# 软件工程考点

### **目录**

[**版本控制 2**](#_Toc6849)

[**迭代开发 3**](#_Toc14397)

[**需求管理 4**](#_Toc12289)

[**需求描述 5**](#_Toc13691)

[**UML类图 7**](#_Toc1644)

[**Observe 10**](#_Toc14448)

[**白盒测试 11**](#_Toc6414)

[**黑盒测试 12**](#_Toc9126)

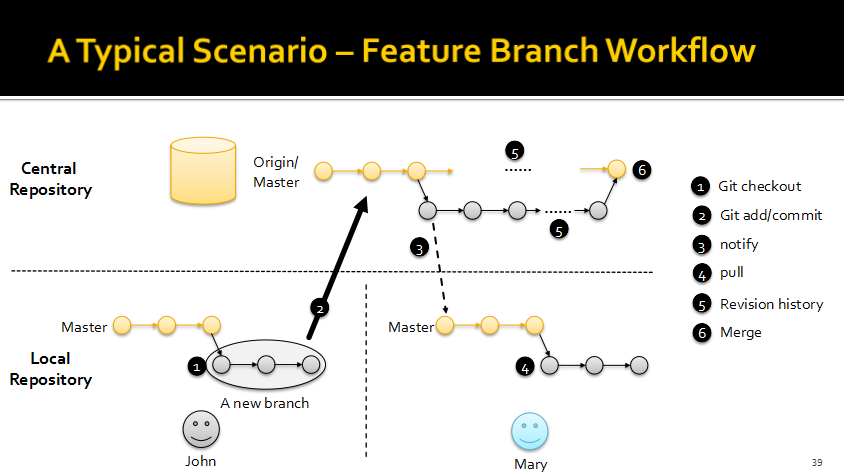
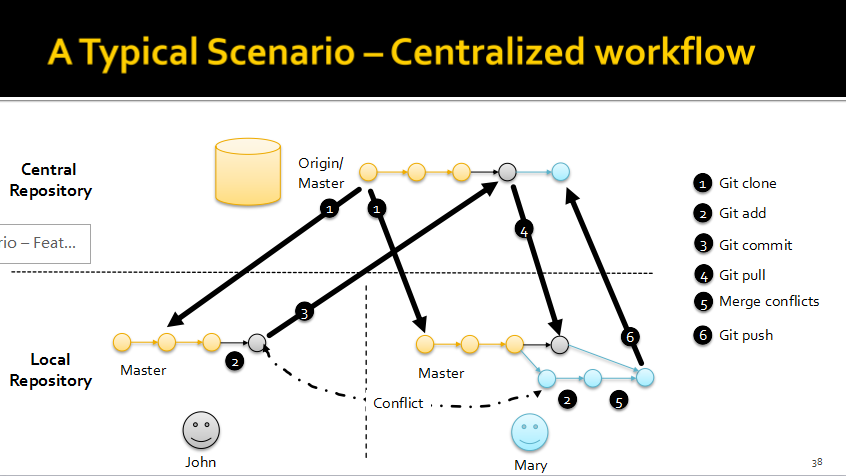
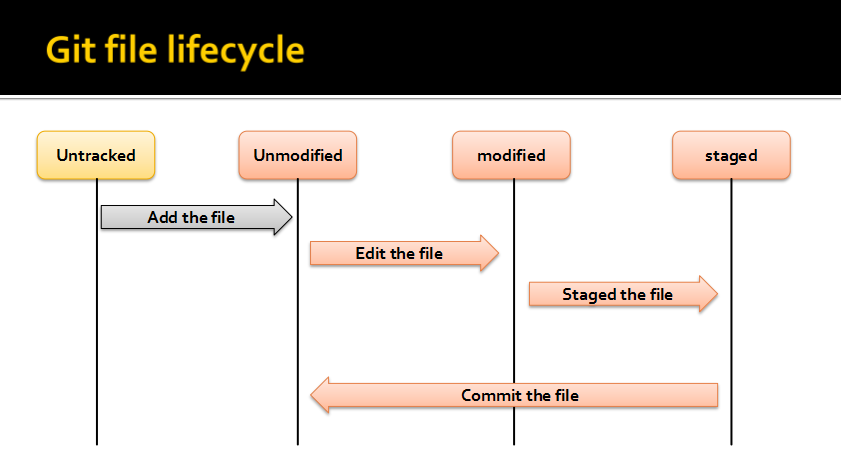
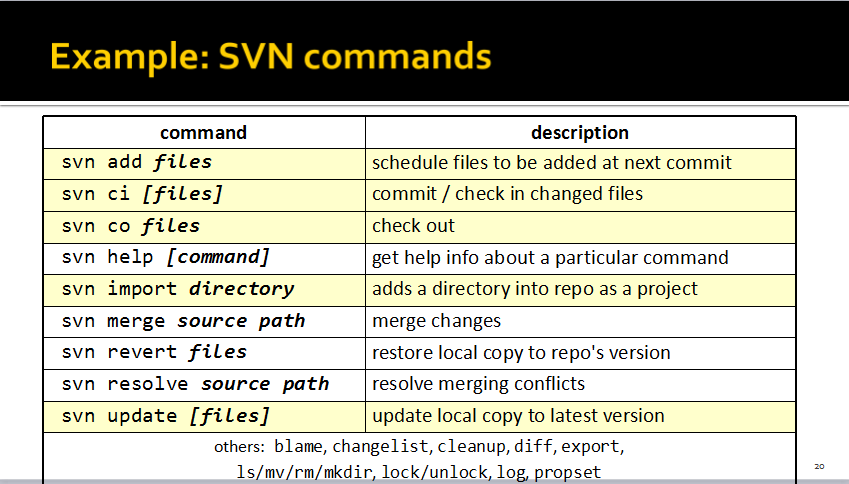
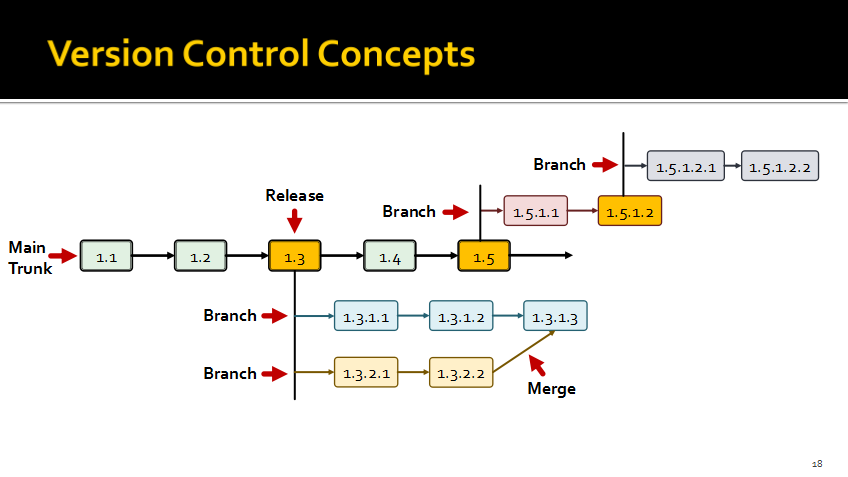
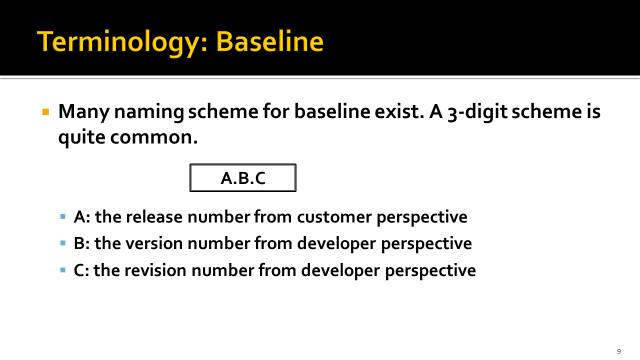
[**关键路径方法 13**](#_Toc3380)

