

Rational统一过程(RUP)

主讲：李妮娅
447898794@qq.com

课程设置

- 学分：2学分
- 理论学时：32学时
- 开课周数：8周

教学书籍

- 《**Rational** 统一过程引论(原书第2版)》
 - 美 **Philippe Kruchten** 编著
 - 周伯生 吴超英 王佳丽 译
 - 机械工业出版社

第一部分 过程

- 第1章 最佳的软件开发实践
- 第2章 **Rational**统一过程
- 第3章 静态结构:过程描述
- 第4章 动态结构:迭代开发
- 第5章 以构架为中心的过程
- 第6章 用例驱动的过程

第二部分 过程 workflow

- 第7章 项目管理 workflow
- 第8章 业务建模 workflow
- 第9章 需求 workflow
- 第10章 分析和设计 workflow
- 第11章 实现 workflow
- 第12章 测试 workflow

第二部分 过程 workflow

- 第13章 配置和变更管理工作流
- 第14章 环境 workflow
- 第15章 实施 workflow

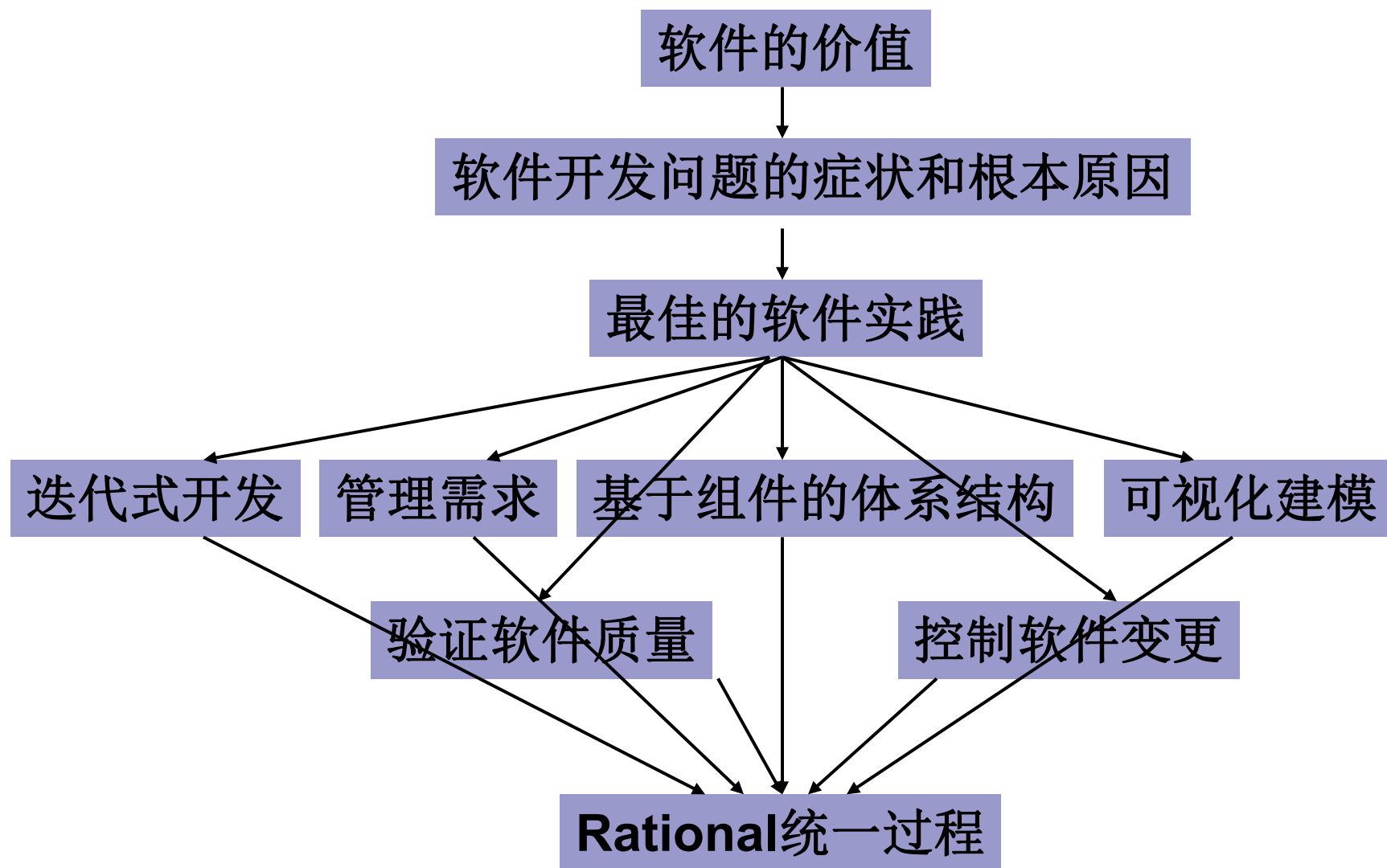
第1章 最佳的软件开发实践

- 1.1 软件的价值
- 1.2 软件开发问题的症状和根本原因
- 1.3 最佳的软件实践
- 1.4 迭代式开发
- 1.5 管理需求
- 1.6 基于组件的体系结构
- 1.7 可视化建模

第1章 最佳的软件开发实践

- 1.8 验证软件质量
- 1.9 控制软件变更
- 1.10 Rational统一过程

第1章章节关系



1.1 软件的价值

- 软件以一种前所未有的方式和形式帮助我们产生和获得信息，并使信息可视化。
 - 从全球范围看，软件业的飞速发展推动着世界经济更快地向前发展。
 - 从个人角度看，软件可以辅助治疗疾病，甚至模拟人类感官。
- 各种在技术上可行且有社会需求的软件密集型系统在规模、复杂性、分布和重要性上都在不断扩张。

这种扩张带来了一系列的技术问题和组织问题。生产和维护软件产品变得越来越难。

1.2 软件开发问题的症状和根本原因

症状

- 对于最终用户的需要理解得不够精确
- 对需求的改变束手无策
- 程序块不兼容
- 软件不易维护或不易扩展
- 对项目严重缺陷的发现较晚
- 软件质量低劣
- 软件性能无法接受
- 开发人员按各自的方式进行开发，软件很难重组
- 一个不可靠的构造和发布过程

