**Chapter 2 – Software Processes**

**软件过程**

1.软件过程是完成软件产品生产的一组相互光联的活动。

2.软件过程模型--软件过程的抽象表示法。每个过程模型从一个特定的角度表现一个过程，只提供过程的某一侧面的信息。

3,有许多不同的软件过程， 但所有软件过程都必须具有下面这些基本的活动。它们是:

软件规格说明 软件的功能以及软件运行的约束

软件开发 必须开发符合规格说明的软件

软件确认确保软件是客户所想要的

软件演化 软件必须演化来满足不断变化的客户需要

4.过程描述还包括:

产品，这是软件过程活动的结果。

角色，反映了参与过程的人在其中的职责。

前置和后置条件，是指在一个过程活动执行前后或产品生产前后满足的条件。

**软件过程模型**

1.瀑布模型:计划驱动的模型。描述和开发的不同阶段。

2.增量式开发:描述、开发和验证交错进行。可以是计划驱动的，也可以是敏捷的。

3.面向复用的软件工程:该系统由现有组件组装而成。可以是计划驱动的，也可以是敏捷的。

**瀑布模型中不同的阶段:**

1.需求分析和定义2.系统和软件设计3.实现和单元测试4.集成和系统测试5.运行和维护

**瀑布模型缺点**

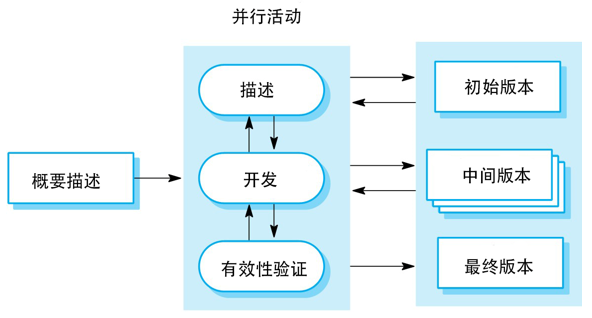
1.瀑布模型的主要缺点是在过程开始后很难适应变化。原则上，一个阶段在进入下一个阶段之前必须完成。

2.不能灵活划分为不同的阶段，难以应对不断变化的客户需求。

* + 只有在全面理解了需求，而且在系统开发过程中不太可能发生重大改变的时候，可以采用瀑布模型。
  + 很少有业务系统具有稳定的需求。

3.瀑布模型主要用于大型系统工程项目，其中一个系统是在几个地点开发的。

**增量式开发**



**增量式开发的优点**

1.降低了适应用户需求变更的成本。

2.在开发过程中更容易得到用户对于己做的开发工作的反馈意见。

3.使更快地交付和部署有用的软件到客户方变成了可能

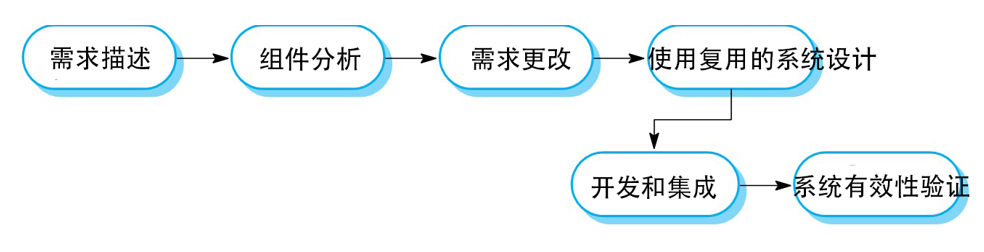
**增量式开发的问题**

1.过程不可见

2.伴随着新的增量的添加， 系统结构会逐渐退化

**面向复用的软件工程**

过程阶段:1.组件分析2.需求修改3.使用复用的系统设计4.开发和集成



**过程活动**

1.真正的软件过程中技术活动、 协作活动、 管理活动交织在一起， 其总体目标是完成一个软件系统的规格说明、设计、实现和测试。

2.对于不同的开发过程， 规格说明、 开发、 验证和进化这4个基本过程活动的组织是不同的。 在瀑布模型中， 它们是顺序组织的， 而在增量式开发中它们是交织在一起的.