[**软件质量属性 14**](#_Toc18557)

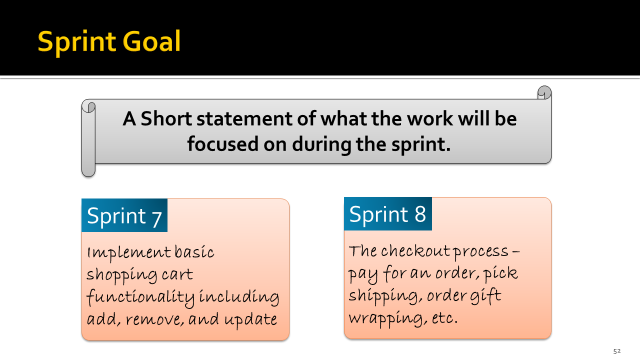
[**软件生命周期 14**](#_Toc13047)

[**良好的编码习惯 15**](#_Toc20984)

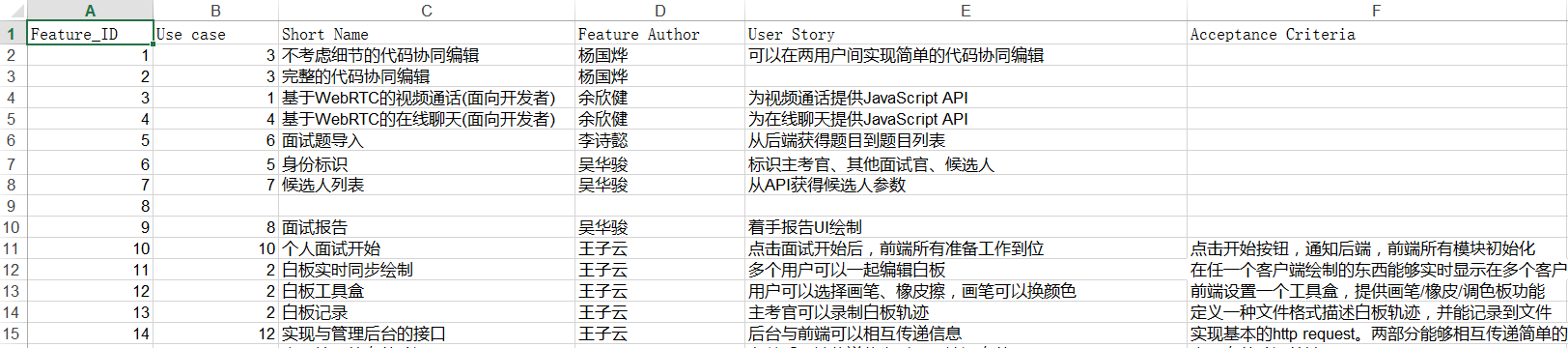
# 版本控制

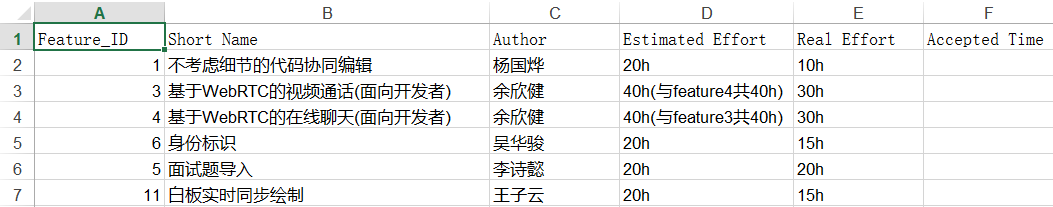


迭代开发

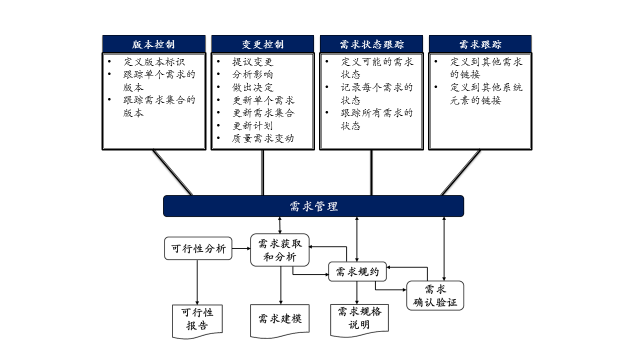
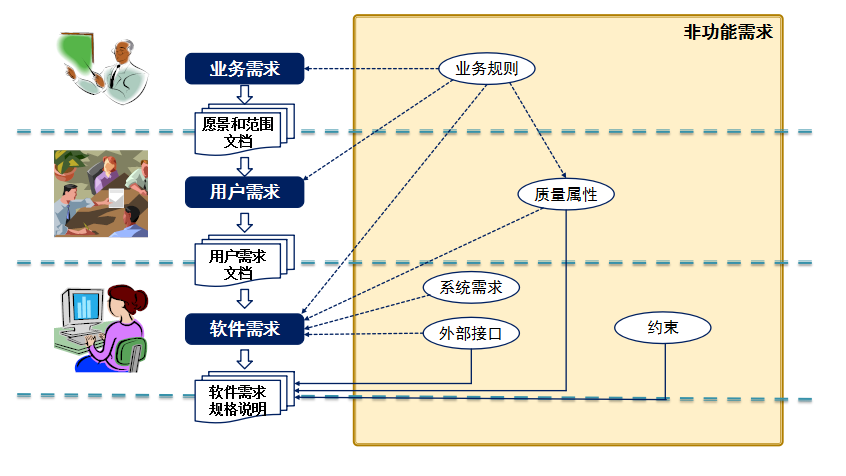