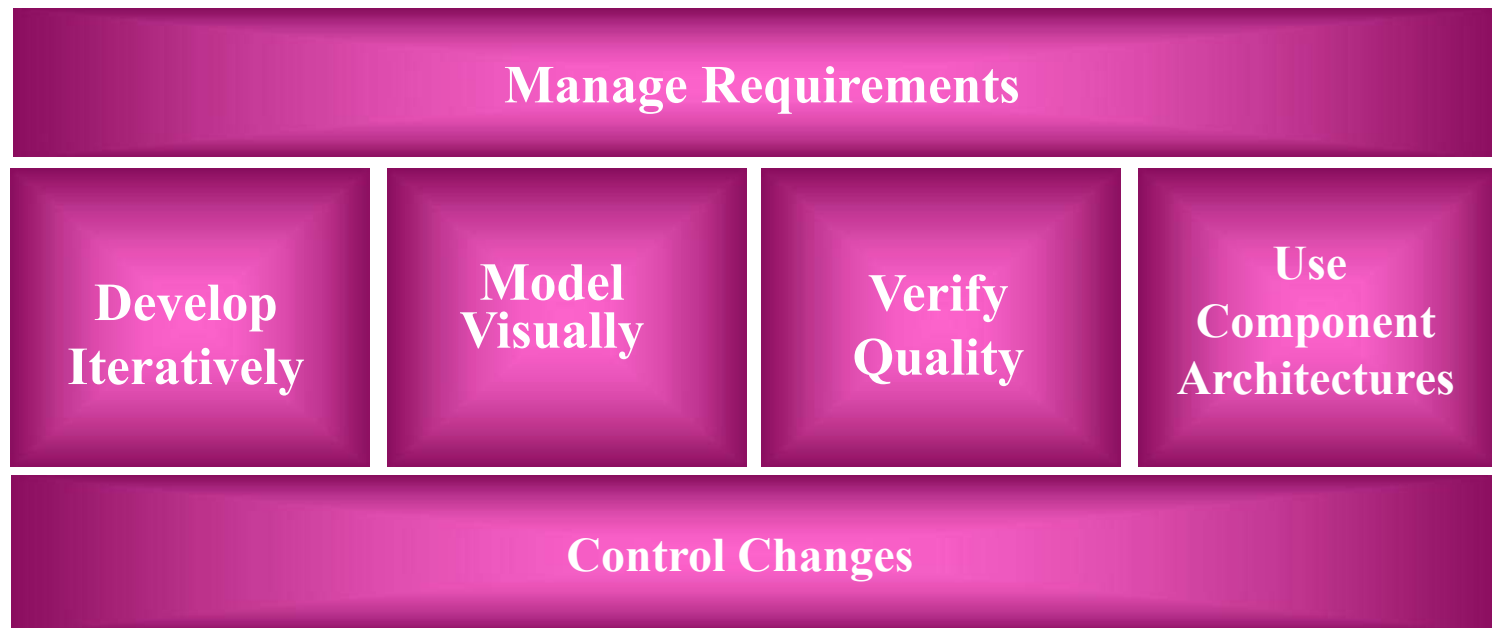
造成失败的原因

- 特别的需求管理
- 模糊和不精确的交流
- 脆弱的构架
- 过度复杂
- 未检测出需求、设计和实现之间的不一致
- 测试的不足
- 对于项目状况的评估过于主观
- 未解决存在的风险
- 无法控制变化的产生和传播
- 自动控制不足

1.3 最佳的软件实践

- 最佳实践：以一种循环的、可预测的方式开发和维护高质量的软件产品。
 - 在商业运作中已经证明这是一种能解决软件开发过程中根本问题的方法。
- 软件开发的六大最佳实践
 - 迭代式开发
 - 管理需求
 - 基于组件的体系结构
 - 可视化建模
 - 验证软件质量
 - 控制软件变更