**需求工程过程中的4个主要活动**

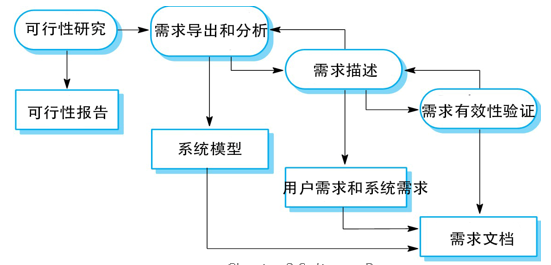
1.可行性研究

2.需求抽取和分析

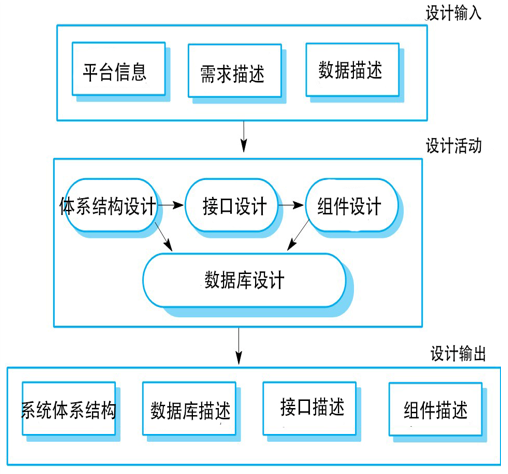
3.需求规格说明

4.需求确认

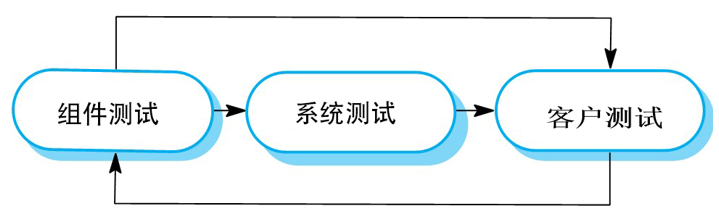
**需求工程过程**



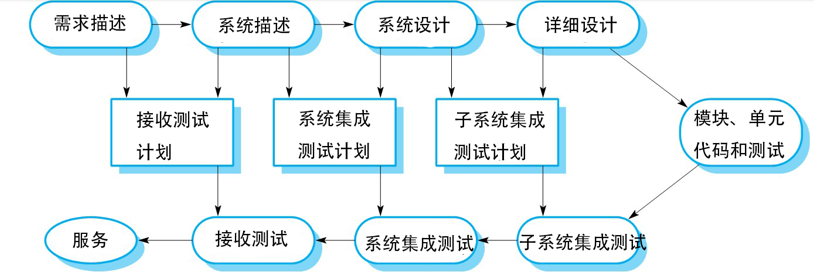
**设计过程的通用模型**



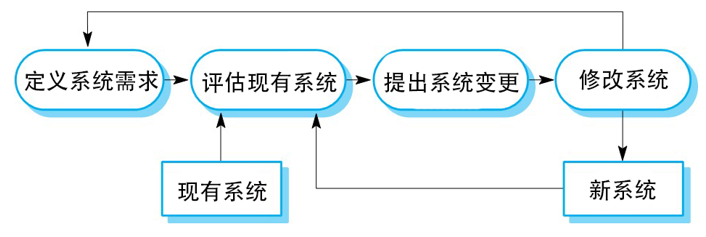
**测试阶段**



**计划驱动的软件过程测试阶段**



**软件系统演化**



**软件原型**

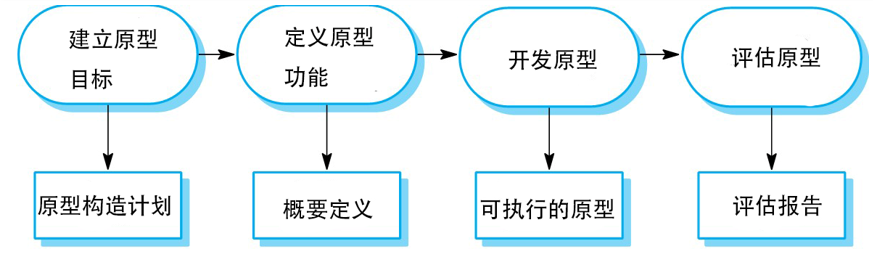
软件原型可以用在：

* + 在需求工程过程中， 原型有助于启发和验证系统需求。
  + 在系统设计过桯中， 原型可用于探索特定软件的解决方案，并支持用户接口设计。
  + 在测试过程中，进行回归测试。

**原型的好处:**

1.提高系统可用性2.更贴近用户的实际需求3.改善设计质量4.提高可维护性5.减少开发工作

**原型开发的过程**



**抛弃原型**

原型应该在开发后丢弃，它们不是生产系统的良好基础:

* + 不可能调整原型以满足非功能性的要求
  + 开发过程中的快速变化必然意味着原型是没有文档的
  + 原型开发过程中的变更可能会破坏系统的结构

原型可能不符合正常的组织质量标准

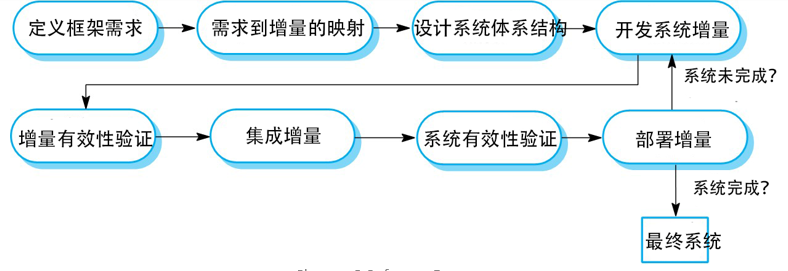
**增量式开发和交付**

增量式开发

* + 以增量方式开发系统，并在进行下一个增量开发之前对每个增量进行评估;
  + 敏捷方法中使用的常规方法;
  + 由用户/客户代理进行评估。

增量式交付

* + 部署供最终用户使用的增量;
  + 更现实的评估软件的实际使用;
  + 难以实现替换系统，因为增量的功能比被替换的系统少。



**增量式交付的优势**

1.客户可以将早期的增量作为原型， 从中获得对后续系统增量的需求的经验。

2.降低项目整体失败的风险。

3.具有最高优先权的服务接受了最多的测试。

**增量式交付的问题**

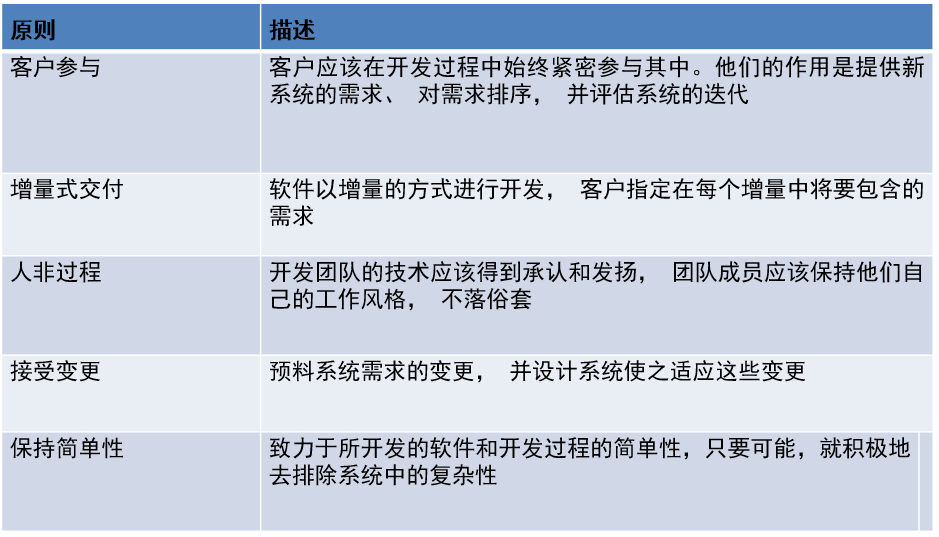
1.当新系统准备替换一个已有的系统时，迭代化交付会有问题。

2.大多数系统需要一组由系统不同部分所使用的基本设施。

3.迭代过程的本质是软件规格说明和软件一起开发的。

**Chapter 3 – Agile Software Development敏捷软件开发**

**敏捷方法的基本原理**



**敏捷方法的适用性**

1.软件企业所开发的用于市场销售的中、小规模产品。

2.组织内的定制化系统开发，其中客户承诺可以参与开发过程， 并且影响软件开发的外部利益相关者和法规不多。

3.因为它致力于小的、 紧密集合的团队， 将它们扩展到大型系统中就会有许多问题。

**敏捷方法和软件维护**

1.如果敏捷方法要成功，它们必须支持维护和原始开发。

2.两个关键问题:

* + 由于开发过程中强调正式文档的最小化， 用敏捷技术开发的系统是否可维护？
  + 敏捷方法是否能有效地用于进化系统以响应用户的变更请求？

3.如果无法维护原有的开发团队，可能会出现问题。

**计划驱动和敏捷开发**

1.计划驱动开发

* + 软件工程的计划驱动方法是基于不同的开发阶段，每个阶段的输出都是预先计划好的。
  + 不一定是瀑布模型-计划驱动，增量开发也是可能的。
  + 迭代发生在活动中。

2.敏捷开发

* + 规范、设计、实现和测试是相互交织的，开发过程的输出是在软件开发过程中通过协商确定的。
  + 敏捷方法依靠好的工具来跟踪不断发展的设计。

**Chapter 4 – Requirements Engineering**

**什么是需求**

1.它可以是服务或系统约束的高层抽象描述，也可以是关于系统功能的详细和正式的定义。

2.这是不可避免的，因为需求可能具有双重功能

* + 可能是投标合同的基础——因此必须对其进行解释;
  + 可能是合同本身的基础——因此必须详细定义;
  + 这两个描述可以被称为需求。

**需求类型**

用户需求:是用自然语言加图的形式给出的、关于系统需要提供哪些服务以及系统操作受到哪些约束的声明。

系统需求:详细地给出系统将要提供的服务以及系统所受到的约束。定义应该实现的内容，因此可以是客户和承包商之间的合同的一部分。

**功能需求和非功能需求**

1.功能需求

* + 包括对系统应该提供的服务、如何对特殊输入做出反应，以及系统在特定条件下的行为的描述。
  + 功能需求可能还需明确声明系统不应该做什么。

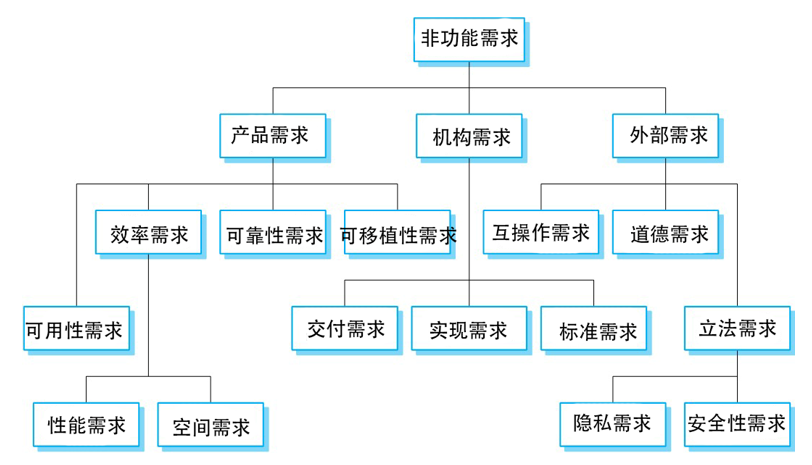
2.非功能需求

* + 对系统提供的服务或功能给出的约束。包括时间约束、开发过程的约束和所受到的标准的约束。
  + 经常适用于整个系统而不是个别的系统功能或服务。

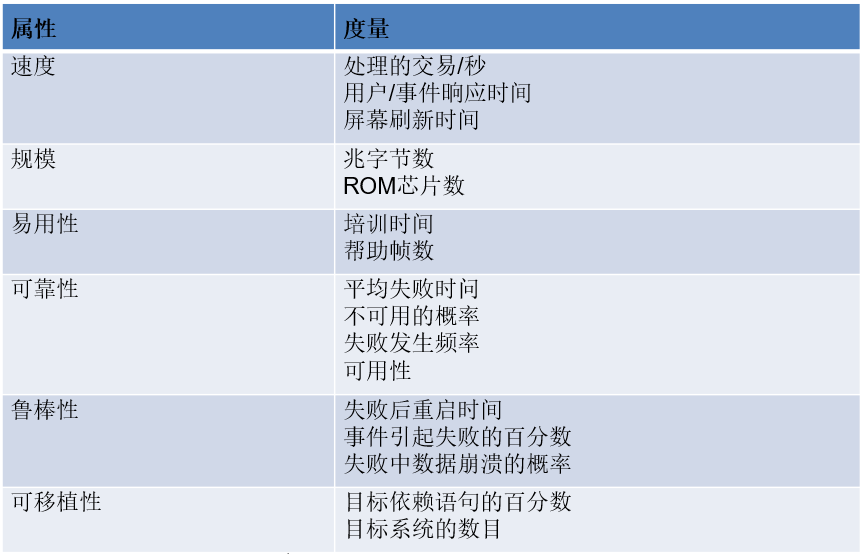
3.邻域需求

* + 系统运行领域的约束条件。

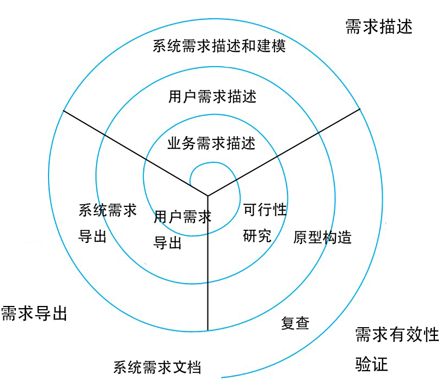
**非功能需求的类型**



**定义非功能需求的量度**



**需求工程过程的螺旋模型**



**需求分析的问题**

1.系统利益相关者井不真正知道他们先要什么。

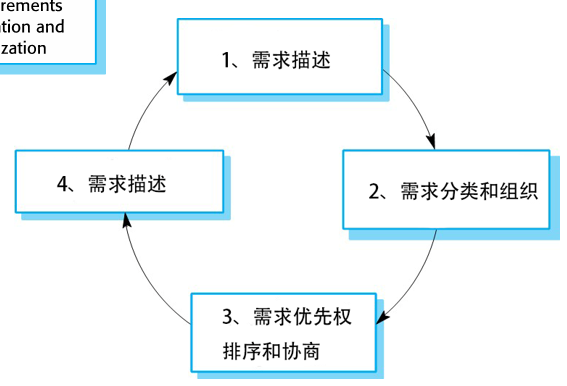
2.系统利益相关者用他们自己的语言表达需求。

3.不同的系统利益相关者有不同的需求。

4.政治上的因素可能影响系统的需求。