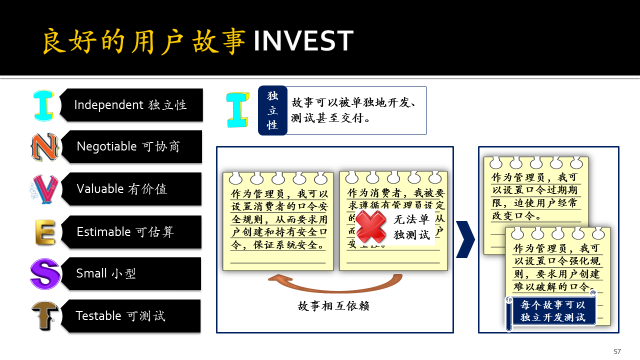
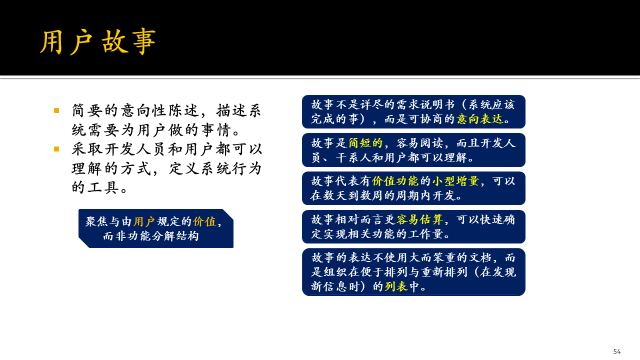
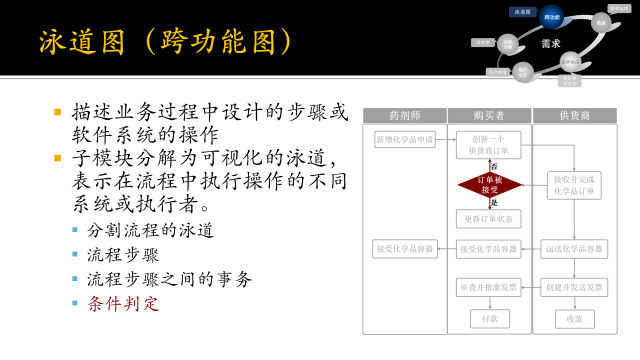
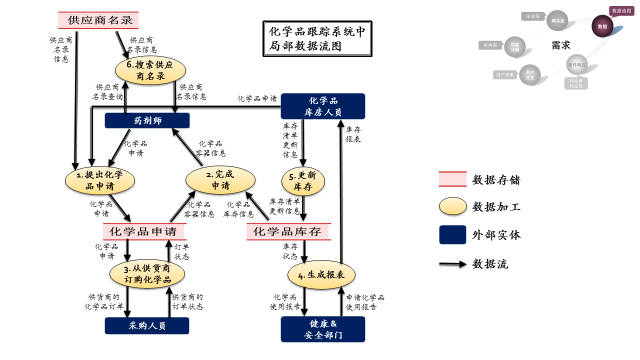
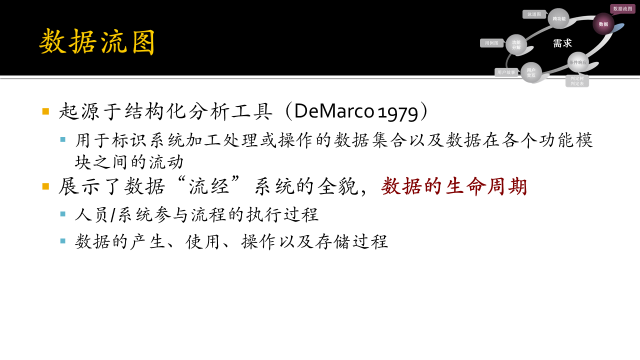




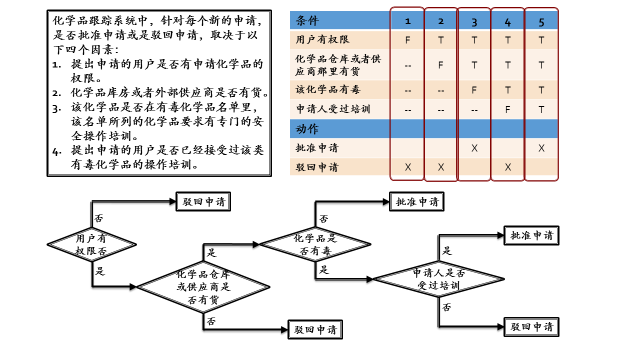
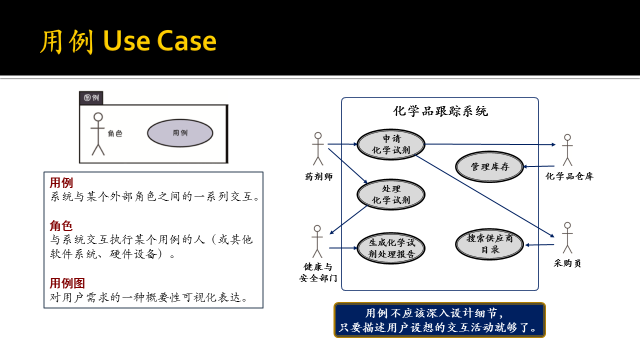