六大最佳实践的关系



1.4 迭代式开发

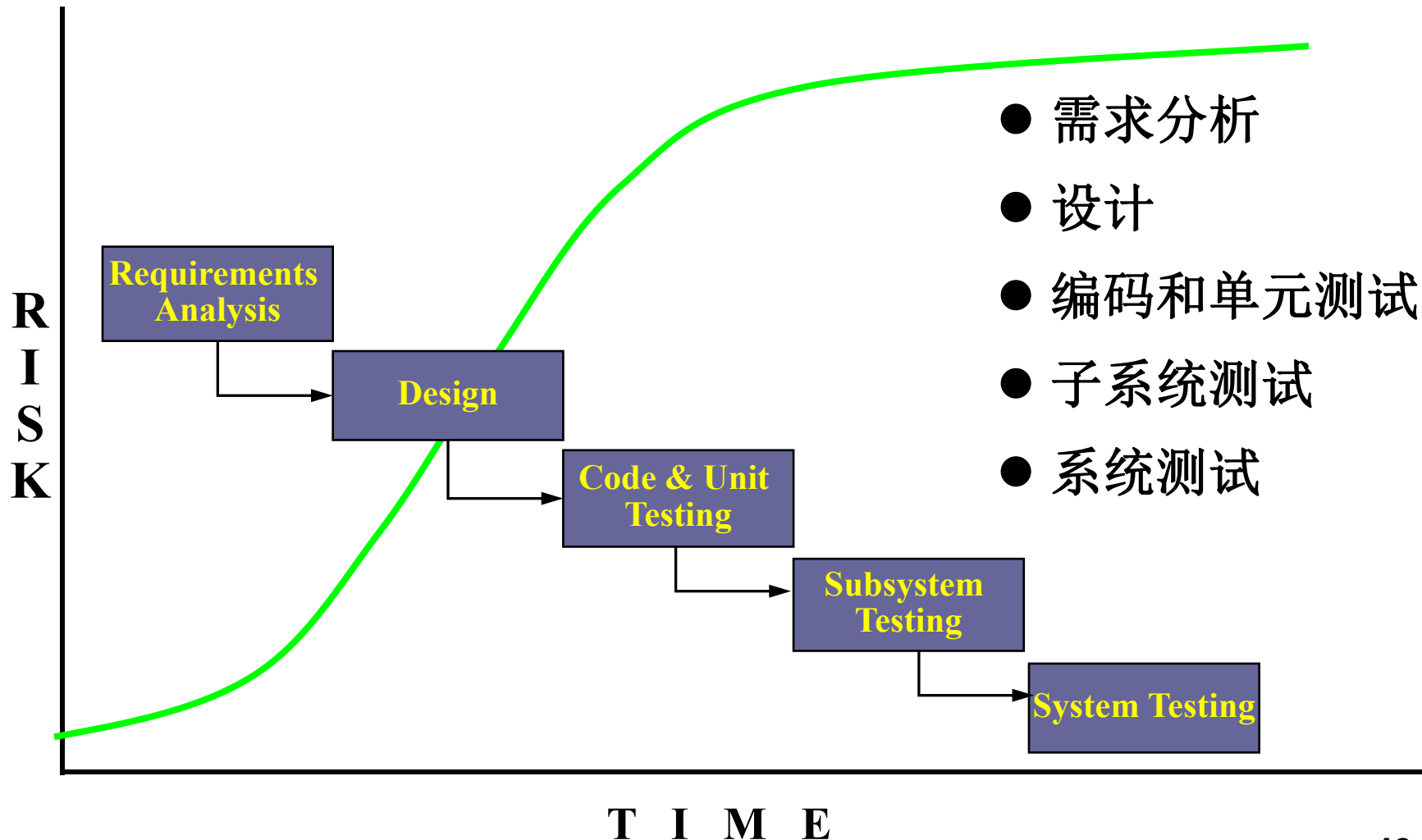
Develop Software Iteratively

■ why

- 随需求变化和理解深入，最初的设计很可能会出现问題。
- 如果设计问题直到后期才被发现，则很有可能导致费用超支，甚至项目失败。

瀑布式生命周期

Waterfall Development: Risk vs. Time



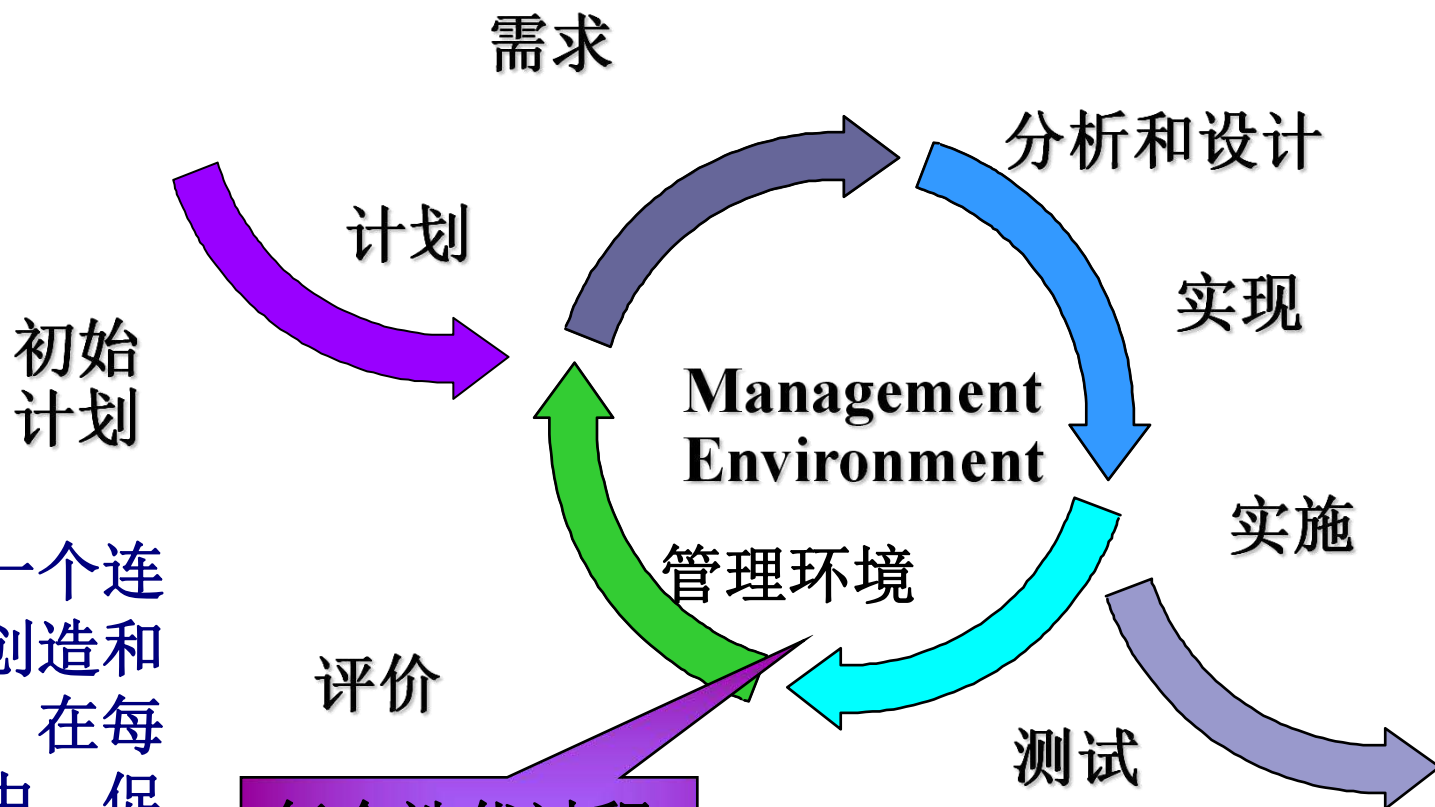
瀑布式方法的问题

- 这种方法最大的问题是，将风险留到后面的阶段。
- 一个早期的设计可能并没有注意到系统的关键需求，而在后期发现设计中的缺陷可能导致巨额的花费，甚至是项目的失败。

假如你不积极地解决你的项目中存在的风险，它们就会积极地解决掉你。

- 瀑布式方法会掩饰项目中真正的风险

另一个瀑布方法：一个迭代和递增的过程



这种方法是一个连续地发现、创造和实现的过程。在每个迭代过程中，促使开发小组以一种循环的、可预测的方式驱动项目制品的生产和制作。

每个迭代过程
产生一个可执
行的发布版本

