SA：软件体系结构。

第一部分

1. **软件危机：**

**随着软件规模越来越大，软件生产与维护越来越困难；**

**软件危机的出现直接导致了软件体系结构以及软件中间件的产生；**

软件体系结构以及软件中间件均是通过提取软件中的共性特征，进行复用来提高软件生产效率和质量

**2.软件危机的表现：**

1. **软件成本日益增长**
2. **开发进度难以控制**
3. **软件质量差**
4. **软件维护困难**

表现拓展：

（1）软件成本在整个计算机系统成本中所占的比例逐年增加。

（2）用户需求变化令软件开发过程很难保证按预定的计划实现，随着人员数量的增加，人员的组织、协调、通信、培训和管理等方面的问题将更为严重

（3）程序员几乎总是习惯性地以自己的想法去代替用户对软件的需求，软件设计带有随意性

（4）没有严格遵循软件开发标准，原来的开发人员可能因各种原因已经离开原来的开发组织

**软件发展已经受到软件生产、运维的高昂成本制约**

**3.软件危机产生根源：**

1. **用户需求不明确：不明确、遗漏、错误等；提出修改要求；双方理解有差距。**
2. **缺乏正确的理论指导：缺乏方法、工具。**
3. **软件规模越来越大：产生疏漏、错误。**
4. **软件复杂度越来越高**