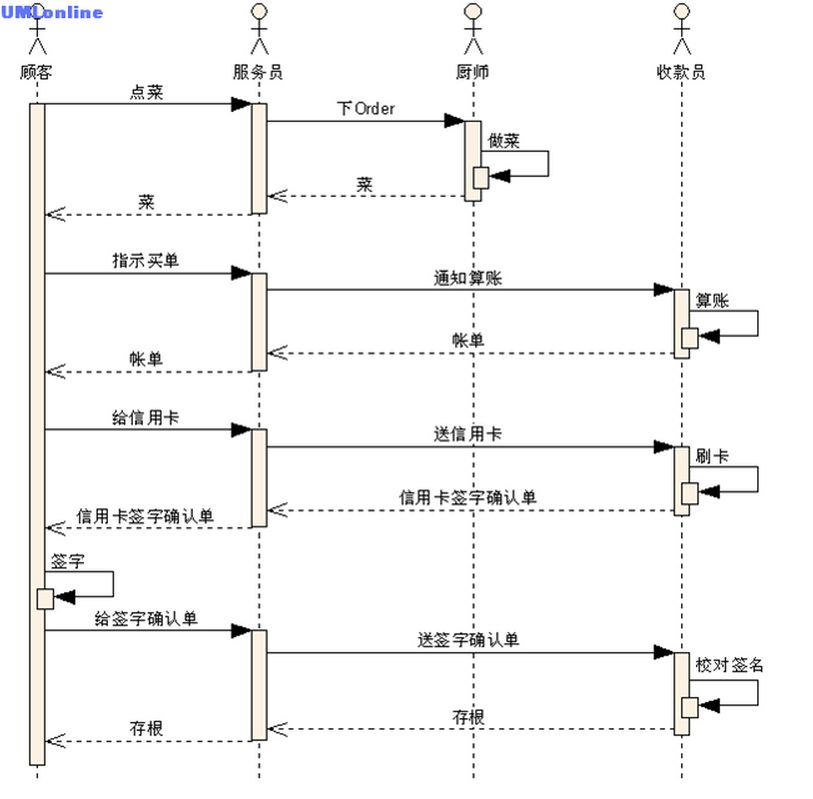
# 需求管理



需求描述  
  

判定表和判定树

序列图by百度  


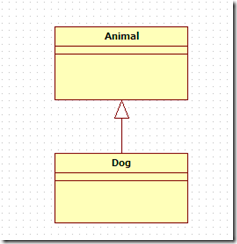
# UML类图

UML定义的关系主要有：泛化、实现、依赖、关联、聚合、组合，这六种关系紧密程度依次加强，分别看一下

**泛化**

**概念：**泛化是一种**一般与特殊**、**一般与具体**之间关系的描述，具体描述建立在一般描述的基础之上，并对其进行了扩展。在程序中是通过继承类实现的。比如狗是对动物的具体描述，在面向对象设计的时候一般把狗设计为动物的子类。

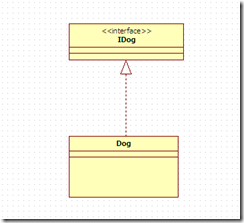
**表示方法：**空心三角形箭头的实线，子类指向父类

[](http://images.cnitblog.com/blog/349217/201309/15113815-bc66cc42f42b46db8f7aa60ca81a5b30.png)

**实现**

**概念：**实现是一种**类与接口**的关系，表示类是接口所有特征和行为的实现，在程序中一般通过类实现接口来描述

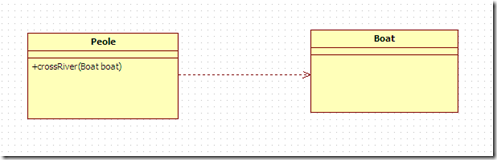
**表示方法：**空心三角形箭头的虚线，实现类指向接口

[](http://images.cnitblog.com/blog/349217/201309/15113816-6c9525327bc746a4820ceffd386648c6.png)

