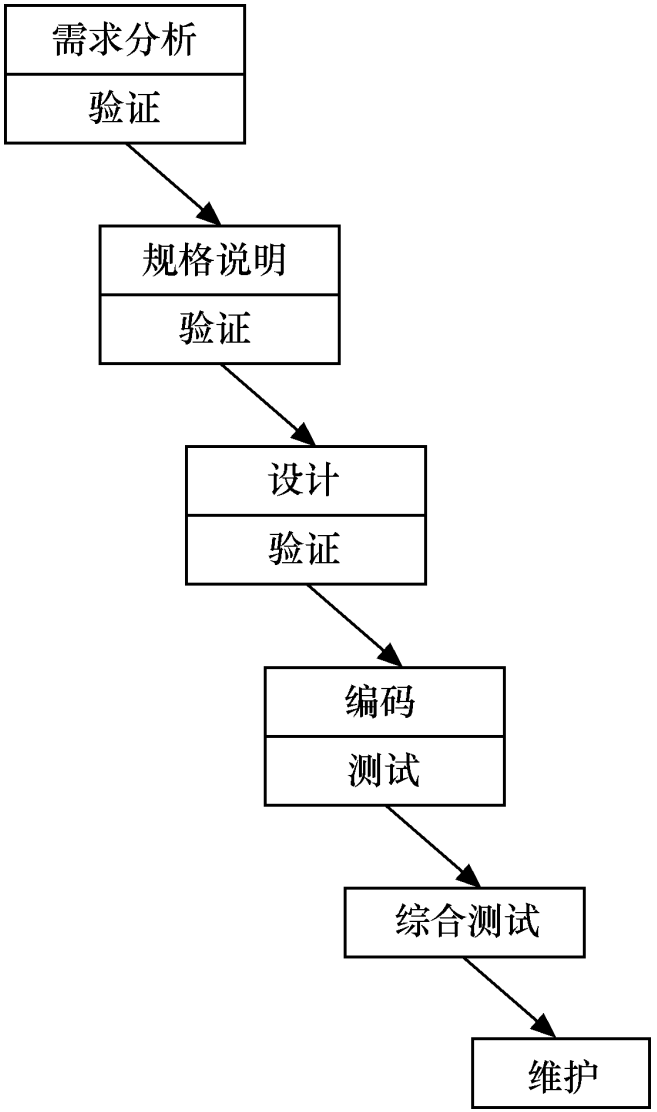
第2章 软件生存期模型

- 瀑布模型
- 快速原型模型
- 增量模型
- 螺旋模型
- 喷泉模型
- 统一过程
- 基于构件的开发模型
- 敏捷过程

2.1 瀑布模型

- **在20世纪80年代之前，瀑布模型一直是唯一被广泛采用的生命周期模型。**
- **传统的瀑布模型如图所示。**



2.1 瀑布模型

- 瀑布模型的特点

➢ **阶段间具有顺序性和依赖性。其中包含两重含义：**

① 必须等前一阶段的工作完成之后，才能开始后一阶段的工作；

② 前一阶段的输出文档就是后一阶段的输入文档。

2.1 瀑布模型

- 瀑布模型的特点

➢ **推迟实现的观点**

① 瀑布模型在编码之前设置了系统分析和系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。

② 清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想。

2.1 瀑布模型

- 瀑布模型的特点

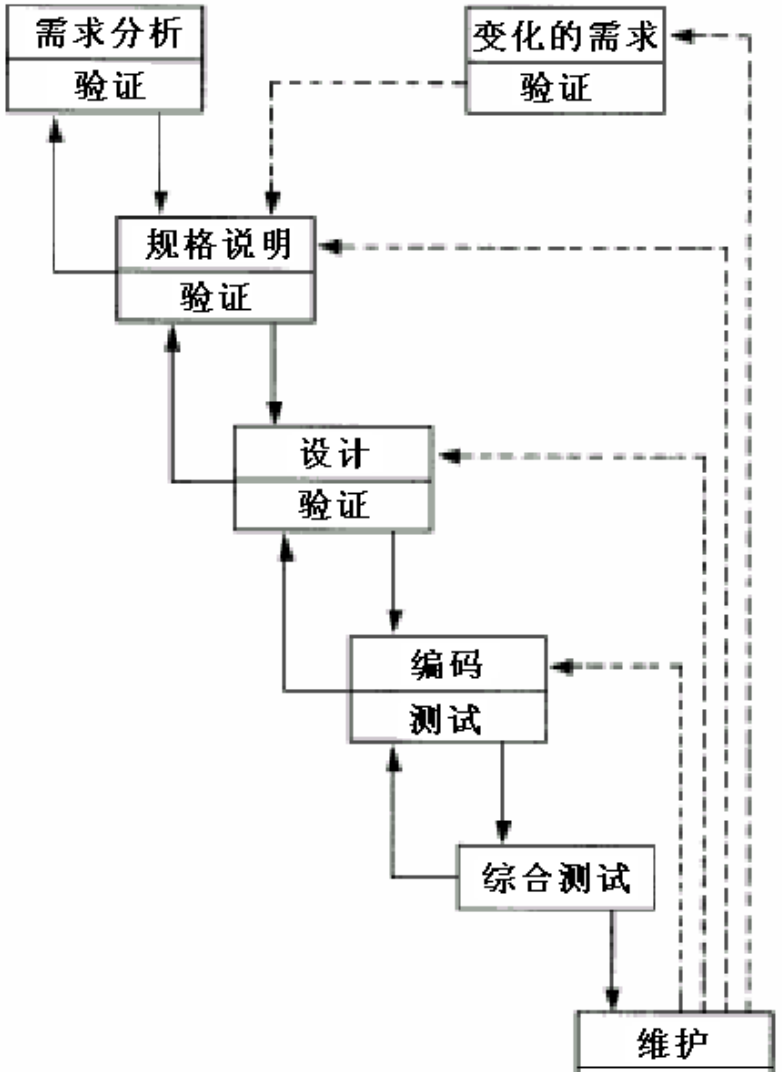
➢ **质量保证的观点**

① 每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。