**4.软件危机解决的途径：**

**要提高软件开发效率，提高软件产品质量，必须采用工程化的开发方法与工业化的生产技术。**

举例：

（1）自顶而下地解决问题

（2）基于重用（复用）的软件生产技术

（3）软件工程、软件过程管理

**5.记忆体系结构定义：**

**软件体系结构包括构件、连接件 和约束，它是可预制和可重构的软件框架结构。**

**构件是可预制和可重用的软件部件，是组成体系结构的 基本计算单元或数据存储单元**

**连接件也是可预制和可重用的软件部件，是构件之间的 连接单元**

**构件和连接件之间的关系用约束来描述**

**软件体系结构 = 构件 + 连接件 + 约束**

**6.理解体系结构的作用：**

（1）体系结构是风险承担者进行交流的手段

（2）体系结构是早期设计决策的体现

1. 体系结构本身就是一种可重用的模型

**7.理解软件中间件。**

中间件是一种“编程抽象”：

–支持共性技术特征

–隐藏信息系统本身的复杂性

中间件是一种“基础架构”：

–一组软件系统/构件：按照编程抽象的要求，实现共性基础特征所代表的能力

–提供可重用的软件框架、软件构件和编程模型

**8.理解软件体系结构与软件中间件的异同点。**

**共同目标：**

**（1）分析、抽取共性特征**

– 软件体系结构抽取不同软件在体系结构级的共性特征

– 软件中间件抽取不同软件均需要的基础服务功能中的共性特征

**（2）通过不同粒度的重用,来提高软件的生产效率,提高软件质量**

**（3）均关注重用。**

**重用什么?**

1）软件体系结构

• 关注重用软件的总体结构框架,软件设计的基本模型

• 是自顶向下重用

2）软件中间件

• 关注重用软件的通用基础业务功能

• 使用频繁的、与具体业务没有直接联系的基础功能被抽象出来，作为中间件

• 是自底向上的重用

**不同：**

**（1）与软件工程的关系：**

**• 软件体系结构**

– 来源于软件工程的研究,20世纪90年代后逐渐发展为独立的分支,但仍于软件工程关系密切

– 关注软件生命周期设计阶段之前的软件体系结构设计阶段

– 实际上可以看成具体业务设计之上的结构设计

**• 软件中间件**

– 与软件工程方向没有直接联系,但中间件的使用直接影响到软件的设计开发维护等阶段

– 是软件开发阶段可利用的基础设施

**（2）与构件的关系：**

**• 软件体系结构**

– 重点关注构件及构件间的连接关系,研究多个构件的组织结构

**• 软件中间件**

– 将提供基础服务功能的构件集合组织在一起，并为其提供运行环境

– 具体业务应用系统的开发可直接利用中间件及 其上运行的构件功能，不再需要专门定制处理

第二部分

1. **记忆软件体系结构风格的定义：**
   1. **定义**

描述特定领域中软件系统家族的组织方式的惯用模式，反映了领域中众多系统所共有的结构和语义特性 并指导如何将各个模块和子系统有效地组织成一个完整的系统。

* 1. **组成**

**一组组件类型。**例如：数据容器，过程，对象

**一组连接件类型/交互机制。**例如：过程调用，事件，管道

**这些组件的拓扑分布**

**一组对拓扑和行为的约束。**例如：数据容器不能。自己存储数据，管道不能是循环的

**一些对风格的成本和收益的非正式描述。**例如：如果你需要重用性并且性能不是很重要，那么可以使用管道风格

1. **理解软件体系结构风格级的复用：**

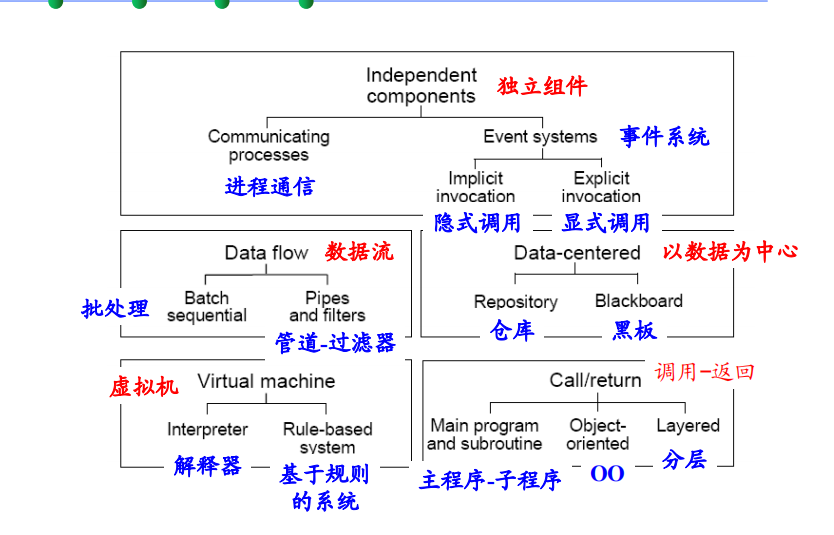
**软件体系结构风格级的复用**，可以使不同的系统可以共享同一个实现代码，一些经过实践证实的解决方案可以可靠地用于解决新的问题。

**只要系统使用规范的方法来组织**，就可以使别的设计者很容易地理解系统的体系结构。

**– 例如**，如果把系统描述为“B/S”风格，那么不需要给出设

计细节也会明白系统是如何组织和工作的。

1. **了解经典体系结构风格的分类。**



1. **理解数据流风格的特征。P19-P23**

数据的可用性决定着处理<计算单元>是否执行

系统结构：数据在各处理之间的有序移动

在纯数据流系统中，处理之间除了数据交换，没有任何其他的交互

基本组件：数据处理。

组件接口：输入端口和输出端口。

从输入端口读取数据，向输出端口写入数据。

计算模型：从输入端口读取数据，经过计算/处理，然后写到输出端口。

连接件：数据流。

单向、异步、有缓冲

拓扑结构：

数据流向无序。

主要关注近似线性的数据流或在限度内的循环数据流。

控制流和数据流：

§控制流(典型例子程序系统)

–主要问题是控制点怎样在程序或系统之间移动

–数据可能跟着控制走，但是并不起推动系统运转的作用

–关注的核心是计算顺序

§数据流

–主要问题是数据怎样在运算单元之间流动

–数据到了，控制（计算）单元便开始工作

–我们关心数据是否可用，转换，潜伏……

1. **理解批处理风格的定义、基本构成。P30-P31**

**定义：**

1.每个处理步骤是一个独立的程序

2.每一步必须在前一步结束后才能开始

3.数据必须是完整的，以整体的方式传递

**典型应用:**

–传统的数据处理

–程序编译/CASE(computer aided software engineering)工具

**基本构成：**

§ 基本组件：独立的应用程序

§ 连接件：某种类型的媒质(magnetic tape)