迭代开发能够...

- 及早发现和修正一些错误的需求理解
- 鼓励用户反馈信息，以明确系统的真实需求
- 使开发小组重视项目中最关键的问题
- 通过不断的迭代测试给出项目状况的客观评价
- 尽早发现需求、设计和实现中的不一致
- 更加平均的分配各环节的工作量
- 在开发中学习和改进
- 通过具体证据了解项目状况

1.5 管理需求

需求是系统必须达到的条件或性能

- 管理软件密集型系统的需求的难点在于动态性：需求在整个软件生命周期中是变化的。
 - 确定一个系统的真正需求是一个连续的过程。
 - 一个新系统或升级系统的出现也会改变用户对系统需求的理解。

动态需求管理包括三项活动：

- 提取、组织系统的功能和约束，并将它们写成文档；
- 估计需求的变化，评估它们会产生的影响；
- 跟踪并记录双方商议的决定。

管理需求能够...

- 规定原则性的方法
- 人员之间的交流建立在已定义的需求之上
- 区分需求的优先级、对需求进行过滤和跟踪
- 尽可能对功能和性能作出客观评价
- 更容易发现系统中的不一致性
- 借助适当的工具支持，运用到外部文档的自动链接，可以为系统的需求、特性和踪迹提供一个知识库。