② 每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。

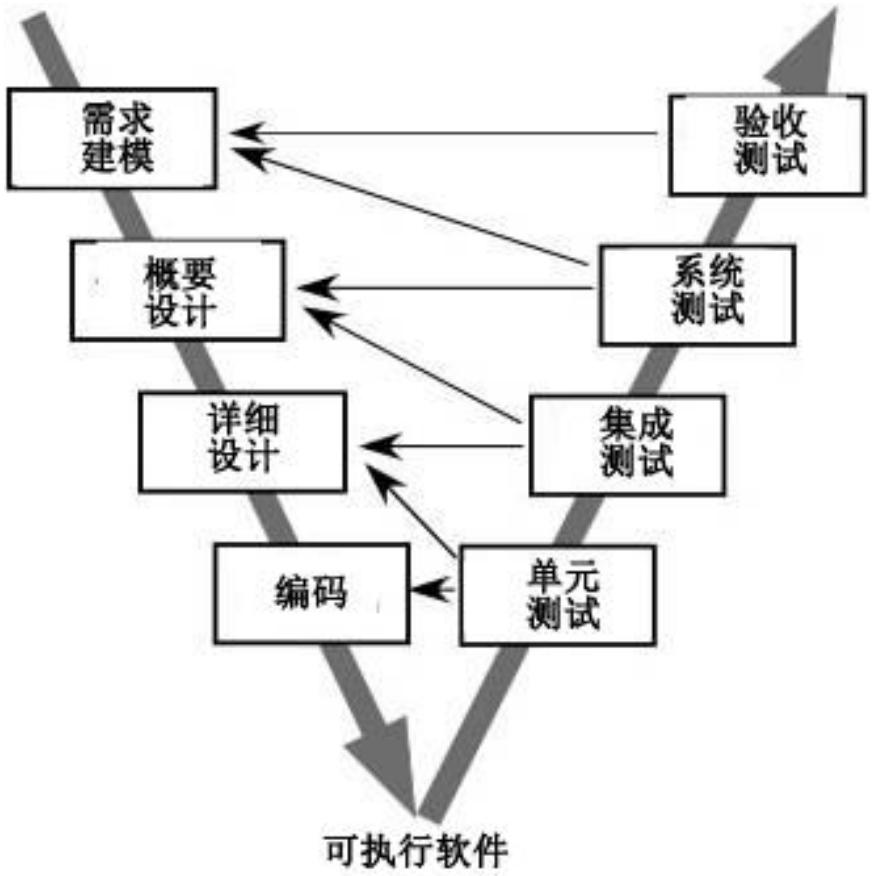
2.1 瀑布模型

- 实际的瀑布模型
  - **实际的瀑布模型是带“反馈环”的，如图所示。**
  - **图中实线箭头表示开发过程，虚线箭头表示维护过程。**



2.1 瀑布模型

- **V模型：瀑布模型的一个变体**
  - **V模型描述了测试阶段的活动与开发阶段相关活动（包括需求建模、概要设计、详细设计、编码）之间的关系。**



2.1 瀑布模型

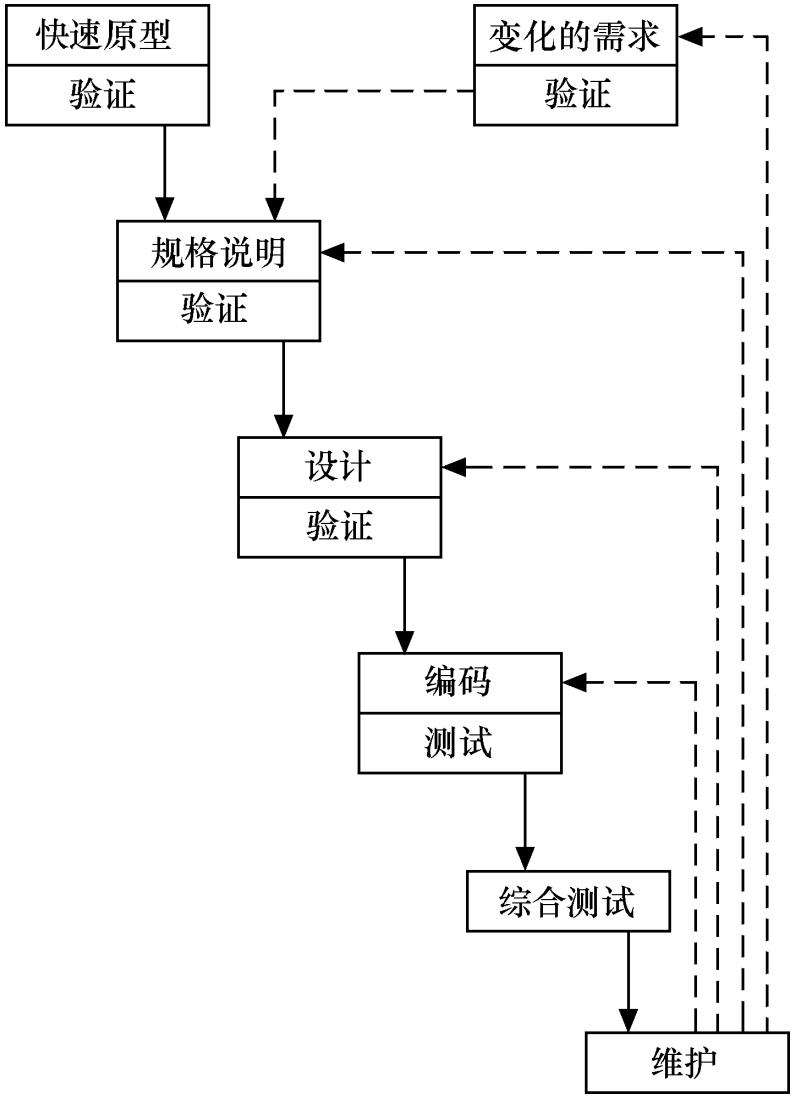
- 瀑布模型的优点
  - **可强迫开发人员采用规范化的方法。**
  - **严格地规定了每个阶段必须提交的文档。**
  - **要求每个阶段交出的所有产品都必须是经过验证的。**

2.1 瀑布模型

- 瀑布模型的缺点
  - **由于瀑布模型几乎完全依赖于书面的规格说明，很可能导致最终开发出的软件产品不能真正满足用户的需要。如果需求规格说明与用户需求之间有差异，就会发生这种情况。**
  - **瀑布模型只适用于项目开始时需求已确定的情况。**

2.2 快速原型模型

- **快速原型是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。**
- **快速原型模型如图所示。**