**依赖**

**概念：**是一种**使用**的关系，即一个类的实现需要另一个类的协助，所以要尽量不使用双向的互相依赖，在程序中一般表现为类A中的方法需要类B的实例作为其参数或者变量，而类A本身并不需要引用类B的实例作为其成员变量。

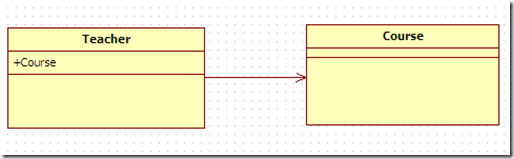
表示方法：虚线箭头，类A指向类B。

[](http://images.cnitblog.com/blog/349217/201309/15113817-e3bfe16c23594617858b82945e2120e1.png)

**关联**

**概念：**表示类与类之间的联接,它使一个类知道另一个类的属性和方法，这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，在程序中被关联类B以类属性的形式出现在关联类A中，也可能是关联类A引用了一个类型为被关联类B的全局变量

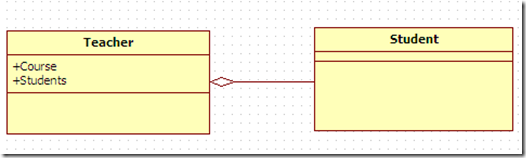
**表示方法：**实线箭头，类A指向类B

[](http://images.cnitblog.com/blog/349217/201309/15113820-dbff6f8616bf419895943b43e0ea1b23.png)

**聚合**

**概念：**聚合关联关系的一种特例，是强的关联关系。聚合是整体和个体之间的关系，即has-a的关系，整体与个体可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享。程序中聚合和关联关系是一致的，只能从语义级别来区分；

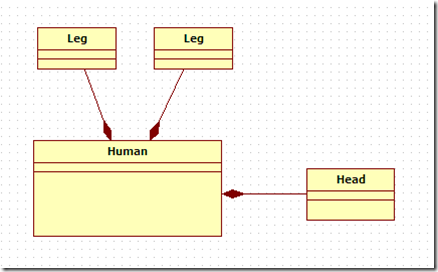
**表示方法：**尾部为空心菱形的实线箭头（也可以没箭头），类A指向类B

[](http://images.cnitblog.com/blog/349217/201309/15113821-f672301acfd5444fb66a9cbb221abb3c.png)

**组合**

**概念：**组合也是关联关系的一种特例。组合是一种整体与部分的关系，即contains-a的关系，比聚合更强。部分与整体的生命周期一致，整体的生命周期结束也就意味着部分的生命周期结束，组合关系不能共享。程序中组合和关联关系是一致的，只能从语义级别来区分。

**表示方法：**尾部为实心菱形的实现箭头（也可以没箭头），类A指向类B

[](http://images.cnitblog.com/blog/349217/201309/15113822-a45fa292737d4c7884acbf250e18e3df.png)

