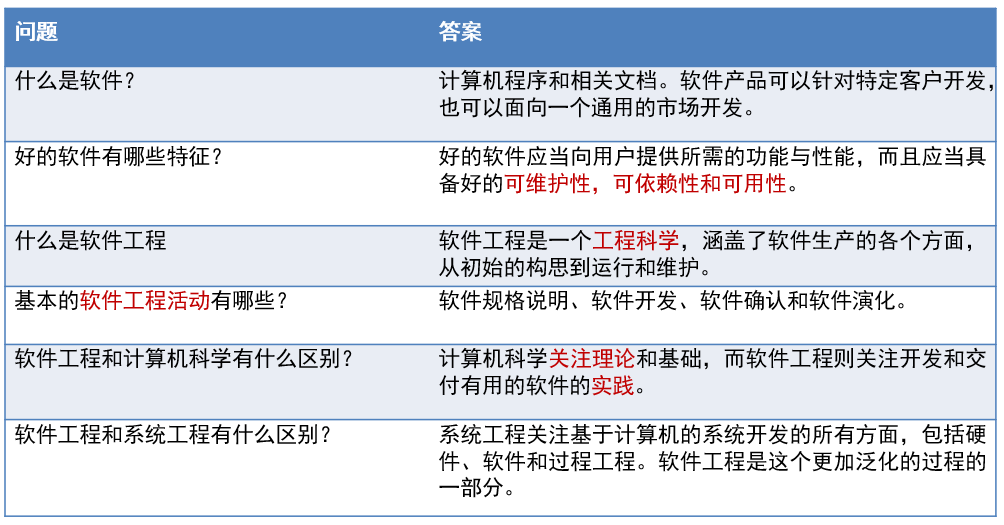
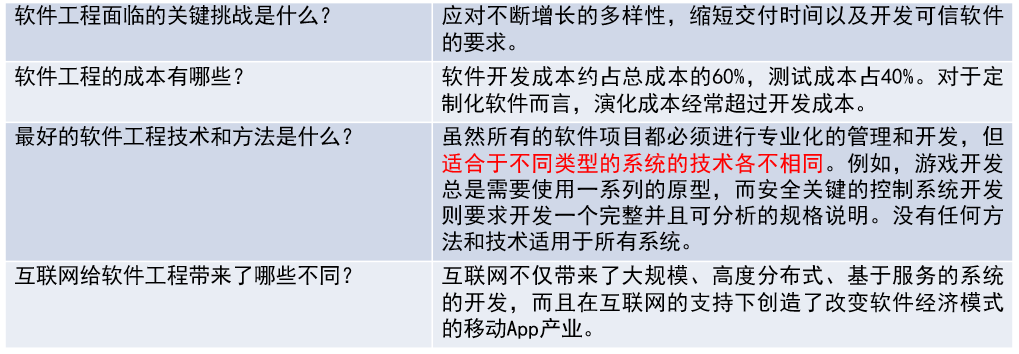
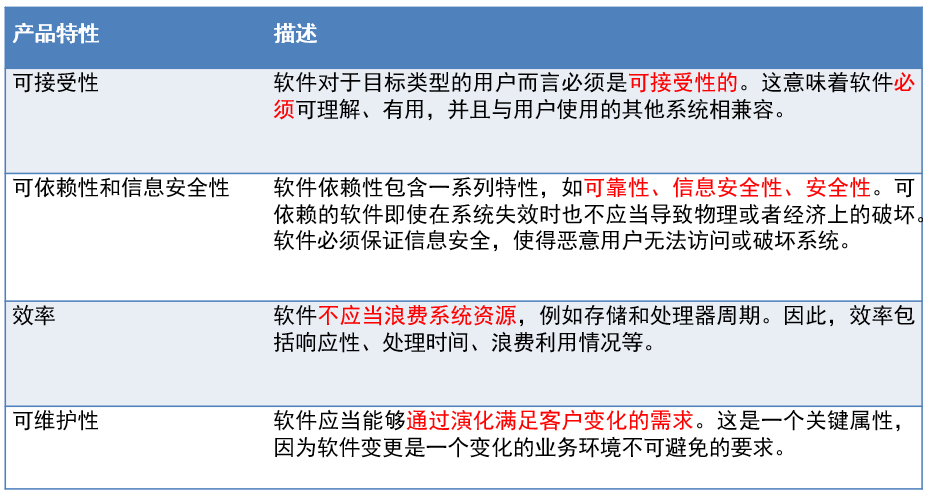
**Ch1 Introduction**

**软件工程的常见问题**





**好的软件的基本属性**



**软件工程的重要性**

1.个人和社会正越来越多地依赖于先进的软件系统。这就要求我们能够以经济而且快速的方式开发出可靠、可信的系统。

2. 运用软件工程方法和技术开发专业化的软件系统，比单纯作为个人编程项目编写程序更加便宜。对于大多数类型的系统，大多数成本是在软件投入使用后更改软件的成本。

**软件过程活动**

软件规格说明，客户和工程师定义所要生产的软件以及对其运行的约束。

软件开发，对软件进行设计和编程实现。

软件确认，对软件进行检查以保证它是客户所需要的。

软件演化，对软件进行修改以反映客户和市场需求的变化。

**影响大多数软件的一般问题**

1.异质性2.企业和社会的变革3.安全与信任4.规模

**应用类型**

1.独立的应用2.基于事务的交互式应用3.嵌入式控制系统4.批处理系统5.娱乐系统6.建模和仿真系统7.数据收集系统8.集成系统

**基于Web系统的软件工程的影响**

1.软件复用已经成为构建基于Web的系统的主流方法

2.基于Web的系统总是增量开发和交付的

3.软件可以使用面向服务的软件工程来实现，其中软件构建是独立的Web服务。

4. AJAX和HTML5等界面开发技术已经出现，这些技术支持Web浏览器中的丰富客户端界面的创建。

**要点**

1.软件工程是一门覆盖软件生产的各个方面的工程学科。

2.软件产品的基本属性是可维护性、 可靠性、 信息安全性、 效率以及可接受性。

3.软件规格说明、开发、确认和演化这些高层活动是所有软件过程中的一部分。

4.软件工程的基本思想适用于所有的软件系统。

5.世界上存在者很多类型的系统。 每一种类型的系统的开发都需要一种与之相适应的软件工程工具和技术。

6.软件工程的基本思想适用于所有的软件系统。

7.软件工程人员对软件工程行业和整个社会负有责任， 不应该只关心技术问题。

8.职业协会颁布的行为准则规定了一系列协会成员应该遵守的行为标准。

9.书中用到的三个学习案例

胰岛素泵控制系统

用于心理健康治疗的患者信息系统

野外气象站

**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**

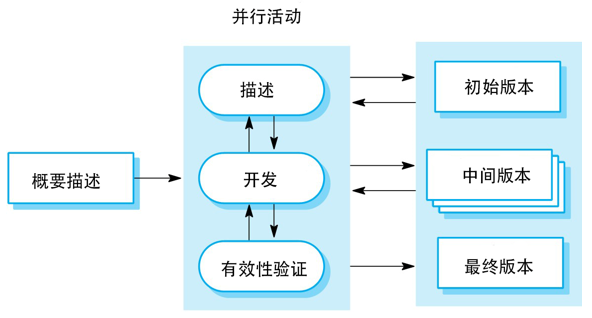
1.瀑布模型的主要缺点是在过程开始后很难适应变化。原则上，一个阶段在进入下一个阶段之前必须完成。