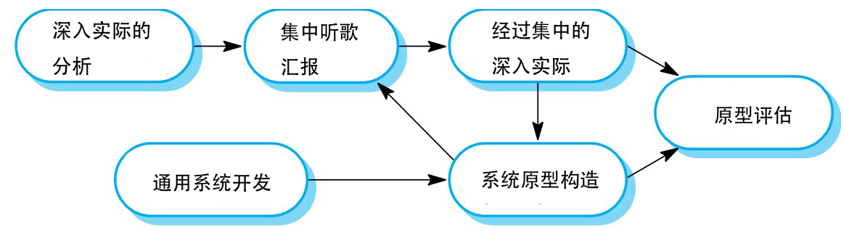
5.进行需求分析的经济和业务环境是动态的。在分析过程期间它不可避免会变化。

**需求抽取和分析过程**



4是需求文档化

**需求分析的深入实际方法和原型**



**需求规格说明**

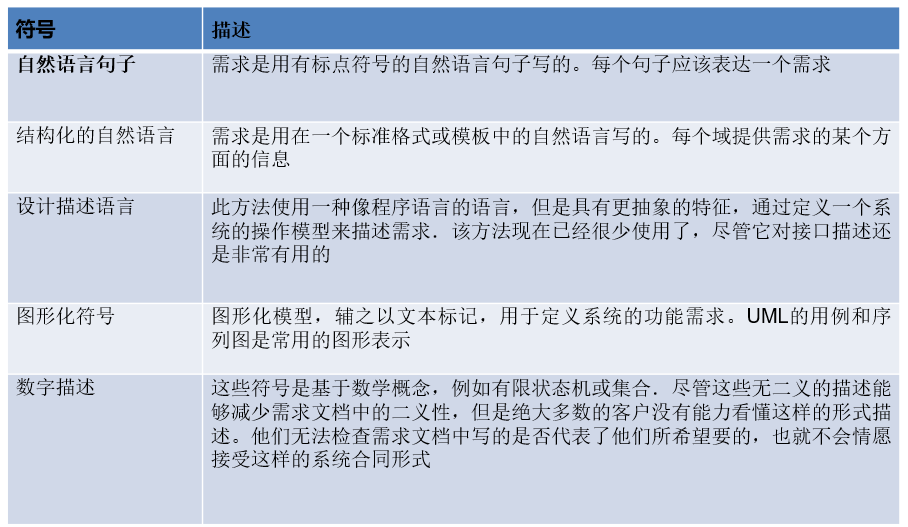
1.需求规格说明就是在需求文档中写下用户需求和系统需求。

2.用户需求必须能够被没有技术背景的终端用户和客户所理解。

3.系统需求是更详细的需求，可能包括更多的技术信息。

4.需求可能是系统开发合同的一部分。

**系统需求规格说明的书写方法**



**需求确认**

1.关注于检查需求是否定义了客户真正想要的系统。

2.需求错误成本很高，因此验证非常重要

**需求检查**

1.有效性检查:系统是否提供最能满足客户需求的功能?

2.一致性检查:是否存在任何需求冲突?

3.完备性检查:是否包含客户要求的所有功能?

4.真实性检查:在现有的预算和技术条件下，能否实现这些需求?

5.可检验性检查:可以检查需求吗?

**评审检查**

1.可验证性:需求是实际可测试的吗?

2.可理解性:需求被正确理解了吗?

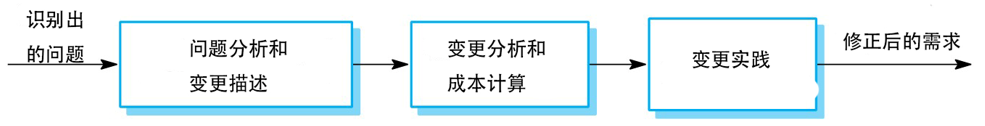
3.可追溯性:是否清楚说明了需求的来源?

4.适应性:需求可以在不影响其他需求的情况下进行更改吗?

**需求管理计划**

1.需求识别2.变更管理过程3.可追溯策略4.工具支持

**需求变更管理**



**Chapter 5 – System Modeling**

**系统建模**

1.系统建模就是建立系统抽象模型的过程， 每一个模型表示一个系统不同的角度或方面。

2.系统建模通常意味着用一些图形符号表示系统， 现在几乎都是用统一建模语言(UML)中的符号。

3.系统建模帮助分析人员理解系统的功能，并使用模型与客户进行沟通。

**UML**

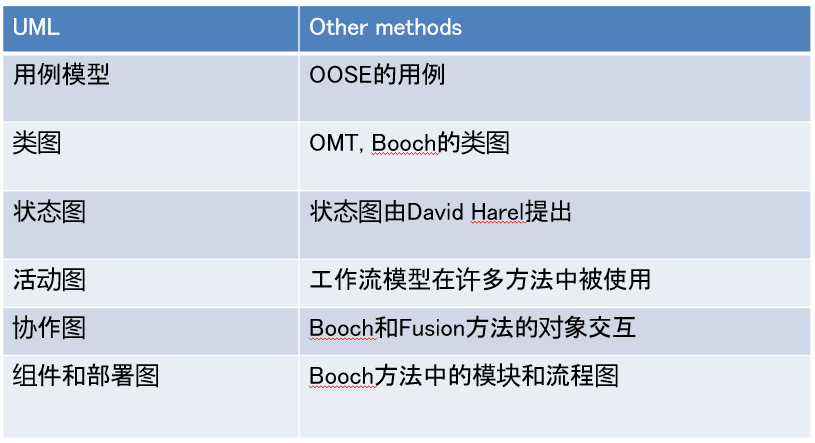
1.UML是一种用于软件产品的文档化、结构化、说明与具体化的标准语言。

2.它可以用于开发生命周期的所有过程以及不同实现技术之间。

3.UML可用于：

* + 主要采用“用例”与“角色”显示系统的边界
  + 利用“交互图”阐述用例的实现
  + 利用“类图”描述系统的静态结构
  + 利用“状态图”给对象的行为建立模型
  + 利用“组件图”和“配置图”显示物理结构
  + 利用类别扩充功能

