# 软件工程介绍

## **软件工程的概念——过程、方法和工具**

### IEEE对软件工程的定义:

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。

在（1）中所述方法的研究。

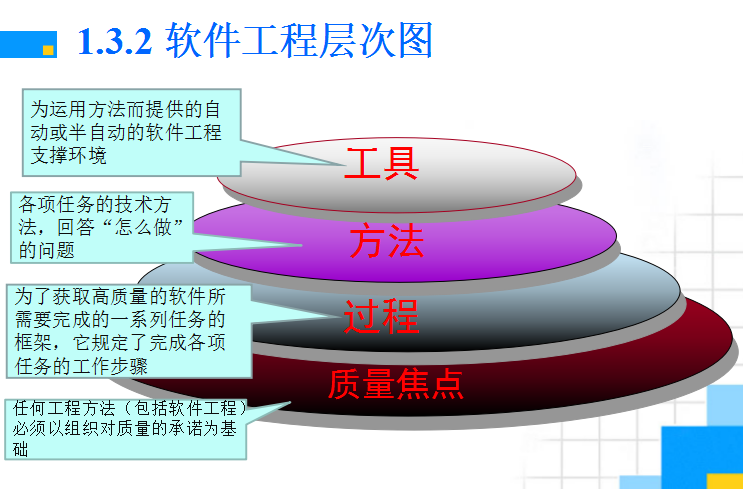
### 软件工程三要素：方法、工具和过程

工具：为过程和方法提供自动化或半自动化的支持

方法：为构建软件提供技术上的解决方法（如何做）

过程：过程定义了一个框架，构建该框架是有效实施软件工程技术必不可少的

## 软件工程层次图

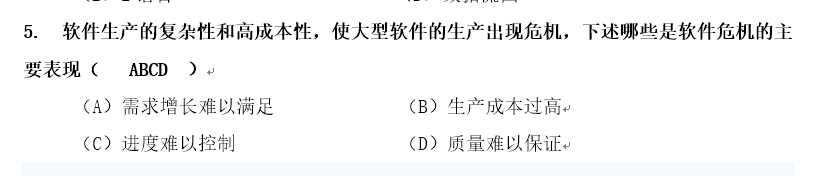


## **软件危机与软件工程的关系、产生的原因及其表现**

软件危机是指在计算机软件开发和维护过程中遇到的一系列严重问题。软件工程主要是针对20世纪60年代的软件危机而提出的。

软件危机的典型表现：

* + - * 1. 成本和进度估计常不准确
        2. 用户的满意度常不高
        3. 质量往往靠不住
        4. 软件通常很难维护
        5. 文档资料不完整、不合格
        6. 软件的成本高，所占比例逐年上升
        7. 软件开发生产率提高的速度慢



产生软件危机的原因：

* + - * 1. 客观原因
      * 软件缺乏“可见性”，管理和控制其开发过程相对困难；
      * 软件大多规模庞大，而复杂性随规模以指数速度上升。
        1. 主观原因

　-错误的认识和做法

* + - * **忽视软件需求分析的重要性**——急于求成，仓促上阵
      * **认为软件开发就是写程序**——编程只占全部工作量的10%--20%，软件配置主要包括程序、文档和数据
      * **轻视软件维护**——其实维护费用占总费用的55%--70%

## 软件神话一些错误认识

### 管理神话：

我们已经有了一本写满软件开发标准和规程的宝典。它无所不包，囊括了我们可能问到的所有问题。

如果我们未能按时完成计划，我们可以通过增加程序员人数而赶上进度。

如果将一个软件外包给另一家公司，则我们可以完全放手不管。

### 用户神话：

有了对项目目标的大概了解，便足以开始编写程序，我们可以在之后的项目开发过程中逐步了解细节。

虽然项目需求不断变更，但是因为软件是弹性的，因此可以很容易地适应变化。

### 从业者神话：

当我们完成程序并将其交付使用之后，我们的任务就完成了。

直到程序开始运行，才能评估其质量

对于一个成功的软件项目，可执行程序是惟一可交付的成果。

软件工程将导致我们产生大量无用文档，并因此降低工作效率。

# 过程模型

## 五个最基本的框架活动：沟通、策划、建模、构建和部署

沟通：与客户之间的交流与写作

策划：为后续的软件工程工作制定计划

建模：包括分析和设计

构建：编码和测试

部署：软件交付用户，用户对其进行评估并反馈意见

## 了解典型的普适性活动

软件项目跟踪和控制； 风险管理；

软件质量保证； 正式技术评审；

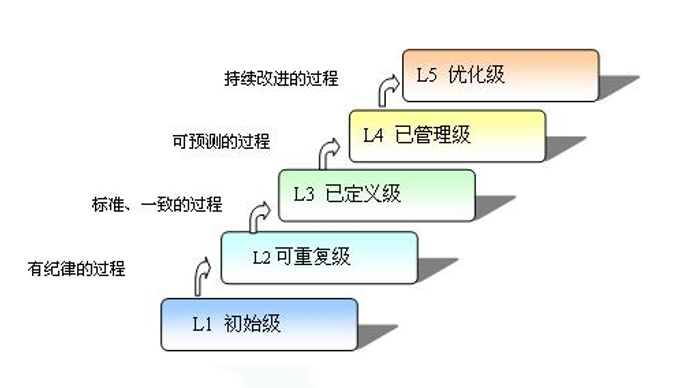
测量； 软件配置管理；

可复用管理； 工作产品的准备和生产

## 了解什么是CMMI

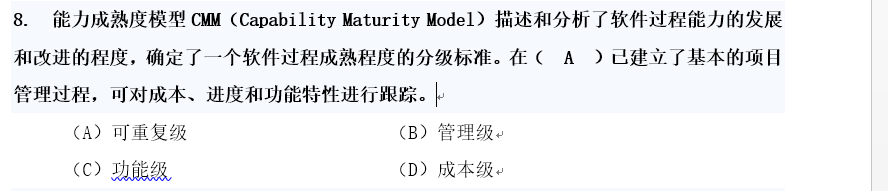
能力成熟度模型集成（CMMI），用于预测软件开发组织所开发的系统和软件工程能力。

CMMI有5个能力成熟度等级：L1初始级、L2可重复级、L3已定义级、L4已管理级、L5优化级。



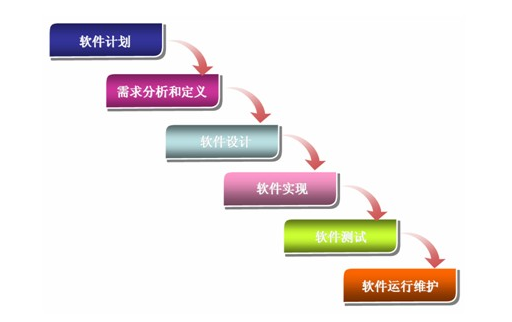
以下来源于网络

|  |  |  |
| --- | --- | --- |
| 能力等级 | 特点 | 关键过程 |
| 第一级 初始级（最低级） | 软件工程管理制度缺乏，过程缺乏定义、混乱无序。成功依靠的是个人的才能和经验，经常由于缺乏管理和计划导致时间、费用超支。管理方式属于**反应式**，主要用来应付危机。过程不可预测，难以重复。 |  |
| 第二级 可重复级 | 基于类似项目中的经验，**建立了基本的项目管理制度，采取了一定的措施控制费用和时间。**管理人员可及时发现问题，采取措施。一定程度上可重复类似项目的软件开发。 | 需求管理,项目计划,项目跟踪和监控,软件子合同管理,软件配置管理,软件质量保障 |
| 第三级 已定义级 | 已将软件过程**文档化、标准化**，可按需要改进开发过程，采用评审方法保证软件质量。可借助CASE工具提高质量和效率。 | 组织过程定义,组织过程焦点,培训大纲,软件集成管理,软件产品工程,组织协调,专家审评 |
| 第四级 已管理级 | 针对制定质量、效率目标，并收集、测量相应指标。利用统计工具分析并采取改进措施。对软件过程和产品质量有定量的理解和控制。 | 定量的软件过程管理和产品质量管理 |
| 第五级 优化级（最高级） | 基于统计质量和过程控制工具，持续**改进软件过程**。质量和效率稳步改进。 | 缺陷预防,过程变更管理和技术变更管 |



## 理解瀑布模型；增量模型；RAD模型；原型模型；螺旋模型；协同开发模型；基于构件模型；形式化方法模型；面向方面模型；统一过程（适用范围、特点、优缺点）

### 瀑布模型（也称为线性模型或传统生存周期，V模型。）



**适用范围：**

* + - * 1. **通常发生在一个已有系统，进行明确定义的适应性调整和增强的时候**
        2. **对于一个新的项目，需求必须是准确定义和相对稳定的**

特点：

* + - * 1. 段间的顺序性和依赖性；
        2. 文档驱动性；
        3. 严格阶段评估；
        4. 开发初期需要清楚全部需求；
        5. 开发周期长、风险大。

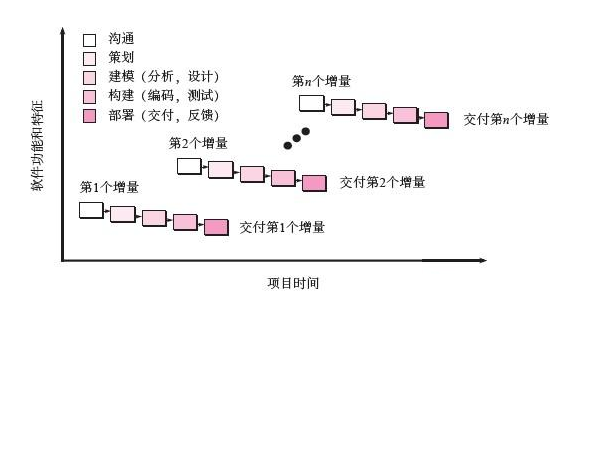
优点：

* + - * 1. 它提供了一个模板，这个模板使得分析、设计、编码、测试和支持的方法可以在该模板下有一个共同的指导。
        2. 虽然有不少缺陷但比在软件开发中随意的状态要好得多。

缺点：

* + - * 1. 顺序太严格。实际工作经常是在多个环节之间来回反馈调整，而不是将一个环节完成后再继续前进。
        2. 产品在最后阶段才与客户见面，从心里学的角度讲有些考验客户。另外，如果此时才发现问题，需要改正，工作量将会很大。
        3. 效率可能不高。
        4. （客户通常难以清楚的描述所有需求。而瀑布模型要求客户明确需求，这就很难适应在许多项目开始阶段必然存在的不确定性）

### 增量模型（以迭代方式运用瀑布模型）



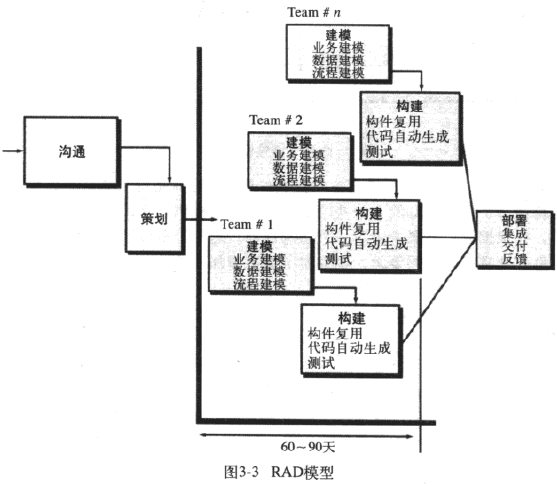
使用范围：初始的软件需求有明确的定义，但是整个开发过程确不宜单纯运用线性模型。同时，迫切需要为用户提供一套功能有限的产品，在后续版本在进行功能扩展和细化

特点：

* + - * 1. 一般来讲，最重要的增量放在前面。
        2. 每次交付的增量产品都是可用的。
        3. 适合于功能可以划分，而且时间不紧迫的情况。
        4. 可以规避一定的风险。如有些技术还不稳定，将这部分放到后边。

优缺点：可以规避一定的风险。如有些技术还不稳定，将这部分放到后边。

### RAD模型：快速应用程序开发,



适用范围:适于工期紧张，又可细分功能，还要有合适的构件。

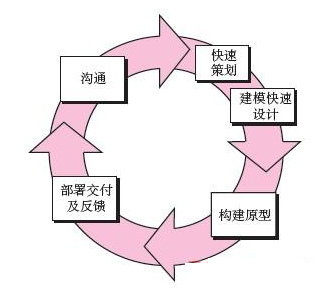
特点：

* + - * 1. 瀑布模型的高速变体，通过基于**构件**的方法快速实现。
        2. 是一种侧重于**短暂的开发周期**的增量软件模型

缺点：

* + - * 1. 需要投入更多的人力。
        2. 各团队要紧密协作。
        3. 只适应于特殊的系统，必须可以合理模块化。
        4. 不适于高性能需求灵活的系统，现有构件不容易轻易满足。
        5. 技术风险很高的情况下，不宜采用该模型。

### 原型开发模型:



演化模型——原型、螺旋、协同开发

适用范围：

* + - * 1. 客户没有详细定义功能和特性需求；
        2. 开发人员可能对算法的效率、操作系统的适应性和人机交互的形式等情况并没有把握。

对原型的基本要求：

* + - * 1. 体现主要的功能
        2. 提供基本的界面风格
        3. 展示比较模糊的部分，以便于确定或进一步明确，防患于未然
        4. 原型最好是可以运行的，最少要在各主要功能模块之间能够建立相互连接

原型的处理方法：

* + - * 1. 抛弃型

在获取的明确需求的基础上，重新设计与开发

成本相对高，小公司一般慎用

* + - * 1. 演化型

在原型的基础上继续开发

优缺点：

* + - * 1. 优点：

能让人（开发者或客户）很快见到产品，有成就感。

能渐进地启发客户提出新的要求或任务。

* + - * 1. 缺点：

容易蒙骗客户，也可能由此给自己带来麻烦。

往往只为结果，而不考虑技术手段，为今后埋下隐患。

系统可能考虑不周全。

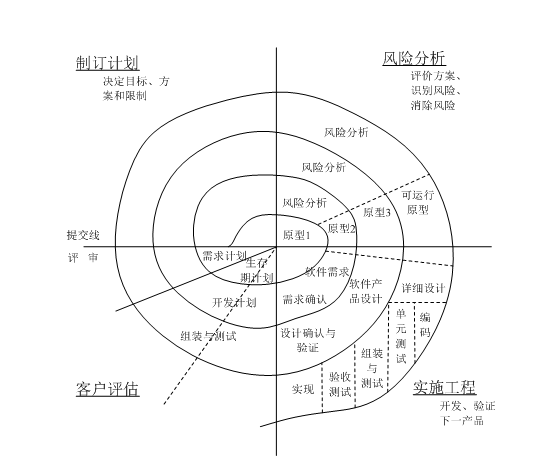
原型开发模型与增量模型相比：

* + - * 1. 增量模型在开发以前基本能确定系统的**需求**，虽然在以后的过程中也可能不断完善；原型开发适应于预先不太清楚系统的需求。
        2. 增量模型的**反馈**可能较少，而原型开发需要不断的大量反馈信息。

### 螺旋模型

特点：

* + - * 1. 结合了原型的迭代性质和瀑布模型的系统性和可控性特点
        2. 风险驱动，引入非常严格的风险识别、风险分析和风险控制
        3. 早期迭代中可能是一个理论模型或原型



适用范围：开发大型系统和软件

螺旋模型与原型开发模型相比：

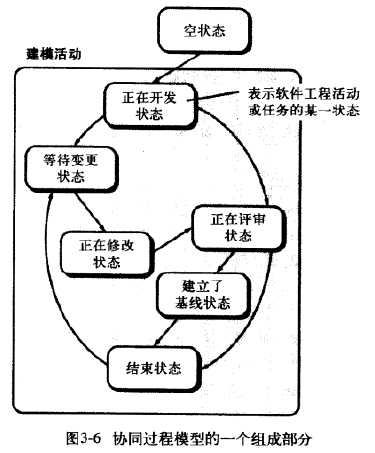
* + - * 1. 螺旋模型虽不像增量模型中对功能有明确界定，但比原型要清晰一些。
        2. 螺旋模型的反馈要求持续于产品的整个生命期。
        3. 适合于大型软件的开发。