§ 表达拓扑结构 ：连接件定义了相应的数据流图

§ 每一步骤必须在前一步骤完全结束之后方能开始

1. **理解管道-过滤器风格的定义、基本构成、优缺点。**

**基本定义：**

场境：数据源源不断的产生，系统需要对这些数据进行若干处理(分析、计算、转换等)。

解决方案：

–把系统分解为几个序贯的处理步骤，这些步骤之间通过数据流连接，一个步骤的输出是另一个步骤的输入：

–每个处理步骤由一个过滤器组件(Filter)实现；

–处理步骤之间的数据传输由管道(Pipe)负责。

每个处理步骤(过滤器)都有一组输入和输出，过滤器从管道中读取输入的数据流，经过内部处理，然后产生输出数据流并写入管道中。

**基本组成：**

组件：过滤器（处理数据流）

连接件：管道（连接一个源和一个目的过滤器）

拓扑结构：连接器定义了数据流图。

过滤器 目标：将源数据变换成目标数据。

管道 作用：在过滤器之间传送数据。

（单向流、可能具有缓冲区、管道先后顺序不影响结果）

不同的管道中流动的数据流，可能具有不同的数据据格式。

**优点：**

（1）由于每个组件的行为不受其他组件的影响，整个系统的行为易于理解

（2）系统中的组件具有良好的隐蔽性和高内聚、低耦合的特点；

1）支持软件复用

2）系统维护和增强系统性能简单

（3）允许对一些如吞吐量、死锁等属性的分析；

（4）支持并行执行：

–每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。

**缺点：**

（1）通常导致进程成为批处理的结构

（2）不适合处理交互的应用

（3）处理两个独立但又相关的数据流是可能会遇到困难

（4）在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性

1. **理解主程序-子过程风格的优点与缺点。**

**优点:**

–有效地将一个较复杂的程序系统设计任务分解成许多易于控制和处理的子任务，便于开发和维护

–已被证明是成功的设计方法，可以被用于较大程序

**缺点:**

–规模：程序超过10万行，表现不好；程序太大，开发太慢，测试越来越困难

–可重用性差、数据安全性差，难以开发大型软件和图形界面的应用软件

–把数据和处理数据的过程分离为相互独立的实体，当数据结构改变时，所有相关的处理过程都要进行相应的修改

–图形用户界面的应用程序，很难用过程来描述和实现，开发和维护也都很困难。

1. **理解面向对象风格的优点与缺点。P114-P115**

**优点**

（1）复用和维护：利用封装和聚合提高生产力

（2）反映现实世界

（3）容易分解一个系统

**缺点**

（1）管理大量的对象：需要考虑怎样确立大量对象的结构

（2）继承引起复杂度，关键系统中慎用

（3）必须知道对象的身份

（4）不是特别适合功能的扩展。为了增加新功能，要么修改已有的模块，要么就加入新的模块，从而影响性能

**9、 理解分层模式及其优缺点**。

**分层模式：**

某一层中的组件一般只与同一级别中的对等实体或较低级别中的组件交互，这种单向交互有助于减少不同级别中的组件之间的依赖性。分为：

严格分层(Strict System Layering)

松散分层(Loosely System Layering)

横切关注(Cross-Cutting Concerns)