# Java Observe

# 白盒测试

覆盖（由弱到强）：

1.语句覆盖：每条语句至少执行一次。

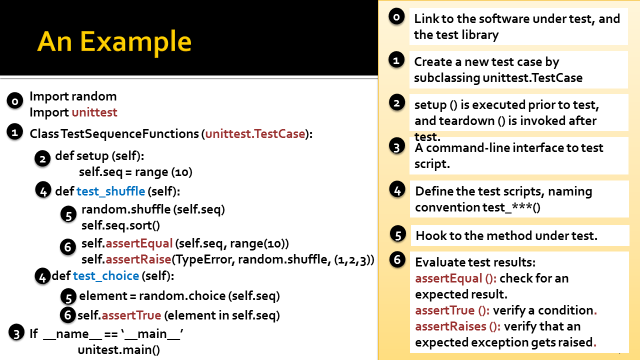
2.判定（分支）覆盖：每个判定的每个分支至少执行一次。

3.条件覆盖：每个判定的每个条件应取到各种可能的值。

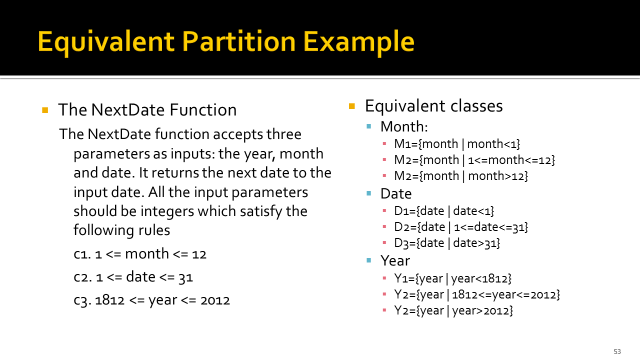
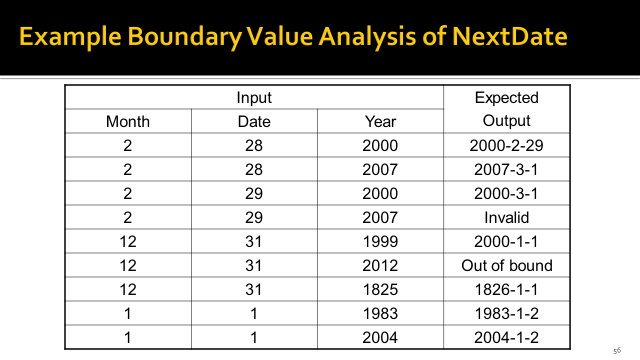
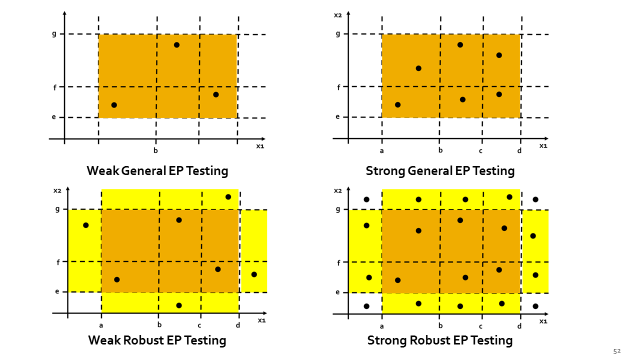
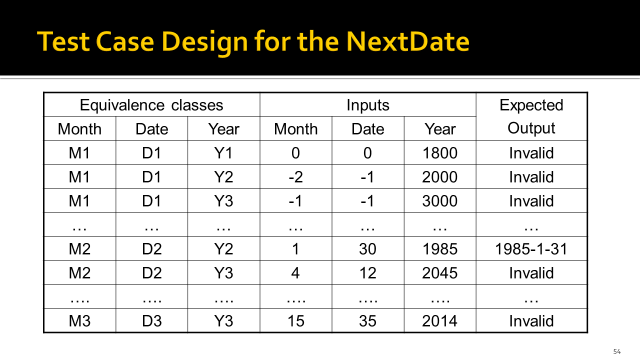
4.判定/条件覆盖：同时满足判定覆盖条件覆盖。