### 协同开发模型（又叫协同工程、并发开发模型）

适用范围：适用于所有类型的软件开发

特点：

* + - * 1. 定义了一个活动的网络，网络上每个活动、动作和任务同时存在。
        2. 过程网络中某一点产生的事件可以触发状态的转换。
        3. 可适用于所有类型的软件开发



### 专用过程模型

特点：具有通用过程模型的特点

适用范围：只适用于一些特定的领域

包括：

* + - * 1. 基于构件的开发
        2. 形式化方法模型
        3. 面向方面的软件开发等

### 基于构件的开发

特点：类似于螺旋模型，本质上是演化模型，需要以迭代的方式构建软件

构件开发的步骤：

* + - * 1. 对所需构件进行评估。
        2. 考虑构件的集成。
        3. 设计系统的软件框架。
        4. 将构件放入框架。
        5. 进行测试。

优点：能够使用软件复用，从而为软件工程师带来极大收益，如缩短开发周期并减少项目开发费用等。

### 形式化方法模型（主要活动是生成计算机软件形式化的数学规格说明）

特点：精密、准确。

缺点：难度大，成本高，可用人力资源少，用户不易理解，有时甚至无法完成。

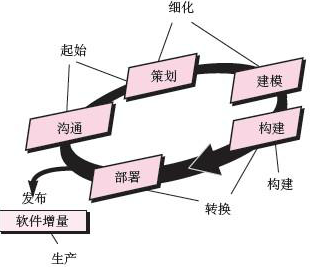
介绍：形式化方法使软件开发人员可以应用严格的数学符号来说明、开发和验证基于计算机的系统

### 面向方面的软件开发

介绍：将系统分成若干相对较独立的组成部分，这些部分称为**方面**。方面性需求定义那些对整个软件体系结构产生影响的横切关注点。面向方面（Aspect-Oriented）技术范围包括面向对象技术。

特点：还不成熟。具有演化型和协同型的共同特点。

### 统一过程（将传统软件模型和敏捷过程模型的优点结合起来，即统一起来。）



统一过程包括：起始，细化，构建，转换，生产等步骤。

起始：

* + - * 1. 包括客户沟通和策划活动
        2. 此时的构架只是主要子系统及其功能、特性的试探性概括。

细化：

* + - * 1. 包括策划活动和通用过程模型的建模活动
        2. 扩展了体系结构，包括软件的五种视图：用例模型、分析模型、设计模型、实现模型和部署模型。

构建：

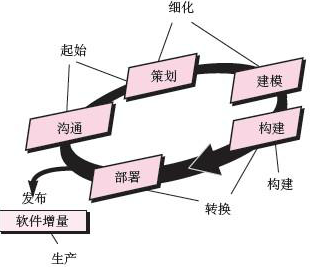
* + - * 1. 与通用软件过程的构建活动相同。
        2. 采用体系结构模型作为输入，开发或获取软件构件，使得最终用户能够操作用例。

转换：

* + - * 1. 软件被提交给最终用户进行Beta测试，用户反馈报告缺陷及必要的变更。
        2. 另外，发布必须的支持信息：用户手册，用户指南及安装步骤等。
        3. 结束时，软件增量成为可用的发布版本。

生产：

* + - * 1. 与通过软件工程的部署一致
        2. 提供运行环境支持，提交并评估缺陷报告和变更请求。



# 敏捷开发

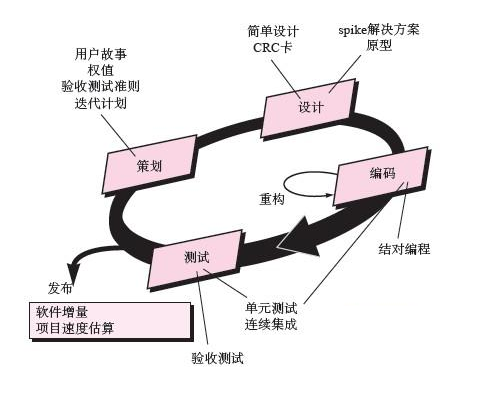
## 掌握敏捷开发宣言



## 理解有哪些敏捷过程模型：（关键实践）极限编程； Scrum 关键思想

### 极限编程

极限编程的基本活动



极限编程（eXtreme Programming, XP）敏捷开发中运用最广泛的方法。适用于小团队开发。包含一些基本活动，力求用最少的精力产出最大的成果，运用已有成果、方法。

包含了 策划、设计、编码和测试4个框架活动的规则和实践

* + - * 1. **策划**

把任务细分，尽量在三周内完成。如果完不成，则再进行细分。细分后做以下工作：1）尽快实现每个任务。2）重要者优先。3）高风险优先。

项目第一个发行版本后，利用已有数据计算进度，以用来安排:1) 后续工作的进度。 2）重新审视以前的安排。

**用户故事**

问题描述：

“作为……（谁），我想要……  
（做什么），为了……（为什么）”。

验收条件

* + - * 1. **设计**

保持尽量**简洁**

尽量使用**已有构件**。

在前进中调整。

* + - * 1. **编码**

常规工作中，先编码，然后开发检测实例。在XP中，提倡先开发检测实例，然后编码。好处：有一个航标指引你前行。--**测试驱动**

提倡**结对编程**，好处：两个人的力量大于一个人的力量。能应付以后的人事变动。

* + - * 1. **测试**

经常的测试。

快速的测试。

阶段性的测试。

便于及时发现问题。

XP验收测试，生产客户可见的测试集

**极限编程实践**：

* + - * 1. 完整团队
        2. 计划游戏
        3. 客户测试
        4. 简单设计
        5. 结对编程
        6. 测试驱动开发
        7. 改进设计
        8. 持续集成
        9. 集体代码所有权
        10. 编码标准
        11. 隐喻
        12. 可持续的速度

### Scrum模型

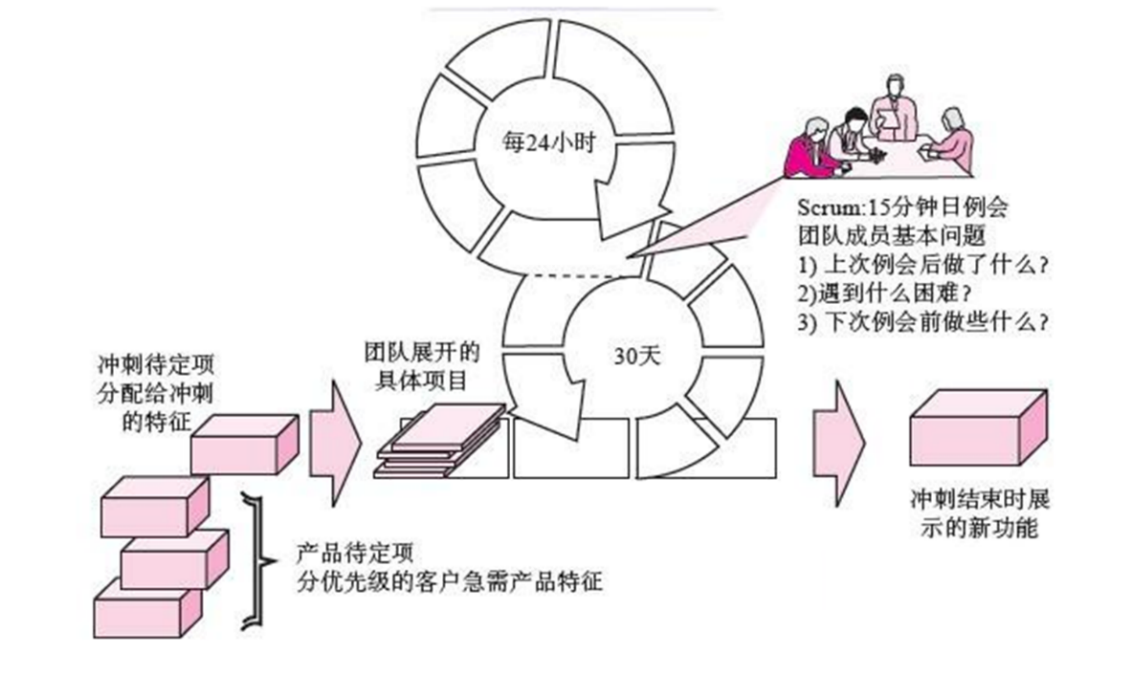
Scrum原则与敏捷宣言一致：

* + - * 1. 组织**小型团队**以达到“沟通最大化、负担最小化、非语言描述、非形式化知识”
        2. 过程对技术和业务变化必须具有**适应性**，以“保证制造具有最好可能的产品”
        3. 过程生产**频繁发布**“可检查、可调整、可测试、可文档化、可构建”的**软件增量**
        4. 开发工作和开发人员分为“清晰的、低耦合的部分或包”
        5. 坚持在产品构建过程中进行**测试和文档化**
        6. Scrum过程提供“在任何需要的情况下都能完成产品的能力”

特点：所有实践围绕一个迭代、增量的过程骨架展开

包括：

* + - * 1. 待定项—一个能为用户提供商业价值的项目需求或特性的优先级列表
        2. 冲刺---短时间内完成特定的任务。
        3. Scrum例会---总结，展望。
        4. 演示---交付部分软件增量。



*图 Scrum过程流*

Scrum方法中只有三种角色：

* + - * 1. **产品负责人**：

代表每位利益相关者的权益，并为项目产出的软件系统负责。规划项目初始总体要求、投资回报目标和发布计划，开发优先级确定。

* + - * 1. **团队：**

责任：开发软件功能

* + - * 1. **ScrumMaster：**

对Scrum 过程负责，向所有项目参与者讲授Scrum方法，负责实施Scrum确保其既符合企业文化，又能交付预期利益，还需督促全体成员遵从Scrum规则和实践。

## 了解一些过程模型（知道这些是敏捷模型）

**自适应软件开发；动态系统开发；Crystal；特征驱动开发**

### 自适应软件开发（ASD)

着眼于人员协作和团队自我组织。

### 动态系统开发（DSDM）

通过在可控项目环境中使用增量原型开发

### Crystal

目的是开发一种提倡“机动性的”的软件开发方法。

### 特征驱动开发（FDD）：

特征:能在更短时间内（两周内）完成的小功能。

# 可行性研究

## 了解可行性研究的目的和任务

### 可行性研究的目的：

是用**最小的代价**在**尽可能短的时间内**确定问题**是否能够解决**。

### 可行性研究的任务：

探索若干种可选系统实现方案，至少从以下3个方面研究每种方案的可行性：

* + - * 1. **技术可行性** 使用现有的技术能实现这个系统吗？
        2. **经济可行性** 这个系统的经济效益能超过它的开发成本吗？
        3. **操作可行性** 系统的操作方式在这个用户组织内行得通吗？

必要时还应该从法律、社会效益等更广泛的方面研究每种解法的可行性。

# 理解需求

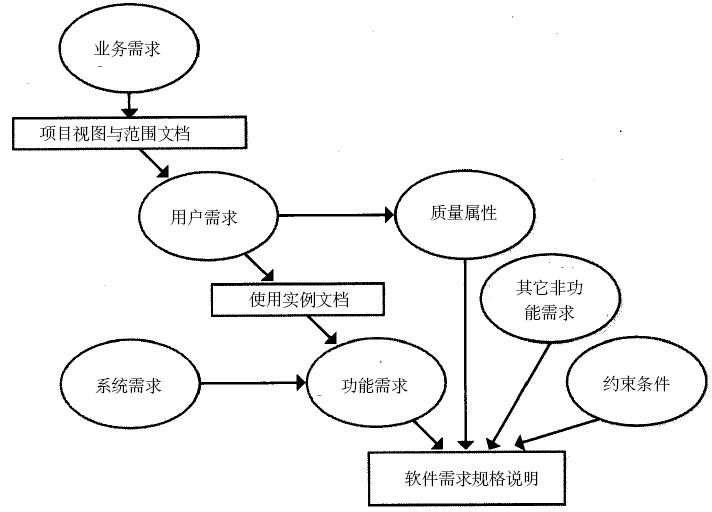
## 为什么需求工程特别困难？

客户说不清楚需求

需求自身不断变动

分析人员或客户理解有误

## 需求分析的三个层次



### 业务需求:

反映了组织机构或客户对系统、产品高层次的目标要求。

### 用户需求:

文档描述了用户使用产品必须要完成的任务。



### 功能需求——也包括非功能需求

定义了开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了业务需求。

## 需求工程中的七个活动

**起始；导出；精化；协商；规格说明；确认；管理**

### 起始

软件工程是询问一些似乎与项目**无直接关系**的问题

泛谈起始，有各种各样的情况

目的是对问题、方案需求方、期望方案的性质、客户和开发人员之间初步的交流和合作的效果，建立基本的理解。

### 导出

询问客户、用户和其他人，系统或产品的目标是什么？想要实现什么？系统和产品如何满足业务的要求? 最终系统和产品如何用于日常工作？

**非常困难**：范围问题、理解问题、易变问题

### 细化

将起始和导出阶段获得的信息进行扩展和提炼

是一个分析建模动作

**细化的最终结果**：

一个精确的需求模型，定义了软件的功能、特征和信息的各个方面。

### 协商

需求工程师必须通过协商的过程, 调节各种冲突

按优先级讨论冲突(每个用户都说他的需求至关重要)