2.2 快速原型模型

- 快速原型模型的优点

(1)有助于满足用户的真实需求。

(2)原型系统已经通过与用户的交互而得到验证，据此产生的规格说明文档能够正确地描述用户需求。

(3)软件产品的开发基本上是按线性顺序进行。

(4)因为规格说明文档正确地描述了用户需求，因此，在开发过程的后续阶段不会因为发现规格说明文档的错误而进行较大的返工。

2.2 快速原型模型

- 快速原型模型的优点

(5)开发人员通过建立原型系统已经学到了许多东西，因此，在设计和编码阶段发生错误的可能性也比较小，这自然减少了在后续阶段需要改正前面阶段所犯错误的可能性。

(6) 快速原型的突出特点是“快速”。开发人员应该尽可能快地建造出原型系统，以加速软件开发过程，节约软件开发成本。

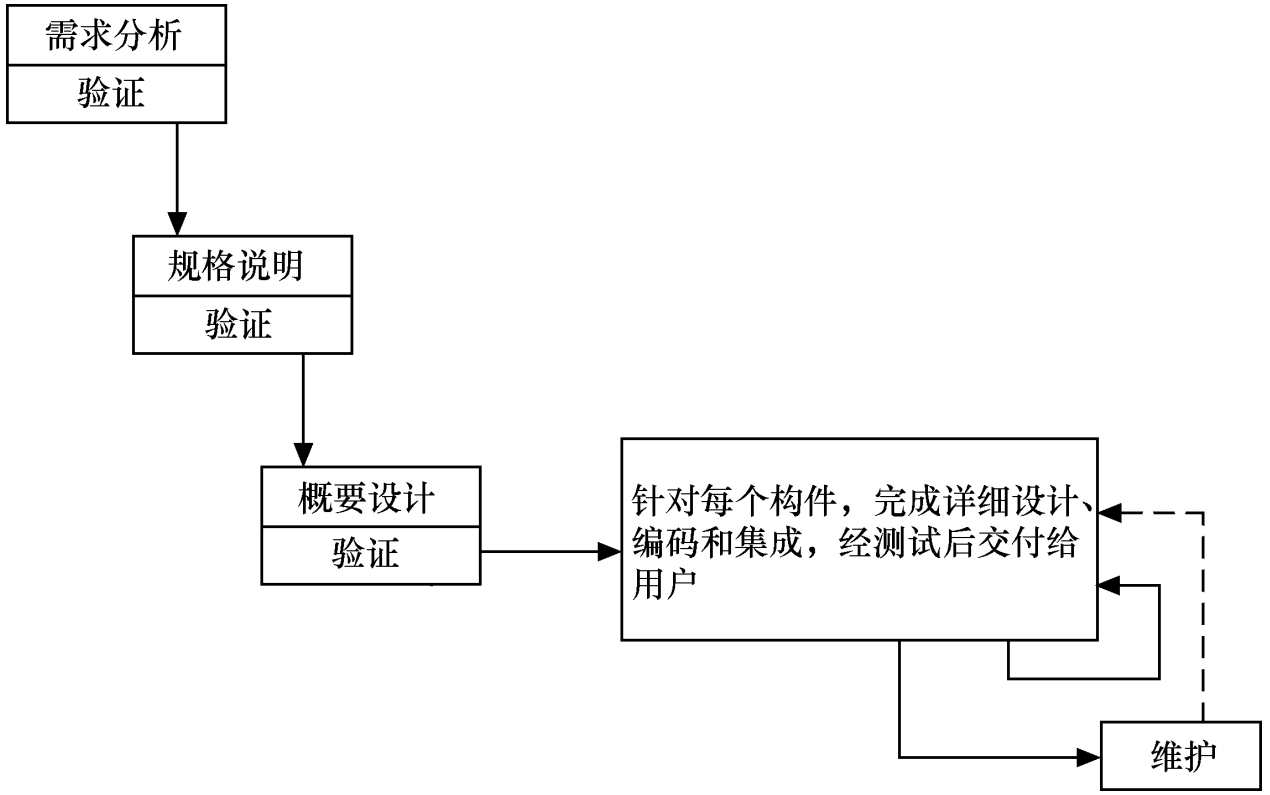
原型的用途是获知用户的真正需求，一旦需求确定了，原型可以抛弃，当然也可以在原型的基础上进行开发。

2.3 增量模型

- 增量模型也称为渐增模型，是Mills等于1980年提出来的。
- 使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码、集成和测试。
- 每个构件由多个相互作用的模块构成，并且能够完成特定的功能。

2.3 增量模型

- **增量模型如图所示。**



2.3 增量模型

- 增量模型的优点

(1)能在较短时间内向用户提交可完成一些有用的工作产品，即从第1个构件交付之日起，用户就能做一些有用的工作。

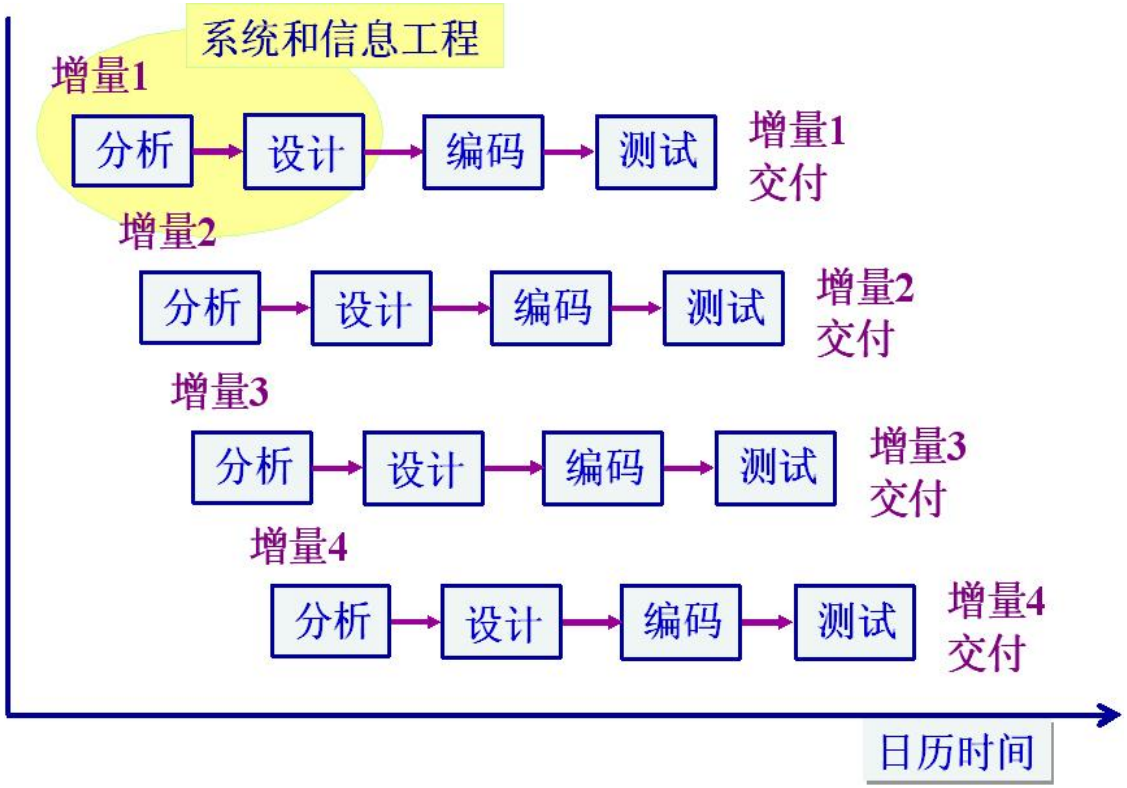
(2)逐步增加产品的功能可以使用户有较充裕的时间学习和适应新产品，从而减少一个全新的软件可能给用户组织带来的冲击。