**优点**

（1）支持基于抽象程度递增的系统设计

（2）局部依赖性

（3）可复用性

（4）可替换性

（5）对标准化的支持

（6）可测试性

**缺点**

（1）并不是每个系统都可以很容易地划分为分层的模式

（2）效率的降低

（3）很难找到合适的、正确的层次抽象方法

1. **理解事件系统的优缺点。**

**优点**

为软件复用提供了强大的支持，功能扩展比较容易

为系统动态演化带来了方便

健壮性

支持实现交互式系统（用户输入/网络

异步执行，不必同步等待执行结果

对事件的并发处理将提高系统性能；

**缺点**

分布式控制方式使系统的同步、验证和调试变得异常困难

数据交换存在全局性能和资源管理的问题

1. **理解黑板风格体系结构的优缺点**

**优点**

（1）便于多客户共享大量数据，它们不用关心数据是何时有的、谁提供的、怎样提供的。

（2）既便于添加新的作为知识源代理的应用程序，也便于扩展共享的黑板数据结构。

**缺点**

（1）不同的知识源代理对于共享数据结构要达成一致，而且，这也造成对黑板数据结构的修改较为困难——要考虑到各个代理的调用。

（2）需要一定的同步/加锁机制保证数据结构的完整性和一致性，增大了系统复杂度

第三部分

1. **理解软件体系结构模型的作用？**

1.以具体的形式来表现系统的框架结构

2.能够帮助人们从全局的角度来把握整个系统的框架结构

3.不同的利益攸关者有着不同的软件质量属性需求，他们所关注的问题是不相同的

4.不同的人从框架结构中所获取的信息也是不相同的，所以，软件体系结构模型应该是多维的，而不是一种单一的结构

1. **理解为什么采用多模型多视图方法。**

1.需求种类的复杂性是采用多模型多视图建模方法的主要因素

2.对于个人级的软件，“用户体验”是软件的灵魂

3.对企业级甚至是行业级的软件，其灵魂有两个：一个是软件带给用户的价值，另一个是软件的品质

4.成为一个软件产品的前提是满足约束，否则就不应该设计、开发、进入市场而成为一个垃圾

1. **了解软件体系结构模型的分类。**

结构模型

框架模型

动态模型

过程模型

功能模型

1. **理解“4+1”视图模型。**

**“4+1”视图模型基本概况**

1.软件体系结构建模方法和过程的具体实践范型

2.Kruchten在1995年提出（RUP、UML）

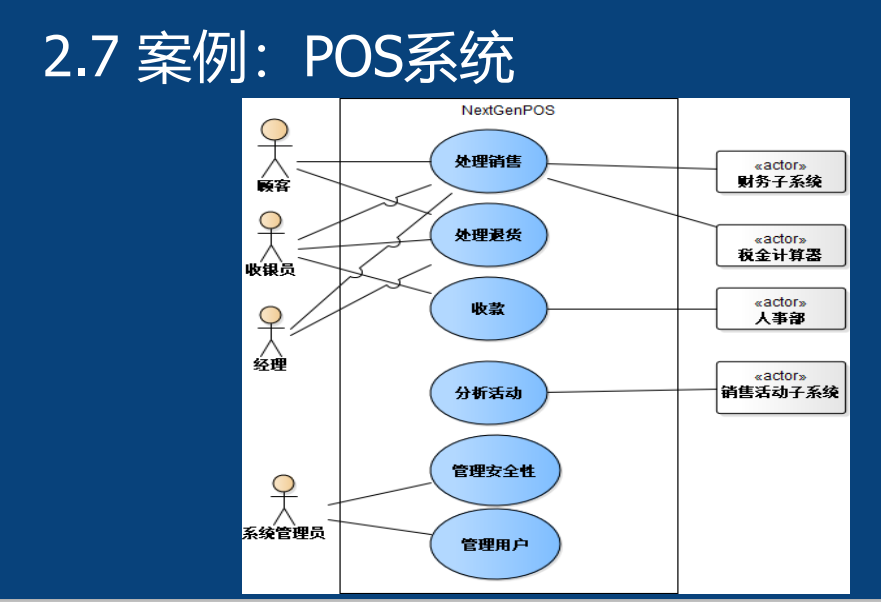
3.从5个不同的视角包括逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件体系结构。