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

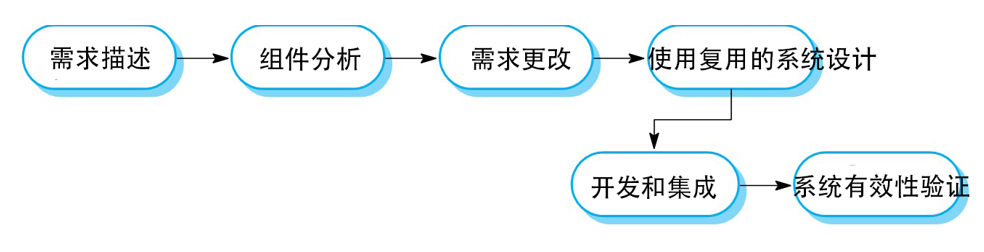
**增量式开发的问题**

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**

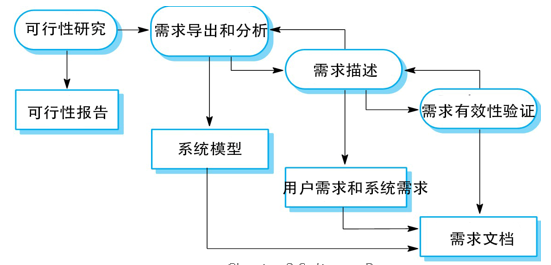
1.可行性研究

2.需求抽取和分析

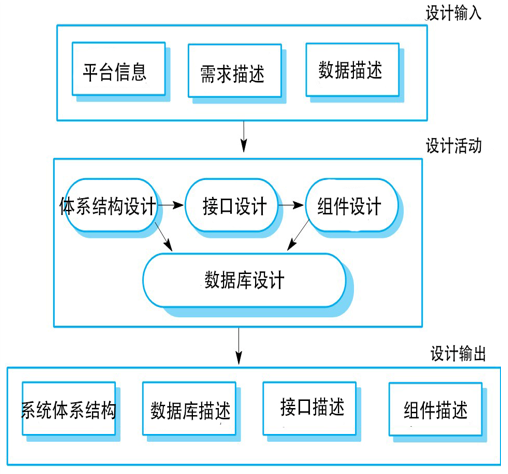
3.需求规格说明

4.需求确认

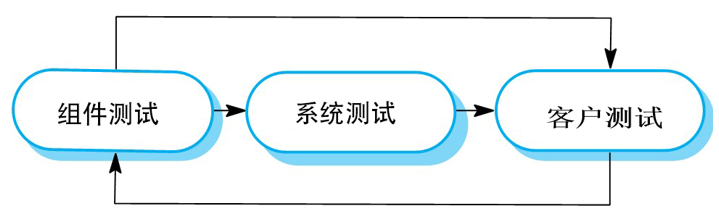
**需求工程过程**



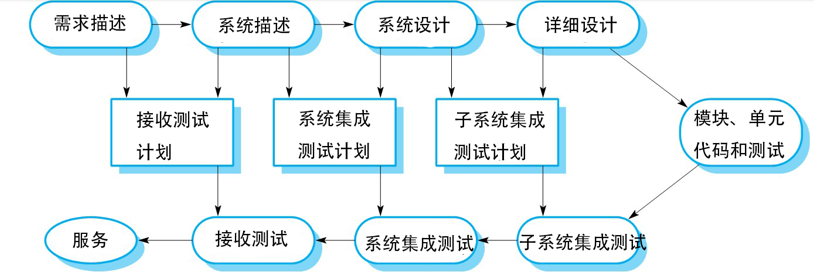
**设计过程的通用模型**



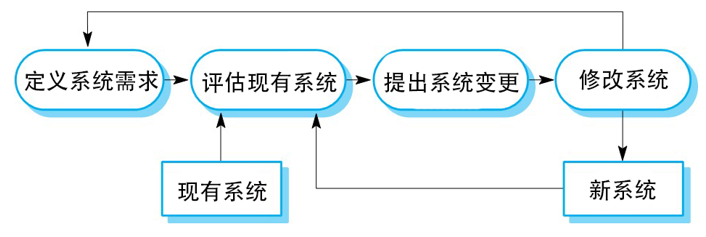
**测试阶段**



**计划驱动的软件过程测试阶段**



**软件系统演化**



**软件原型**

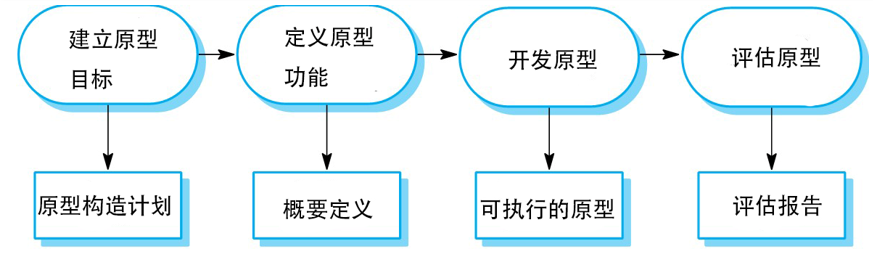
软件原型可以用在：

* + 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

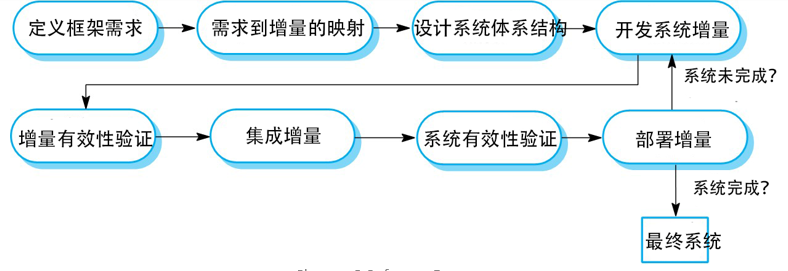
**增量式开发和交付**

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。

**要点**

1.软件过程是产生一个软件系统的一系列活动。软件过程模型是这些过程的抽象表示。

2.一般过程模型描述软件过程的组成。一般过程摸型实例包括瀑布模型、增量式开发、面向 复用的开发。

3.需求工程是开发软件描述的过程。

4.设计和实现过程是将需求描述转换为一个可运行的软件系统的过程。

5.软件有效性验证是检查系统是否与它的描述相一致， 以及是否符合系统用户的真正需要的过程。

6.软件进化是修改已存在的软件系统以适应用户新的需求的过程。 变更是一个持续的过程，软件必须在变更过程中保持可用。

7.过程应包含应对变更的活动。 这可能包含一个原型构造阶段， 以帮助避免在需求和设计上的错误决定。

8.过程应该适应迭代开发和交付， 这样变更时就不会对整个系统带来于扰。

9.Rational统一过程是现代一般过程模型， 其特点是由阶段（开端， 细化、 构造和转换）所构成，但是它把活动（需求、 分析和设计等）和阶段相区别。

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

**敏捷方法**

1.在20世纪80年代和90年代初，敏捷方法被提出。这些方法:

* + 关注代码而不是设计
  + 基于软件开发的迭代方法
  + 旨在快速交付工作软件，并快速发展以满足不断变化的需求。

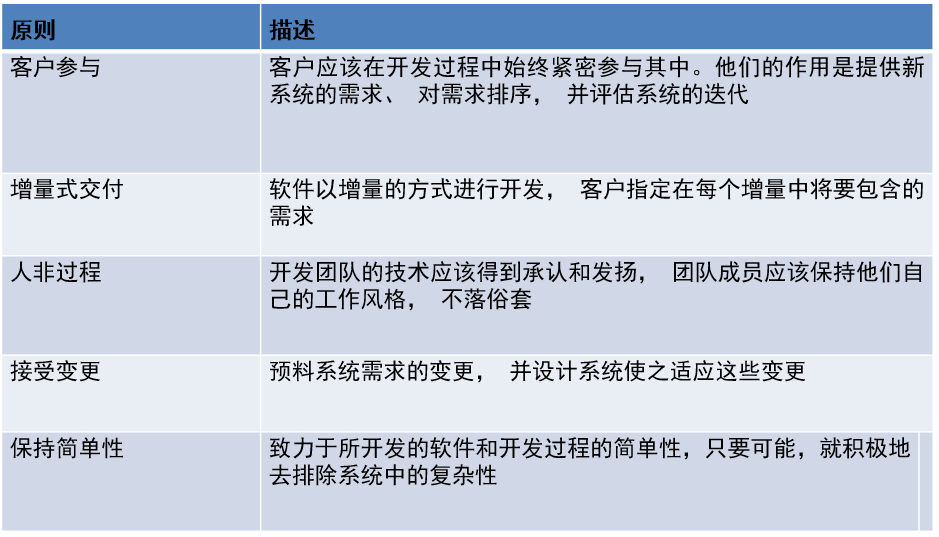
2.敏捷方法的目标是减少软件过程中的开销(例如，通过限制文档)，并且能够快速响应需求的变化，而不需要过多的返工。

**敏捷宣言**

我们在开发过程中发现更好的软件开发方法， 并帮助他人这样做。 通过这项工作我们得到如下评估：

* + 个体和交互胜过过程和工具
  + 可工作的软件胜过全面的文档
  + 客户协作胜过合同谈判
  + 响应变更胜过遵循计划

**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。

**极限编程和敏捷原则**

1.极限编程(XP)采用“极限”方法进行迭代开发。

* + 新版本可以每天构建几次;
  + 每2周向客户交付增量;
  + 必须为每个构建运行所有测试，并且只有在测试成功运行时才接受构建。

2.通过频繁的小的系统发布支持 “增量交付”。

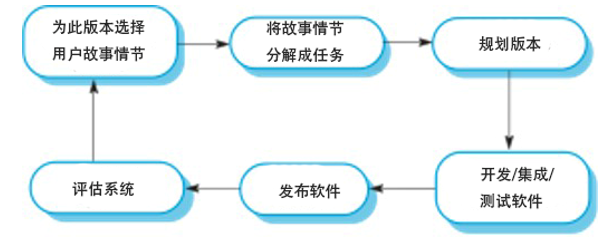
3.通过在开发团队中持续地约见客户实现“客户参与” 。

4.通过结对编程、 系统代码的共同拥有权、 以及不包含超长工作时间的可持续的开发过程来支持“人而不是过程” 。

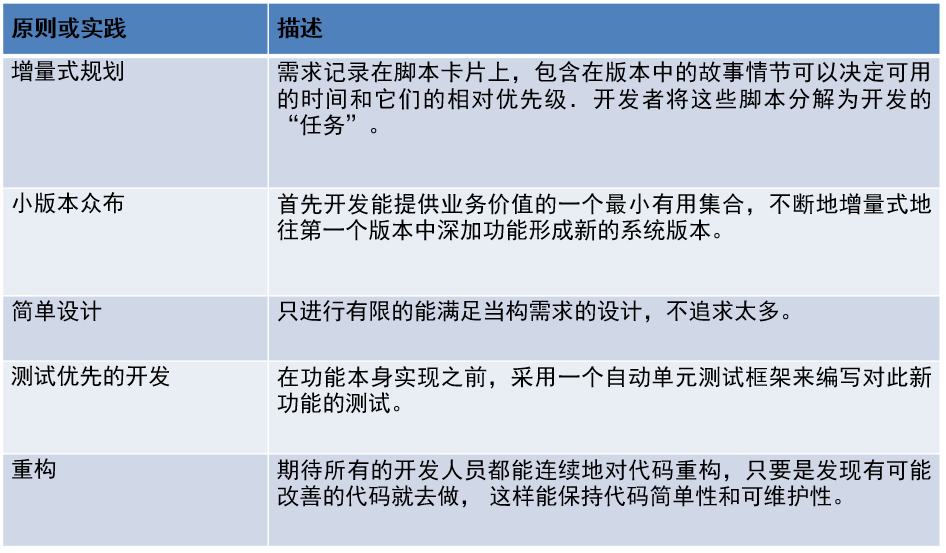
5.对变更的支持是通过经常性的系统版本发布。

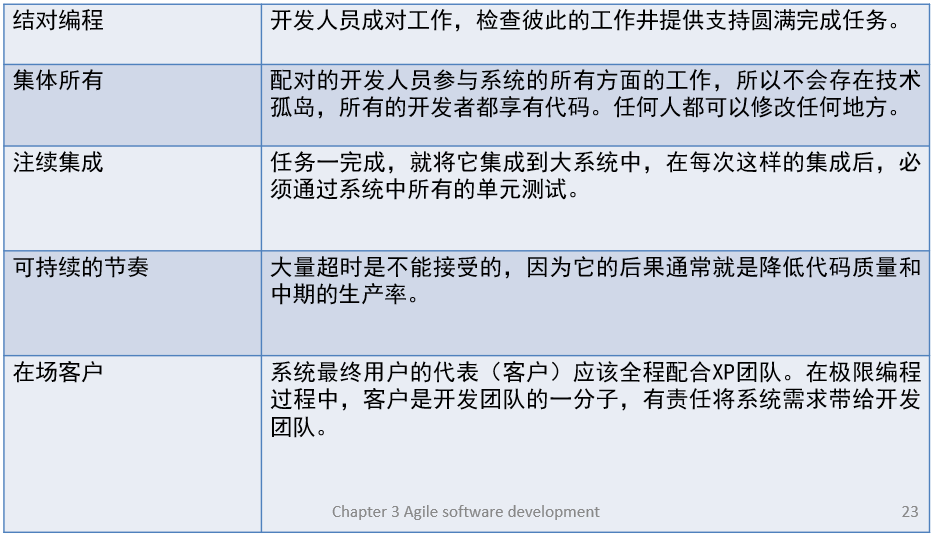
6.通过持续的重构来维护系统的简洁性。

**极限编程的版本循环**



**极限编程实践**





**重构**

1.这提高了软件的可理解性，从而减少了文档的需要。

2.更改更容易，因为代码结构良好且清晰。

3.但是，有些更改需要架构重构，这要昂贵得多。

**重构的例子**

1.类的层次结构重组以消除重复代码。

2.整理和重命名属性和方法，使它们更容易理解。

3.调用程序库中的方法替换代码。

**结对编程**

支持对于系统的共同所有权和共同责任的思想。

扮演了非正式的评审过程的角色． 因为每一行代码至少由两个人看着。

它鼓励重构，整个团队都可以从中受益。

**Scrum**

1.Scrum方法是一个通用的敏捷方法， 但它主要是注重迭代开发的管理， 而不是专门的管理敏捷软件工程的技术方法。

2.Scrum有3个阶段.