(3)项目失败的风险较低，虽然在某些增量构件中可能遇到一些问题，但其他增量构件将能够成功地交付给客户。

(4)优先级最高的服务首先交付，然后再将其他增量构件逐次集成进来。因此，最重要的系统服务将接受最多的测试。

2.3 增量模型

- 增量构件开发
  - **每个增量构件应当实现某种系统功能，因此增量构件的开发可以采用瀑布模型的方式，如图所示。**



2.3 增量模型

- 采用增量模型需注意的问题

(1)在把每个新的增量构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。

(2)软件体系结构必须是开放的，即向现有产品中加入新构件的过程必须简单、方便。

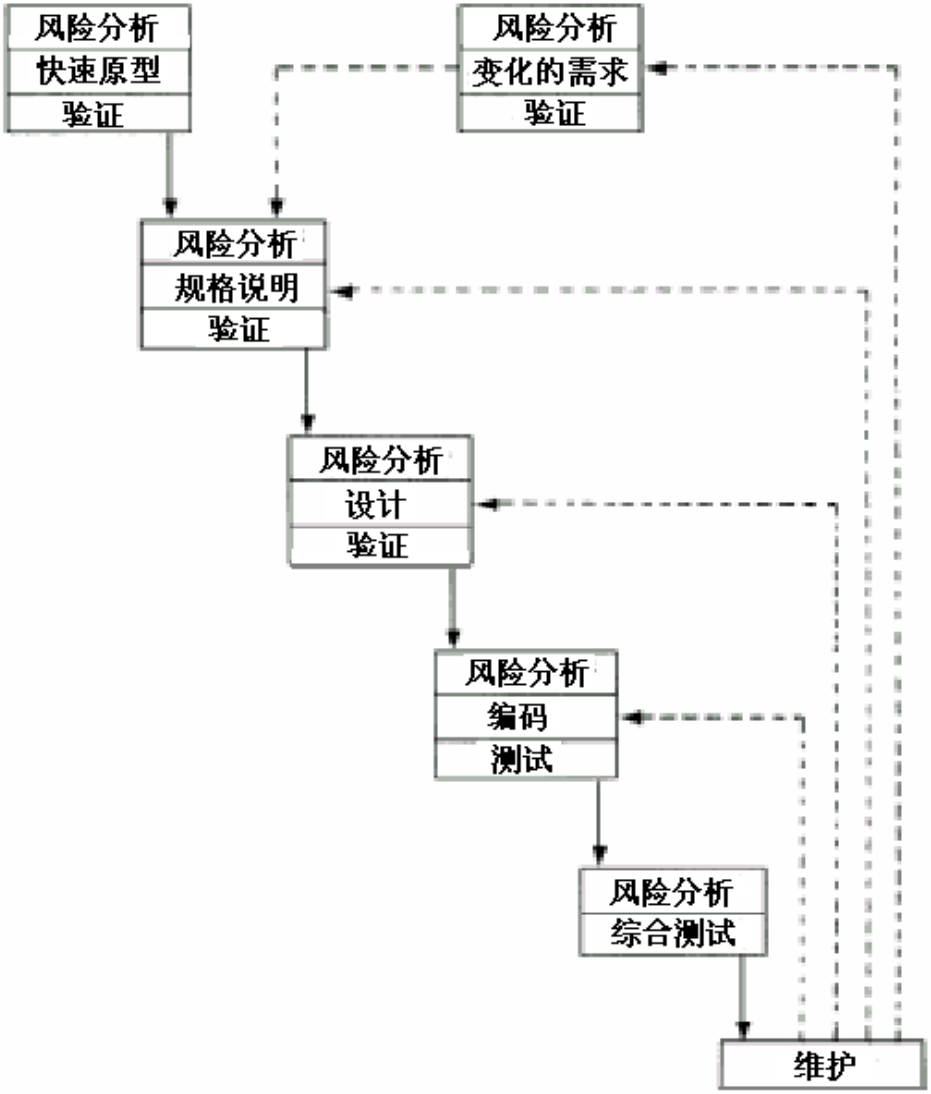
因此，采用增量模型比采用瀑布模型和快速原型模型更需要精心的设计。

2.4 螺旋模型

- **螺旋模型最初是Boehm于1988年提出来的。**
- **该模型将瀑布模型与快速原型模型结合起来，并且加入两种模型均忽略了的风险分析。**
- **螺旋模型的基本思想是，使用原型及其他方法来尽量降低风险。**

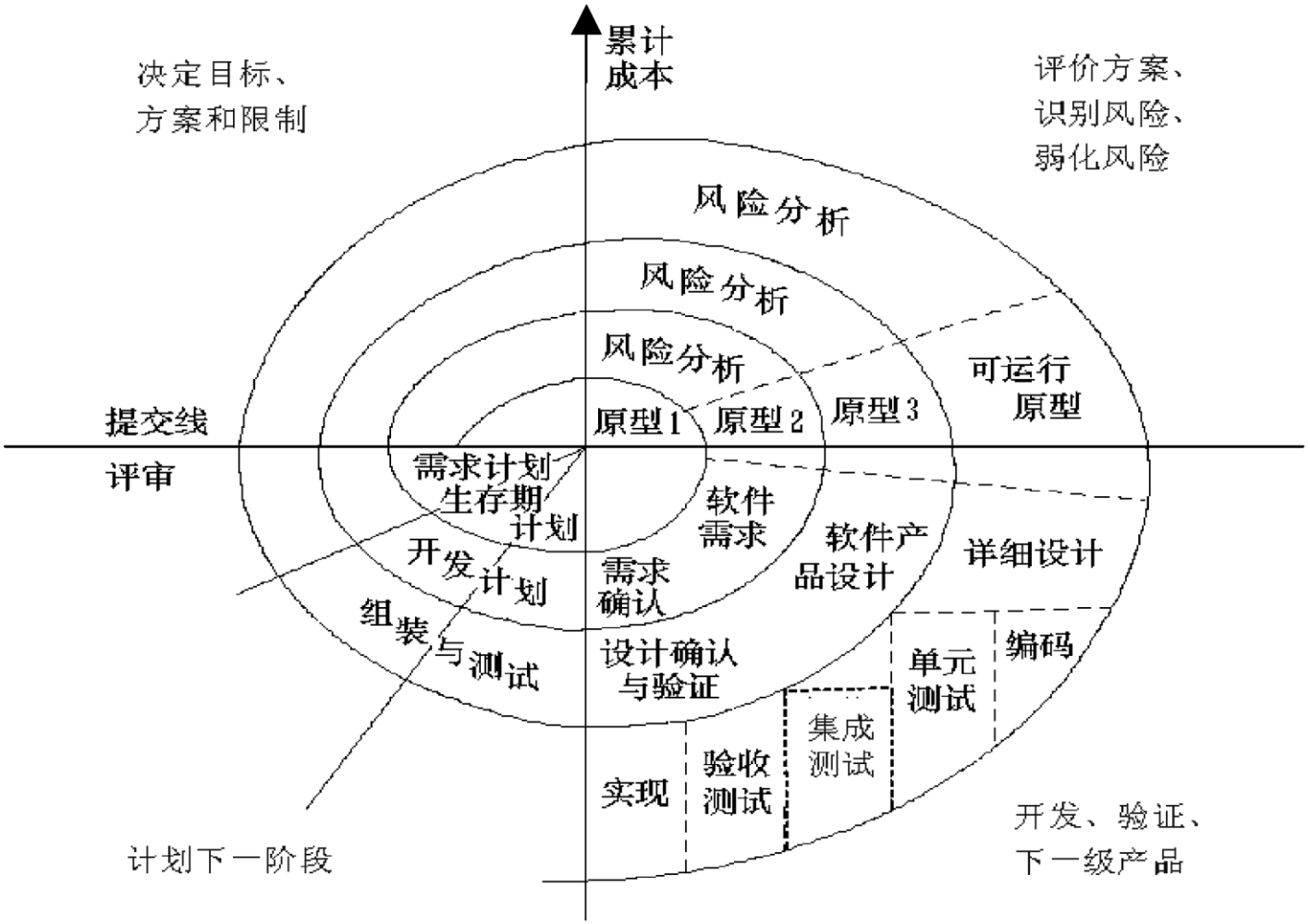
2.4 螺旋模型

- **理解这种模型的一个简便方法，是把它看做在每个阶段之前都增加了风险分析过程的快速原型模型。**



2.4 螺旋模型

- 完整的螺旋模型



2.4 螺旋模型

- 完整的螺旋模型

- 在螺旋模型中，软件过程表示成一个螺线，而不是像以往的模型那样表示为一个具有回溯的活动序列。
- 在螺线上的每一个循环表示过程的一个阶段。
- 每个阶段开始时的任务是确定该阶段的目标、为完成这些目标选择方案及设定这些方案的约束条件。接下来的任务是，从风险角度分析上一步的工作结果，努力排除各种潜在的风险，通常用建造原型的方法来排除风险。如果成功地排除了所有风险，则启动下一步开发步骤，在这个步骤的工作过程相当于纯粹的瀑布模型。最后是评价该阶段的工作成果并计划下一个阶段的工作。