1.6 基于组件的体系结构

■ 一个系统的构架决定：

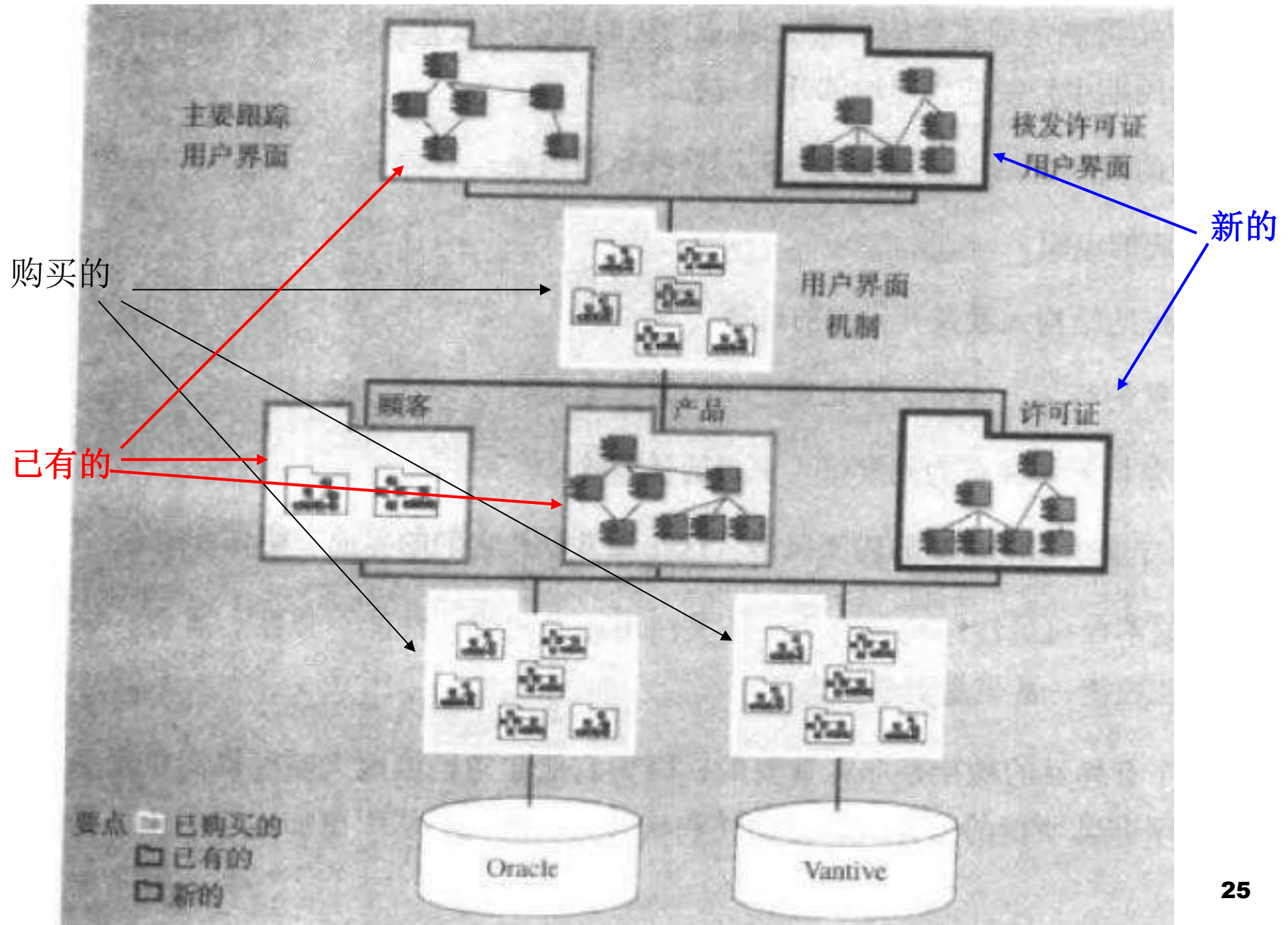
构架

- 软件系统的组织
- 构成系统的结构元素及其界面
- 结构元素的行为，通过这些元素之间的协作来说明
- ...

构架要考虑的因素

- 基本因素：结构、行为