* + 第一个是规划纲要阶段，建立大致的项目目标和设计软件体系结构。
  + 接下来是一系列的冲刺循环，每个循环开发出一个系统增量。
  + 最后，项目结束阶段总结项目，完善需要的文档，如系统帮助和用户手册，并总结从项目中学到的经验。

3.Scrum 过程



4. Scrum 优点

1.产品被分解成一组可管理的和可被理解的块。

2.不稳定的需求并不阻碍工程进展。

3.整个团队的所有事情都是可见的， 因此改善了团队的沟通。

4.用户看到增量的及时交付， 且得到对于产品如何工作的反馈。

5.客户和开发者之间彼此信任， 创造了积极的文化， 每个人都希望项目成功。

**要点**

1.敏捷方法是一种专注于快速开发的增量式开发，频繁地发布软件、降低过程开销、 生产高质量的代码。

2.极限编程是一种著名的敏捷方法，它集成了一系列好的编程经验． 例如频繁地软件发布，连续软件改善和客户参与到软件开发团队。

3.极限编程的一个特别长处是在创建程序特征之前开发自动测试。在增量集成进系统的时候所有的测试必须成功执行。

4.Scrum方法是一种提供项目管理框架的敏捷方法。它的核心是一组冲剌循环，开发一个系统增量是有固定的时间周期的。规划是基于积压的工作的优先权安排的，选择高优先权的任务开始冲刺循环。

**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。

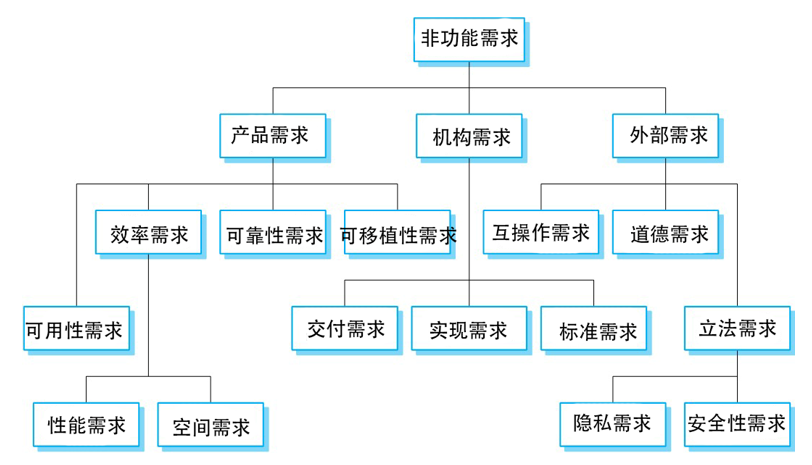
2.非功能需求

* + 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。

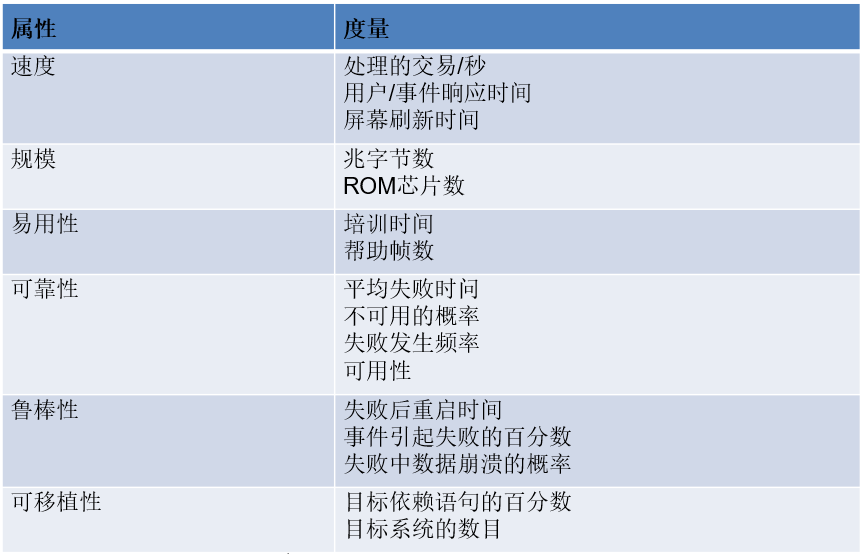
3.邻域需求

* + 系统运行领域的约束条件。

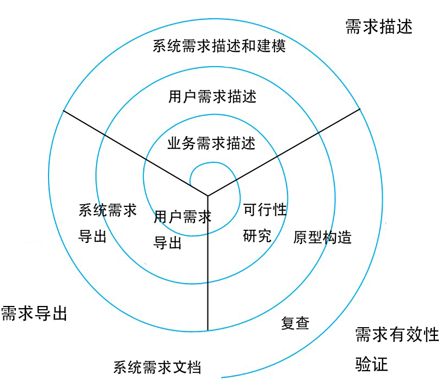
**非功能需求的类型**



**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。

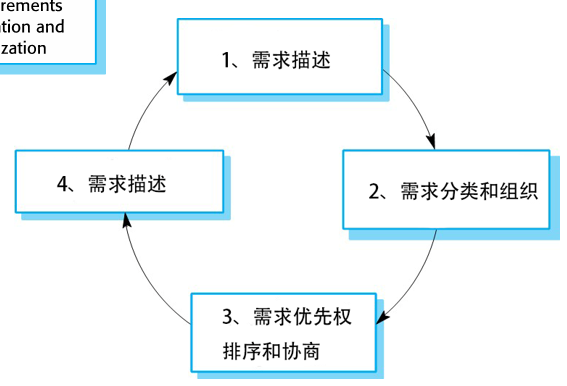
2.系统利益相关者用他们自己的语言表达需求。

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。

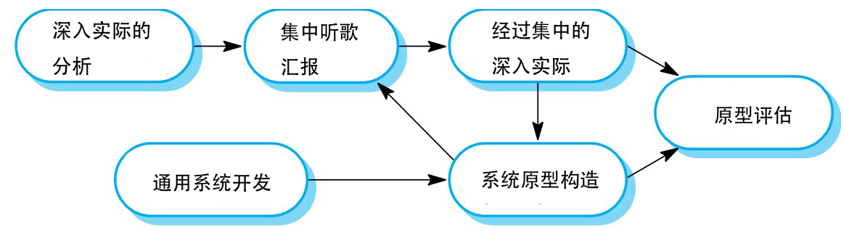
5.进行需求分析的经济和业务环境是动态的。在分析过程期间它不可避免会变化。

**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**

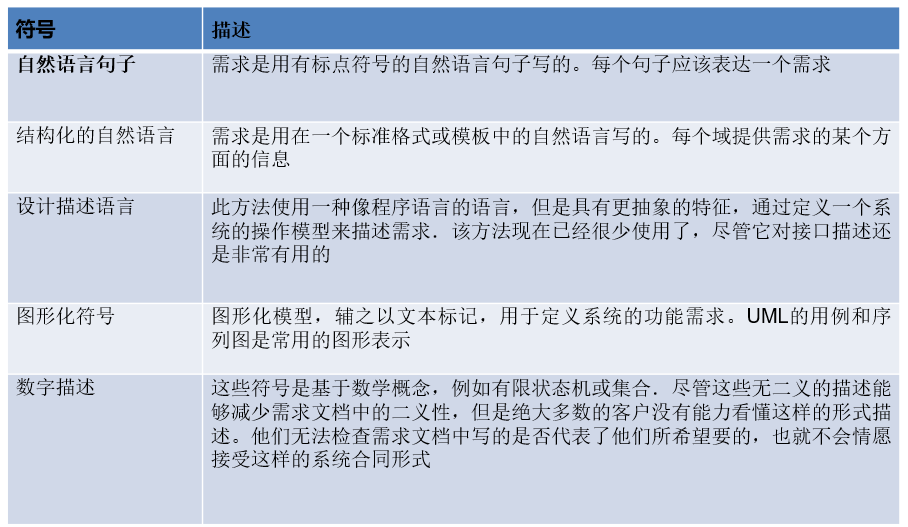
1.需求规格说明就是在需求文档中写下用户需求和系统需求。

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?

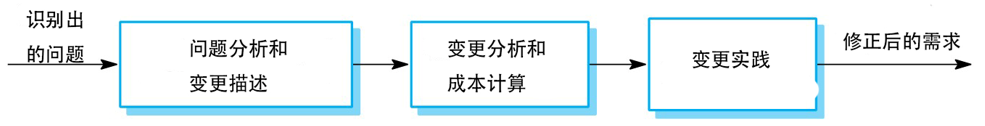
3.可追溯性:是否清楚说明了需求的来源?