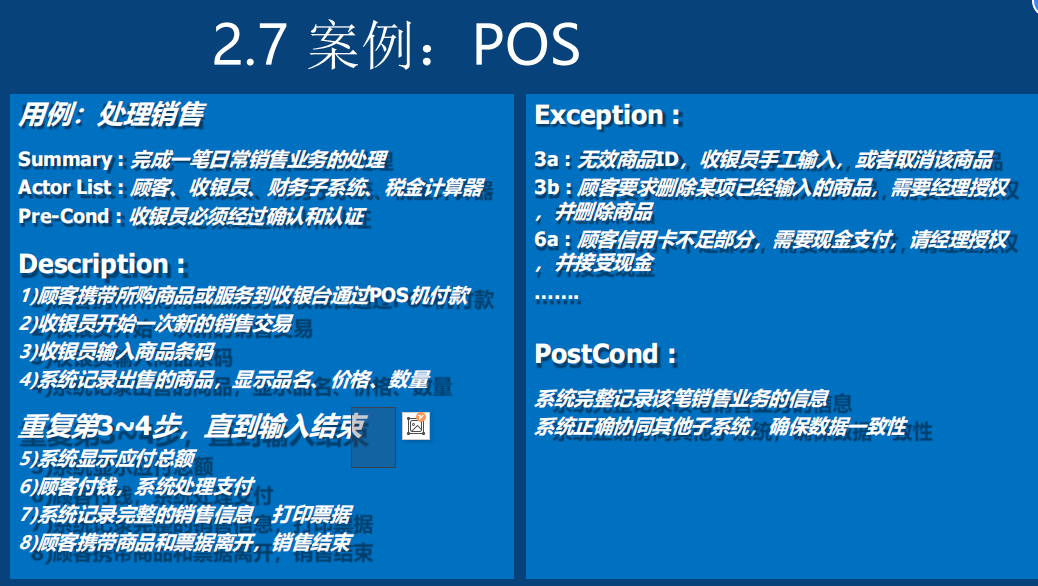
4.每一个视图只关心系统的一个侧面，5个视图结合在一起才能反映系统的软件体系结构的全部内容