识别和分析风险

粗略“估算”开发工作量，并评估每项需求对项目成本和交付时间的影响

使用迭代，删除、细化或修改需求，以便各方达到一定的满意度

### 规格说明

把前面的成果用文字或其它方式明示出来。

可以是一份写好的**文档**，一套**图形化**的模型，一个形式化的**数学模型**，一组使用**场景**，一个原型或上述各项的任意组合。

### 确认

要**检查规格说明**以保证：

所有的系统需求已被无歧义地说明；不一致性、疏漏和错误已被检测出并被纠正；工作产品符合为过程、项目和产品建立的标准。

由**第三方（通常为评审组）完成**

### 管理

用于帮助项目组在项目进展中**标识、控制和跟踪需求**以及**变更需求**的一组活动。

解决方法：特征跟踪表、来源跟踪表、依赖跟踪表、子系统跟踪表、接口跟踪表等。

工具EasyRM, Rational RequisitePro

## 导出需求有哪些方法

访谈； 面向数据流自顶向下求精；

协同需求获取； 快速建立软件原型；

质量功能部署； 用户场景

### 访谈

正式访谈——事先准备好的具体问题

非正式访谈——自由问答

### 面向数据流自顶向下求精

数据决定了需要的处理和算法，是需求分析的出发点

需求分析的目标之一是**在可行性研究得到的高层数据流图基础上，把数据流和数据存储定义到元素级**。

通过对未知数据和其需要的算法的请教，深入认识系统

分析员借助数据流图、数据字典和IPO图向用户解释输入数据是怎么一步步转变成输出数据的。

### 协同需求获取

面向团队的需求获取方法

利益者和开发人员的团队共同完成：**确认问题，为解决方案的要素提供建议，协商不同的方法，以及说明初步的解决方案的需求集合**。

### 快速建立软件原型

是最准确、最有效、最强大的需求分析技术

构造原型的要点是：应该事先用户能看得见的功能（如屏幕演示或打印报表），可省略目标系统的“隐含”功能（例如，修改文件）

快速原型的特性：快速、容易修改

### 质量功能部署（QFD）

是把**顾客或市场的要求**转化为设计要求、零部件特性、工艺要求、生产要求的多层次演绎分析方法。

QFD对需求的分类：

* + - * 1. 常规需求：

也称普通需求，包含客户对项目的最基本需求，是客户对整个项目最为关心的部分。

* + - * 1. 期望需求：

客户可能没有表达明确或没有明确提出的需求，但是会让客户提升对项目的满意度。

* + - * 1. 意外需求：

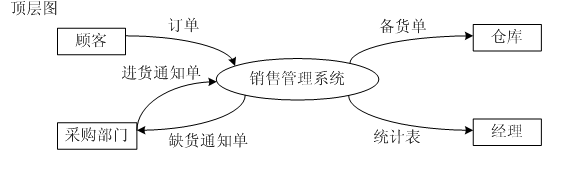
也称兴奋需求，如果实现会给客户带来惊喜，但是如果无法实现也不会受到客户责备。

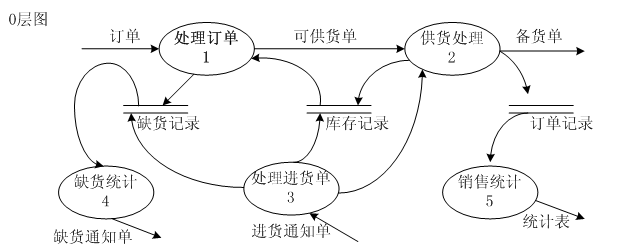
### 用户场景

通常被称为**用例**，它提供了系统将如何被使用的描述

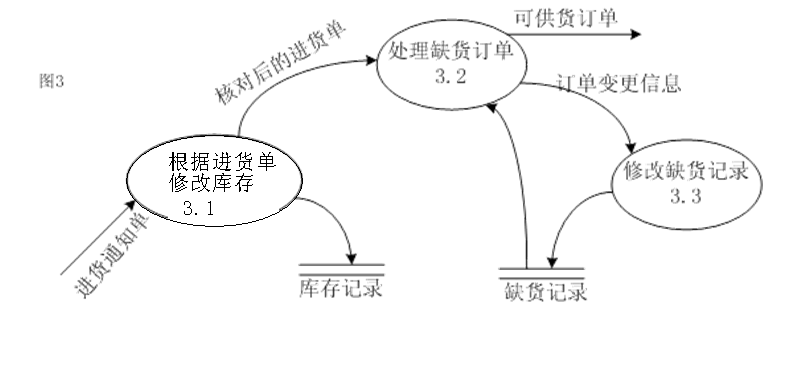
关注用户将如何使用该功能

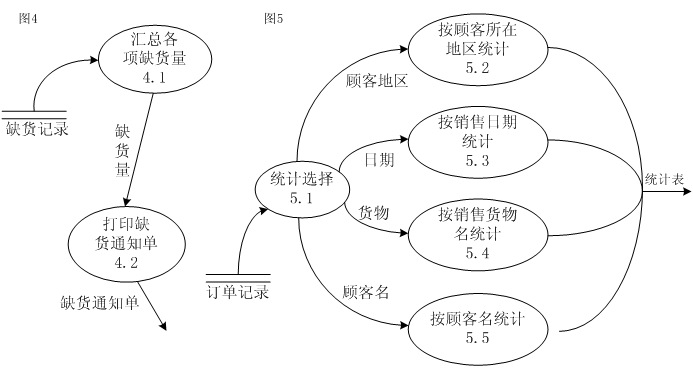
## 掌握数据流图画法











注意事项：

* + - * 1. 一个数据流一条线
        2. 0层图中，没有指明来源的数据，不要画出来
        3. 1层图中，没有指明来源的数据，要从某个数据存储里去取出来
        4. 注意所有细节，以及事件的先后顺序和发起人
        5. 题目只给了操作，要自己脑补数据

eg:“系统会安排租赁者与房主见面”

系统=》房主：看房安排

房主=》系统：看房安排确认

<https://wenku.baidu.com/view/cbacfeb0b9d528ea80c7791e.html>

# 需求建模

## 掌握分析建模的方法都有哪些

### 结构化分析：

着重考虑数据和处理。

### 面向对象分析

对各种**对象**加以研究，包括其对应的数据和处理。

关注于**定义类**和影响客户需求的类之间的**协作**方式。

## 能够根据要求绘制：

**数据流图；概念类图；**状态图；**用例图；**活动图

## 熟悉基本加工逻辑说明的三种方法

结构化英语（PDL）； 判定表； 判定树

### 结构化英语的词汇表由

英语命令动词

数据词典中定义的名字

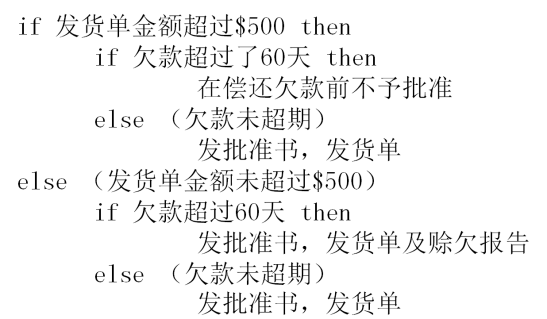
有限的自定义词

逻辑关系词 IF\_THEN\_ELSE、

* + - * 1. CASE\_OF 、 WHILE\_DO、
        2. REPEAT\_UNTIL等组成。

其基本控制结构有三种：

* + - * 1. 简单陈述句结构：避免复合语句
        2. 重复结构：while\_do 或repeat\_until 结构
        3. 判定结构：if\_then\_else 或case\_of 结构



### 判定表

如果数据流图的加工需要依赖于多个逻辑条件的取值，使用判定表来描述比较合适。



### 判定树

判定树也是用来表达加工逻辑的一种工具。有时候它比判定表更直观。



# 设计工程

## 理解设计和需求的目标有什么不同

软件需求：解决“做什么”

软件设计：解决“怎么做”

## 理解设计的软件设计的目标和任务

### 软件设计的目标和任务

根据用信息域表示的软件需求，以及功能和性能需求，进行：

* 数据设计
* 系统结构设计
* 过程/构件设计
* 接口设计

1. **数据设计**将类模型转化为设计类的实现以及软件实现所需要的数据结构。
2. **体系结构设计**定义了软件的主要结构化元素之间的关系，可满足系统需求的体系结构和模式以及影响体系结构实现方式的约束。
3. **接口设计**描述了软件和协作系统之间，软件和使用人员之间是如何通信的。
4. **过程设计/构件设计**则是把结构成份转换成软件的过程性描述。在编码步骤，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件。

软件设计任务：

* 从工程管理的角度来看，软件设计分两步完成：
  + - * 1. 概要设计（总体设计）：
    - 将软件需求转化为数据结构和软件的系统结构。
      * 1. 详细设计（过程设计，模块设计）：
    - 通过对结构表示进行细化，得到软件的详细的数据结构和算法。

软件设计目标

### 软件设计的目标

功能性(Functionality)：

通过评估程序的特征集和能力、所提交功能的通用性以及整个系统的安全性来评估

易用性（Usability)

通过考虑人员因素、整体美感、一致性和文档来评估。

可靠性（Reliability）

通过测量故障的频率和严重性、输出结果的精确性、平均故障时间、故障恢复能力和程序的可预见性来评估

性能（Performance)

通过处理速度、响应时间、资源消耗、吞吐量和效率来度量

可支持性(Supportability）

综合了可扩展性、可适应性和可用性。此外，还包括可测试性、兼容性、可配置性、系统安装的简易性和问题定位的容易性。

## 理解软件设计各层设计理念及关系

### 抽象：

过程抽象：具有明确和有限功能的指令序列，如“开”门

数据抽象：描述数据对象的冠名数据集合，如“门”

### 体系结构：

软件的整体结构和这种结构为系统提供概念上的完整性的方式。

程序各功能模块的编排顺序、作用方式、具体的数据结构。可以用多种模型来完成一个体系结构设计。

### 模式：

设计模式描述了在某个特定场景与可能影响模式应用和使用方式的“影响力”中，解决某个特定的设计问题的设计结构。

### 关注点分离：

任何复杂问题如果被分解为可以独立解决和（或）优化的若干块，该复杂问题能够更容易被处理。

一个关注点是一个特征或行为，被指定为软件需求模型的一部分。

### 模块化：

软件被划分为可以独立命名的、可寻址的构件，有时被称为模块

两个问题综合时的理解复杂度通常要大于每个问题各自的理解复杂度之和

“分而治之”，将一个策略问题分解成可以管理的若干块，这样更容易解决问题。

### 信息隐蔽：

每个模块对其他所有模块都隐蔽自己的设计策略

隐蔽意味着通过定义一系列独立的模块,可以得到有效的模块化，模块间只交流所必须的信息。

为修改提供很大的益处。

### 功能独立：

模块化、抽象概念和信息隐蔽的直接结果。

通过开发具有“专一”功能和“避免”与其他模块过多交互的模块。

具有独立模块的功能更易开发。

模块的独立程度可以由两个定性标准来度量：

* + - * 1. 内聚——模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量
        2. 耦合——模块之间的互相连接的紧密程度的度量

### 求精：

层次结构的开发将通过逐步分解功能的宏观陈述（过程抽象），直至形成程序设计语言的语句。

精化促使设计者在原始陈述上**细化**，并随着每个精化（细化）的持续进行，将提供越来越多的细节。

### 重构：

重构是使用这样的一种方式改变软件系统的过程：**不改变代码的外部行为而是改进其内部结构**。

**检查**冗余、未使用的设计元素、低效或不必要的算法、低劣或不恰当的数据结构以及设计不足，**修改**以获取更好设计。

### 设计类：

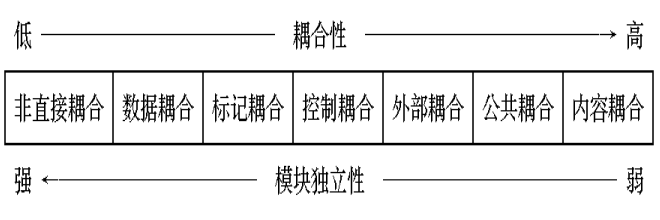
分析类关注用户或客户可见的问题，描述问题域中的某些元素。

设计类

* + - * 1. 通过提供设计细节，精化分析类。
        2. 创建一组新的设计类，实现软件的基础设施以支持业务解决方案。

## 了解内聚和耦合的不同种类

### 耦合



非直接耦合：两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。非直接耦合的模块独立性最强。

数据耦合：一个模块访问另一个模块时，彼此之间是通过**简单数据参数** (不是控制参数、公共数据结构或外部变量)来交换输入、输出信息的。

标记耦合：一组模块通过参数表传递**记录信息**，就是标记耦合。这个记录是某一数据结构的子结构（如**数组名、文件名**等），而不是简单变量。

控制耦合：如果一个模块通过传送开关、标志、名字等**控制信息**，明显地控制选择另一模块的功能，就是控制耦合。

外部耦合：一组模块都访问**同一全局简单变量**而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。

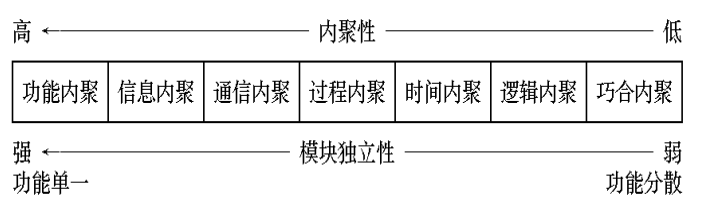
公共耦合：若一组模块都访问同一个**公共数据环境**，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。

内容耦合：如果发生下列情形，两个模块之间就发生了内容耦合

* + - * 1. 一个模块直接访问另一个模块的内部数据
        2. 一个模块不通过正常入口转到另一模块内部;
        3. 两个模块有一部分程序代码重迭(只可能出现在汇编语言中);
        4. 一个模块有多个入口。

实际上，模块之间是混合式的耦合。原则：尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。

### 内聚



功能内聚：一个模块中各个部分都是**完成某一具体功能**必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的。则称该模块为功能内聚模块。

信息内聚：这种模块**完成多个功能，各个功能都在同一数据结构上操作**，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表），因此，它是一个信息内聚的模块。

通信内聚：如果一个模块内各功能部分都使用了相同的**输入数据**，或产生了相同的**输出数据**，则称之为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。

过程内聚：使用流程图做为工具设计程序时，**把流程图中的某一部分划出组成模块**，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。

时间内聚：时间内聚又称为经典内聚。这种模块大多为多功能模块，但**模块的各个功能的执行与时间有关**，通常要求所有功能必须在同一时间段内执行。例如初始化模块和终止模块。

逻辑内聚：这种模块把几种相关的功能组合在一起，每次被调用时，**由传送给模块的判定参数**来确定该模块应执行哪一种功能。

巧合内聚：巧合内聚 (偶然内聚)。当模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。

# 进行体系结构设计

## 理解什么阶段要做体系结构设计

一般在**方案阶段**就会**确定结构体系**，因为这个关系造价，且比重很大。[初步设计](https://www.baidu.com/s?wd=%E5%88%9D%E6%AD%A5%E8%AE%BE%E8%AE%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYmvc4P1NbnjN9PjcLmHTs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnkPWRzn1R4PWcvnWcdrHmY)阶段基本就有确定的大致体系

## 掌握体系结构风格的分类及各类的主要特点

### 以数据为中心的体系结构

**数据存储**驻留在这种体系结构的中心，其他构件经常访问（增删改查）数据存储。

提高了**可集成性**，便于现有构件修改和新构件加入。

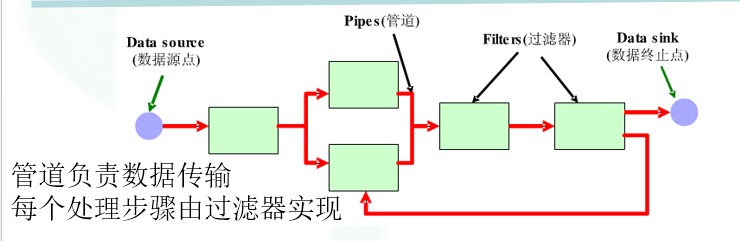
### 数据流体系结构

数据服从输入—变换—输出的简单流程

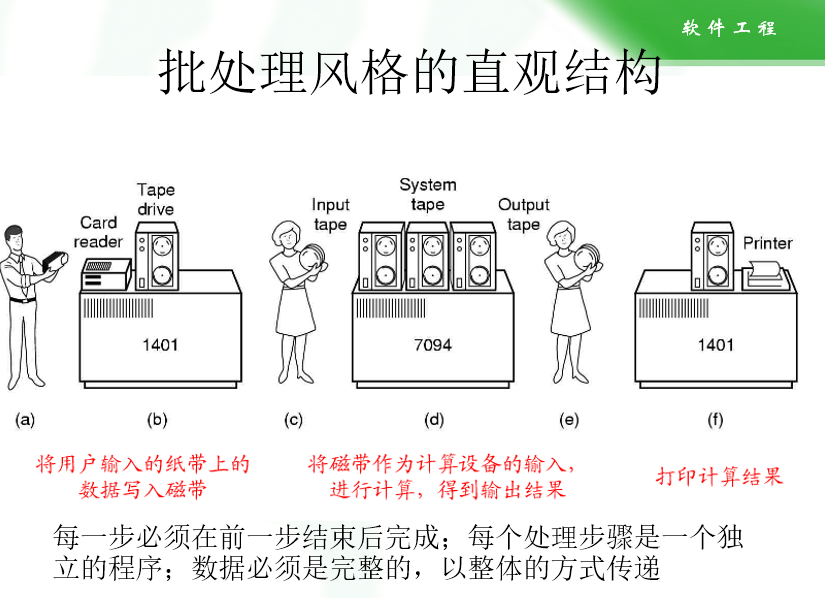
管道和过滤器结构拥有一组被称为过滤器的构件，每个构件独立于上游和下游而工作。

两种典型的数据流风格：

* + - * 1. 管道-过滤器（Pipe-And-Filter）



* + - * 1. 批处理（Batch Sequential)