2.4 螺旋模型

- 螺旋模型的4项活动

➢ **螺线上的每一个循环可划分为4个象限，分别表达了4个方面的活动。**

(1)目标设定——定义在该阶段的目标，弄清对过程和产品的限制条件，制订详细的管理计划，识别项目风险，可能还要计划与这些风险有关的对策。

(2)风险估计与弱化——针对每一个风险进行详细分析，设想弱化风险的步骤。

(3)开发与验证——评价风险之后选择系统开发模型。

(4)计划——评价开发工作，确定是否继续进行螺线的下一个循环。如果确定要继续，则计划项目的下一个阶段的工作。

2.4 螺旋模型

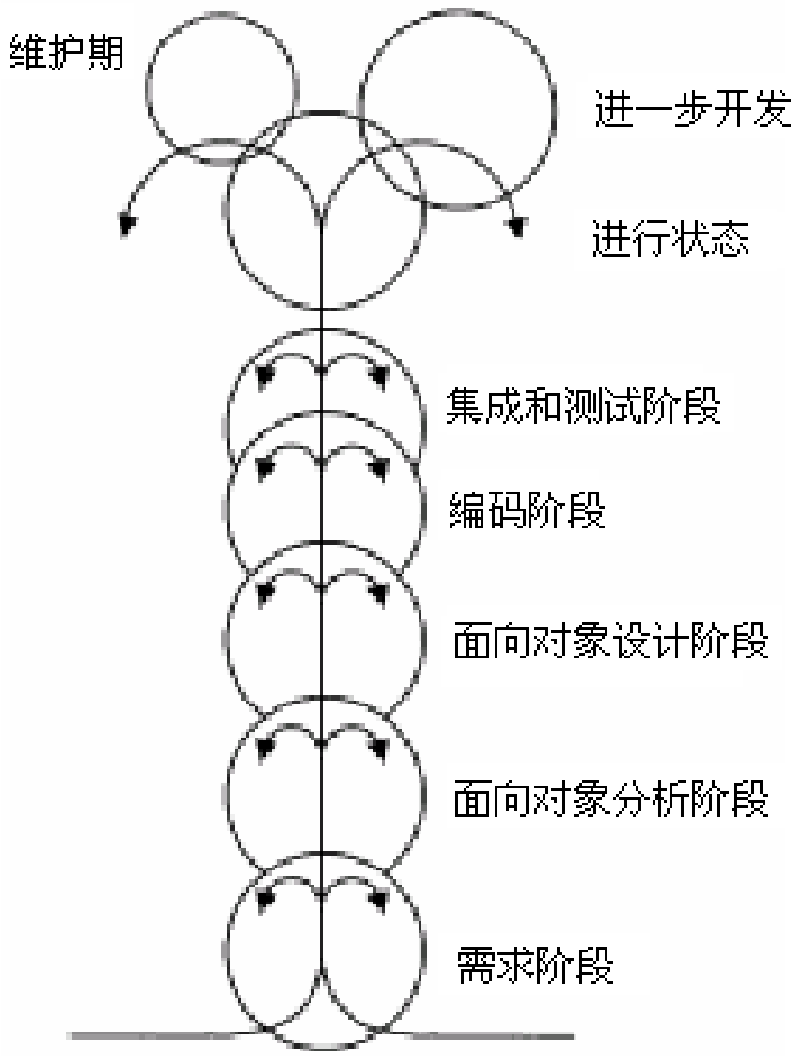
- 螺旋模型的优点
  - **对可选方案和约束条件的强调有利于已有软件的重用，也有助于把软件质量作为软件开发的一个重要目标。**
  - **减少了过多测试或测试不足所带来的风险。**
  - **在螺旋模型中维护只是模型的另一个周期，因而在维护和开发之间并没有本质区别。**

2.4 螺旋模型

- 螺旋模型的缺点
  - **螺旋模型是风险驱动的，因此要求软件开发人员必须具有丰富的风险评估经验和这方面的专门知识，否则将出现真正的风险：当项目实际上正在走向灾难时，开发人员可能还以为一切正常。**

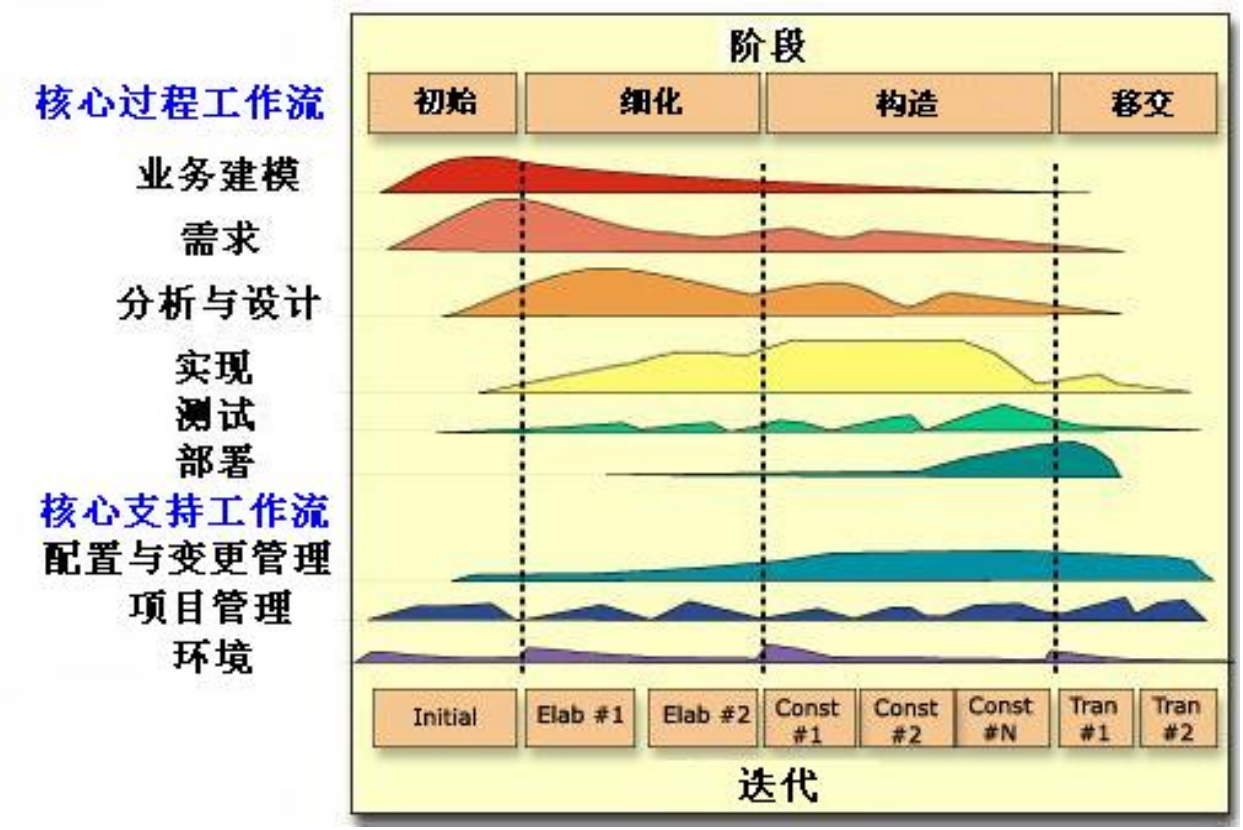
2.5 喷泉模型

- **喷泉模型是典型的面向对象生命周期模型。**
- **“喷泉”一词体现了迭代和无间隙特性。图中代表不同阶段的圆圈相互重叠，这明确表示两个活动之间存在重叠。**