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**



**要点**

1.一个软件系统的需求描述了系统应该做什么以及定义系统运行时和实现时的约束。

2.功能需求是有关系统一定要提供的服务或者是必须执行的计算的描述。

3.非功能需求约束所开发的系统和所采用的开发过程。

4.这些需求与系统的总体特性相关， 且作用于系统整体。

5.软件需求文档是经过认可的系统需求描述。 它应该被很好地组织，以便既可以为系统客户使用又可以为软件开发者使用。

6.需求工程过程是一个迭代的过程，包括需求导出、描述和有效性验证。

7.需求的导出和分析是一个反复的过程， 可以看做是一种螺旋式活动——需求发现， 需求分类和组织， 需求协商和需求文档化。

8.你可以使用一系列的技术来获取需求，包括访谈、场景、用例和深入实际。

9.需求有效性验证是检查需求的有效性 、 一致性 、 完备性 ， 真实性和可检验性。

10.业务 、 组机机构和技术上的变更不可避免地导致软件系统需求的变更。 需求管理就是管理和控制这些变更的过程。

**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**

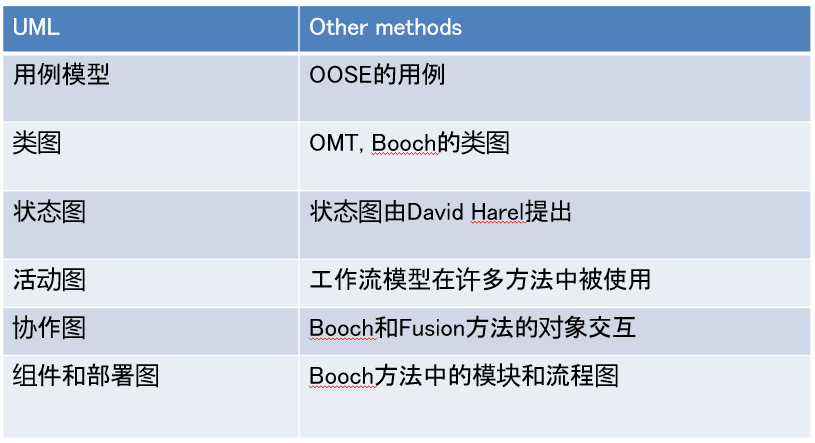
1.UML是一种用于软件产品的文档化、结构化、说明与具体化的标准语言。

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**

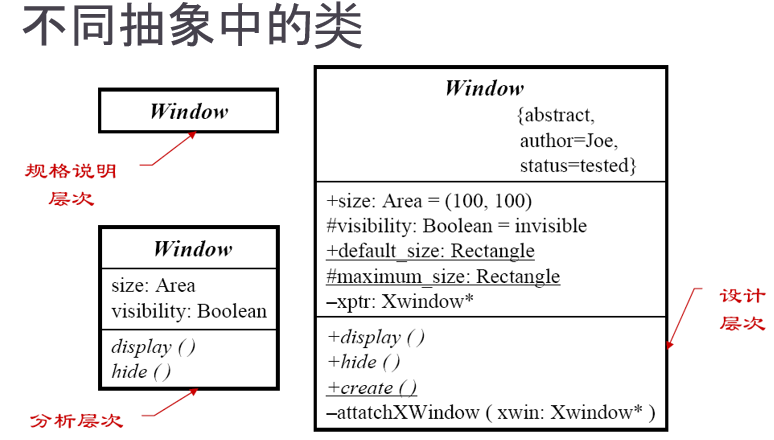
1.事物:类、接口、包、组件等。

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

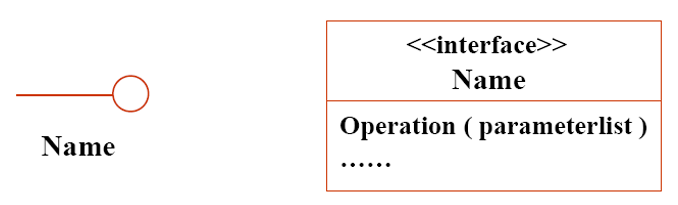
**结构事物:对象类**



**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



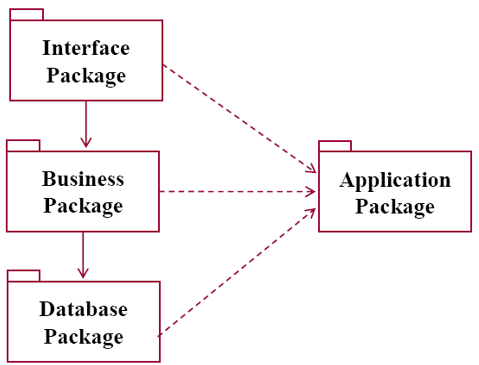
**行为事物:交互**

对象之间的消息交换

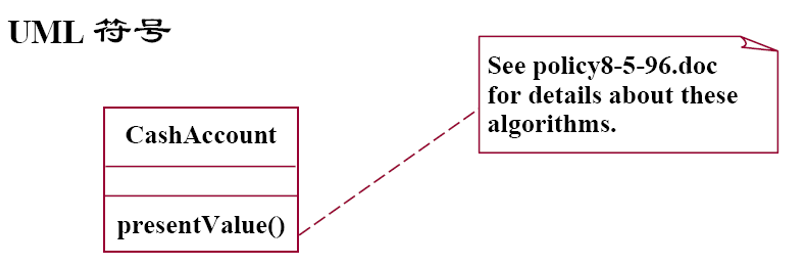


**分组事物:包**

把一些事物放在一个包中

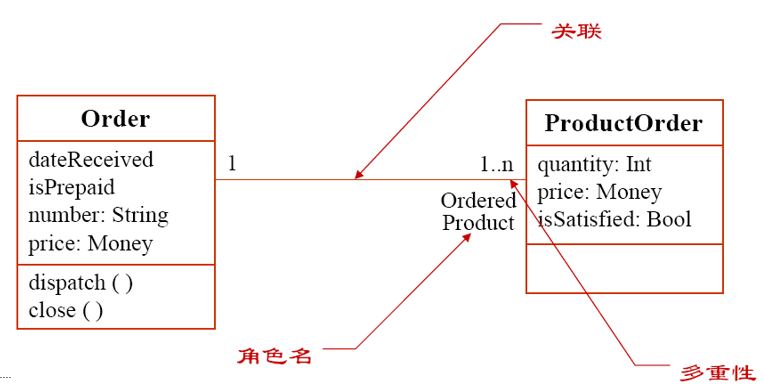


**注释事物:注释**

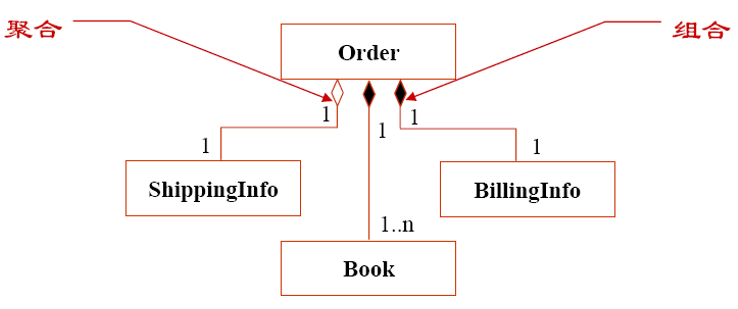


**UML 关系**

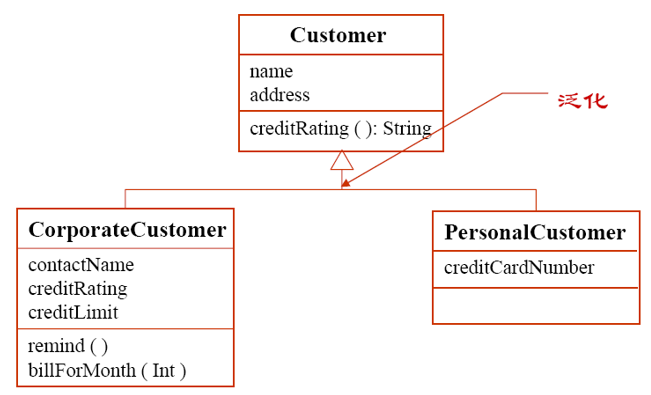
**关联**



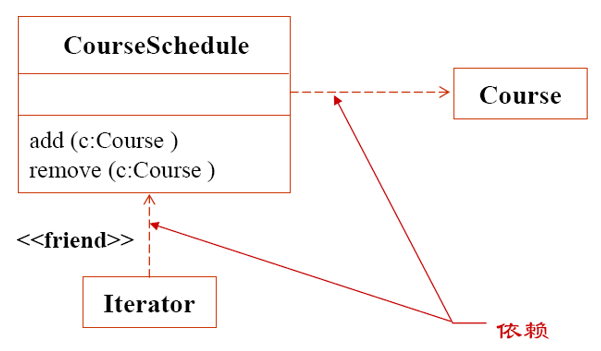
**特殊关联**



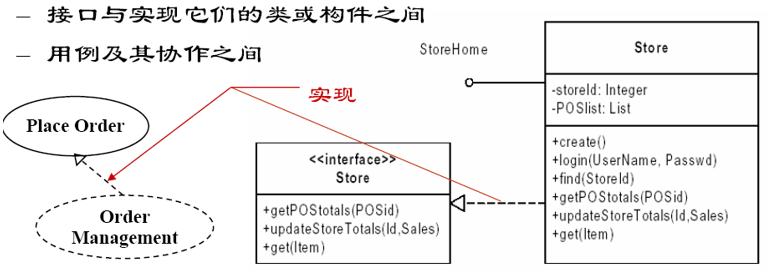
**泛化:一般事物与特殊事物之间的关系**



**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。

**要点**

1.模型是忽略了一些系统细节的系统抽象表示。 建立互补的系统模型来表示系统的上下文 、 交互、结构和行为。

2.上下文模型描述所建模的系统是如何在含有其他系统和流程的环境中工作的。

3.用例图和时序图用来描述系统用户之间或系统用户和其他系统之间的交互。 用例描述的是系统和外部参与者之间的交互，时序图通过表示系统对象之间的交互为用例图添加更多的信息。

4.结构模型表示的是系统的体系结构。 类图用来定义系统中类的静态结构以及类之间的关联关系。

5.行为模型用来描述可执行系统的动态行为。 我们可以从系统处理数据的角度和事件激励系统产生响应的角度来建立这种模型。

6.活动图用来为数据的处理过程建摸，其中每一个活动图代表一个处理步骤。

7.状态图用来为响应内外部事件的系统行为建模。

8.模型驱动工程是一种软件开发的方法，其中的系统表示为可自动转换为可执行代码的模型的集合。

**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。

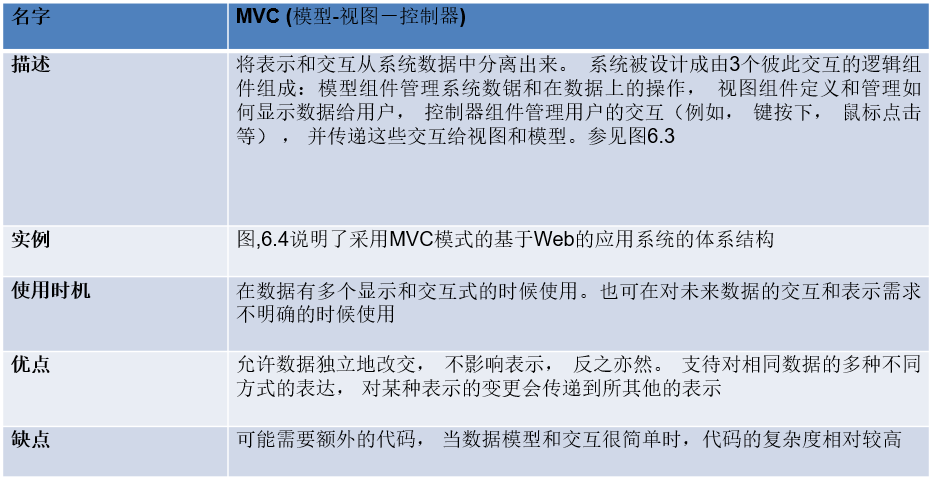
2.进程视图，显示在运行时系统是如何组织为一组交互进程.

3.开发视图， 它显示了软件是如何为了开发而被分解的。

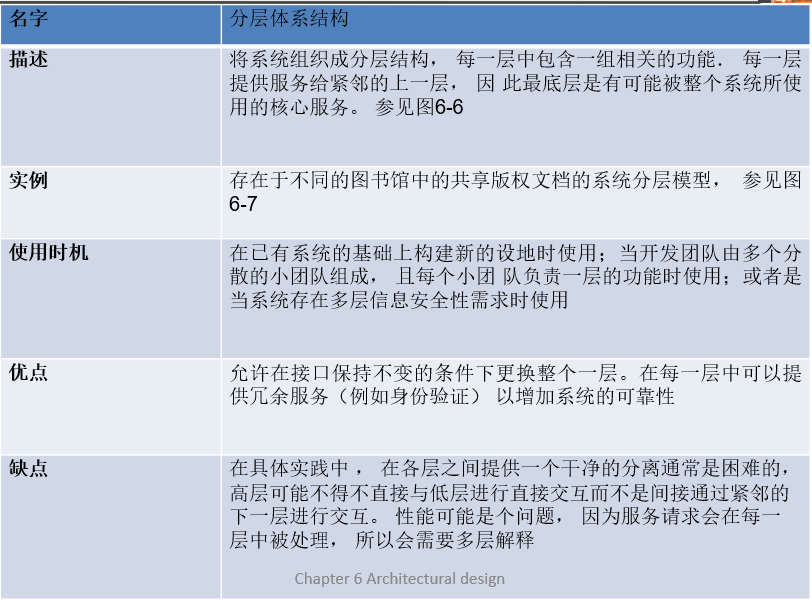
4.物理视图， 它显示了系统硬件和系统中软件组件是如何分布在处理器上的。

5.相关的使用用例或场景(+1)