### 调用和返回体系结构

主程序/子程序体系结构

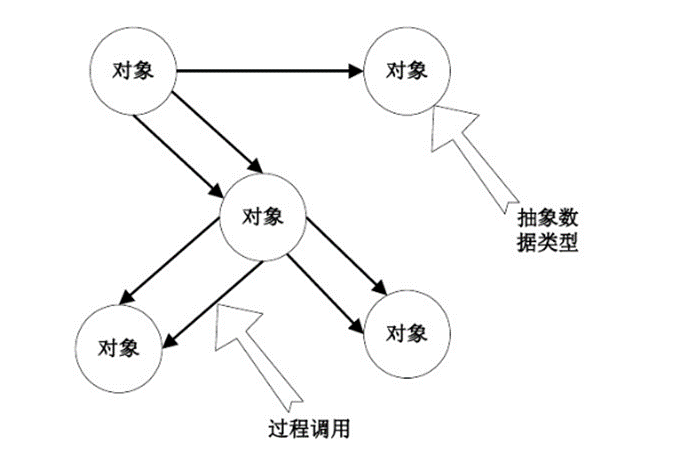
主程序调用一组程序构件，这组程序构件又去调用其程序构件

远程过程调用体系结构

主程序/子程序的构件分布在多个计算机上。

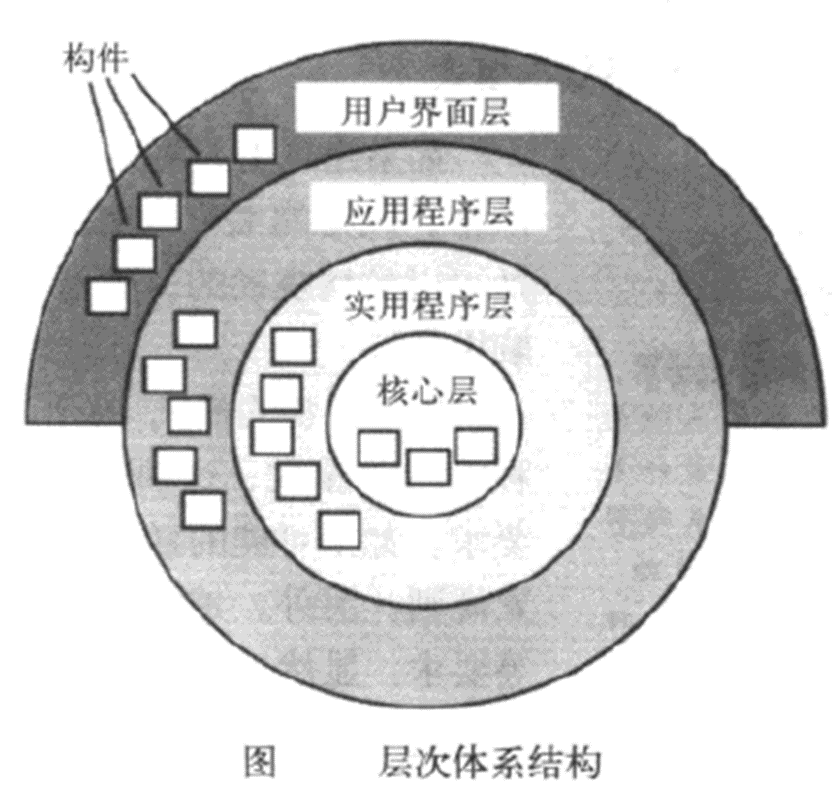
### 面向对象体系结构。

封装了数据和必须应用到该数据上的操作，构件间通过信息传递进行通信和合作



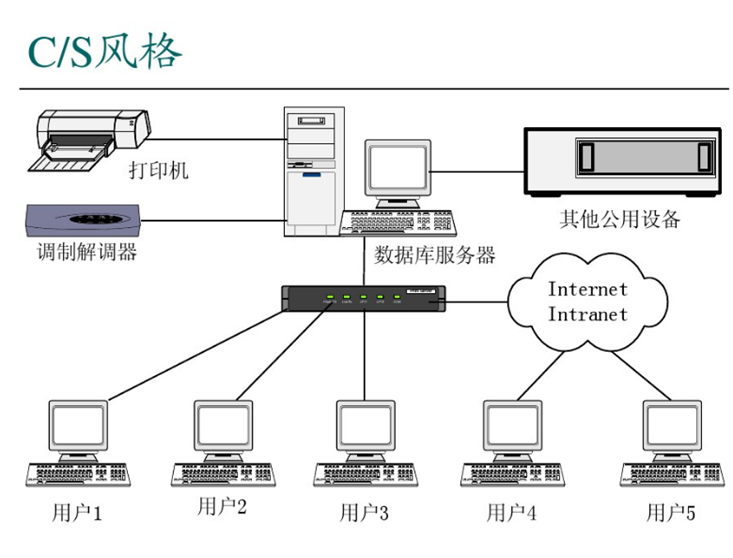
### 层次体系结构

各模块实现功能的层次不一样。



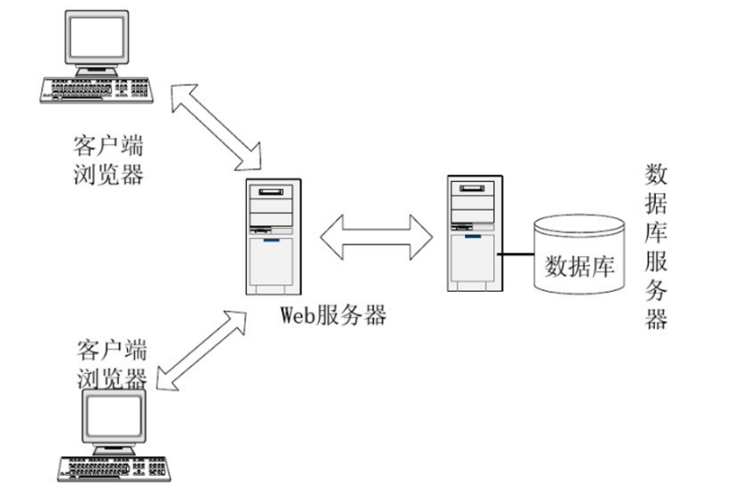
### C/S风格

数据库服务器、客户应用程序、网络



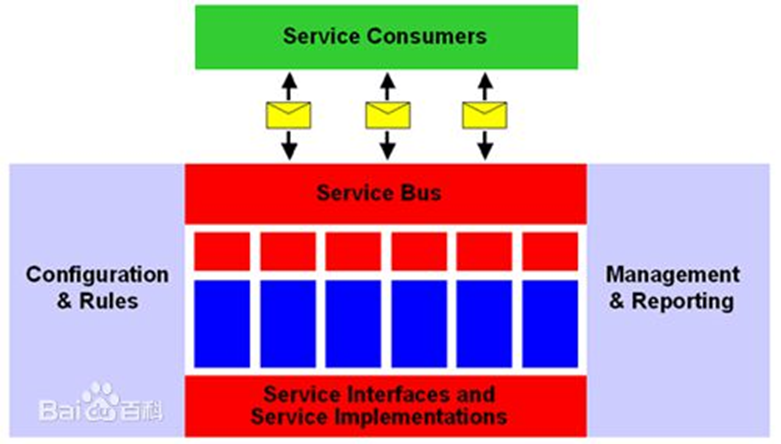
### B/S风格

浏览器/Web服务器/数据库服务器



### SOA

整个应用程序被设计和实现为一组相互交互的服务

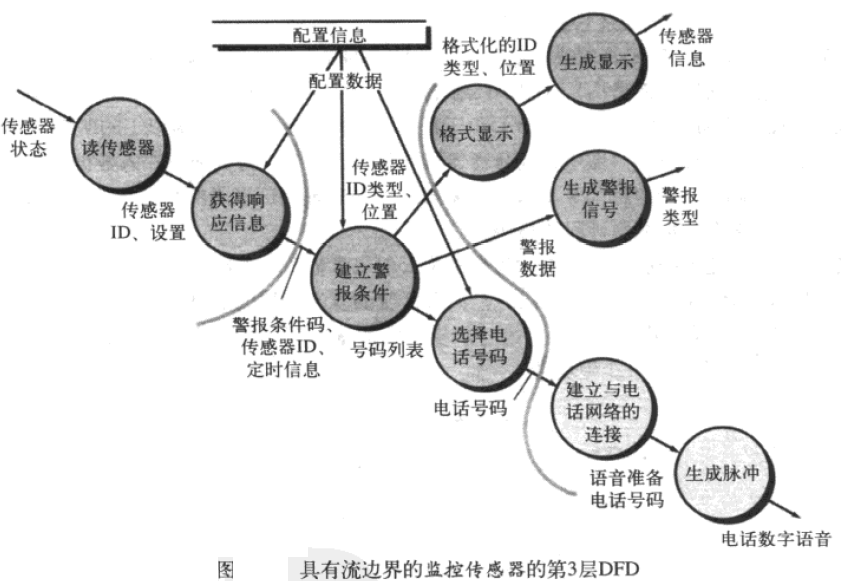


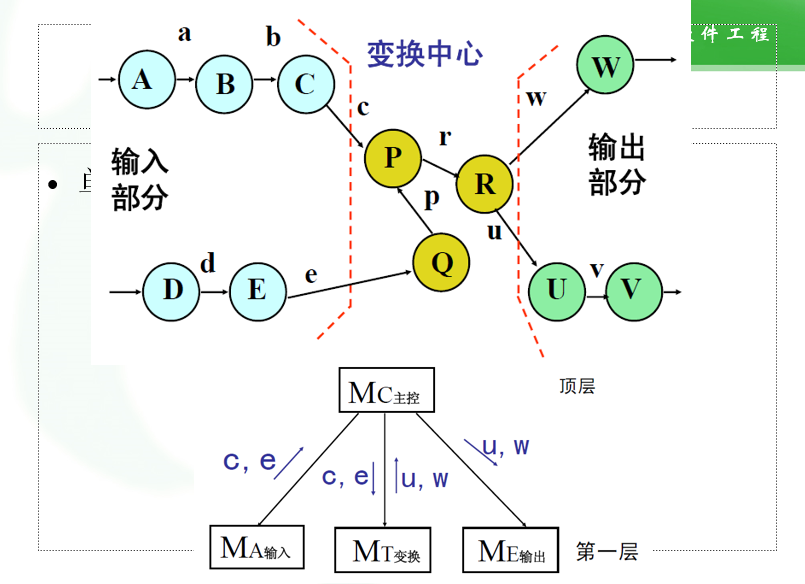
## 掌握映射数据流到“调用和返回”体系结构的方法

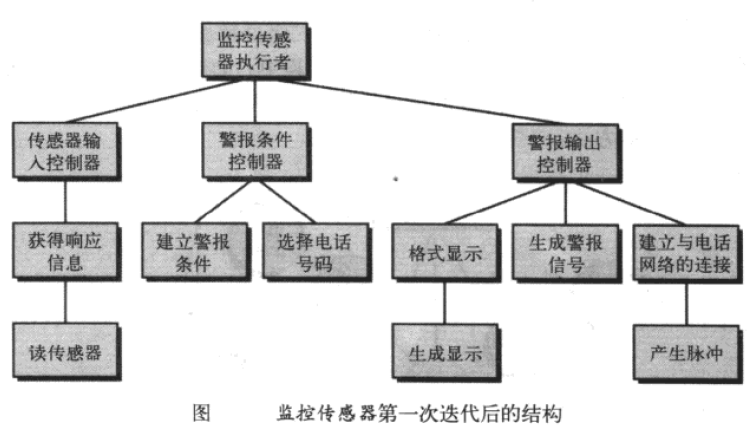
PPT 8.6

分为变换映射和事务映射

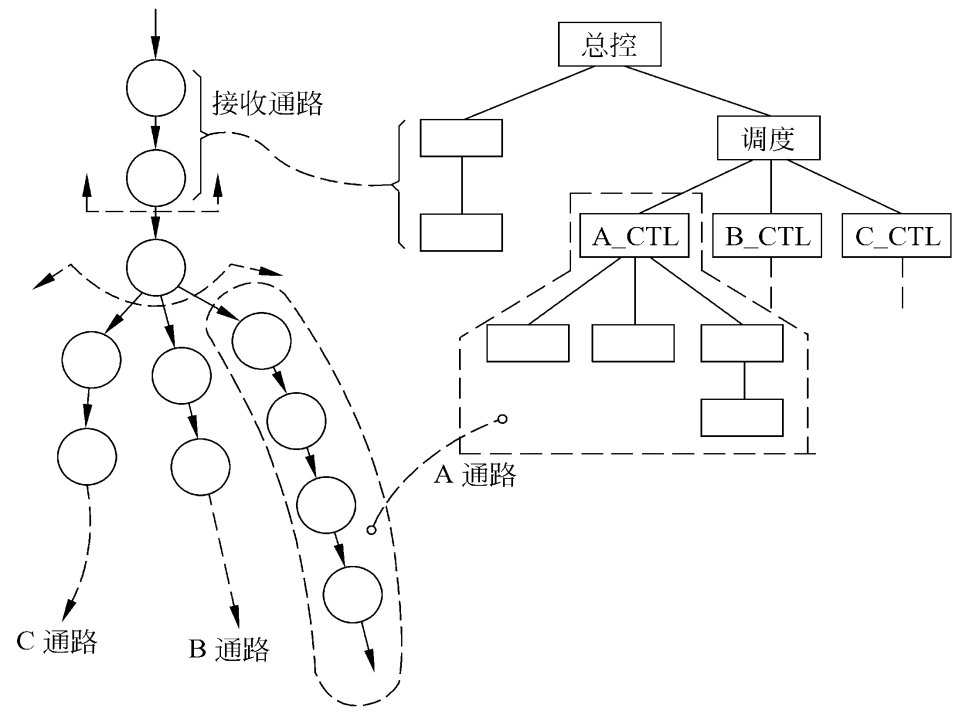
### 变换映射

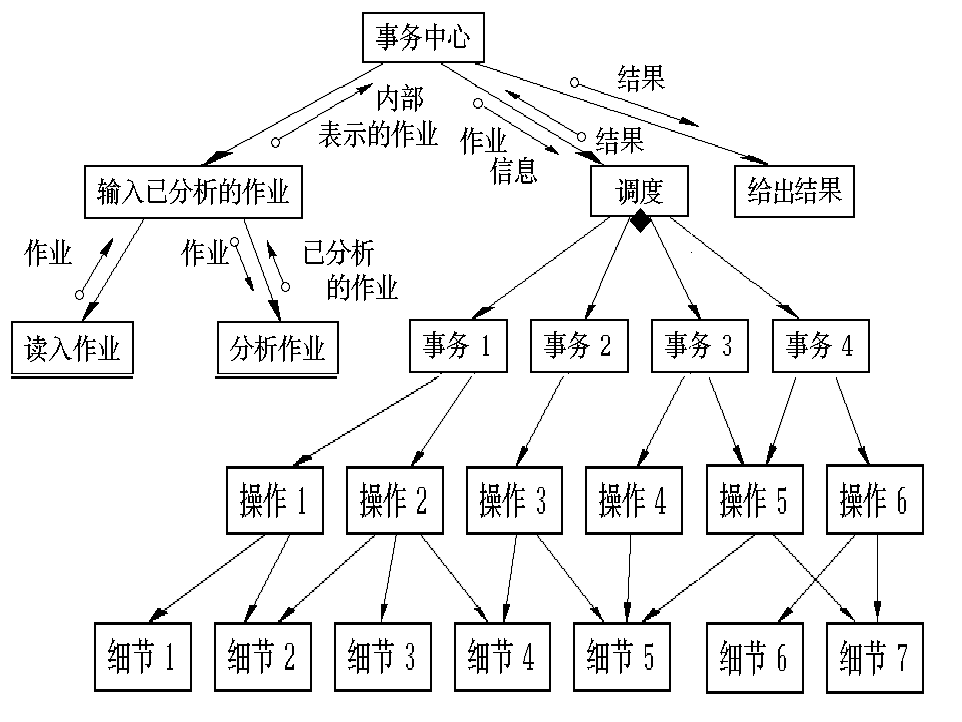


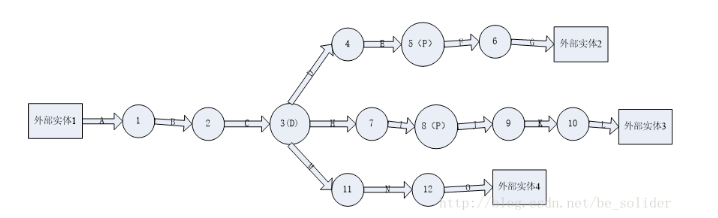


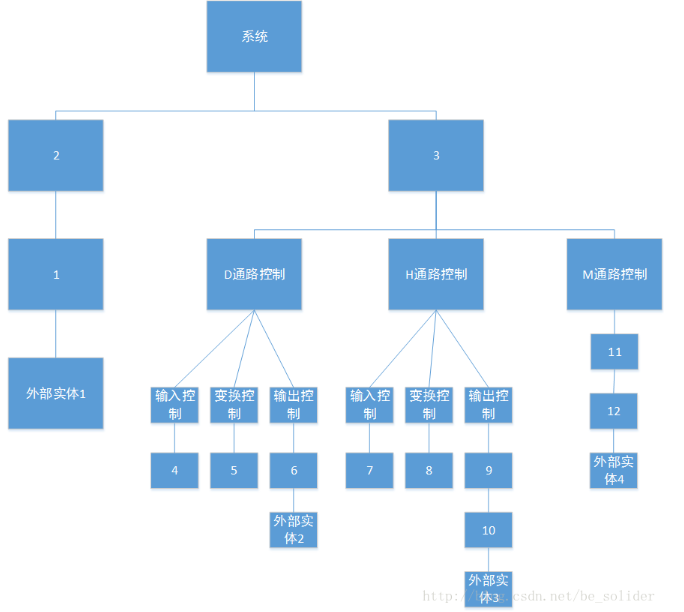


### 事务映射









# 构件级设计建模

## 理解什么是构件

通俗地讲，**构件是一段程序**，该程序能完成一个相对独立的功能，并有一定的通用性。

它是软件中一个模块化的构件块。

正式定义：系统中某一模块化的、可部署的和可替换的部件，该部件封装了实现并暴露一系列接口。

## 了解构件设计的基本原则

### 开关原则（The Open-Closed Principle ,OCP）：

**模块应该对外延有开放性，对修改具有封闭性。**

即设计者应该采用一种无需对构件自身内部（代码或者内部逻辑）做修改就可以进行扩展（在构件所确定的功能域内）的方式来说明构件。

### Liskov替换原则（Liskov Subsitution Principle, LSP）：

**子类可以替换它们的基类**

* + - * 1. 源自基类的任何子类必须遵守基类与使用该基类的构件之间的隐含**约定**（前置条件、后置条件）。

### 依赖倒置原则（Dependency inversion principle, DIP）：

**依赖于抽象，而非具体实现。**

构件依赖的其它构件（不是依赖抽象类，如接口）愈多，扩展起来就愈困难

抽象可以比较容易地对设计进行扩展。

### 接口分离原则(Interface Segregation principle, ISP)：

**多个用户专用接口比一个通用接口要好**

设计者应该为每一个主要的客户类型都设计一个特定的接口。

如SafeHome中FloorPlan类用于安全和监督功能，两处操作有些不同，监视功能多了关于摄像头的操作，定义两个接口。

### 将多个构件组织起来的原则（打包原则）：

**发布复用等价性原则-**--对类打包管理，同时升级。

**共同封装原则**----一同变更的类应该合在一起。

**共同复用原则**----可能一起被复用的类才能打包到一块。

## 理解构件设计中要完成的任务1-7步

### 一个典型的构件级设计步骤：

标识出所有与问题域相对应的类

确定所有与基础设施域相对应的类

* + - * 1. GUI构件、操作系统构件、对象和数据管理构件

细化所有不能作为复用构件的类

* + - * 1. 说明消息的细节流
        2. 为每个构件确定适当的接口
        3. 细化属性并定义数据类型和结构
        4. 描述每个操作中的处理

说明持久数据源（数据库或文件）等相关类

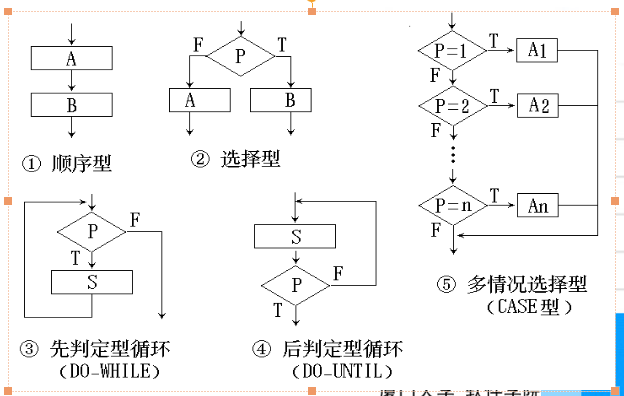
开发并细化类的行为表示

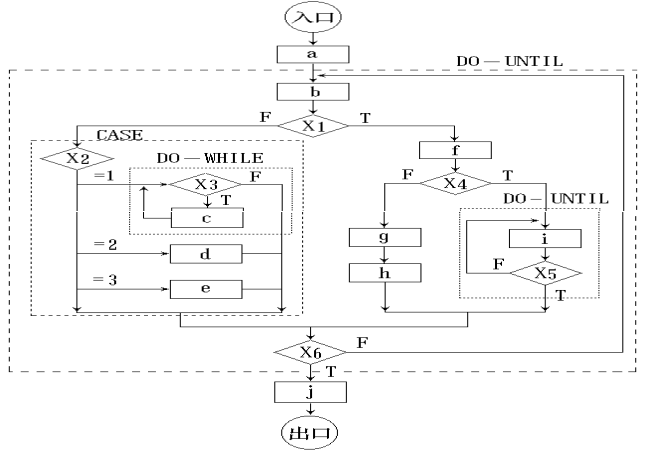
细化部署图

反省和检查现有的设计

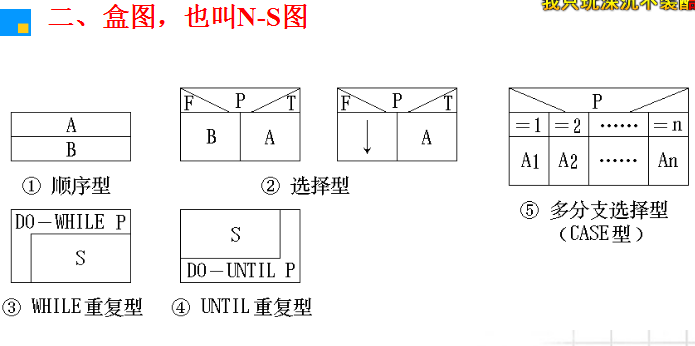
## 掌握传统构件设计的方法：会画程序流程图、盒图(N-S图)、PAD图（了解）

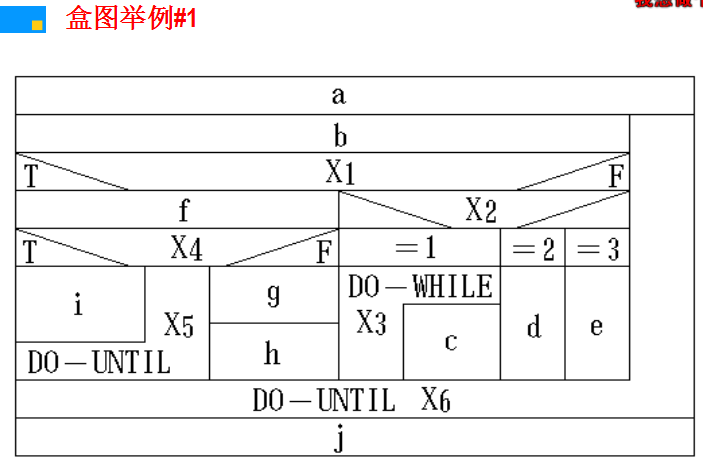
### 流程图：





### 盒图：

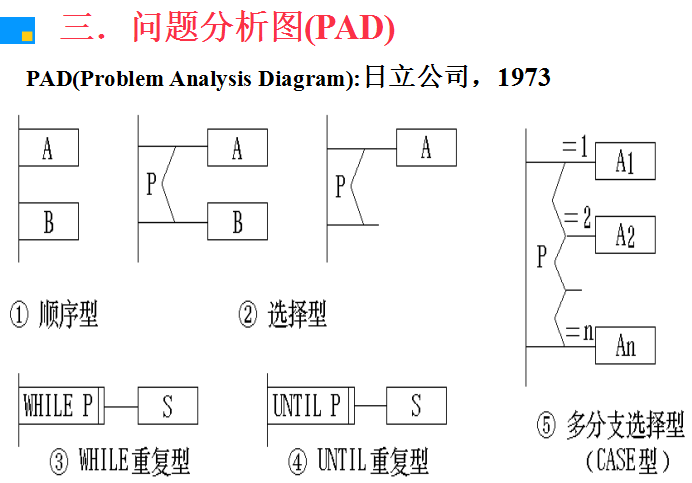


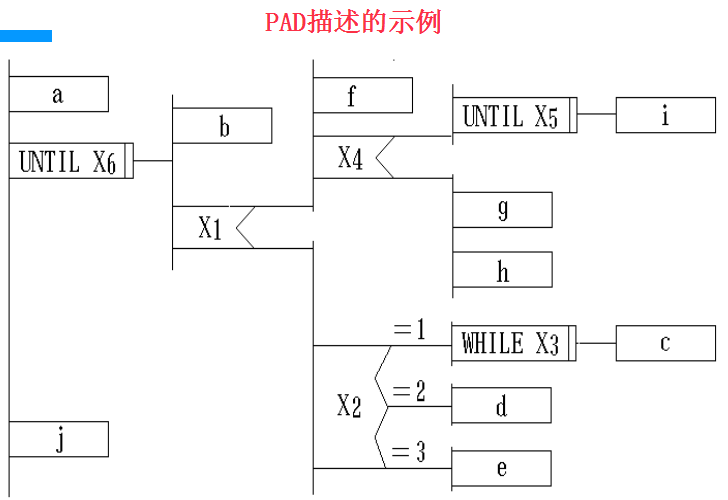


### PAD图：

**PAD特点：**

* + - * 1. 结构清晰，层次分明，易读；
        2. 支持逐步求精的设计思想；
        3. 容易将PAD自动转换为高级语言源程序。





# 完成用户界面设计

## 掌握人机界面设计的黄金规则

### 置于用户的控制之下

用户希望**控制计算机**，而不是被计算机控制

能充分满足用户的需求，具体表现在：

* + - * 1. 以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式。
        2. 提供灵活的交互
        3. 允许用户交互被中断和撤销
        4. 当技能级别增长时可以使交互流线化并允许定制交互—如宏
        5. 使用户与内部细节隔离开来
        6. 设计应允许用户与出现在屏幕上的对象直接交互