第四部分

1.**理解用例图，认真理解课件中的用例例子**





1. **理解顺序图，认真理解课件中的顺序图例子**（P33、P45-47）
2. **理解活动图**

描述一项任务执行过程中所完成的工作（动作）

 描述对象内部的工作

 显示如何执行一组相关的动作，以及这些动作如何影响它们周围的对象

 显示用例的实例如何执行动作以及如何改变对象状态

 说明一次业务流程中的人（参与者）和对象是如何工作的

**活动图与用例模型互为补充，主要用于需求分析阶段**

**活动图中的基本要素包括**

活动（动作）、转移、分支、分叉和汇合、泳道、对象流等

**4.比较状态图与交互图、活动图。**

**交互** 

对共同工作的对象群体的行为建模

 动态行为

 **状态机** 

对单个对象的行为建模

 有时，可以对单个“完整系统”的行为建模

 说明对象在它的生命期中响应事件所 经历的状态序列以及对那些事件的响应

 动态行为建模

**活动图** 

强调从活动到活动的控制流，多个业务角色

5.**理解顺序图和通信图的区别。**

1.顺序图和通信图都来自UML的元模型中相同的信息，二者在语义上是等价的，可以相互转化

2.通信图通过对象之间的交流，通过在链接上附着带顺序号的消息来显示对象之间是如何被连接的结构关系，而顺序图则没有

3.顺序图显示消息的返回、有对象生命线、控制焦点，而通信图则没有。

第五部分

1. **理解软件可靠性的定义。**

定义：

在规定的条件下，在规定的时间内，软件不引起系统失效的概率，该概率是系统输入和系统使用的函数，也是软件中存在的错误的函数；

在规定的时间周期内，在所述条件下程序执行所要求的功能的能力。

可靠性是软件的13个质量因素中最关键、最重要的。

软件故障产生的原因是软件缺陷，但软件缺陷并不一定导致故障的产生，高缺陷率的软件的可靠性不一定就差。软件失效意味着软件运行中断或者无法完成所规定的任务。

1. **理解软件不可靠的主要原因。**
2. 不完善的需求定义
3. 客户与开发人员缺乏交流
4. 偏离软件需求
5. 逻辑设计错误
6. 编码错误
7. 编码与文档不一致
8. 缺少测试过程
9. 接口定义错误
10. 不受控的变更
11. **了解软件体系结构的风险分析方法。**
12. 动态方法
13. 构建依赖图
14. **了解基于体系结构描述的软件测试。**

软件体系结构测试与程序测试有所不同,它是检查软件设计的适用性, 这种测试不考虑软件的实现代码,所以基于实现和说明的程序测试方法对软件体系结构测试并不适用。

与传统的软件测试一样,基于体系结构的软件测试也需要研究 测试内容、测试准则、测试用例、测试充分性及测试方法等问题。

第六部分

1. **理解软件的功能属性和质量属性。**

功能属性概念：

功能属性是系统够完成所期望工作的能力

在开发过程中，功能要求是惟一首先要考虑的

功能属性与质量是正交的，因为功能属性并不能规定安全性、性能、可用性和易用性的等级

功能在很大程度上是独立于结构的。实际上，如果功能是系统的惟一需求话，整个系统是一根本没有内部结构的单一模块

功能属性所关心的是：它如何与质量交互，以及如何限制质量属性的

当质量属性很重要时，SA 会限制各种功能的分配

质量属性的概述：

商业目的决定了SA必须满足的一些质量性质。

重新设计系统往往不需要改变功能，而是为了满足某些质量属性。

没有任何一个质量属性完全依赖于设计、实现和部署的。

软件结构确定了SA对质量属性的支持，但不是唯一的。

系统的性能：是一个既依赖于、又不完全依赖于SA的质量属性。

质量属性的实现不能以孤立的方式实现。

质量属性包括：

1. 可用性：系统能正常运行的时间比。与系统故障及其后果相关。
2. 可修改性：能快速的以较高的性价比对系统运行更改的能力。有关变更的成本问题，关注修改什么、何时以及谁来进行更改。
3. 性能：系统的响应能力。与时间有关。
4. 安全性：是系统向合法用户提供服务的同时，阻止非授权使用的能力。
5. 可测试性：是经过测试揭示软件缺陷的容易程度，通常是基于运行的测试。
6. 易用性：关注的是对用户来说完成某个期望任务的容易程度和系统所提供的用户支持的种类。
7. 可靠性：系统能够长时间运行的能力。
8. 可移植性：系统能在不同计算机环境下运行的能力。
9. 功能性：系统完成所期望工作的能力。
10. 可变性：SA经过扩充或更改而变成新SA的能力。
11. 可子集性：支持子系统生成的能力。
12. 概念完整性：能在各个层次上将系统设计统一起来的根本思想。
13. **了解软件体系结构评估方法。**
14. SAAM方法
15. ATAM方法
16. ARID方法
17. 其他方法

第七部分

1. **理解构件的定义。**
2. 构件是预先构建的；
3. 构件只能通过接口访问的(黑盒)；
4. 构件是具有可分离性的；
5. 构件具有可组装和可部署等特征的
6. 构件是依靠支持技术和工具的。

定义：

1. 一个软件构件是可执行软件的一个可分离的单元
2. 只能通过构件的接口来访问它的服务
3. 必须能得到其接口的细节
4. 可以与其它构件实现互操作
5. 需要某种环境的支持
6. 在进行了必要的组装和配置过程之后,它才能以接口规定的方式来使用