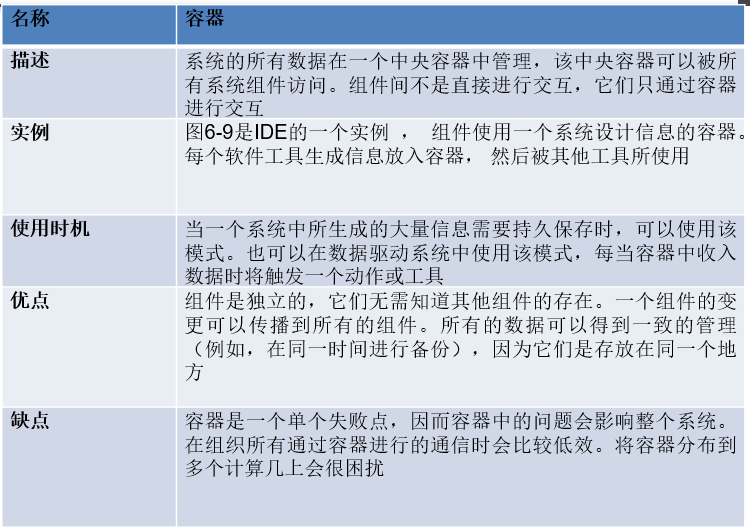
**模型-视图－控制器(MVC)摸式**



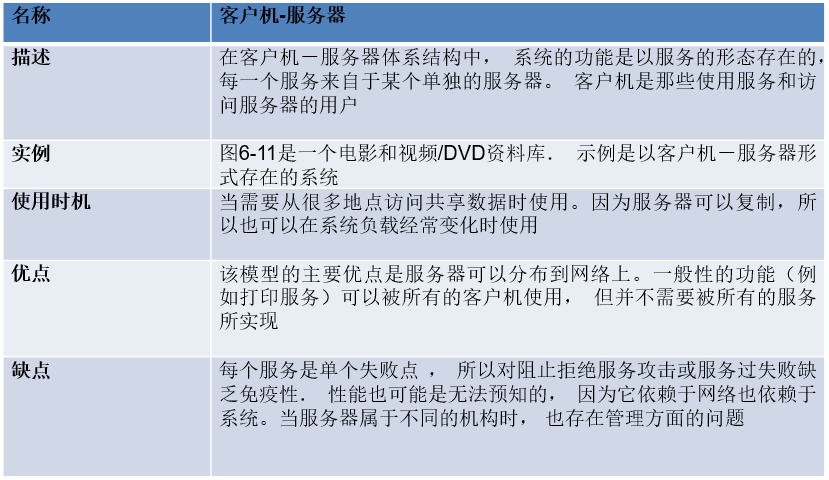
**分层体系结构模式**



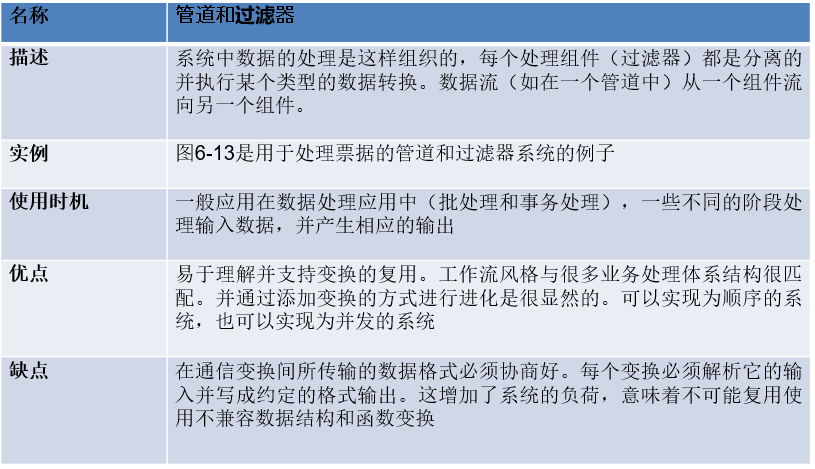
**容器模式（知识库模式）**



**客户机－服务器模式**



**管道和过滤器模式**



**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

**要点**

1.软件体系结构是有关软件系统如何组织的描述。

2.体系结构设计决策包括对应用类型的决策、 系统分布的决策、 所使用的体系结构风格的决策 。

3.体系结构可能会从许多不同的视角和视图被文档化。 可能的视图包括概念视图 、 逻辑视图、进程视图 、 开发视图。

4.体系结构模式是复用通用的（一般的）系统体系结构知识的一种方法。 它描述体系结构，解释这样的体系结构的使用时机以及它的优缺点。

5.应用系统体系结构的一般模型能帮助我们理解应用的运作， 比较相同类型的应用， 验证应用系统设计的有效性并能达到大粒度的复用。

6.事务处理系统是交互式系统 ， 允许数据库中的信息被很多远程用户访问和修改。

7.语言处理系统用来将文本从一种语言翻译成另一种语言， 来执行输入语言所定义的指令。 它们包括一个翻译器和一个执行生成的语言的抽象机。

**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。

**要点**

1.软件设计和实现是相互交织的活动。设计的细节层次取决于待开发系统的类型以及是否使用计划驱动或敏捷开发方法。

2.面向对象的设计过程包括系统体系结构的设计、 识别系统中的对象、 运用不同的对象模及描述设计， 以及文挡化组件接口。

3.面向对象的设计过程中会产生不同的模型。 这些模型包括静态模型（类模型、 泛化模型、 关联模型）和动态模型（时序模型、 状态机模型）。

4.要对组件的接口进行精确的定义， 这样其他的对象才可以使用它们。 可以使用UML接口模板来定义这些接口 。

5.开发软件时， 我们总是要考虑复用现存组件的可能性， 无论是对组件、 服务还是对整个系统的复用。

6.配置管理是管理系统变更的过程。 当一个团队的人员协同开发软件时， 这个过程是十分必要的。

7.多数软件开发过程属于宿主机－目标机开发模式。 要用宿主机上的IDE开发软件， 然后传输到目标机运行。

8.开源开发指公布系统中的源代码， 这意味着很多人可以提出更改建议并改善软件。

**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。

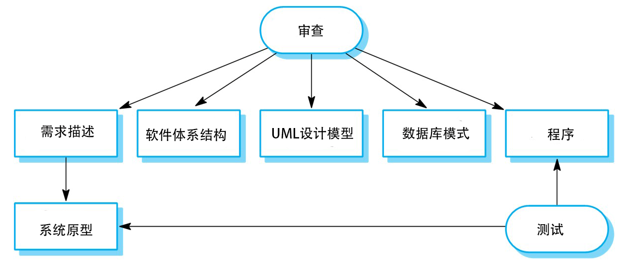
过程:1向开发人员和系统客户演示软件满足其需求2一次成功的测试表明，系统按预期运行

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**



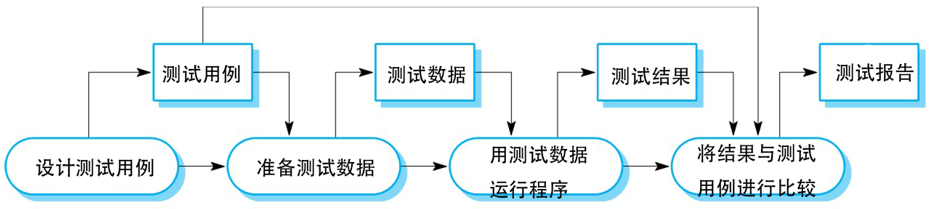
**软件审查的优点**

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**

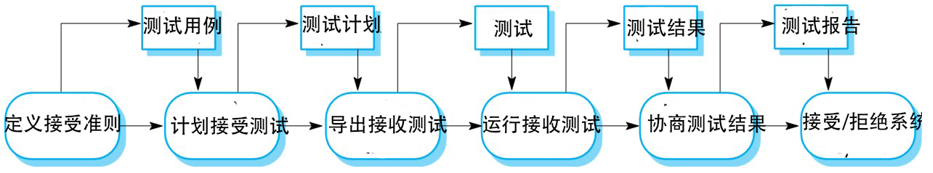
1.代码覆盖：原则上， 我们所写的每个代码片段都至少有一个测试。 因此， 我们可以确信系统中的所有代码都实际执行过。

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**



**要点**

1.测试只能证明系统中存在错误。 它不能说明系统中不再有缺陷。

2.开发测试是软件开发小组的责任。 独立的测试小组应该在系统发布给客户之前负起测试系统的责任。

3.开发测试包括单元测试， 即测试单个对象和方法；组件测试， 即测试相关的一组对象； 系统测试， 即测试部分或完整的系统。

4.测试软件时， 我们应该根据经验和原则把软件打散来选择测试案例的类型， 选择那些在其他系统中已经证明是能有效地发现缺陷的那些测试案例类型。

5.有可能的话， 我们应该编写自动化测试。 将这个测试嵌人到程序中， 这样的程序可以每次当系统一有变化就运行。

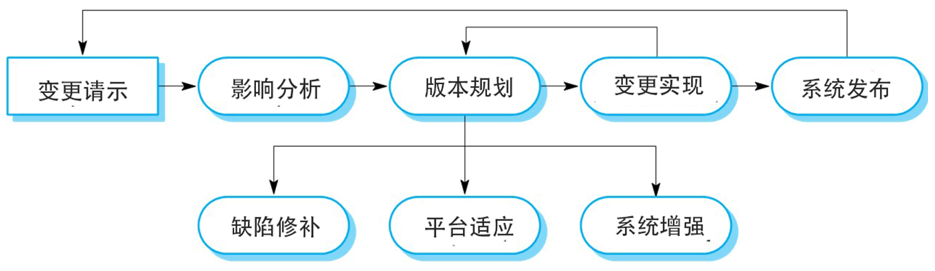
6.测试优先开发是开发的一种方法， 在代码测试之前就巳经写好测试。

7.情景测试是非常有用的， 因为它可以复制系统的使用。

8.接收测试是一个用户的测试过程， 目的是决定这个软件是否足以进行部署。 以及能否用在它的实际操作环境中。

**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**

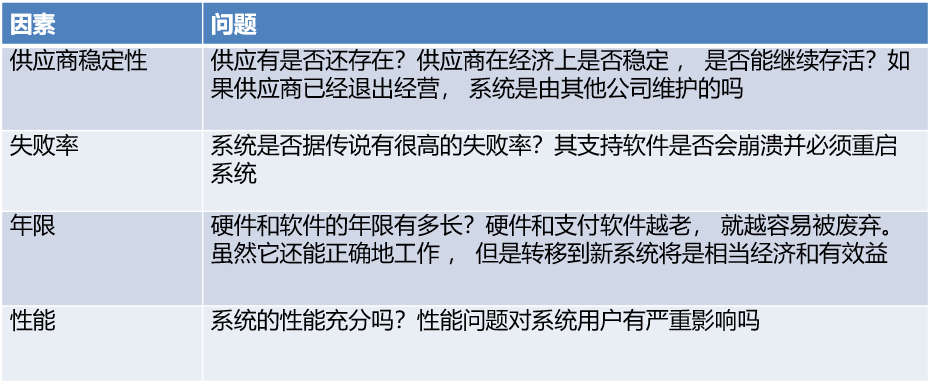
1.系统的使用:如果系统只是偶尔被使用或被很少一部分人使用， 那么它们含有较低的业务价值。系统使用较少，那么含较低的业务价值