**UML和其他方法**



**UML的元素**

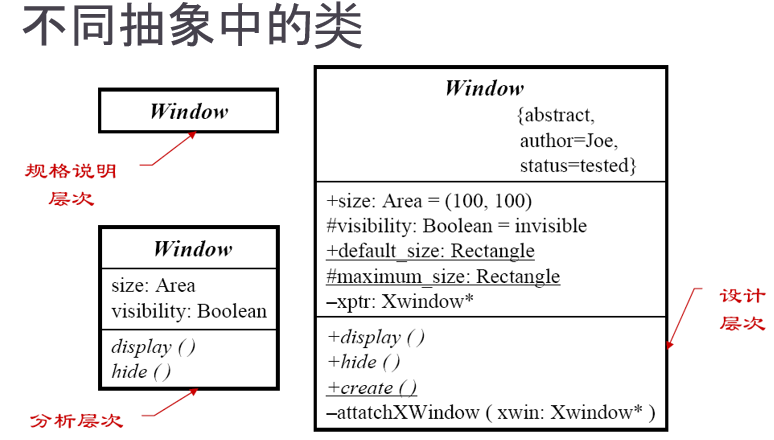
1.事物:类、接口、包、组件等。

2.关系:泛化，依赖，关联，实现

3.图:由事物和关系组成,用例图、类图、序列图……和其他图

4.规则:命名, 范围 , 一些限制

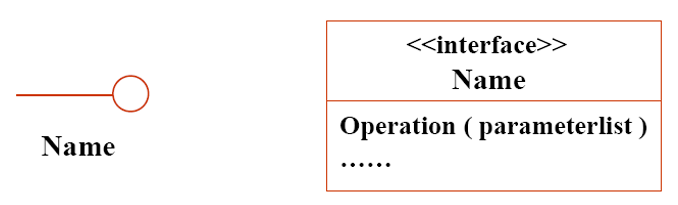
**结构事物:对象类**



**结构事物:接口**

1.类或组件所提供的服务

2.类或组件对外可见行为的描述



**结构事物:组件**

组件是物理的



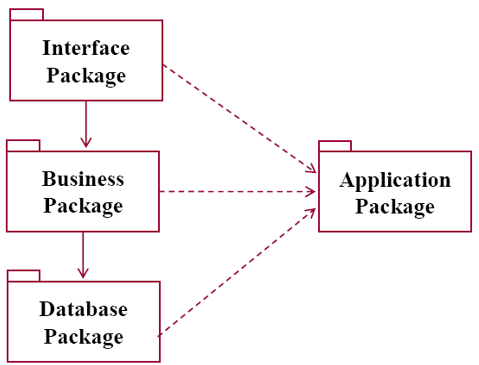
**行为事物:交互**

对象之间的消息交换

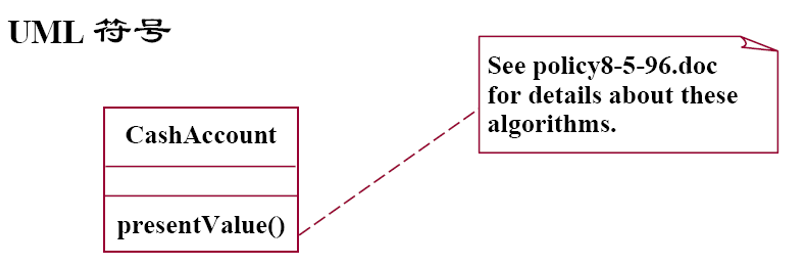


**分组事物:包**

把一些事物放在一个包中

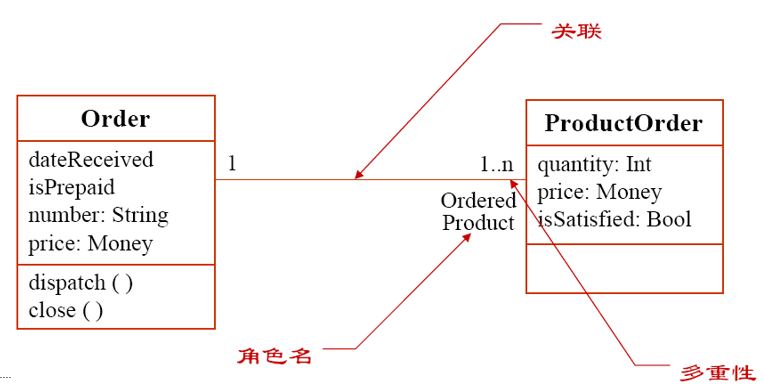


**注释事物:注释**

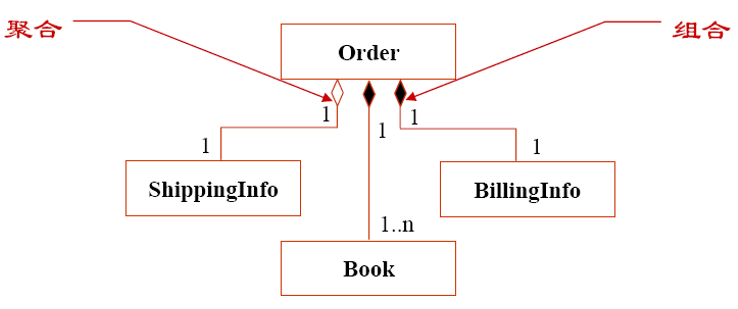


**UML 关系**

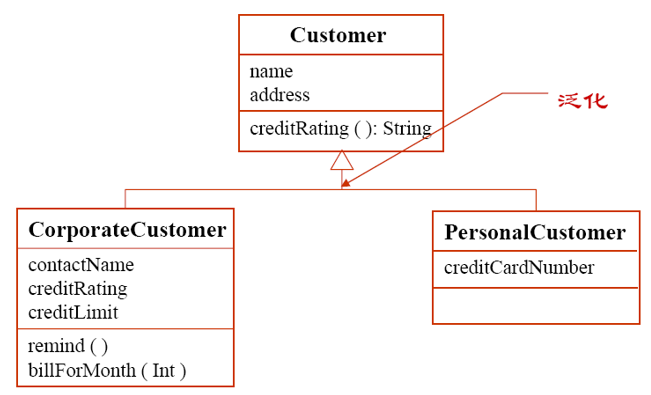
**关联**



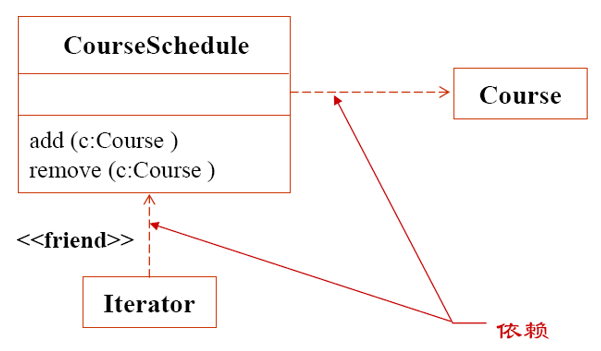
**特殊关联**



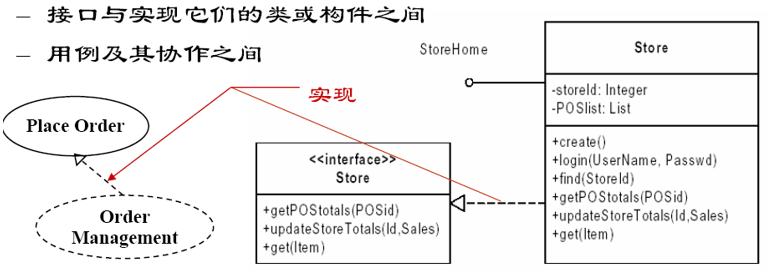
**泛化:一般事物与特殊事物之间的关系**



**依赖:说明一个事物的变化会影响另一个使用它的事物**



**实现:一部分由另一部分实现**



**上下文模型**

1.上下文模型用于界定系统上下的边界-显示系统边界之外的内容。

2.社会和机构等非技术性因素可能影响系统的边界定义。

3.体系结构模型用于描述系统和相关系统之间的联系。

4.系统边界是用来定义系统内部和外部的。

5.系统边界的位置对系统需求有着深刻的影响。

6.界定一个系统的边界是一个政治判断

**交互模型**

1.为用户交互建模有助于识别用户需求.

2.为系统与系统之间的交互建模可以突出可能出现的通信问题.

3.为系统各部分之间的交互建模有助于分析所提出系统结构能否实现系统所需的功能及其可靠性.

4.用例图和顺序图用来建立交互模型.

**结构模型**

1.软件的结构模型表示的是系统的构成，表示为组件构成系统以及组件之间的关系.

2.有的结构模型是静态模型，表示系统设计的结构；有的是动态模型，表示系统执行时的构成.

3.我们建立结构模型以讨论和设计系统体系结构.

**对象类聚合模型**

1.聚合模型显示了聚合类是如何由其他类组成的。

2.聚合模型类似于语义数据模型中的局部-整体关系**。**

**行为模型**

1.行为模型是描述系统运行时的动态行为的模型，表示当系统响应来自所处环境的刺激时所发生的或有可能发生的事情。

2.这样的刺激有两种：

* + 数据 一些数据到达必须由系统处理。
  + 事件 某些触发系统处理的事件发生。事件可能会有相关联的数据，但并不总是这样。

**数据驱动模型**

1.许多业务系统是主要由数据驱动的数据处理系统， 它们由输人数据所控制， 很少处理外部事件。

2.数据驱动模型描述一个动作序列， 该动作序列涉及输人数据的处理和相关输出的产生。

3.数据驱动模型非常有用， 因此它们可用来表示系统中端到端的过程。

**事件驱动模型**

1.实时系统通常是事件驱动的，数据处理很少。例如，固定电话交换系统通过产生拨号音来响应诸如“挂断”之类的事件。

2.事件驱动模型表示系统对内外部事件的响应方式。