第八部分

1. **理解软件产品线。**

背景：

**“软件产品线”** 是CMU(Carnegie Mellon University) 的SEI (Software Engineering Institute) 提出的。

**一个产品线(Product Line)**：是共享一组公共的、可管理的特性，并且满足特定市场需求的产品集合。

**产品线方法**必将成为新世纪中占主导地位的软件生产模式

（1） 产品的灵活性是市场的必然需求，而产品线将通过裁剪，生产出满足特定用户或用户群需要的产品。

（2） 从开发者的角度，产品线的成功在于产品之间通过共性的共享，达到了生产上经济的目的。

概念：

**一个软件产品线是满足下列性质的一组软件系统：**

1. 共享一组相同的、可管理的特性集合
2. 满足一类特定的市场需求
3. 以预先规定的方式基于公共核心资产集开发

**在一个软件产品线中，新产品的形成通过以下步骤：**

（1）从公共资产库中选取合适的构件

（2）使用预定义的变化性机制进行必要裁剪,如参数化、继承

（3）必要时增加新的构件

（4）在整个产品线范围内共同的SA指导下,进行构件组装,

**新产品（系统）的开发从“创造编程”变为“集成组装”**

使用产品线的好处和代价

**好处1：**产品线体系结构提供了在产品线中进行系统开发的结构，构件间的关系和约束。一旦定义好了产品线体系结构，意味着产品线中 所有产品的系统设计已基本完成。

**代价1**：产品线体系结构必须支持产品线内部固有的变化性，所以除了定义构件本身和构件之间的约束（必选的，可选的，可替换的），还要定义在产品线中开发系统时构件使用和演化的原则，增加了产品线体系结构定义的复杂性。

**好处2：**设计决策、数据结构、算法、文档、编码和调试信息等都属于可复用资产，它们在产品线的所有产品中可被反复使用。

**代价2：**因为可复用资产要适应不同产品之间的差异，所以要求可复用资产有足够通用的特性，同时要保证性能不被降低，增加了资产设计和实现的复杂性。

**好处3：**为软件开发购买的开发环境、配置管理工具、设备管理工具可以在整个产品线中使用，相当于投资的“分期付款”。

**代价3：**要求这些可复用的工具和环境有足够的适应性，适应一个产品线的通用性又可以适应单独产品的变化性 ，因此对这些工具和环境的要求较高。

**好处4：**产品线是面向特定领域中的共性产品，开发人员具有适应整个产品线的经验，可以按照需要随时转换项目，提高生产效率。

**代价4：**专业人员的培训代价高。

产品线方法是技术和管理的综合实践

**1. 领域分析技术**－用于理解产品线中内在的共性和变化性，开发出领域模型以便派生出体系结构和软件构件。

**2. 软件体系结构技术**－产品线方法是一种基于体系结构的开发实践，它涉及软件体系结构的定义和表示、体系结构的适用性评价、体系结构的抽取和一致性评价。

**3. 再工程技术**－用于支持构件的提取，从遗产系统中开采出产品线构件，并对遗产系统的演化进行规划。

**4. 技术管理**

**5. 组织管理**

面临的挑战

1. **文化上**－产品线策略意味着软件组织和管理者对他们产品开发的 直接控制减少了，对其它组织的依赖增加了，这意味着一种思想观念上的转变。
2. **战略计划上**－产品线的规划不仅是对一组相关系统的管理过程，还需要考虑用户的长期需要和现有产品线的能力，对未来的发展作出长远规划；
3. **需要折衷**－产品线方法需要用户作出折衷，是“独自开发一个恰好是我所需要的系统”，还是“利用产品线开发一个与我的需要非常接近的系统，但可以节省开发的代价和时间”；
4. **资源的所有权**－产品线构件归谁“所有”？在目前的体制下，这的确是一个容易引起纠纷的问题。这就需要整个软件开发和管理组织的重心从当前的程序获取转移到商业化的构件开发上来；
5. **提拔和奖励**－当前的开发方法主要是提拔和奖励那些提交了最终 系统的开发人员，采用产品线方法后还应该提拔和奖励那些开发构件和促进了产品线开发中构件复用的人员。