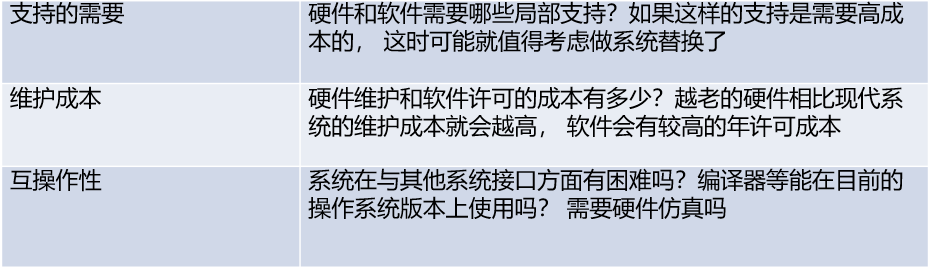
2.支持的业务流程:引入不合适的业务流程，系统业务价值降低

3.系统的可靠性:系统可靠性比高，系统业务价值降低

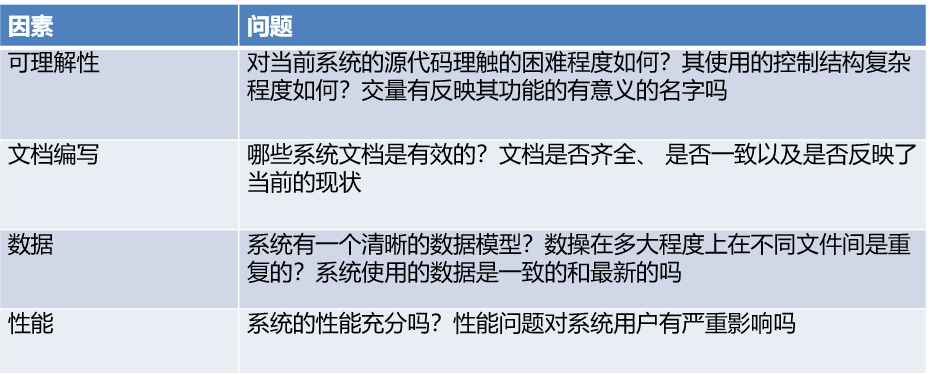
4.系统的输出:业务依赖系统输出，系统业务价值高

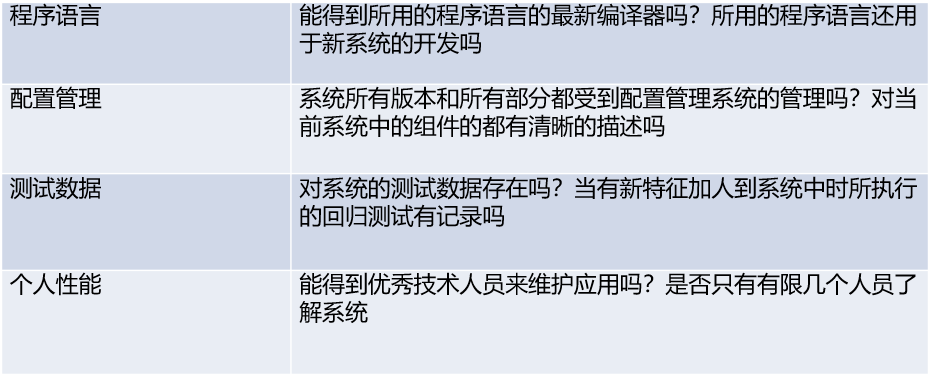
**环境评估中考虑的因素**





**应用评估中使用的因素**





**要点**

1.软件开发与进化应当是一个集成的、完整的 、 增量式的过程， 它可以用螺旋模型表示。

2.对于定制系统来说，软件维护的费用一般超过了软件开发的费用。

3.软件进化过程是由变更请求驱动的， 包括变更影响分析、 版本规划和变更实现。

4.Lehman定律， 例如变更是持续的观点，描述了许多来自于系统进化长期研究的领悟。

5.软件维护有3种基本类型：修改软件中的缺陷， 使软件适应不同操作环境， 以及向系统中添加或修改功能。

6.软件再工程的目标在干改善系统结构和文档 ， 使其更容易理解和变更。

7.重构是一种预防性维护，即对程序进行更改以保留其功能。

8.对遗留系统的业务价值、应用软件质量以及应用软件环境进行评估，然后决定是否更换、转换和维护系统。

**Chapter 22 – Project Management**

**成功标准（目标）**

1.在约定的时间将软件产品交付给客户；

2.将全部成本控制在预算之内；

3.交付的软件产品满足客户的要求．

4.保持一个愉悦并且运作良好的开发团队。

**管理活动**

*1.项目规划*:项目管理者负责规划、 评估以及调度项目开发， 并给人员分配任务。

*2.工作报告*:项目管理者通常有责任向客户和开发此软件的公司的经理汇报项目进展情况。

*3.风险管理*:项目管理者必须评估井且监测某些可能影响项目的风险， 并在问题出现时立刻采取行动。

*4.人员管理*:项目管理者负责管理一个团队。 他们必须为他们的团队挑选人员并且建立能够使团队高效运作的工作方式。

*5.提出书面建议*:软件项目的第一个阶段可能要写出完成该项目的建议书， 在建议书中要写清楚项目的目标和实现该目标的方法。

**常见的项目 、 产品和业务风险的实例**



**风险管理分类**

1.项目风险 是影响项目进度或项目资源的风险。

2.产品风险 是影响开发中软件的质量或性能的风险。

3.业务风险 是影响软件开发机构或软件产品购买机构的风险。

**风险管理过程**

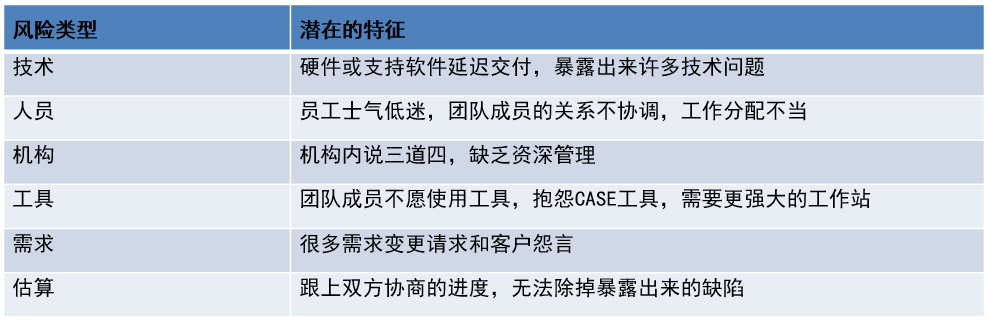
1.风险识别:识别可能的项目风险、 产品风险和业务风险。

2.风险分析:评估这些风险出现的可能性及其后果。

3.风险规划:制定计划说明如何规避风险或最小化风险对项目的影响。

4.风险监控:对整个项目的风险进行监控。

**风险因素**



**人格类型**

人格类型也会对激励产生影响：1面向任务型2面向自我型3面向交互型

**要点**

1.软件工程项目要想按进度、 按预算进行开发， 完善的软件项目管理是必要的。

2.软件项目管理与其他的工程管理有明显区别。 软件产品是无形的。 软件项目可能很独特或者有创新， 这样就没有实在的经验以供借鉴。 软件过程没有传统工程过程那么成熟。

风险管理现在被看做最重要的项目管理任务之一。

3.风险管理包括识别并评估重大的项目风险， 从而判断这些风险发生的可能性大小及其后果的严重程度。对很有可能发生并有潜在严重性的风险， 应该制定有关规避、 管理和解决可能发生的风险的计划， 包括对风险发生的分析和当风险发生时的应对措施。

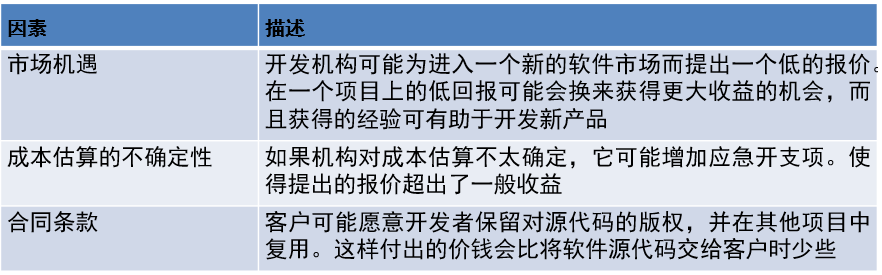
4.对一个人的激励因素会包括与其他人的交互作用、 得到管理层和同行的赏识， 以及得到个人发展的机会。

5.软件开发小组不宜太大且要有凝聚力。 影响小组效率的关键因素是小组中的人员 、 小组组织的方式以及组员之间的沟通。

6.一个小组内部的沟通受许多因素的影响， 如小组成员的职位 、 小组的规模、 小组的性别组成、 小组成员的个性以及可用的沟通渠道。

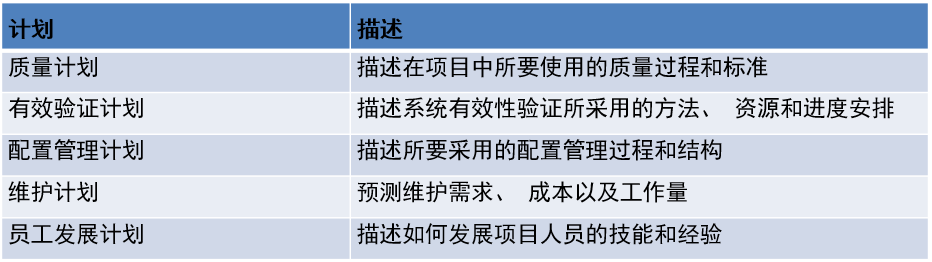
**Chapter 23 – Project planning**

**影响软件报价的因素**

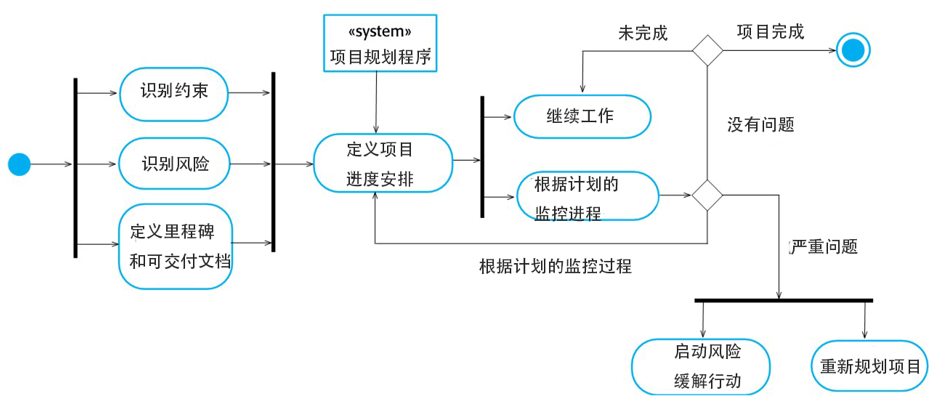




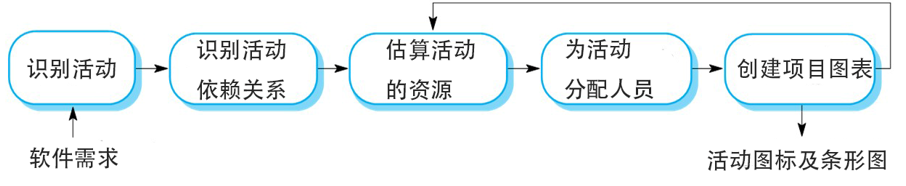
**项目辅助计划**



**项目规划过程**



**项目进度安排过程**



**进度安排问题**

1.估计问题的难度以及开发解决方案的成本是困难的。

2.工作效率与从事一项任务的人数不成正比。

3.将人员添加到一个延迟的项目中会使其延迟，这是因为通信开销。

4.意外总是会发生。要留有应急计划。

**敏捷计划阶段**

1.发布规划，包括对未来几月的展望以及决定系统的发布版本中应该包含的功能。

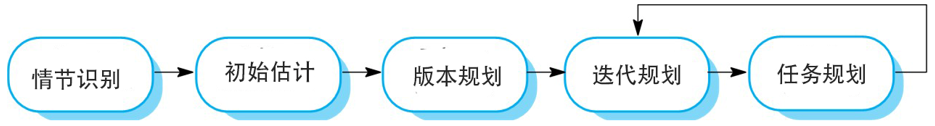
2.迭代规划，包括短期的展望，主要是规划系统的下一个增量。一般这将花费整个团队2~ 4个星期的工作量。