2.6 统一过程

- **由Booch、Jacobson及Rumbaugh提出，统一过程模型如图所示。**



2.6 统一过程

- 统一过程的工作流

➢ **在统一过程中，有6个核心工作流。**

① 业务建模工作流。用商业用例为商业过程建立文档。

② 需求工作流。目标是描述系统应该做什么，确保开发人员构建正确的系统。为此，需明确系统的功能需求和非功能需求（约束）。

③ 分析和设计工作流。其目标是说明如何做。结果是分析模型和设计模型。

2.6 统一过程

④ **实现工作流**。用分层的方式组织代码的结构，用构件的形式来实现类，对构件进行单元测试，将构件集成到可执行的系统中。

⑤ **测试工作流**。验证对象之间的交互、是否所有的构件都集成了、是否正确实现了所有需求、查错并改正。

⑥ **部署工作流**。制作软件的外部版本、软件打包、分发、为用户提供帮助和支持。

2.6 统一过程

- 统一过程的阶段

➢ **统一过程有4个阶段，分别是初始阶段、细化阶段、构造阶段和移交阶段。**

① **初始阶段。初始阶段主要关注项目计划和风险评估，其目的是确定是否值得开发目标信息系统。**

② **细化阶段。细化阶段关心定义系统的总体框架，其目标是：细化初始需求（用况）、细化体系结构、监控风险并细化它们的优先级、细化业务案例以及制订项目管理计划。**

2.6 统一过程

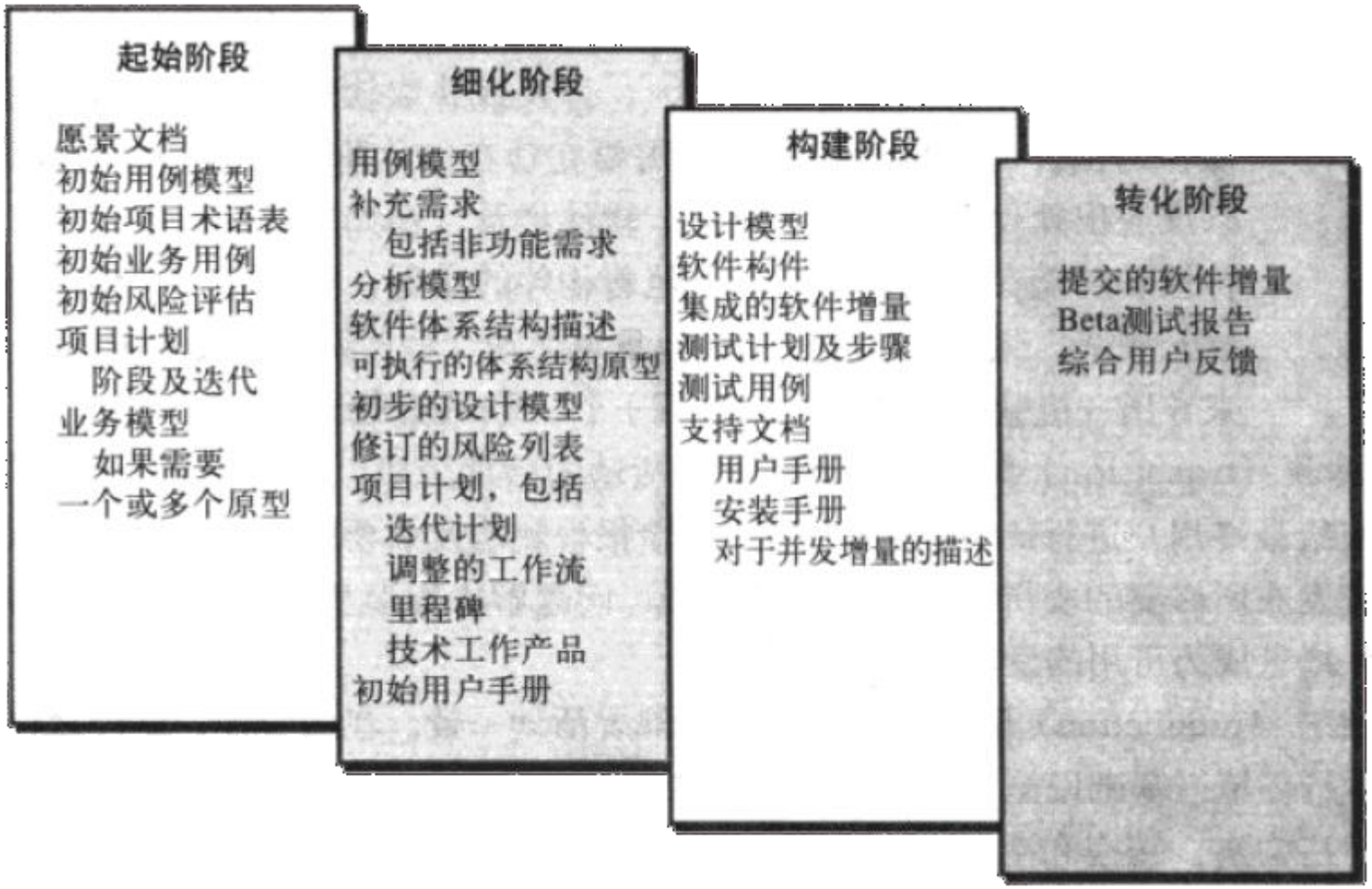
- 统一过程的阶段

③ 构造阶段。构造阶段是建立系统，构造信息系统的第1个具有操作质量的版本，以能够交付给客户进行β测试的版本结束，有时称为测试版本。

④ 移交阶段。移交阶段包含β测试时期，以发布完整的系统而终止，其目标是确保信息系统真正满足客户的需求。

2.6 统一过程

- 主要工作产品



2.7 基于构件的开发模型

- **基于构件的软件工程（component-based software engineering，CBSE）是强调使用可复用的软件“构件”来设计和构造基于计算机的系统的过程。**

2.7 基于构件的开发模型

- **Clements对CBSE给出了如下描述。**

CBSE正在改变大型软件系统的开发方式。CBSE体现了Frod Brooks和其他人支持的“购买，而非构造”的思想。就如同早期的子程序将程序员从考虑编程细节中解脱出来一样，CBSE将考虑的重点从编码转移到组装软件系统。

考虑的焦点是“集成”，而不再是“实现”。

这样做的基础是假定在很多大型软件系统中存在足够多的共性，使得开发可复用的构件来满足这些共性是可行的。

2.7 基于构件的开发模型

- 当软件团队使用传统的需求获取技术确定了待开发软件的系统需求时，该过程开始。
- 体系结构设计完成后，并不立即进行详细设计任务，而是针对每一系统需求考虑以下问题:

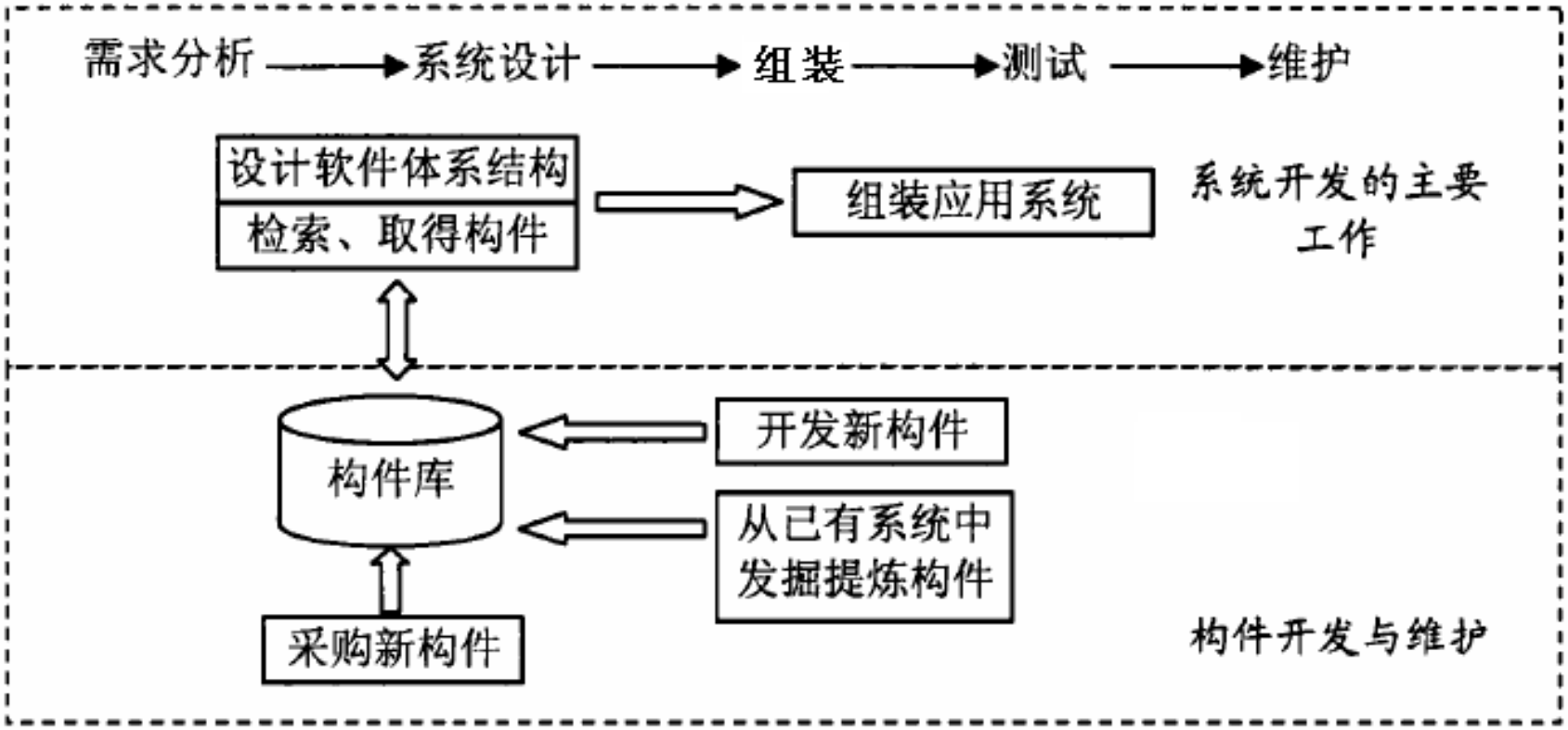
(1)现有的商品化构件（commercial off-the-shelf，COTS）是否能够实现该需求？

(2)内部开发的可复用构件是否能够实现该需求？

(3)可用构件的接口与待构造系统的体系结构是否相容？

2.7 基于构件的开发模型

- 基于构件的开发模型如下图。



2.7 基于构件的开发模型

- 开发步骤

➢ **不考虑构件的开发技术，基于构件的开发模型由以下步骤组成：**

(1)对于该问题领域的基于构件的可用产品进行研究和评估。

(2)考虑构件集成的问题。

(3)设计软件架构以容纳这些构件。

(4)将构件集成到架构中。

(5)进行充分的测试以保证功能正常。

2.7 基于构件的开发模型

- 典型的构件模型

(1)OMG/CORBA。对象管理组织发布了公共对象请求代理体系结构（OMG/CORBA），一个对象请求代理提供了多种服务，使得可复用构件（对象）可以与其他构件通信。

(2)Microsoft COM/DCOM/.NET。微软公司开发了构件对象模型（COM），此模型提供了构件的规格说明,在Windows操作系统，一个应用系统中可以使用不同厂商生产的构件。

