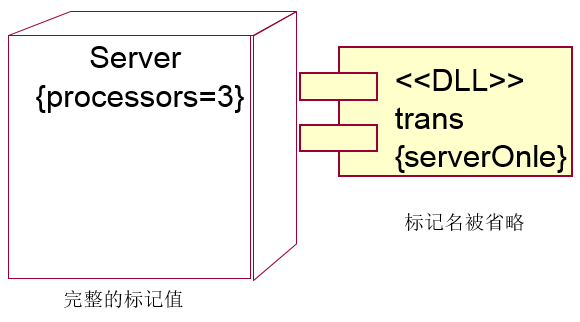
## UML中的三种扩展机制

### 构造型Stereotype

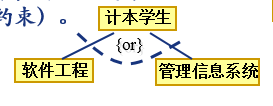
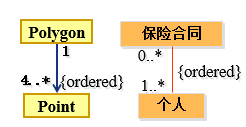
* + **在UML中, 构造型(stereotype)被定义为是对UML词汇（建模元素）的扩充，用来描述和已有的UML建模元素类似，但又对特定的问题领域有特殊意义的建模元素。**
  + **新的构成（用标记值表示）**
  + **新的语义（可用约束表示）**
  + **自己的标识符（文本的和图形化的）**
  + **两种形式**
  + **记名的构造型（named stereotype）**
  + **构造型的图标形式**
* 

from

### 标记值 tagged value

* + **在UML中，标记值被定义为是对UML建模元素的构成(property)的扩充，用于为此建模元素增加新的规格说明。**
  + **在特定的情形下，将有必要在建模元素的基本构成之外再增加一些构成,  此构成就是标记值**
  + **在UML里，标记值被图形化地表达为一个字符串，此字符串用花括弧括( {} )起来，被放置到原建模元素的名字的下方**
* 

### 约束 contraint

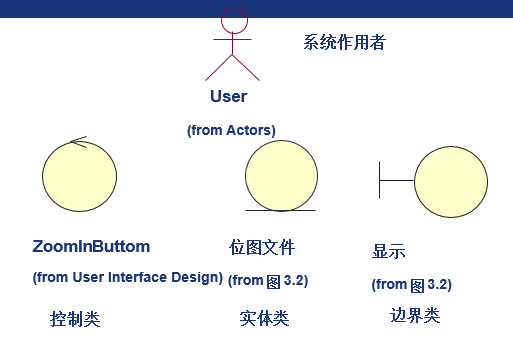
* **约束的定义：在UML里，约束用来扩充UML建模元素的语义，以便增加新的规则或修改已有的规则**
  + **在UML里，约束被图形化为一个文本串，此文本串被括在一对花括号内，并被放置在被约束的建模元素附近**
* 

### 标准拓展

#### 文档(documentation)

* **文档(documentation)是一个标记值，可以用于所有的UML建模元素。其取值可以是对此建模元素的注释、描述或解释**

#### 标准构造型

* **1)、系统作用者**
* **系统作用者（actor）是在进行软件系统的用例分析时必须用到的概念**
* **控制类（control class）**
  + **控制类代表一类控制或启动交互的对象。**
* **边界类（boundary class）**
  + **边界类的定义：边界类代表处于系统边界上，不但和系统内部对象交互，而且又和系统外部的系统作用者交互的一类对象。**
* **实体类(entity class)： 实体类是一类被动的对象，它本身不会启动交互，可以参加多个用例的交互，并且存活于任何单独的交互之外。**
* **实体类图形化表示**
  + **实体类被图形化描述为和一条短直线在底部相切的圆圈。**
* 

## UML中所包含的10种图形及各自的作用。

下面四种模型图主要用来描述静态结构

（1）类图包含类、接口、协同及其关系，它用来描述逻辑视图的静态属性

（2）对象图包含对象及其关系，它用来表示类图的类的对象在系统运行过

程中某一时刻的状态，对象也是软件系统的逻辑视图的一个组成部分。

（3）组件图描述系统的物理实现，包括构成软件系统的各部件（运行文件）

的组织和关系，类图里的类在实现时最终会映射到组件图的某个组件。一个

组件可以实现多个类。组件图是软件系统实现视图的组成部分。

（4）分布图描述系统的组件在运行时在运行节点上的分布，一个节点可包

含一个或多个组件。分布图是软件系统分布视图的组成部分。

描述软件系统的动态特性的, 使用

（1）用例图描述系统的边界，和其上的动态行为，图中包括：用例（use

case），系统作用者（actor）及其之间的(关联)关系。用例图是用例视图的

重要组成部分。

（2）序列图和协作图用来描述一组对象之间的动态交互。以用来描述系统的

动态特性、外部的动态特性、内部的动态特性。

（3）状态图和活动图用于描述对象的动态特性。

状态图强调对象对外部事件

的响应及相应的状态变迁

活动图描述对象之间控制流的转换和同步机制。

## 用例图的作用。

1. 描述和决定系统的功能需求，帮助客户和软件开发人员形成一致意见。

2. 给出系统应该做什么且与内容一致的可视化描述，使之成为在开发全过程中研讨系统需求和进行系统设计的依据。

3. 在软件测试阶段作为系统测试的基础。建立系统实现的各个对象类和系统操作与功能需求之间的可追踪关系。

## 用例图的主要构成部分。

用例图的三个组成部分：执行者、系统边界和用例