**算法成本建模**

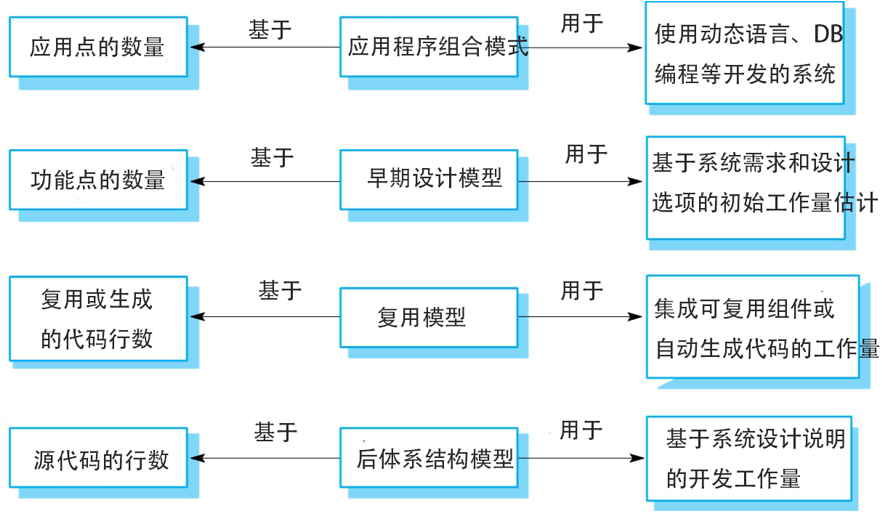
Effort = A × SizeB  ×M

A 是一个组织有关的常数；B反映大型项目的不成比例的努力；M是一个乘数因子，反映产品、流程和人的属性等；size 一个对软件的代码规模或是用功能点或应用点表示的功能的估算。

**XP中的规划**



**COCOMO 估计模型**



**应用组合模型**

1.支持原型设计项目和有广泛重用的项目。

2.根据开发人员标准的应用程序（对象）点/月的生产率来估算。

3.采用CASE工具的也考虑在内。

4.计算公式为

* + PM = ( NAP × (1 - %reuse/100 ) ) / PROD
  + PM是以人月为单位的工作量，NAP是交付系统的应用程序点的总数， PROD是应用点生产率，%reuse 是在开发中利用的代码量的估计

**乘法因子**

1.产品属性:关注正在开发的软件产品的要求特性。

2.硬件属性:由硬件平台强加给软件的限制和约束。

3.人员属性:考虑人员在项目工作上的经验和能力。

4.项目属性:关心的是软件开发项目的具体特点。

**要点**

1.系统的价格并不仅仅取决于它的开发成本估计，它可能会因市场和机构的优先事项进行调整。

2.计划驱动开发是围绕一个详细定义的项目计划进行组织的。项目计划定义了项目活动 、 计划的工作量 、 活动进度安排和每项活动的负责人。

3.项目进度安排需要创建有关项目计划的各种图形化表示。 用来表示活动期限和人员使用时间的条形图是在进度安排中中最常使用的。

4.极限规划游戏是让整个团队成员都参与到项目规划中来。计划做成增量式的， 如果问题出现， 计划将被调整 ，通过减少软件的功能性而不是延期交付一个增量， 修改计划。

5.软件的估算技术可能是基于经验的， 管理者对需要的工作量进行判断；或者是使用算法，需要的工作量通过使用其他的估算工程中的参数计算得到。

6.COCOMO II 成本计算模型是一个比较成熟的成本估算模型 ， 它将项目、 产品 、 硬件以及人员因素和产品大小和复杂属性都考虑在内。

**Chapter 24 - Quality Management**

**软件质量管理**

1.确保软件产品达到所需的质量水平。

2.3个重要的关注点：

* + 在机构层面， 质量管理与建立能生产高质量软件的机构过程框架和标准相关。
  + 在项目层面， 质量管理包括专门的质量过程的应用、对所规划的过程的执行情况的检查。
  + 在项目层面，质量管理同样关注于为项目确立一个质量计划。 质量计划应该给出项目的质量目标， 定义应该使用什么样的过程和标准。

**软件达到的目标**

1.开发过程是否遵循编程和文档化标准？

2.软件是否得到了正确的测试？

3.软件是否足够可靠能被投人使用？

4.软件性能是否对于正常使用是合格的？

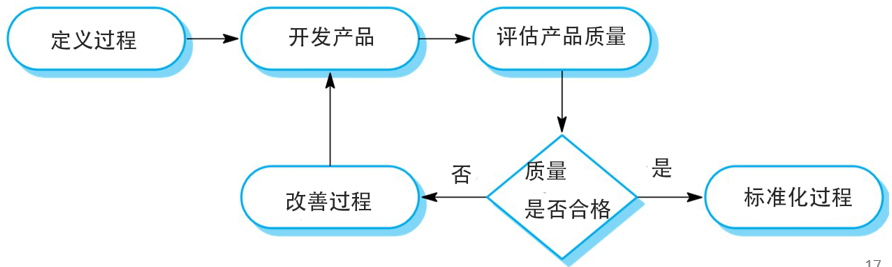
5.软件是否可用？

6.软件是否结构良好并且易于理解？

**软件质量属性**



**基于过程的质量**



**软件标准**

1.标准定义了产品或过程所需的属性。它们在质量管理中起着重要的作用。

2.标准可以是国际、国家、组织或项目标准。

3.产品标准定义了所有软件组件应该表现出的特征，例如通用的编程风格。

4.过程标准定义了软件过程应该如何实施。

**软件标准的重要性**

1.软件标准封装了最成功的，至少是最恰当的软件开发经验。

2.软件标准提供了一个框架，围绕这个框架才能实现质量保证过程。

3.软件标准还有助于工作的连贯性，有一个人着手进行的工作别人可以接着做

**标准存在的问题**

1.在软件工程师看来，它们可能是不相关或者不是最新的。

2.他们往往涉及太多的形式化内容。

3.如果软件工具不支持这些标准，那么为了能使文档和标准联系起来，则需要繁琐的手工劳动。

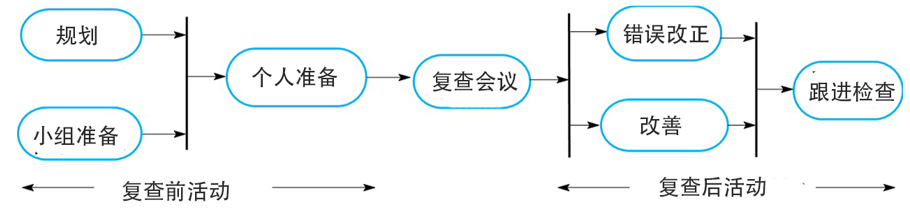
**复查与审查**

这种方法通过一组人检查软件过程的全部或者一部分、系统或者相关文档以发现潜在的问题.

目标不同的几种审查方法

* + 设计或程序检查 (产品)
  + 过程审查 (产品和过程)
  + 质量审查 (产品和标准)

**软件复查过程**

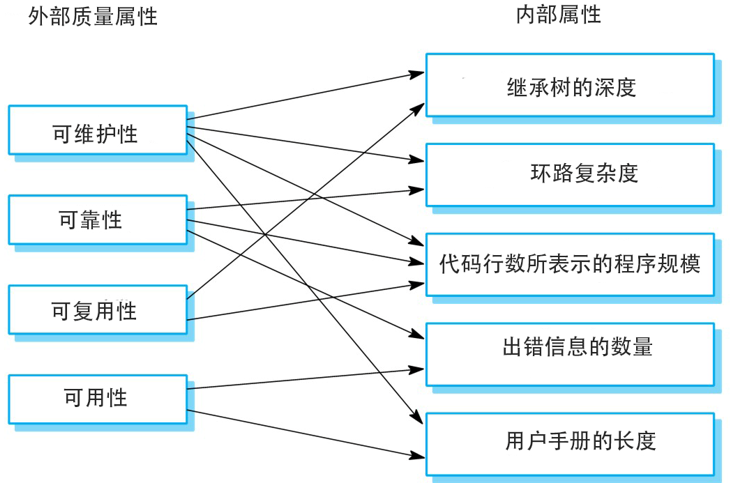


**软件产品度量方法**

1.给系统质量属性赋值:通过度量系统组件的特性， 比如回路复杂性， 并将这些度量综合起来， 就能评估系统质量属性， 比如可维护性。

2.找出质量低于标准的系统组件:度量能识别那些特性背离某些规范的个别组件。例如， 可以度量组件以发现那些有着最高复杂性的问题组件。 由于复杂度高难于理解， 这些组件更可能包含错误。

**软件的内在和外在关系**



**工业度量的问题**

1.引入一个机构量度程序的投资回报是无法量化计算的。

2.现在还没有软件度量的标准， 或是没有标准化的度量和分析过程。

3.在许多公司， 软件过程是非标准化的，没有很好定义也是很难控制的。

4.许多关于软件度量和量度的研究主要侧重于基于代码的量度和计划驱动开发过程。 然而， 越来越多的软件的开发是通过配置ERP系统、COTS系统。

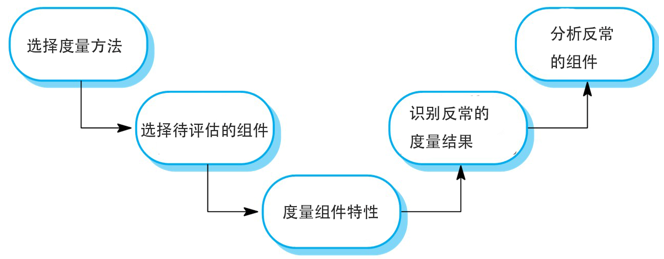
5.引入度量增加了额外的开销。

**产品量度**

产品量度分类：

* + 动态度量，通过对执行中的程序度量所收集到的。
  + 静态量度， 通过对系统各种表现形式（如设计、 程序或文档等）度量所收集到的。
  + 动态量度用于评估一个程序的效率和可靠性。
  + 静态量度则用于评估一个软件系统或系统组件的复杂性、易懂性和可维护性。