### 减少用户的记忆负担

减少对短期记忆的要求

建立有意义的缺省

定义直观的快捷方式

界面的视觉布局应该基于真实世界的象征

以不断进展的方式揭示信息—不同抽象层次

### 保持界面一致

允许用户将当前任务放入有意义的环境中—提供指示器，如窗口题目、图标、一致的颜色编码等

在应用系统家族内保持一致

和用户已有的合理的交互模型保持一致—比如一些常用快捷键

## 理解人机界面设计中要理解哪些元素？

### 用户分析

* 用户特性分析：

用户类型

外行型； 初学型； 熟练型； 专家型（了解系统内部构造，要求提供修改和扩充系统能力的界面）

用户特性度量

使用的频度； 需要自由选择界面的情况；思维能力；生理能力和技能

### 任务分析和建模

也称用户工作分析

是系统内部活动的分解

分析方法：自顶向下，逐步进行功能分解

步骤：

* + - * 1. 任务分配：用户的任务、计算机的任务、两者混合的任务
        2. 对象细化
        3. 工作流分析
        4. 层次表示

### 显示内容分析

要考虑内容的格式和美感

* + - * 1. 不同类型的数据是否放置在固定位置？如照片
        2. 用户是否可定制内容的屏幕位置？
        3. 是否对所有内容赋予适当的屏幕标识？
        4. 为了便于理解，应如何划分长篇报告？
        5. 对于大集合的数据，是否存在直接移动到摘要信息的机制？
        6. 输出图形的大小是否需要适合所使用显示设备的限制？
        7. 如何使用颜色来增强理解？
        8. 出错信息和警告应如何呈现给用户？

### 工作环境分析

物理环境分析

工作场所的文化氛围

## 了解一些界面设计模式

### 完整用户界面。为高层结构和导航提供设计指导

* + 模式：***高层导航***
  + 简要描述：提供高层菜单，通常带有一个图像，能够直接掉转到任一个系统主要功能

### 页面布局。负责页面概括组织（用于站点）或者清楚的屏幕显示（用于需要进行交互的应用系统）

* + 模式：***层叠***
  + 简要描述：呈现层叠状的标签卡，伴随着鼠标每一下点击的选择，显示指定的子功能或者分类内容。

### 表格和输入。考虑了完成表格级输入的各种设计方法。

* + 模式：***填充空格***
  + 简要描述：允许在“文本框”中填写文字与数字数据。

### 表。为创建和操作各种列表数据提供设计指导。

* + 模式：***有序表***
  + 简要描述：用来显示长记录列表，可以在任何一列上选择排序机制进行排序。

### 直接数据操作。解决数据编辑、数据修改和数据转换问题。

* + 模式：***现场编辑***
  + 简要描述：为显示位置上的特定类型内容提供简单的文本编辑能力。

### 导航。辅助用户在层级菜单、Web页面和交互显示屏幕上航行。

* + 模式：***面包屑***
  + 简要描述：当用户工作于复杂层次结构的页面或者屏幕显示时，提供完全的导航路径。

### 搜索。对于网站上的信息或保存在可以通过交互应用访问的持久存储中的数据，能够进行特定内容的搜索。

* + 模式：***简单搜索***
  + 简要描述：提供在网站或者持久数据源中搜索由字符串描述的简单数据项的能力。

### 页面元素。实现Web页面或者显示屏的特定元素

* + 模式：***向导***
  + 简要描述：通过一系列的简单窗口显示来指导完成任务，使得用户能够一次一步地完成某个复杂的任务。

### 电子商务。主要针对于站点，这些模式实现了电子商务应用中的重现元素。

* + 模式：***购物车***
  + 简要描述：提供一个要购买的项目清单。

### 其它。模式不能简单地归类到前面所述的任一类中，在某些情况下，这些模式具有领域的依赖性或者只对特定类别的用户适用。

* + 模式：***进展指示器***
  + 简要描述：为某一正在进行的操作提供进展指示。

# 基于模式的软件设计

### 了解模式的种类

无生产力(nongenerative)的模式

只描述问题和背景，但不提供明确的解决方案。

有生产力(generative)的模式

识别能描述系统中重要的和可重复方面的模式

### 模式的种类

体系结构模式

描述了很多可以用**结构化方法**解决的设计问题

数据模式

描述了重现的面向数据的问题以及用来解决这些问题的数据建模解决方案

构件模式

涉及与开发**子系统和构件**相关的问题、它们之间相互通信的方式以及它们在一个较大的体系中的位置。

界面设计模式

描述公共用户界面问题及具有影响因素的解决方案。

WebApp模式

解决WebApp时遇到的问题，而且往往包括很多前面提到的一些其他模式。

移动模式

描述在开发移动平台的解决方案时碰到的问题。

## 了解用模式思考的基本思路

保证理解全局。

检查全局，提取在此抽象层上表示的模式。

从“全局”模式开始设计，为将来的设计工作建立环境或架构。

“在环境的内部工作”是在更低的抽象层上寻找有助于设计方案的模式。

重复步骤1～4，直到完成完整的设计。

通过使每个模式适应将要构建的软件细节对设计进行优化。

理解**每个模式不是独立的实体**是很重要的。

# 质量概念

## 什么是软件质量

在最一般的意义上，软件质量可以这样定义：

在一定程度上应用有效的软件过程，创造有用的产品，为生产者和使用者提供明显的价值。

软件质量是对明确陈述的功能和性能**需求**、明确记录的**开发标准**以及对所有专业化软件开发应具备的**隐含特征**的符合度。

## 软件质量要素

* **功能性**，子属性：适应性、准确性、互操作性、依从性和安全性
* **可靠性**，子属性：成熟性、容错性和可恢复性
* **易用性**，子属性：可理解性、易学习性和可操作性
* **效率**，子属性：时间表现和资源表现
* **可维护性**，子属性：可分析性、可修改性、稳定性和可测试性
* **可移植性**，子属性：适应性、可安装性和可替代性。