- 其他因素：用途、功能、性能、弹性、重用、经济和技术限制，美学等等。
- 建立弹性的构架是非常重要的，它的好处在于：
 - 大幅度提高重用率
 - 使软件和硬件相互独立，从而更加适应变化
 - 提高可维护性

一个基于组件的软件构架



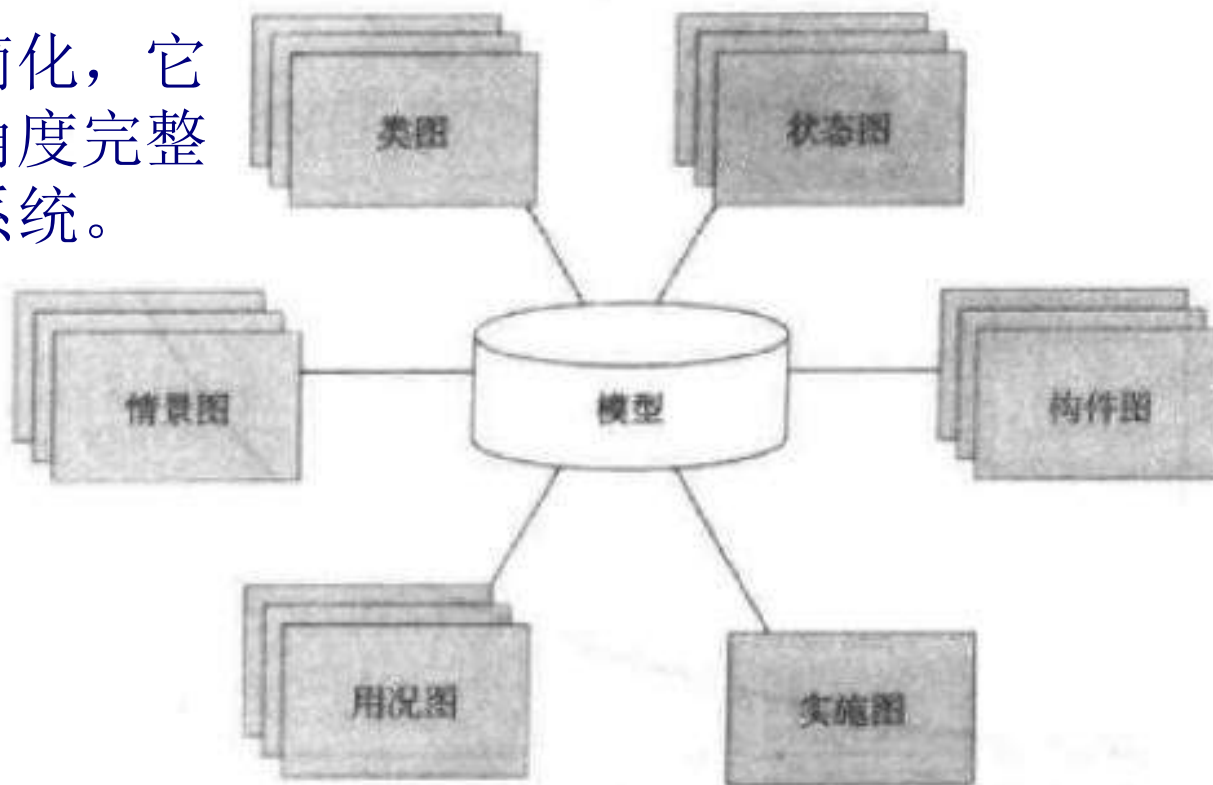
基于组件的体系结构能够...