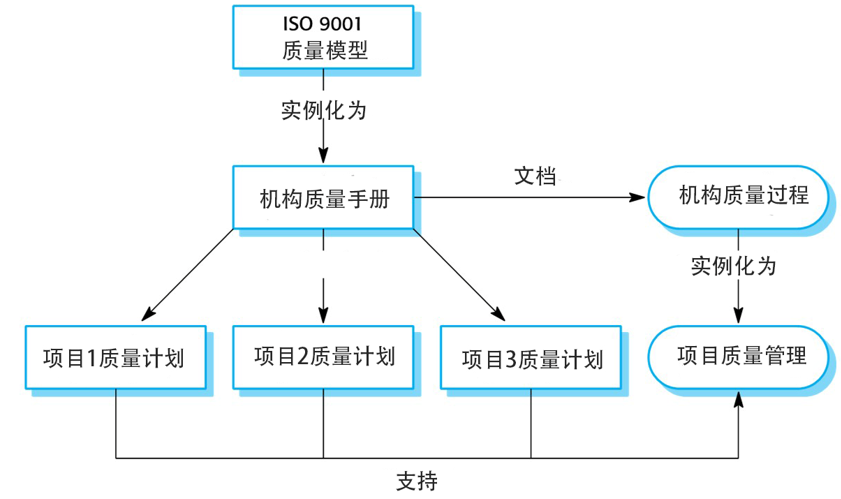
**产品度量过程**



**ISO 9001 核心过程**



**ISO 9001 和质量管理**



**要点**

1.软件质量管理就是确保软件有较少的缺陷数，并达到可维护性、 可靠性、 可移植性等既定标准。

2.质量管理活动包括为过程和产品制定标准， 并为检测是否符合这些标准而建立过程。

3.软件标准对质量保证来说非常重要 ，因为这些标准是对 “成功实践” 的认同。

4.机构的质量手册应该编制一套质量保证规程，可以根据ISO 9001标准中的通用模型进行编制。

5.对软件过程产生的可交付物进行复查需要有检查质量标准执行情况的团队人员参加。

6.在程序审查或同行评审中，一个小团队系统地检查代码。他们仔细地读这些代码并寻找可能出错和易被遗漏的地方。

7.软件度量可以用于收集有关软件和软件过程的量化数据。

8.产品质量量度对于暴露存在质量问题的异常组件具有特别重要的意义。

**Chapter 25 – Configuration Management**

**CM 活动**

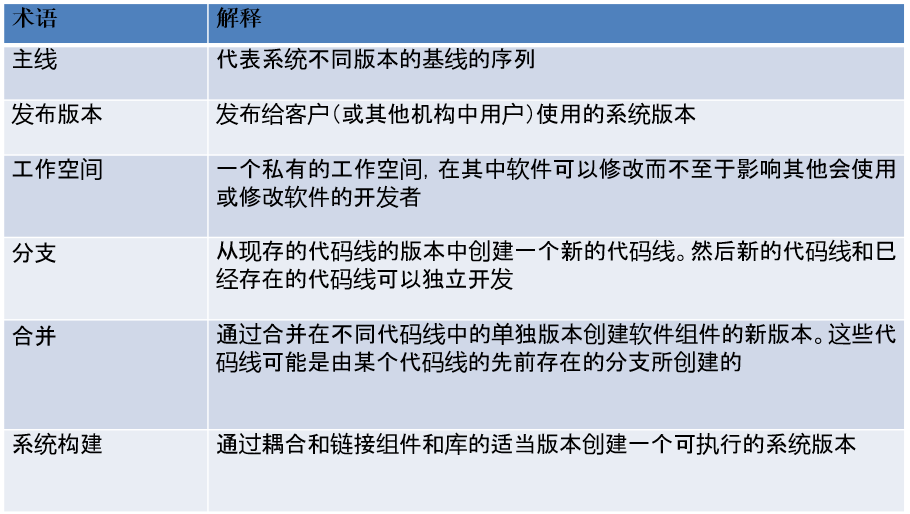
1.变更管理:包括跟踪来自客户和开发者的软件变更请求，计算做出这些变更的花费并估计其影响，决定是否变更。

2.版本管理:包括跟踪系统组件的多个版本，确保由不同开发者对组件做出的变更不会彼此干涉。

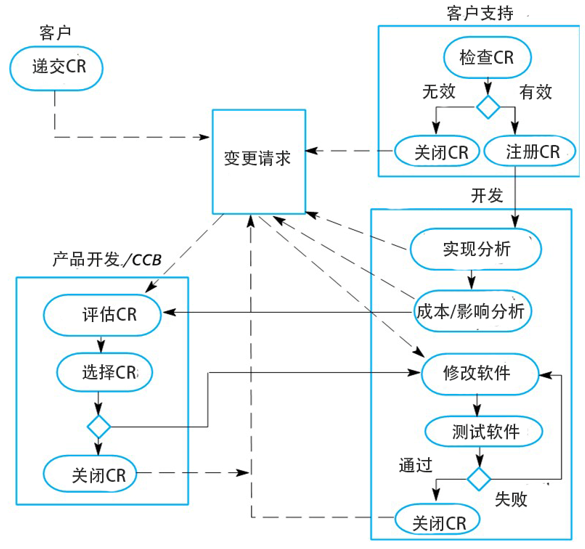
3.系统构建:是一个组装程序组件、数据和库的过程，然后把这些组件编译链接成一个可执行系统。

4.发布管理:包括准备对外发布的软件，持续跟踪已经发布以供客户使用的系统版本。

**配置管理术语**



**变更管理过程**



**变更请求时需要考虑的因素**

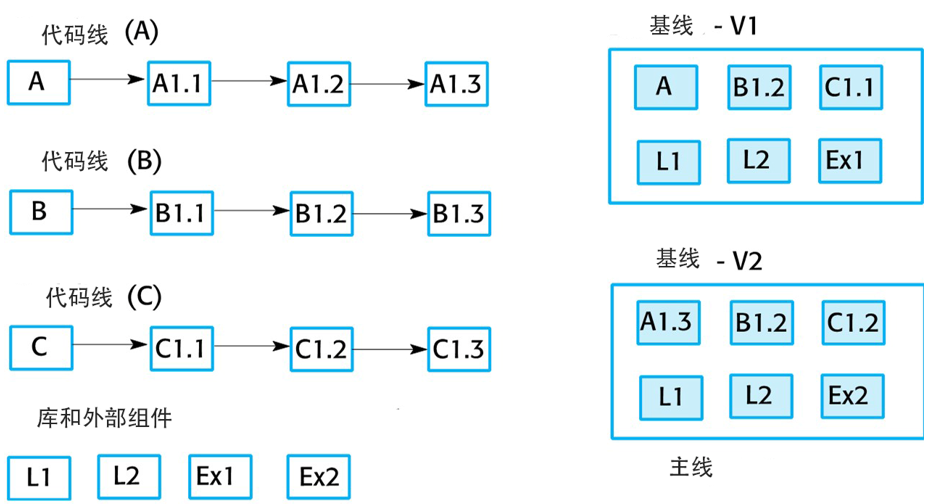
1.不做变更的后果2.变更的好处3.受变更影响的用户数4.进行变更的成本5.该产品发布周期

**代码线和基线**

1.代码线就是源代码版本的序列，一个晚期的版本是由某个早期版本发展而来。

2.基线是对一个特殊系统的定义:可能会使用一种配置语言来详述基线， 以便用户定义一个特殊系统版本所包括的组件。

3.基线很重要，因为开发者常常不得不重建一个完整系统的特定版本。



**文件识别**

1.源代码校验和:源代码文件的签名是从文件中数据计算出的校验和。校验和函数使用源代码文件作为输入，计算出一个独一无二的值。如果改变源代码（即使是一个字符），就将生成不同的校验和值。这样就肯定能够判断有着不同的校验和的文件是不相同的。

2.时间戳:因为源文件和目标文件通过名称联系起来， 而不是明确的源文件签名， 由于要生成相同名字的目标文件，一般不可能在同一个目录同一时间建立源代码组件的不同版本。

3.校验和:在时间戳的情况下，通常的做法是当你重新编译一个组件，它不会覆盖目标代码，相反，它会生成一个新的目标代码文件和源代码的签名标签。并行编译是可能的，并且不同版本的组件可以在同一时间被编译。

**持续集成**



**发布版本管理**

1.系统的发布版本是分发给客户的系统版本。

2.对于大众市场软件，通常可能定义两种类型的发布 ， 一种是主要发布，用于交付重要的新功能；另一种是小型发布， 用于修复漏洞和修复用户报告的问题。

3.对于定制软件或者软件产品线，系统的特别发布版本可能 必须为每个客户或者个人客户生产，可能同时运行几种不同的系统版本。

**发布版本追踪**

1.如果发生问题，可能需要重新生产已经交付给特定客户的软件。

2.一个系统的发布版本生成时，必须编制文档以保证将来可以重新准确地复制它。

3.这一点对定制的 、生命周期长的嵌入式系统尤为重要，比如那些控制复杂机器的。

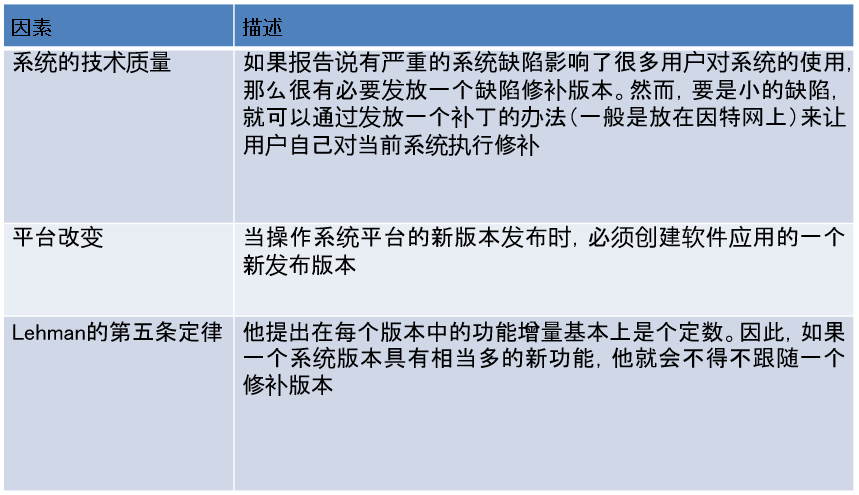
**发布版本重新生产**

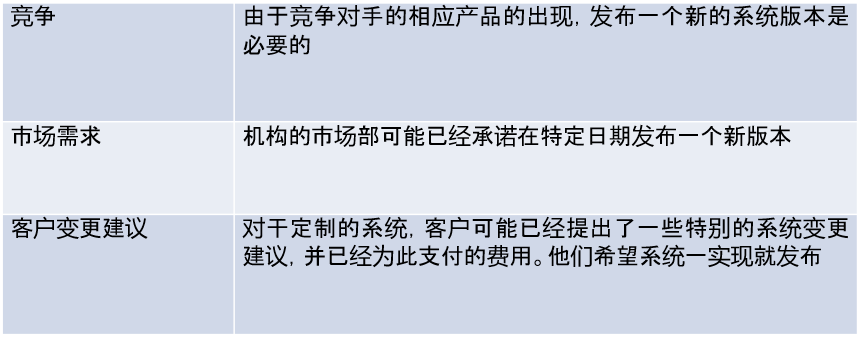
1.为发布版本编制文档，必须记录用来产生可执行代码的源代码组件的特定版本。

2.也要保存源代码文件、 相应可执行代码以及所有的数据和配置文件。

3.还应该记录操作系统的版本、 库 、 编译程序和建立软件的其他工具。

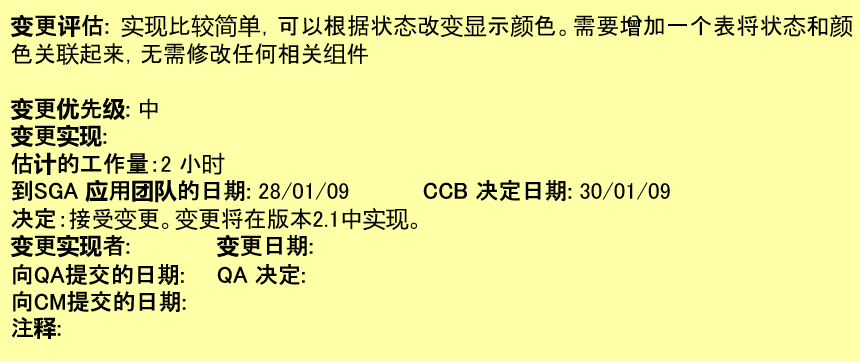
**影响系统发布规划的因素**





**部分完成的变更需求表**





**要点**

1.配置管理是对开发软件系统所做的管理。在维护一个系统时，配置管理团队的作用是保证系统的变更是在受控状态下完成的，并且详细记录已经实现的改变。

2.主要配置管理过程包括变更管理、 版本管理、 系统构建以及发布管理。 存在能够支持所有这些过程的软件工具。

3.变更管理包括评估来自系统客户和其他信息持有者的变更提议，并判断在系统新版本中实现这些变更的费用是否合适。

4.版本管理包括追踪软件组件由于变更而产生的不同版本。

5.系统构建是将系统组件装配成一个在目标计算机系统上的可执行程序的过程。

6.软件应该频繁地重新构建， 在新版本构建之后立即进行测试。 这样可以更容易察觉最后一次构建引入的错误和问题。

7.系统的发布包括可执行代码、 数据文件 、配置文件和文档。 发布管理包括决定何时发布一个系统，准备好所有待发布的信息，并为每一个系统发布版本编制好文档。