## 如何实现软件质量

### 软件工程方法

采用适当的分析和设计方法

### 项目管理技术

进度管理、风险规划、变更管理等

### 质量控制

如检查代码、一系列的测试步骤、测量等

### 质量保证

质量保证还包含审核和报告功能，用以评估质量控制过程的有效性和完整性

# 质量保证

## 了解软件质量保证的要素

### 标准：IEEE、ISO及其他标准化组织制定了一系列广泛的软件工程标准和相关文件。软件质量保证的任务是要确保遵循所采用的标准，并保证所有的工作产品符合标准。

### 评审和审核：技术评审是由软件工程师执行的质量控制活动，目的是发现错误。审核是一种由SQA人员执行的评审，意图是确保软件工程工作遵循质量准则。如对评审过程审核。

### 测试：软件测试是一种质量控制功能，它有一个基本目标——发现错误。SQA的任务是要确保测试计划适当和实施有效，以便最有可能实现软件测试的基本目标。

### 错误/缺陷的收集和分析：改进的唯一途径是衡量如何做。软件质量保证人员收集和分析错误和缺陷数据，以便更好地了解错误是如何引入的，以及什么样的软件工程活动最适合消除它们。

### 变更管理：变更是对所有软件项目最具破坏性的一个方面。如果没有适当的管理，变更可能会导致混乱，而混乱几乎总是导致低质量。软件质量保证确保进行足够的变更管理实践。

### 教育：每个软件组织都想改善其软件工程实践。改善的关键因素是对软件工程师、项目经理和其他利益相关者的教育。软件质量保证组织牵头软件过程改进，并是教育计划的关键支持者和发起者。

### 供应商管理：可以从外部软件供应商获得三种类型的软件：（1）简易包装软件包(例如微软Office)；（2）定制外壳(通过可以根据购买者需要进行定制的基本框架结构)；（3）合同软件(按客户公司提供的规格说明定制设计和构造)。软件质量保证组的任务是，通过建议供应商应遵循的具体的质量做法，并将质量要求作为与任何外部供应商签订合同的一部分，确保高质量的软件成果。

### 安全防卫：软件质量保证确保应用适当的过程和技术来实现软件安全。

### 安全：软件质量保证可能负责评估软件失效的影响，并负责启动那些减少风险所必需的步骤。

### 风险管理：软件质量保证组应确保风险管理活动适当进行，且已经建立风险相关的应急计划。

## 了解软件质量保证的任务、目标和度量

### 任务：

编制项目质量保证计划。

参与项目的软件过程描述的编写。

评审软件工程活动，以验证是否符合规定的软件过程。

审核指定的软件工作产品以验证是否遵守作为软件过程一部分的那些规定。

确保根据文档化的规程记录和处理软件工作和工作产品中的偏差。

记录各种不符合项并报告给高层管理人员。

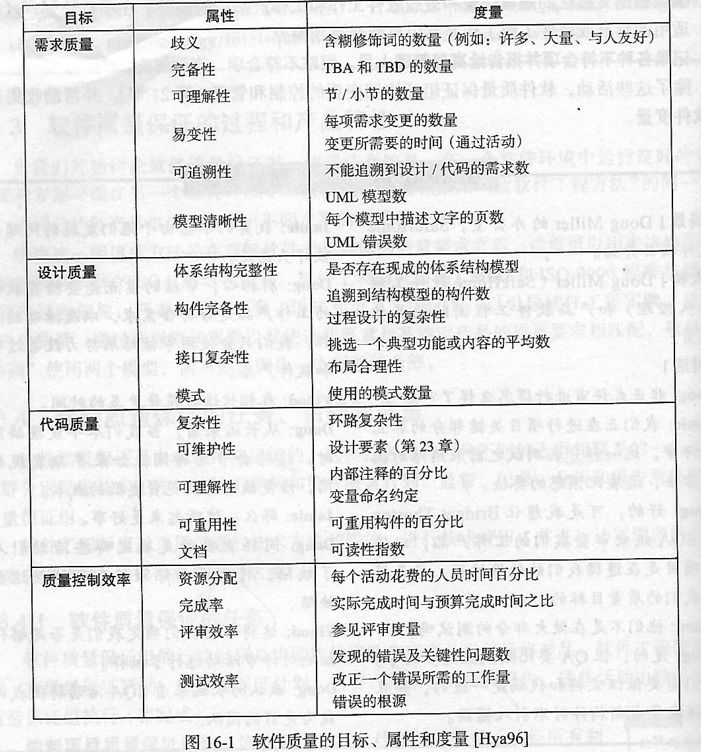
### 目标和度量：

**需求质量**: 软件质量保证必须确保软件团队严格评审需求模型，以达到高水平的质量。

**设计质量**:软件团队应该评估设计模型的每个元素，以确保设计模型显示出高质量，并且设计本身符合需求。SQA寻找能反映设计质量的属性。

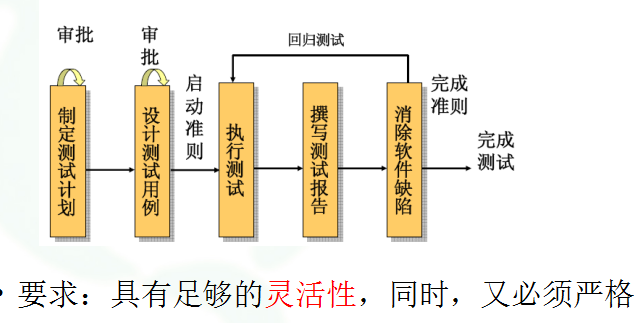
**代码质量**。源代码和相关的工作产品必须符合本地的编码标准，并显示出易于维护的特点。SQA应该找出那些合理分析代码质量的属性。

**质量控制有效性**。软件团队应使用有限的资源，在某种程度上最有可能得到高品质的结果。SQA分析在评审和测试上的资源分配，评估是否以最有效的方式进行分配的。



# 软件测试策略

## 理解软件测试的步骤



## 软件测试策略 （详细见PPT14章）

单元测试：目标；侧重点

集成测试：目标；各种集成方法及优缺点

确认测试：目标；α测试和β测试

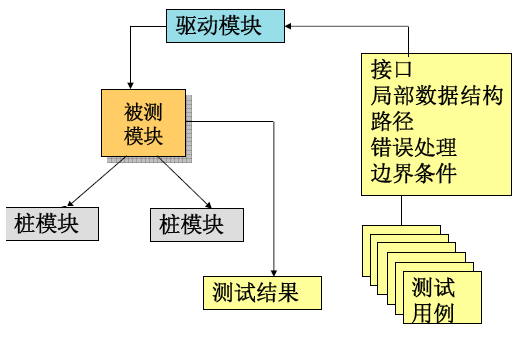
系统测试：目标

### 传统单元测试

侧重点：

* + - * 1. 侧重于软件设计的最小单元（软件构件或模块）的验证工作
        2. 侧重于构件的内部处理逻辑和数据结构
        3. 侧重于模块的算法细节和穿过模块接口的数据
        4. 着重从以下几个方面对模块进行测试：**模块接口、局部数据结构、重要的执行通路、出错处理通路**和**边界条件**等。

目标：利用构件级设计秒杀后作为指南，测试重要的控制路径以发现模块内的错误。



### 传统集成测试

侧重：旨在发现与接口相关的错误

目标：利用已通过单元测试的构件建立设计中描述的程序结构

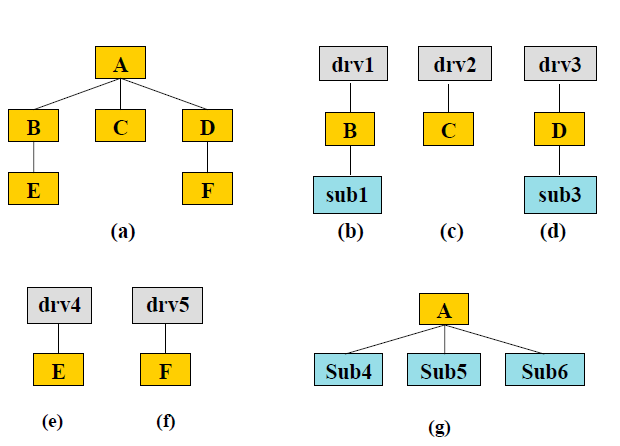
**一步到位**与**增量集成**

一次到位：

* + - * 1. 又称为非增式组装，一次性组装或大爆炸集成，这种集成将所有单元在一起编译并进行一次性测试。
        2. 缺点：

一次成功的可能性不大

当发现缺陷时，没有多少线索能够用来帮助确定缺陷位置。



增式组装

逐步组装成较大的系统，边组装边测试，分为：（具体见14章PPT 25页）

* + - * 1. 自顶向下的增殖方式

缺点：需要建立桩模块；重要而复杂的算法模块一般在底层

优点：较早的发现主要控制方面的问题

* + - * 1. 自底向上的增殖方式

优点：不需要建立桩模块，建立驱动模块一般比建立桩模块容易；容易出问题的部分在早期解决；易于并行测试，提高效率

缺点：最后才接触到主要的控制

### 面向对象中的单元测试

测试对象是类，类包含属性、操作等，有些类之间有类似的属性与操作，此时可以考虑同时测试这些指标

面向对象的软件的类测试，是由封装在该类中的操作和类的状态行为驱动的

### 面向对象中的集成测试

**基于线程的测试**

集成**响应系统的一个输入或事件**所需的一组类，每个线程被集成并分别测试，应用回归测试以保证没有产生副作用。

**基于使用的测试**

* + - * 1. 通过测试那些几乎不使用服务器类的类(称为独立类)来开始系统的构造。
        2. 在独立类测试完成后，下一层的使用独立类的类，称为依赖类，被测试。
        3. 这个依赖类层次的测试序列一直持续到构造完整个系统。

### 确认测试

在传统软件和面向对象软件间没有明显差别。

**始于**集成测试的结束。

验证软件的功能和性能及其它特性是否与**用户要求**一致。

步骤：

* + - * 1. 进行**有效性测试（**黑盒测试）：

制定测试计划、测试步骤，设计测试用例

确定所有的功能、性能均满足要求

* + - * 1. 软件**配置复查**
        2. α测试和β测试
        3. 验收测试：由用户用实际数据进行测试。考虑软件的功能、性能、可移植性、兼容性、可维护性
        4. α测试：

由一个用户在**受控环境下**进行的测试。

最终用户在开发者的场所进行。

目的是评价软件产品的FLURPS（功能、局域化、可使用性、可靠性、性能、支持），产品的界面和特色.

* + - * 1. β测试：

由多个用户在**实际使用环境下**的测试。

用户定期向开发者报告软件运行的问题。

主要衡量产品的FLURPS

* + - * 1. 确认测试完成的标志：

功能和性能与用户的要求一致，用户接受

应交付的文档：

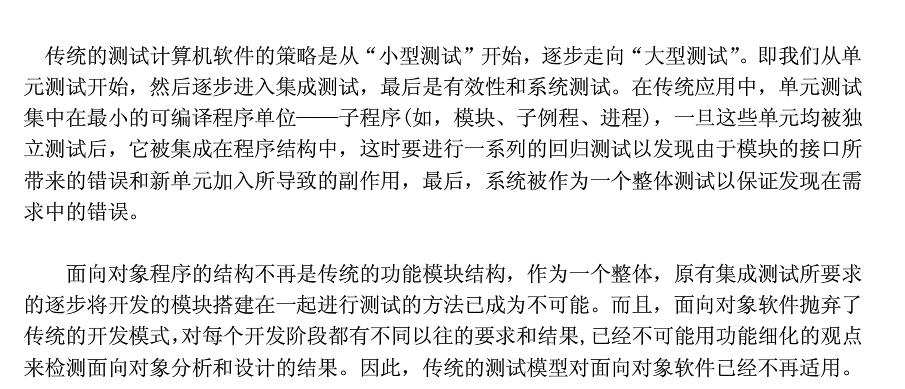
确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告。

### 系统测试

系统测试是基于**实际应用环境**对计算机系统的一种多方位的测试，每一种测试都具有不同的目的，但所有的测试都是为了检验各个系统成分能否正确集成到一起并且是否能完成预定的功能。