(3)Sun JavaBean构件。JavaBean构件系统是一个可移植的、平台独立的、使用Java程序设计语言开发的CBSE基础设施。

2.8 敏捷过程

- 敏捷过程模型
  - **2001年，Kent Beck等17名编程大师发表“敏捷软件开发”宣言：**

我们正在通过亲身实践以及帮助他人实践的方式来揭示更好的软件开发之路，通过这项工作，我们认为：

个体和交互胜过过程和工具；

可工作软件胜过宽泛的文档；

客户合作胜过合同谈判；

响应变化胜过遵循计划。

2.8 敏捷过程

- 敏捷过程

➢ 任何一个敏捷过程都可以由所强调的3个关键假设识别出来，这3个假设可适用于大多数软件项目。

(1) 提前预测哪些需求是稳定的、哪些需求会变化非常困难。同样的，预测项目进行中客户优先级的变化也很困难。

(2) 对很多软件，设计和构建是交错进行的。事实上，两种活动应当顺序开展以保证通过构建实施来验证设计模型，而在通过构建验证之前很难估计应该设计到什么程度。

(3) 从制订计划的角度来看，分析、设计、构建和测试并不像我们所设想的那么容易预测。

2.8 敏捷过程

- 敏捷原则

(1)我们最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。

(2)即使在开发的后期，也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。

(3)经常交付可运行软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。

(4)在整个项目开发期间，业务人员和开发人员必须天天都在一起工作。

(5)围绕有积极性的个人构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。

2.8 敏捷过程

- 敏捷原则

(6)在团队内部，最富有效果和效率的信息传递方法是面对面交谈。

(7)可运行软件是进度的首要度量标准。

(8)敏捷过程提倡可持续的开发速度。责任人、开发者和用户应该能够保持一种长期、稳定的开发速度。

(9)不断地关注优秀的技能和好的设计会增强敏捷能力。

(10)简单是必要的。

(11)好的架构、需求和设计出自于自组织团队。

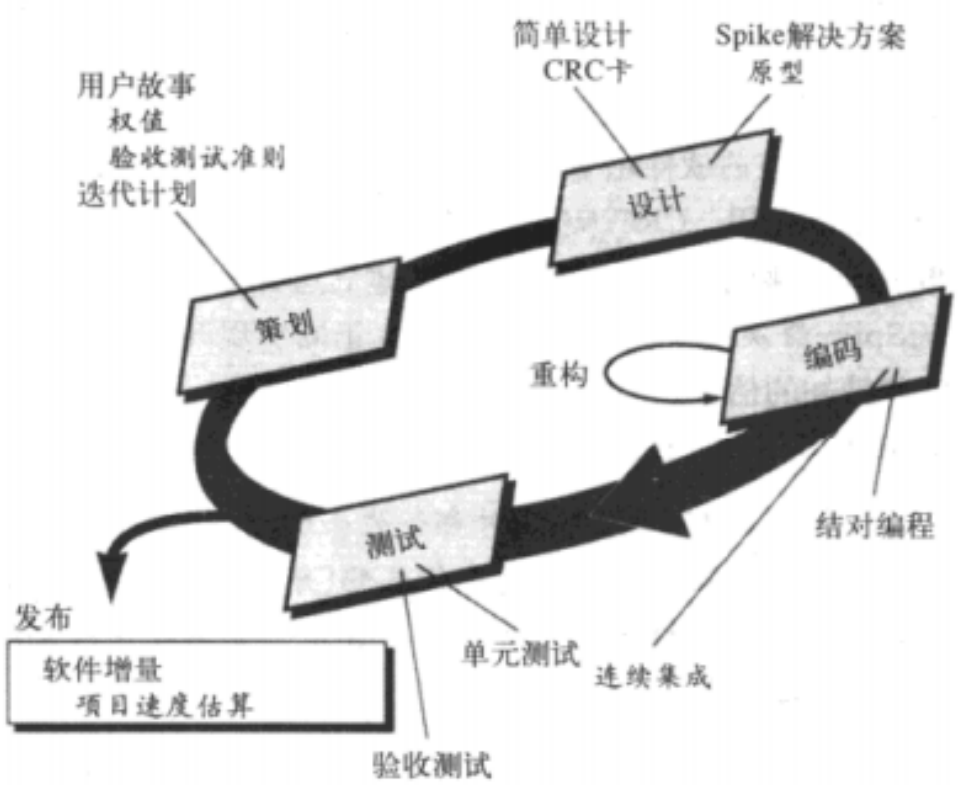
(12)每隔一定时间，团队会反省如何才能更有效地工作，并相应调整自己的行为。

2.8 敏捷过程

- 人的因素
- 敏捷开发团队成员及团队本身必须具备的一些特点：
  - **基本能力**
  - **共同目标**
  - **精诚合作**
  - **决策能力**
  - **模糊问题解决能力**
  - **相互信任和尊重**
  - **自我组织**

2.8 敏捷过程

- 极限编程（eXtreme Programming，XP）
  - **使用最广泛的敏捷过程，最初由Kent Beck提出。XP包含了策划、设计、编码和测试4个框架活动的规则和实践。**



2.8 敏捷过程

- 极限编程的框架活动

① **策划**

- 开始于建立“用户故事”
- 敏捷团队评估每一个故事并给出成本（cost）
- 故事被分组用于可交付增量
- 对发布日期做出承诺
- 在第一个发行版本（软件增量）之后，“项目速度” 用于帮助建立后续发行版本（软件增量）的发布日期

2.8 敏捷过程

- 极限编程的框架活动

② 设计。XP设计严格遵循KIS（keep it simple，保持简洁）原则，通常更愿意使用简单设计而不是更为复杂的表述。

- 严格遵守**KIS**原则
- 鼓励使用**CRC**（类-责任-协作者）卡片
- 在设计中遇到困难, **XP**推荐立即建立这部分设计的可执行原型，实现并评估设计原型
- 鼓励“重构”—一种迭代的改进内部程序设计的方法

2.8 敏捷过程

- 极限编程的框架活动

③ 编码。

- 建议在开始编码之前为每一个故事开发一系列单元测试
- 鼓励 “结对编程”

④ 测试。

- 所有的 单元测试每天都要执行。
- “验收测试” 由客户定义，将着眼于客户可见的、可评审的系统级的特征和功能。

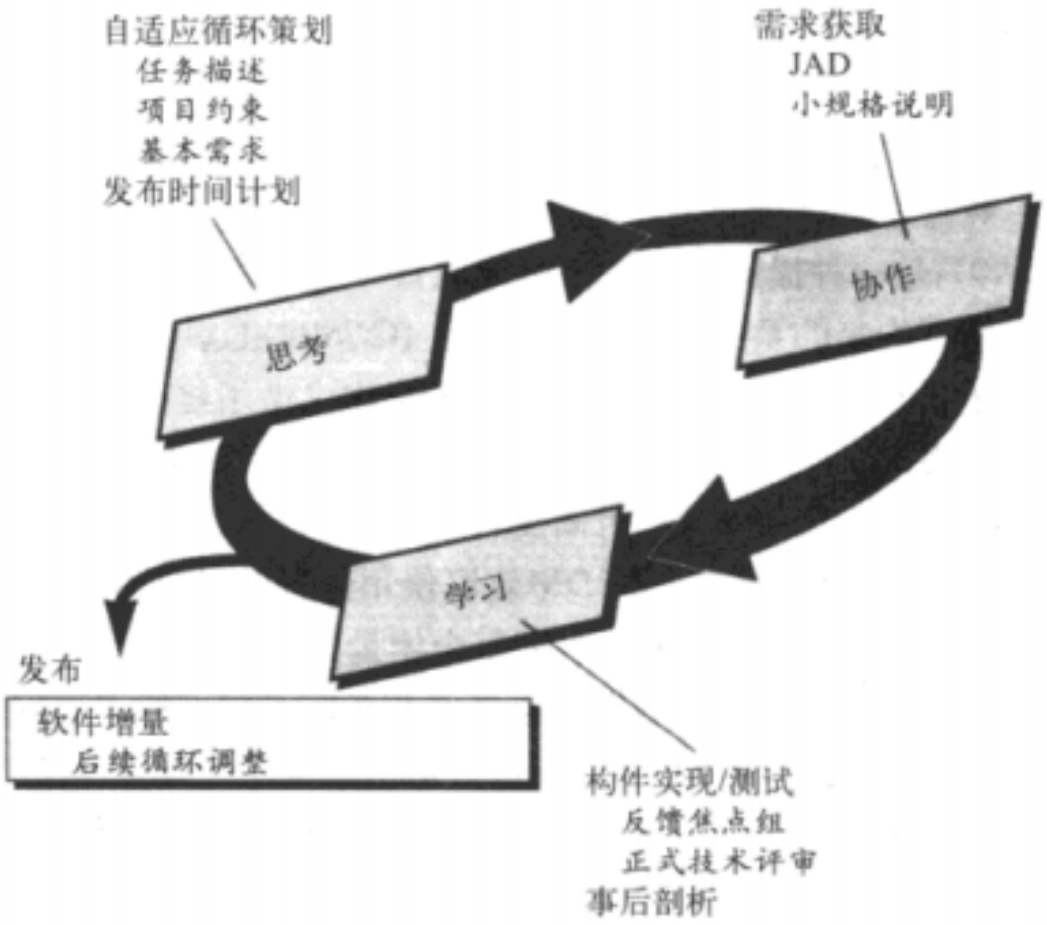
2.8 敏捷过程

- 自适应软件开发

- **自适应软件开发（adaptive software development，ASD）是由Jim Highsmith提出的；**
- **它可作为构建复杂软件和系统的一项技术，其基本概念着眼于人员合作和团队自组织。**

2.8 敏捷过程

- 自适应软件开发模型



2.8 敏捷过程

- 自适应软件开发的阶段

(1)思考：思考过程中，开始项目规划并完成自适应循环策划。自适应循环策划通过使用项目开始信息，来确定项目所需的一系列软件增量发布循环。

(2)协作：受激励的人员以超越其聪明才智和独创成果的方式共同工作，协作方法是所有敏捷方法中不断重现的主旋律。

(3)学习：当ASD团队成员开始开发作为自适应循环一部分的构件时，其重点是在完成循环的过程中学习尽可能多的东西。

That's All!