- 组件有利于创建有弹性的构架
- 模块化使得人们可以逐个关注系统中易改变的不同元素
- 通过使用标准化的框架和商业上可获得的组件来提高重用。
- 组件为配置管理提供一个非常自然的基础。
- 可视化建模工具为基于组件的开发提供自动化的工具。

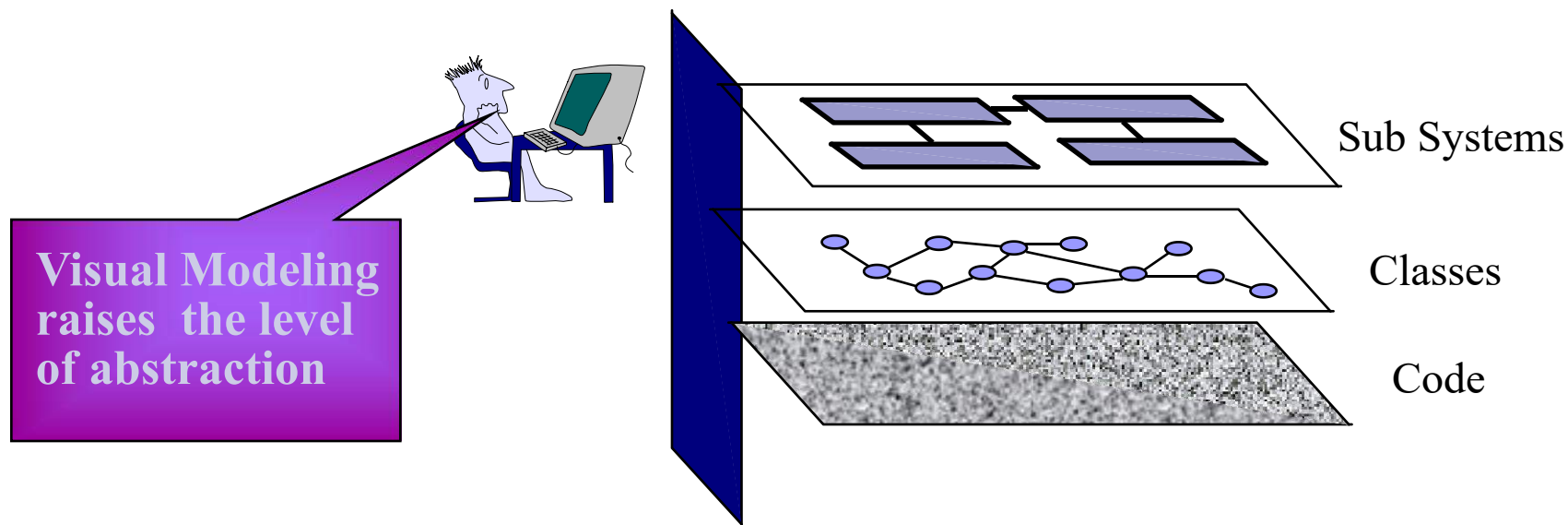
1.7 可视化建模

- 建模帮助开发团队将一个系统构架的结构和行为可视化、具体化、结构化，并写出文档记录。

模型是现实的简化，它从一个特定的角度完整地描述了一个系统。



- 可视化建模可以帮助开发团队提高管理复杂软件的能力。
 - 捕获架构和组件的结构和行为
 - 展示系统各个要素之间的协作方式
 - 维护设计和实现之间的一致性
 - 促进清晰的交流