包括：恢复测试、安全测试、压力测试、性能测试、部署测试

## 面向对象的测试策略的不同点（课本P284左右，下面答案来自网络）

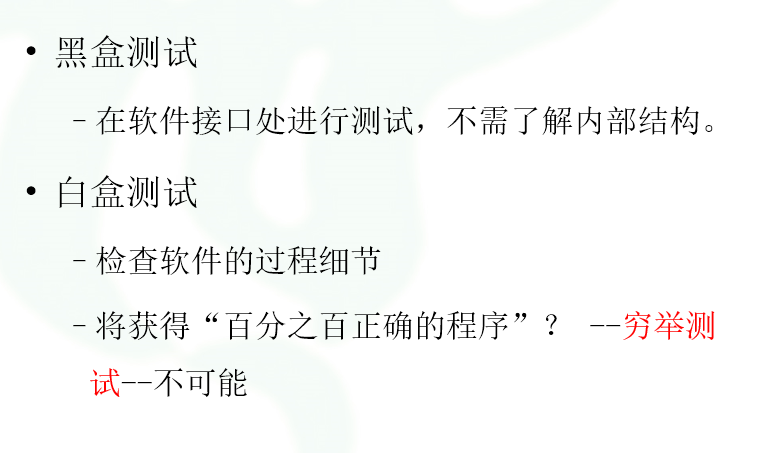


# 测试战术

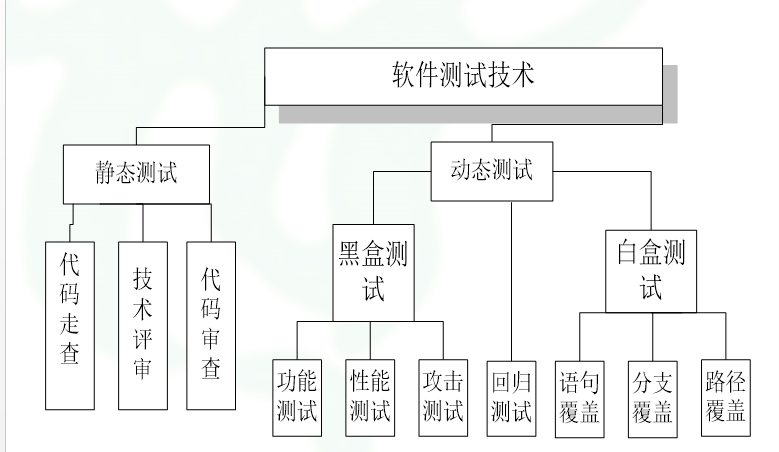
## 测试技术分类

### 从是否需要执行被测试软件的角度分类（静态测试和[动态测试](http://baike.baidu.com/view/106213.htm)）。

### 从测试是否针对[软件结构](http://baike.baidu.com/view/600142.htm)与算法的角度分类（白盒测试和黑盒测试）。

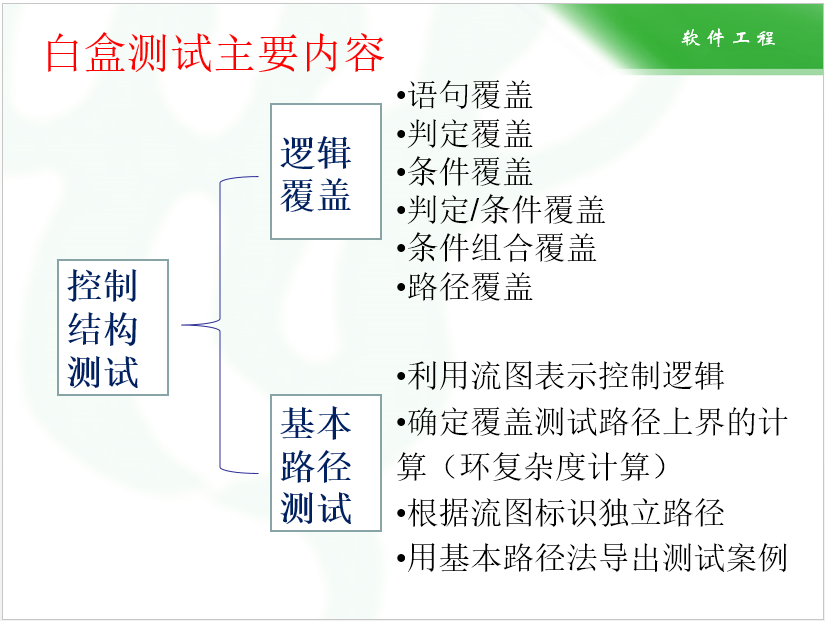


### 从测试的不同阶段分类（单元测试、集成测试、系统测试、验收测试）。



## 掌握白盒测试的主要方法，并能根据要求设计测试用例

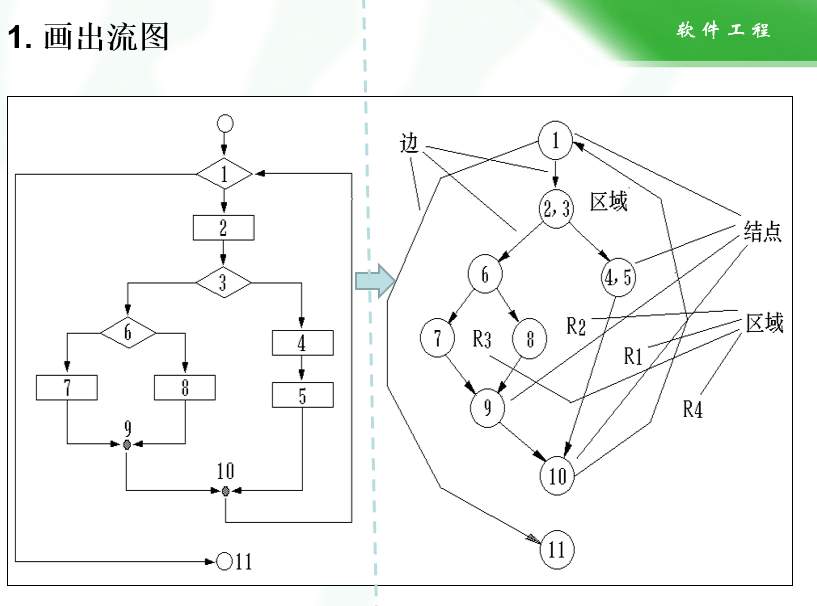
主要：条件组合测试、**基本路径测试**

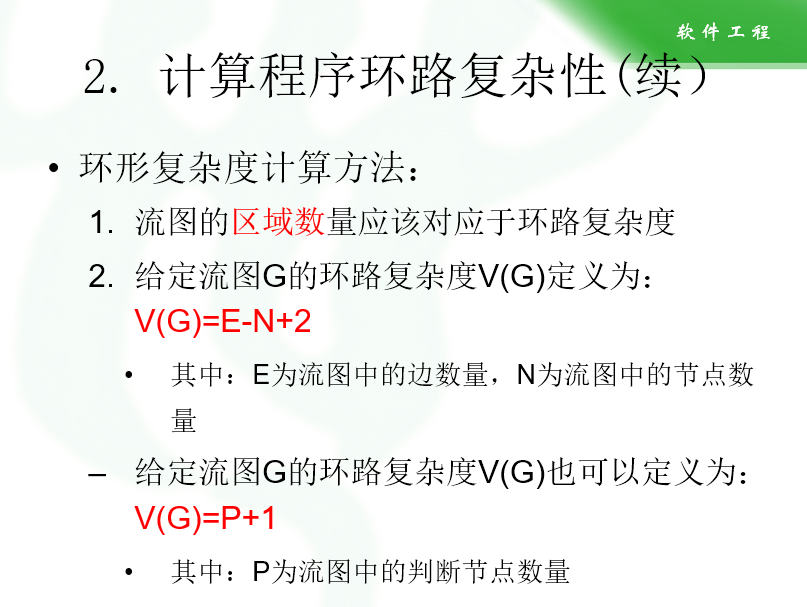


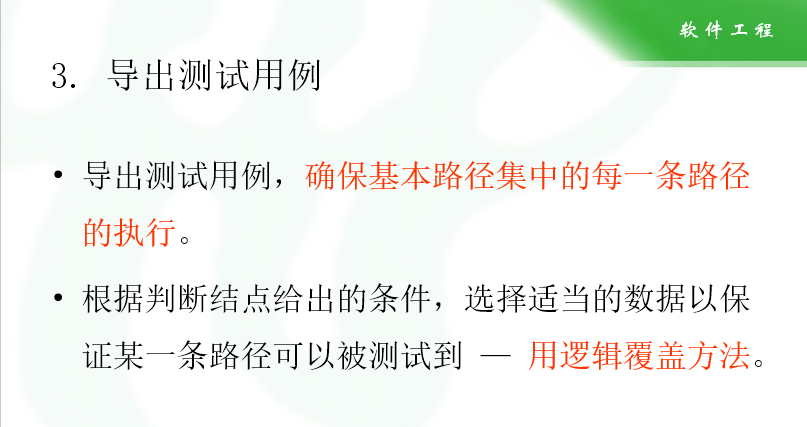
### 基本路径测试：

基本路径测试方法把覆盖的路径数压缩到一定限度内，**程序中的循环体最多只执行一次**。

它是在程序控制流图的基础上，**分析控制构造的环路复杂性**，**导出基本可执行路径集合，设计测试用例的方法**。设计出的测试用例要保证在测试中，程序的每一个可执行语句至少要执行一次。







### 条件组合测试：

每个判定表达式中条件的各种可能组合都至少出现一次。

## 掌握黑盒测试的主要方法，并能根据要求设计测试用例：

主要：**等价类划分**、边界值、错误推测法

### 等价类划分：

使用这一方法设计测试用例要经历**划分等价类**（列出等价类表）和**选取测试用例**两步。

等价类的划分有两种不同的情况：

* + - * 1. 有效等价类：是指对于程序的规格说明来说，是合理的，有意义的输入数据构成的集合。
        2. 无效等价类：是指对于程序的规格说明来说，是不合理的，无意义的输入数据构成的集合。

在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。

* + - * 1. 设计一个新方案以**尽可能多地覆盖**尚未被覆盖的有效等价类；重复这一步骤直到所有有效类都被覆盖为止
        2. 设计一个新方案以覆盖**一个且仅一个**尚未被覆盖的无效等价类；重复这一步骤直到**所有无效类**都被覆盖为止。（通常程序执行一个错误后即不继续检测其它错误，故每次只测一个无效类）





### 边界值分析：

边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。通常边界值分析法是作为对等价类划分法的补充，这种情况下，其测试用例来自等价类的边界。

怎样用边界值分析法设计测试用例？

* + - * 1. 首先确定边界情况。通常输入或输出等价类的边界就是应该着重测试的边界情况。
        2. 选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值。

### 错误推测法：

错误推测法的基本想法是：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。

## 理解面向对象的继承相关的测试、随机测试和类间测试

继承并不能排除对所有派生类进行全面测试的需要

# 安全性工程

## 资产都包括什么？

资产包括数据信息、文件、程序、硬盘驱动器的存储空间、系统内存甚至处理器的容量。

## 什么是显露度

资产损失的价值被称为显露度。

## 安全性分析包括什么？

安全性分析包括**需求获取、威胁建模、风险分析、测度设计和正确性检查**。除去商业理由以外，这些任务包括系统的功能性和非功能性细节的考量。

# 软件配置管理

## 了解软件配置项内容（写的各种文档都是）

软件过程的输出信息主要分为三类：

* + - * 1. **计算机程序**（源代码和可执行程序）
        2. **描述计算机程序的文档**（针对技术开发者和用户）
        3. **数据**（包含在程序内部的数据，或程序外部的数据）

在软件过程中产生的所有信息项总称为软件配置

## 理解配置管理包含的内容（看书P307）

软件配置管理（Software Configuration Management, SCM）

是指通过执行版本控制、变更控制等规程，以及使用合适的配置管理软件，来保证所有配置项的完整性和可跟踪性。配置管理是对工作成果的一种有效保护。

## 理解基线的定义

由正式技术评审而得到的软件配置项(SCI)的正式文本构成了基线。

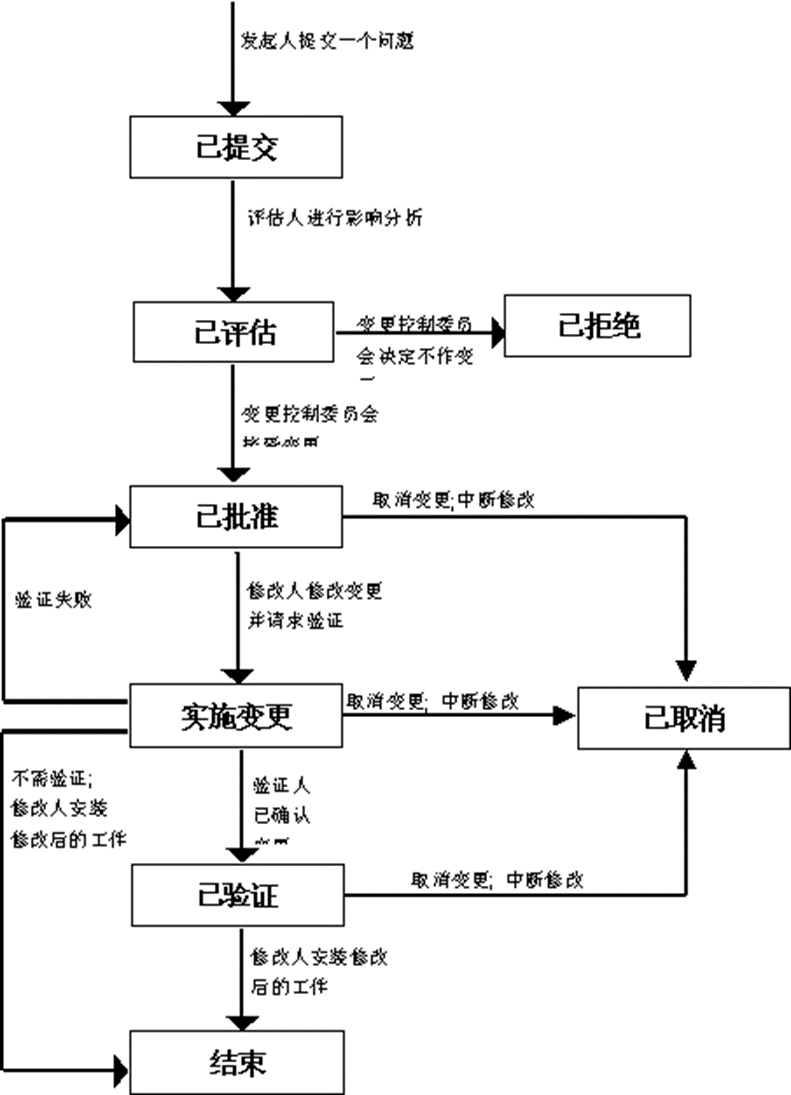
基线定义：已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能够修改它。

基线中的配置项被“**冻结**”了，**不能随意修改**

## 理解变更控制流程（不需要画那个图，但是要理解意思）

变更控制的目的是防止配置项被随意修改而导致混乱。

P316



# 项目管理概念

## 理解项目管理涉及的范围（4个P）

人员、产品、过程、项目

### 人员

“ 人的因素”非常重要

美国卡内基.梅隆大学开发了一个人员管理能力成熟度模型，定义以下关键的实践区域：

* + - * 1. 招募、选择、业绩管理、培训、报酬、个人事业发展、组织和工作设计以及团队精神或企业文化培养。

### 产品

在制定项目计划之前，首先确定产品的目标和范围，考虑可选的解决方案，识别技术和管理上的限制。

软件开发者和客户必须一同定义产品的目标和范围。

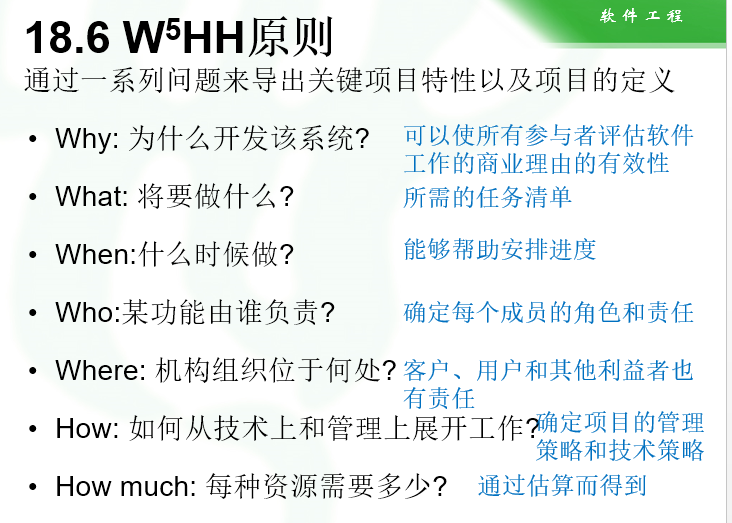
### 过程

在前面提到的软件过程框架下，可以制定软件开发的综合计划

### 项目

为了避免项目失败，必须了解实施成功的项目管理的关键因素，还要确定计划和监控项目的一目了然的方法。

## 了解W5HH原则



## 检查点(Check Point)、里程碑(Mile Stone)、基线(Base Line):意思

### 检查点(Check Point):

* + 指在规定的时间间隔内对项目进行的检查与复审工作，通过比较实际进展与计划进度的差距，并根据差距进行调整。

### 里程碑(Mile Stone):

* + 完成阶段性工作的标志，往往是一些重要活动的完工，或重要文档的交付，或阶段评审的通过。

### 基线(Base Line):

* + 指一个（或一组）配置项在项目生命期的不同时间点上通过正式评审而进入正式受控的一种状态。基线其实是一些重要的里程碑。基线一旦建立后，以后的任何更改都需要受到控制。

# 过程和项目度量（看书）

## 理解项目度量的概念和项目度量的步骤

软件度量是对收集到的原始数据，采用一些数学函数来计算，以测量过程、项目、产品的性能。度量范围分为：过程度量和项目度量。

项目度量的步骤：

* + - * 1. 建立历史数据基线；
        2. 对工作量等的估算；
        3. 将实际工作量等的测量与估算值比较，以控制项目的进度；
        4. 收集技术度量、评价设计质量、测试等的方法。
        5. 记录和跟踪所发现的错误；
        6. 补充历史数据基线。

## 掌握项目度量方法有哪些；掌握面向功能的度量和面向规模的度量，掌握功能点的计算方法（5各方面+14个因子）

### 面向对象的度量

### 面向用例的度量

### 面向规模的度量：

面向规模的软件度量是通过规范化质量和/或生产率的测量而得到的，这些测量基于所生产软件的“规模”。



每个项目产生一组简单的面向规模度量：

* + - * 1. 每千行代码(KLOC)的错误数。
        2. 每千行代码(KLOC)的缺陷数。
        3. 每行代码(LOC)的成本。
        4. 每千行代码(KLOC)的文档页数。
* 除此之外，还能够计算出其他有意义的度量：
  + - * 1. 每人月错误数。
        2. 每人月代码行(LOC)。
        3. 每页文档的成本。

### 面向功能的度量

面向功能度量用一种称为“**功能点**”(Function Point, FP)的测量。

使用软件所提供的功能的测量作为规范化值。

因为“功能”不能直接测量，所以必须通过其他直接的测量来导出。

**功能点**是基于软件信息领域的可计算的(直接的)测量及软件复杂性的评估而导出的。

5个方面：

* + - * 1. **用户输入数**：计算每个用户输入，它们向软件提供面向应用的数据。输入应该与查询区分开来，分别计算。
        2. **用户输出数**：计算每个用户输出，它们向用户提供面向应用的信息。输出是指报表、屏幕、出错信息，等等。
        3. **用户查询数**：一个查询被定义为一次联机输入，它导致软件以联机输出的方式产生实时的响应。每一个不同的查询都要计算。
        4. **文件数**：计算每个逻辑的主文件(如数据的一个逻辑组合，它可能是某个大型数据库的一部分或是一个独立的文件)。
        5. **外部接口数**：计算所有机器可读的接口(如磁带或磁盘上的数据文件)，利用这些接口可以将信息从一个系统传送到另一个系统。