3.这种模型建立在以一个假设为基础的， 即系统状态是有限的， 并且事件（激励）可能引起从一种状态向另一种状态的转变。

**状态机模型**

1.这些模型对于来自系统外部和内部的事情的系统响应行为进行建模。

2.它们表示系统对激励的反应，因此经常用于实时系统建模。

3.状态机模型将系统状态表示为一个节点，而把事情表示为连接这些节点的弧线。当一个事件发生时，系统从一个状态移动到另一个状态。

4.状态图是UML不可分割的一部分，用于表示状态机模型。

**模型类型**

1.计算无关模型(CIM):此模型为系统中使用的重要的领域抽象建模。 CIM有时也被称为领域模型

2.平台无关模型(PIM):此模型在没有它的实现作为参考下为系统的运行建模。 我们通常使用UML模型来描述PIM, UML模型表示静态系统结构和系统对内外部事件的响应。

3.平台相关模型(PSM) :PSM是由平台无关模型转化得到的， 其中每一个应用平台都有一 个独立的PSM。原则上 ， PSM应有好几层， 每一层都增加了一些特定的平台细节。

**Chapter 6 – Architectural Design**

**清晰体系结构的好处**

1.信息持有者之间的沟通

* + 体系结构可以作为不同的信息持有者之间讨论的焦点。

2.系统分析

* 分析系统是否能满足非功能性需求。

3.大规模复用

* + 体系结构能在系统之间复用
  + 可以开发产品线架构。

**体系结构和系统特征**

1.性能:将关键业务本地化，减少沟通。使用大而不是细粒度的组件。

2.信息安全性:使用在内层具有关键资产的分层体系结构。

3.安全性:在少数子系统中本地化安全关键特性。

4.可用性:包含冗余组件和容错机制。

5.可维护性:使用小粒度的、可更换的组件。

**体系结构识图**

1.逻辑视图，它显示了系统中对象和对象类的一些主要抽象。

2.进程视图，显示在运行时系统是如何组织为一组交互进程.

3.开发视图， 它显示了软件是如何为了开发而被分解的。

4.物理视图， 它显示了系统硬件和系统中软件组件是如何分布在处理器上的。

5.相关的使用用例或场景(+1)

**应用类型**

1.数据处理应用:数据批处理过程中没有用户的干预

2.事务处理应用:处理来自用户对信息的请求并更新数据库数据

3.事件处理系统:系统行为依赖系统环境事件的触发

4.语言处理系统:用户意图通过形式化语言被系统处理和解释

**Chapter 7 – Design and Implementation**

**过程阶段(开发一个系统设计)**

1.了解并定义上下文和与系统的外部交互。

2.设计系统体系结构。

3.识别出系统中的主要对象。

4.开发设计模型。

5.定义对象接口。

**识别对象类的过程**

1.对系统的自然语言描述做文法分析。

2.使用应用领域中的真实实体名字.

3.使用行为方法，参与行为并起重要作用者即可视为对象。

4.基于情景的分析识别，识别出需要的对象、属性和操作。

**设计模型**

设计模型表示系统中包含的对象或对象类， 同时也描述了实体间的关联与关系。

1.静态模型通过系统对象类及其之间的关系来描述系统的静态结构。

2.动态模型描述了对象之间的动态交互。

3.子系统模型，给出对象的逻辑分组即子系统。

4.时序模型说明对象交互的序列。

5.状态机模型说明单个对象如何响应事件来改变它们的状态。

6.其他模型包括用例模型、聚合模型、泛化模型等。

**设计模式**

1.模式的一般定义是:“上下文中问题的解决方案”。

2.设计模式是一种对问题和解决方案的抽象知识的复用方法。

3.“模式”是对问题和解决方案的基本内容的描述。

**为什么用模式?**

1.设计面向对象的软件是困难的，而设计可重用的面向对象的软件则更加困难.