5.条件组合覆盖：每个判定中各条件的每一种组合至少出现一次。

6.路径覆盖：使程序中每一条可能的路径至少执行一次。



# 黑盒测试

# 关键路径方法

规划步骤：

指定各个活动

确定这些活动的顺序

绘制网络图

估计每个活动的完成时间

标识关键路径

随着项目进展更新CPM图

关键路径是通过网络的最长持续时间路径。

活动所在，不能拖延，不拖延项目。

为了加速项目，有必要减少关键路径中活动所需的总时间。

参数

EST 最早开始时间：活动可以开始的最早时间，因为活动必须首先完成。

EFT 最早完成时间：EST +活动持续时间

LFT 最长完成时间：活动可以在不拖延项目的情况下完成的最近时间

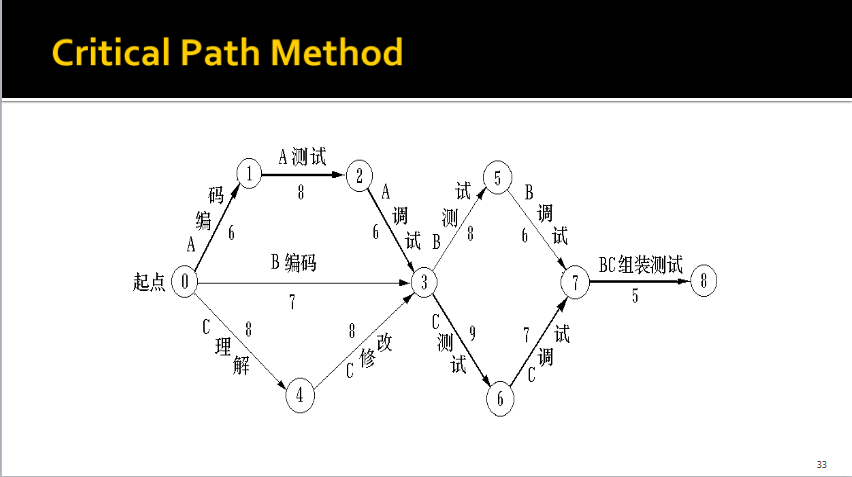
LST 最早开始时间：LFT - 活动持续时间

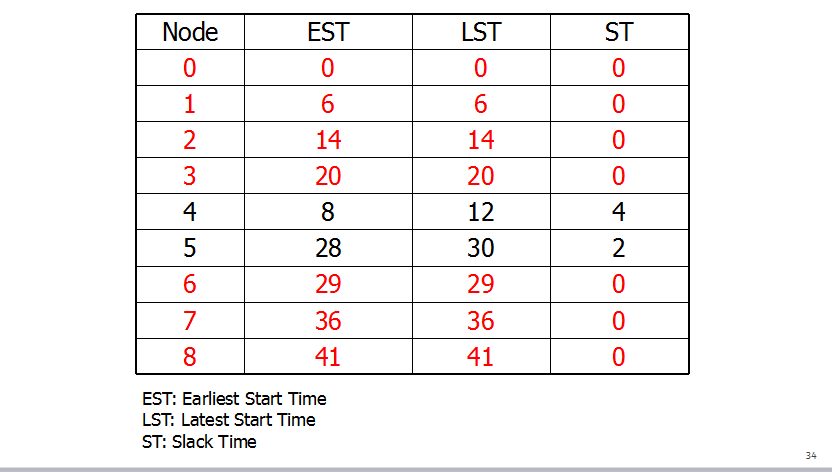
ST 松弛时间： LST-EST=可用时间 - 实时

关键路径

最长路径

对于路径中的每个节点，松弛时间= 0，即LST = EST，EFT = LFT





# 软件质量属性

可靠性 Reliability：软件可靠性是软件在给定的时间间隔及给定的环境条件下，按设计要求，成功地运行程序的概率。

健壮性 Robustness：系统遇到非法输入、软件错误、异常操作等情况下，仍能够正确运行的能力。

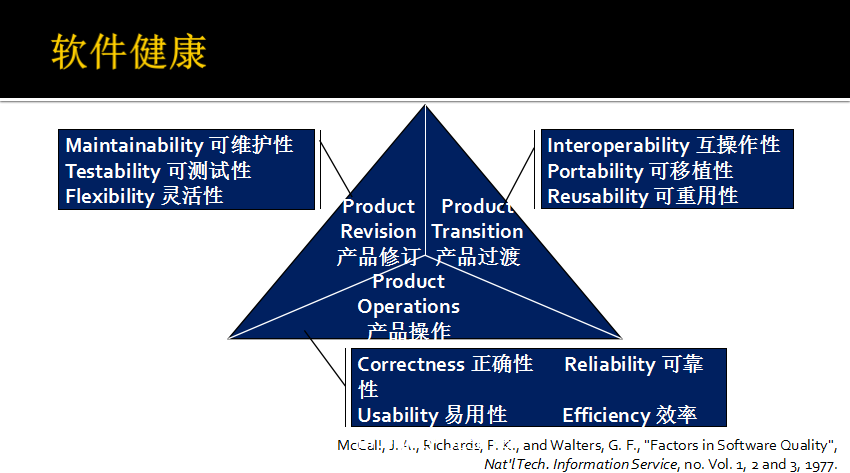
易用性 Usability：是指在指定条件下使用时，软件产品被理解、学习、使用和吸引用户的能力。

可维护性：是指软件产品可被修改的能力，修改可能包括修正，改进或软件适应环境、需求和功能规格说明中的变化。

可移植性：是指软件产品从一种环境迁移到另一种环境的能力。

互操作性：是指软件产品与一个或多个规定系统进行交互的能力

...



# 软件生命周期

需求：识别，分析和指定用户的需求

设计：定义如何构建系统以满足要求

编码：编程软件

测试：验证和验证系统

维护：在操作期间维护软件

# 良好的编码习惯

**Programming with Assertions**

注意他们的代码是健壮的

确保每个假设都在防御性代码中明确捕获

需要明确定义的垃圾输入行为

仔细考虑他们写的代码，因为他们写它

编写代码，保护自己免受其他人（或自己）的愚蠢

**The layout and presentation of source code**

避免无谓的争论，并对他人的意见敏感

足够谦卑，知道他们不总是对的

知道代码布局如何影响可读性，并努力写出最清晰的代码

即使它与自己的个人喜好相矛盾，也会采用已用风格

**Naming**

意识到名字的重要性，并尊重他们

考虑命名，并为他们创建的一切选择适当的名称

平衡考虑命名因素：名称长度，清晰度，上下文等

保持更大的视野，使得他们的名字在一个项目中可用（或项目）

**Self-Documenting Code**

寻求写清楚，自我记录的代码

尝试写最少量的必要文件

考虑将维护他们的代码的程序员的需求

**Code Comments**

尝试写几个真正好的注释

写注释解释为什么

集中于写好的代码，而不是过多的评论

写有用的合理的注释

**Bad Code Comments**

写注释解释如何

不介意如果注释只对自己有意义

充满坏代码与许多注释

使用冗余注释填充其源文件