采用下面的方式计算功能点：

* + - * 1. FP=总计数值×［0.65＋0.01×ΣFi］
        2. 其中“总计数值”是从上页图中得到的所有条目的总和。
        3. Fi (i=1 到14)是基于对下面的问题的回答而得到的“复杂度调整值”(0 到5)。
        4. 等式中的常数和信息域值的加权因子是根据经验确定的。

Fi考虑因素；

* + - * 1. 系统需要可靠的备份和复原吗？
        2. 需要数据通信吗？
        3. 有分布处理功能吗？
        4. 性能很关键吗？
        5. 系统是否在一个已有的、很实用的操作环境中运行？
        6. 系统需要联机数据项吗？
        7. 联机数据项是否需要在多屏幕或多操作之间切换以完成输入？
        8. 需要联机更新主文件吗？
        9. 输入、输出、文件或查询很复杂吗？
        10. 内部处理复杂吗？
        11. 代码需要被设计成是可复用的吗？
        12. 设计中需要包括转换及安装吗？
        13. 系统的设计支持不同组织的多次安装吗？
        14. 应用的设计方便用户修改和使用吗？

*Fi*（*i*＝1..14）是复杂性校正值，它们应通过逐一回答上述提问来确定。

*Fi*的取值0..5：

0 没有影响 1 偶然的

2 适中的 3 普通的

4 重要的 5 极重要的

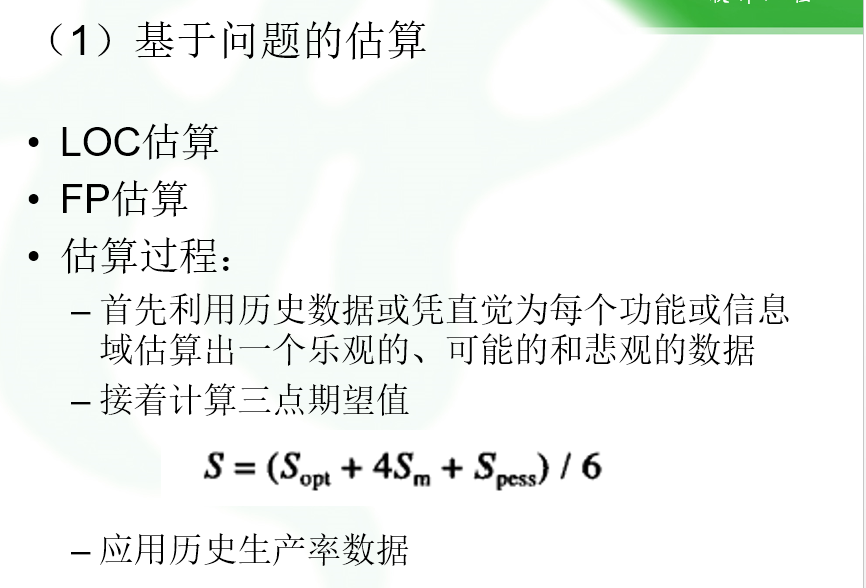
*SUM*（*Fi*）是求和函数。

# 软件项目估算

## 掌握软件规模估算的方法：（看书）

掌握基于问题的估算；了解基于过程的估算；了解基于用例的估算

### 基于问题的估算





### 基于过程的估算：

将过程分解为一组较小的任务，并估算完成每个任务所需的工作量

将各阶段的平均劳动力价格应用于每个软件过程活动

计算每个功能及框架活动的成本和工作量

### 基于用例的估算：

* 用例可以用于估算，但只有在用例描述的“**结构层次”**的情况下考虑
  + 该结构层次中每一层次都可以由不超过10个用例来描述，而每个用例包括的场景不超过30个

## 了解估算模型的总体结构，了解COCOMO Ⅱ（3个方面，最后算出来一个对象点）

### 估算模型总体结构

典型的估算模型是通过回归分析从以往软件项目中收集到的数据而得到的。这种模型的总体结构： E = A + B×(ev) C

其中，A、B、C为经验常数，E是工作量（人.月）、ev是估算变量（LOC或FP）



### COCOMO Ⅱ应用组装模型：

计算对象点

* + - * 1. （用户界面的）屏幕数
        2. 报表数
        3. 构造应用可能需要的构件数

确定复杂度

* + - * 1. 将每个对象实例归类到三个复杂度级别之一：
    - 简单；中等；困难
      * 1. 复杂度是以下变量的函数：客户和服务器数据表的数量和来源，以及视图或版面的数量

根据复杂度，对屏幕、报表和构件的数量进行加权。

若采用基于构件的开发或一般的软件复用时，还要估算复用的百分比，并调整对象点数

NOP= 对象点\*[（100-复用的百分比）/100]

* + - 其中NOP为新的对象点

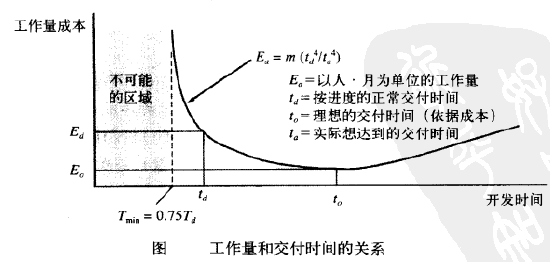
根据NOP进行工作量估算时，先确定”生产率” PROD=NOP/人.月

估算工作量 = NOP/PROD

# 项目进度安排

## 理解人员和工作量之间的关系（一定不是线性关系）

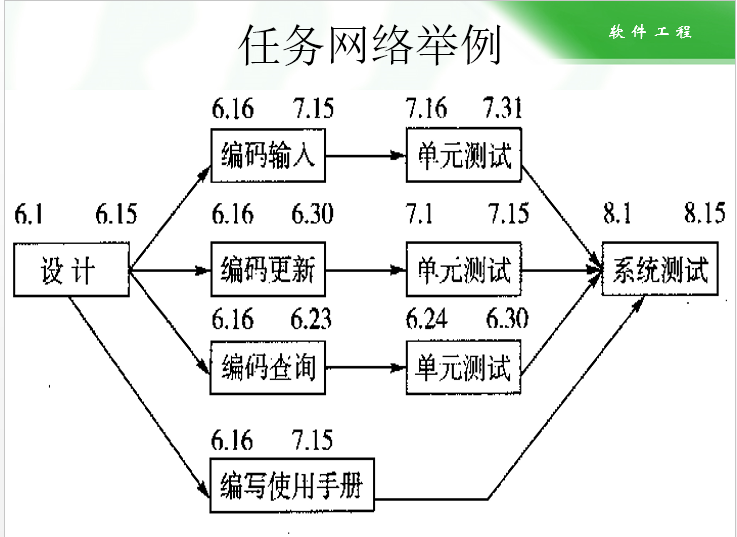
完成项目的时间与投入项目中的人员的工作量之间存在着**高度非线性关系**。

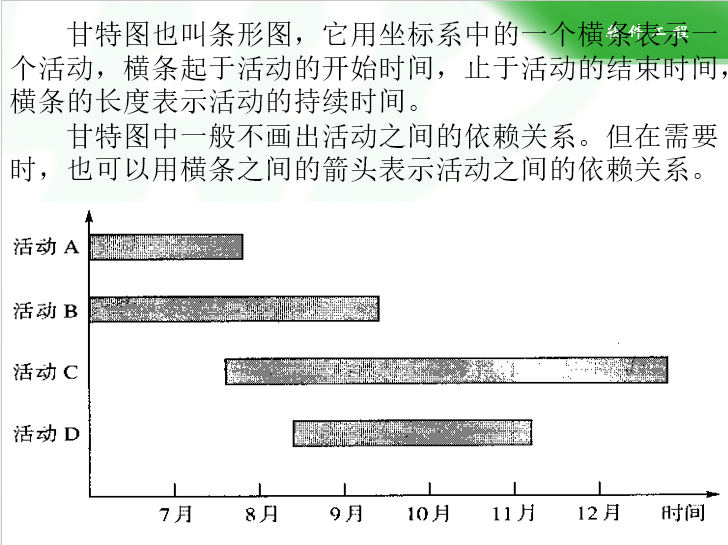


每种软件项目估算技术最终都归结为对完成软件开发所需人月(或者人年)的估算.

在项目后期增加人手通常产生一种破坏性影响，其结果是使进度进一步拖延。

## 会画任务网络和甘特图（进度安排的时候，注意关键路径）

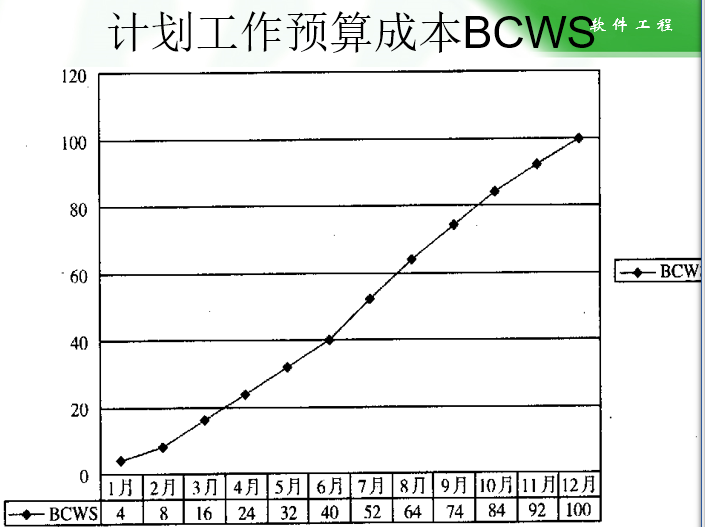




## 理解获得值分析（分析进度偏差和费用偏差）

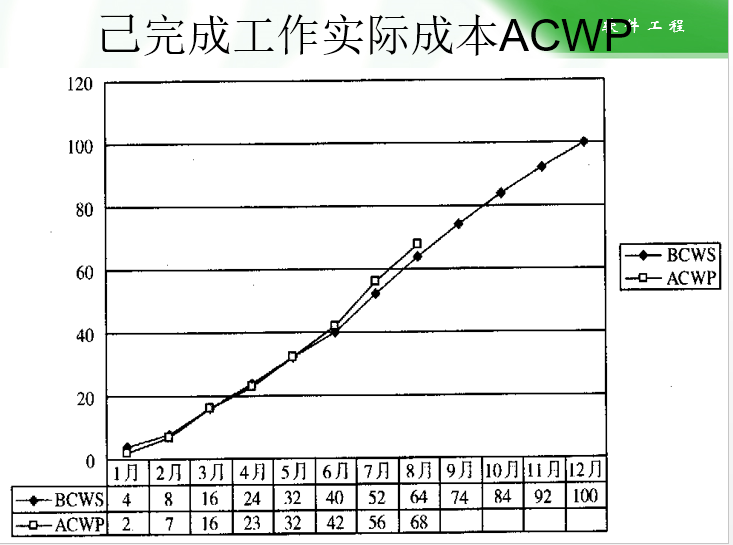
为了进行获得值分析，首先从项目的费用预算中计算出，在项目过程里，到某个时间点为止，预计的费用综合是多少。我们把在规定的时间内所有计划执行活动的**己批准预算费用总和**称为**计划工作预算成本**，记作**BCWS**。

如果将项目过程中每个时间点的BCWS连成一条曲线，就可以得到上图所示的BCWS曲线图。



在项目执行过程中，我们定期记录项目到**当前时间为止的总费用**、我们把后一费用称为**己完成工作实际成本**，记作**ACWP**。

如果把项目过程中获得的每个时间点的ACWP连接在一起，就可以得到下图所示的ACWP曲线。

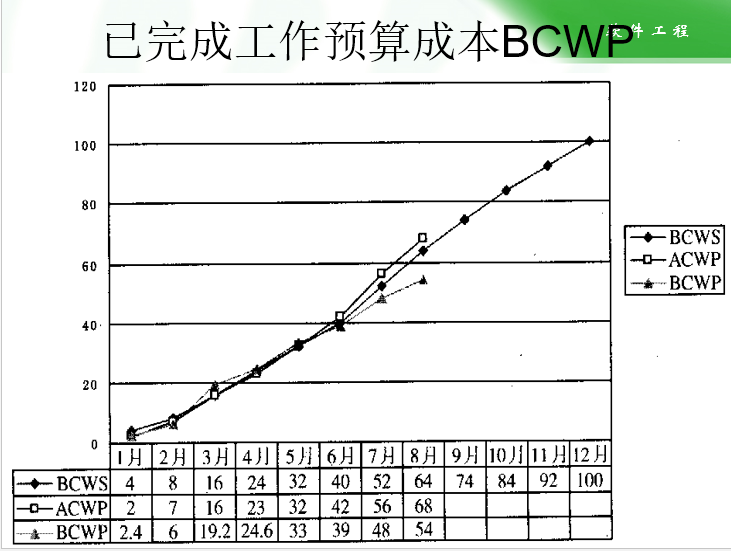


BCWS曲线和ACWP曲线的偏离反映了项目预算费用和实际费用的偏差。但这一偏差不能完全说明项目的费用和进展情况。

要考虑完成的工作量

为了更准确地反映项目实际进程与计划进程的偏差，我们再引入**已完成工作预算成本**的概念。

* + 即，首先记录到某个时间为止，项目已完成的工作量，然后根据我们的费用预算，把这个工作量转换成预算费用，也就是找出在项目预算中，要完成这些工作量，需要多少费用。**已完成工作预算成本**通常被记为**BCWP或EV**，这个数值也就是我们在前面所说的**挣值（盈余值）**了。



有了项目某个时间点的 BCWS、ACWP和 BCWP数据，我们就可以对项目的费用和进展情况进行定量的分析了。

* 例如：
  + - * 1. 项目的**费用偏差**CV＝BCWP－ACWP
        2. 项目的**进度偏差**SV＝BCWP－BCWS
        3. **费用执行情况**指数 CPI＝BCWP/ACWP
        4. **进度执行情况**指数SPI＝BCWP/BCWS
        5. 预测完工费用＝总预算费用／CPI

# 风险管理

## 了解面对风险的两种策略（防火、救火）

### 被动风险策略：

管理风险对项目成功非常关键。人们对待风险有两种态度:

* + 一种是被动态度，可比作**救火模式。**
  + 另一种是主动态度，可比作**防火模式。**

大多数软件项目组还是仅仅依赖于被动风险策略。

* + 最多不过是针对可能发生的风险来监督项目，直到它们发生，才处理。
  + 更普遍的情况是，对风险不闻不问，直到发生了错误，才赶紧采取行动，试图纠正错误。
  + 当这样的努力失败后，“危机管理” 接管一切，这时项目已经处于真正的危机中了。

### 主动风险策略：

* 主动策略早在**技术工作开始之前**就已经启动了。
  + **标识**出潜在的风险，**评估**其出现的概率及产生的影响，且按重要性加以排序，然后，软件项目组建立一个计划来**管理风险**。
* 主要的目标是**预防风险**，但因为不是所有的风险都能够预防，所以，项目组必须建立一个意外事件的计划

## 了解识别风险、预测风险、风险缓解、监控的策略

**识别风险**是试图系统化地确定对项目计划(估算、进度、资源分配)的威胁。

每一类风险又分为两个不同的类型：

* + - * 1. 一般性风险
    - 对每一个软件项目而言都是一个潜在的威胁。
      * 1. 特定产品的风险。
    - 特定产品的风险只有那些对当前项目的技术、人员、及环境非常了解的人才能识别出来。
    - 为了识别特定产品的风险，必须检查项目计划及软件范围说明，并给出以下问题的答案：“本项目中有什么特殊的特性可能会威胁到我们的项目计划？”

风险预测，又称风险估算，试图从两个方面评估每一个风险:

* + - * 1. 风险发生的可能性或概率;
        2. 以及如果风险发生了，所产生的后果。

一个有效的策略必须考虑三个问题：

* + - * 1. 风险避免。
        2. 风险监控。
        3. 风险管理及应急计划。