2.有经验的设计师会重用过去有效的解决方案.

3.结构良好的面向对象的系统循环(可再现的)类和对象的模式.

4.了解过去工作过的模式可以使设计人员的工作效率更高，从而使设计更加灵活和可重用

**GoF设计模式的基本元素**

1.模式名称:为模式提供一个简洁、有意义的名称可以改善开发人员之间的沟通

2.问题:我们将在何处使用此模式的问题和上下文是什么?在使用此模式之前必须满足哪些条件?

3.解决方案:1构成设计模式的元素的描述,2强调他们的关系、责任和协作,3不是具体的设计或实现，而是一种抽象的描述

4.效果:1使用模式的结果2包括对可重用性、可移植性和可扩展性的影响

**设计模式的优点**

1.获取专业知识，并以标准的形式使非专业人员可以访问它

2.通过提供一种公共语言来促进开发人员之间的交流

3.使成功的设计更容易重用，避免降低可重用性的替代方案

4.促进设计修改

5.提高设计文档

6.提高设计的可理解性

**复用层次**

1.抽象层:这一层中并不是直接复用软件， 而是运用软件设计中的成功抽象。

2.对象层:在这一层可直接复用库中的对象， 代替自己编写代码。

3.组件层:组件是在应用程序系统中复用的对象和对象类的集合。

4.系统层:在这一层， 我们复用整个应用系统。

**开发平台工具**

1.集成编译器和句法导向的编辑系统，能够创建、 编辑和编译代码。

2.编程语言调试系统。

3.图形编辑工具， 例如编辑UML模型的工具。

4.测试工具， 如JUnit，可以在新版本的程序上自动运行一组测试。

5.项目支持工具， 能够帮助我们为不同的开发项目组织代码。

**Chapter 8 – Software Testing**

**程序测试目标**

1.向开发者和用户展示软件满足了需求。

2.为了找出软件中的缺陷和不足， 即软件的行为是不正确的、 所不希望的或不符合它的描述的。

**有效性测试和缺陷测试**

1.有效性测试可达成第一个目标:即在这一阶段中使用系统所希望的使用方式的一组测试案例来测试系统的表现。

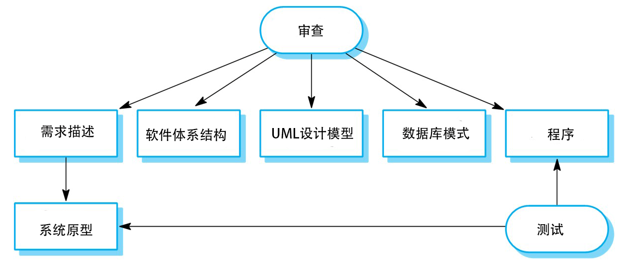
过程:1向开发人员和系统客户演示软件满足其需求2一次成功的测试表明，系统按预期运行

2.缺陷测试可达成第二个目标:即在这一阶段测试案例是设计来暴露系统的缺陷。 测试案例可以是故意模糊的并且不需要反映系统通常是怎样使用的。

缺陷测试

过程：1发现软件中行为不正确或不符合其规范的错误或缺陷2成功的测试是使系统执行不正确，从而暴露系统缺陷的测试。

**审查和测试**



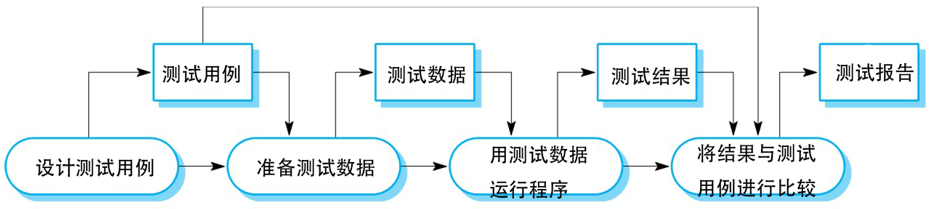
**软件审查的优点**

1.在测试期间，一个错误可能会掩盖其他错误。因为审查是一个静态的过程，我们不必关心错误之间的相互作用。

2.审查一个系统的不完整版本不需要额外的代价。如果一个程序不完整，我们需要开发特殊测试工具来测试可用的部分。

3.除了搜索程序缺陷，审查也可以考虑一个程序更广泛的质量属性，如符合标准、可移植性和可维护性。

**软件测试过程模型**



**测试阶段**

1.开发测试，即在开发中进行系统测试来发现故障和缺陷。

2.发布测试，即一个测试小组对一个系统的完整版本进行测试， 然后发布给用户。

3.用户测试，即系统的用户或是潜在的用户在他们自己的环境中测试这个系统。

**接口测试**

1.接口类型

* + 参数接口 在这些接口中， 主要是数据和函数指针， 由一个组件传递到另一个组件。
  + 共享内存接口 在这些接口中， 有一个被子系统共享的内存块。
  + 程序接口 在这些接口中，有子系统封装的一组程序，这些程序可以被其他子系统调用。
  + 消息传递接口 在这些接口中，子系统通过消息传递来请求其他子系统上的服务。

2.接口错误：1接口误用2接口误解3时序错误

3.接口测试准则1设计一组测试集， 使得对传给外部组件的参数的值选择紧靠取值范围边缘的那些值。2当有指针从接口传递时 ， 总用空指针参数来测试接口。3当组件通过程序接口被调用时 ， 设计一些容易引起组件失败的测试。4在消息传递系统中进行强度测试。5当组件间通过共享内存来交互时 ， 可以设计一种测试 ， 使其对激活组件的次序有所改变。

**测试驱动开发过程活动**

1.从识别所需要的功能增量开始。 这个通常应该比较小， 用几行代码就可以实现。

2.针对此功能编写一个测试并实现为一个自动测试。

3.然后运行此测试， 以及所有已实现的其他测试。 最初， 我们并没有实现这个功能， 因此这个新的测试将是失败的。

4.然后是实现这个功能， 并重新运行这个测试。

5.一旦所有的测试成功 ， 我们可以转去实现下一个功能块。

**测试驱动开发的优势**

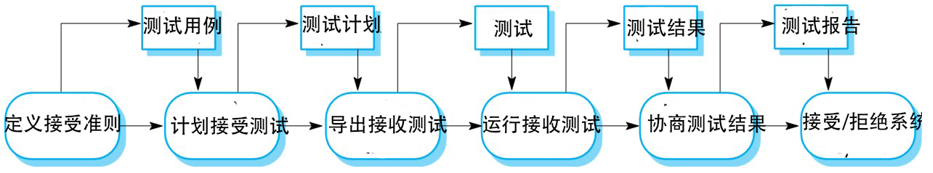
1.代码覆盖：原则上， 我们所写的每个代码片段都至少有一个测试。 因此， 我们可以确信系统中的所有代码都实际执行过。

2.回归测试：随着一个程序的开发， 一个测试套件也增量式开发出来。

3.简化调试:当一个测试失败的时候， 问题出在何处是很明显的。 新写的代码需要审查和修改。

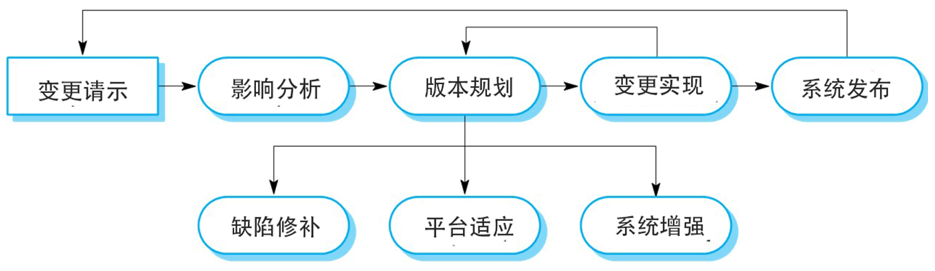
4.系统文档:测试本身就表现为一种文档形式 ， 它描述了代码应该做什么。

**接收测试过程**



**Chapter 9 – Software Evolution**

**软件进化过程**



**软件维护的类型**

1.修补软件缺陷:改变一个系统以纠正其不足之处来满足需求。

2.使软件适应不同操作环境:改变一个系统，使其在不同的环境(计算机、操作系统等)中运行。

3.增加或修改系统功能:修改系统以满足新的需求。

**维护花费原因**

1.团队稳定性:如果相同的工作人员参与一段时间，维护成本就会降低。

2.合同的责任:系统的开发人员可能没有维护的合同责任，因此没有为将来的变更而设计的动机。

3.人员技术水平:维护人员一般都缺乏经验， 而且不熟悉应用领域。

4.程序年龄和结构:随着程序不断的变更， 其结构受到了破坏。它们变得越来越不容易理解和变更。

**复杂性度量**

可维护性预测可以通过评估系统组件的复杂度来实现。

复杂度依赖于：1控制结构的复杂度；2数据结构的复杂度；3对象、方法（过程）和模块的规模。

**再工程的优势**

1.较小的风险:新开发软件是高风险的。有可能存在开发的问题、人员的问题和系统描述的问题等。

2.较小的成本:再工程的成本较之重新开发一个软件的成本来说要小得多。

**再工程工程活动**

1.源代码转换:将代码转换到一个新的语言。

2.反向工程:对程序进行分析并理解它。

3.程序结构改善:对程序结构分析并修改使之易于理解。

4.程序模块化:重构程序结构。

5.数据再工程:改变程序处理数据以反映程序变更。

**再工程成本因素**

1.软件质量的重新设计。2.再工程的工具支持。3.必需的数据转换的程度。4.再工程专家人员的可用性。

**遗留系统类别**

1.低质量、低业务价值:这类系统是抛弃的候选对象。

2.低质量、 高业务价值:这些系统正在为业务做出重要贡献， 不能抛弃。 不过， 其低质量意味着运行的成本很高， 所以这类系统有待于转换或者以合适的系统替代。

3.高质量、 低业务价值:用COTS替代，抛弃或维护。

4.高质量、 高业务价值:正常的系统维护应该继续进行。

**业务价值评估的问题**

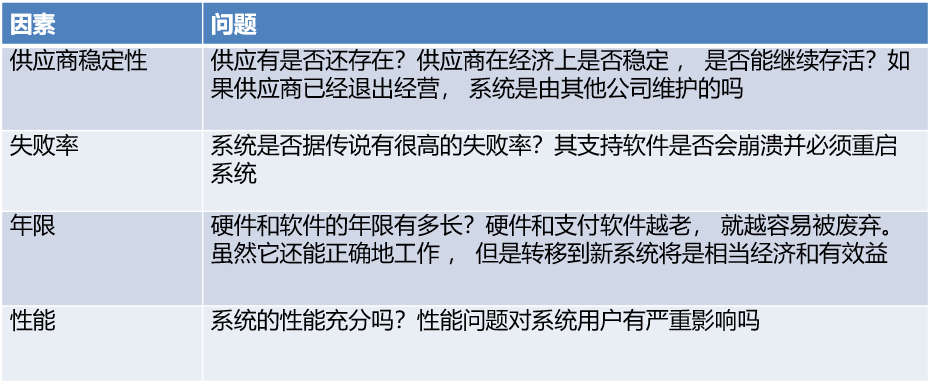
1.系统的使用:如果系统只是偶尔被使用或被很少一部分人使用， 那么它们含有较低的业务价值。系统使用较少，那么含较低的业务价值

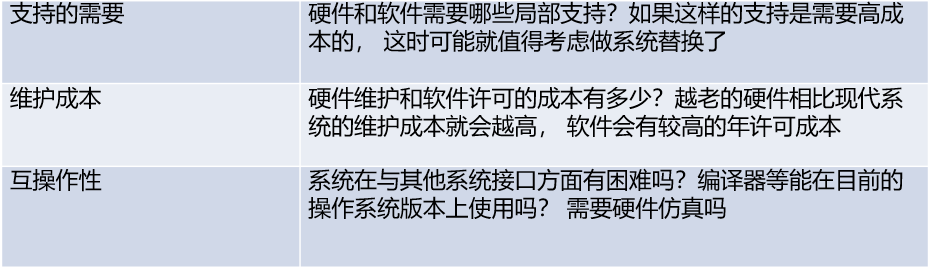
2.支持的业务流程:引入不合适的业务流程，系统业务价值降低

3.系统的可靠性:系统可靠性比高，系统业务价值降低

4.系统的输出:业务依赖系统输出，系统业务价值高

**环境评估中考虑的因素**





**应用评估中使用的因素**