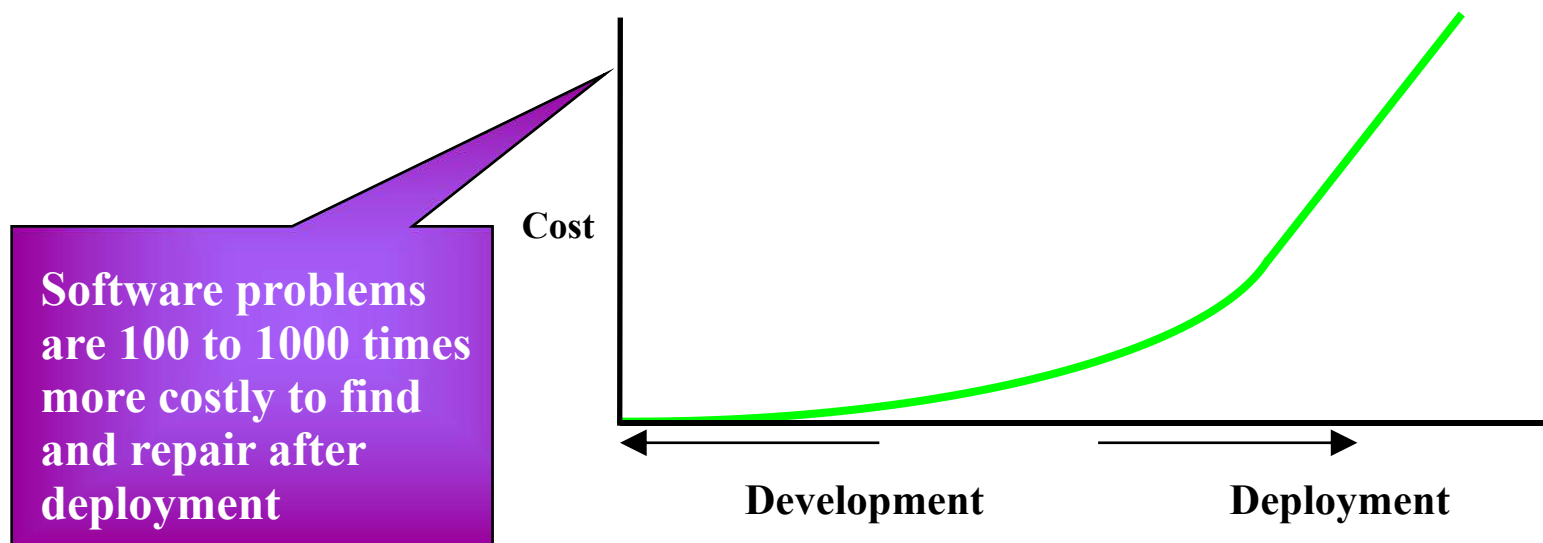


可视化建模能够...

- 毫无二义性地说明行为
- 毫无二义性地理解软件设计
- 非模型和不灵活的构架被暴露出来
- 必要时可以隐藏细节
- 容易揭示不一致性

1.8 验证软件质量

- 完成软件实施之后再去查找并修改软件问题要比早期进行这项工作多花**100到1000倍**的费用。因此从功能、可靠性、应用性和系统性能等多方面不断地对软件质量进行评估是非常重要的。



软件质量验证能够...

- 客观评估项目状况(评价的是检测结果，而非书面文档)
- 揭示出需求、设计和实现中的不一致
- 关注高风险领域，提高这些领域的质量水平和效率
- 提早发现缺陷，从而降低修改成本
- 测试功能、可靠性和性能

1.9 控制软件变更

- 迭代开发软件密集型系统面临的最大挑战：
 - 安排不同组、不同地点的开发人员同时工作于多个迭代过程、发布版本、产品和平台中。
- 为解决上述问题，要建立用于管理软件变更和其他开发制品变更的循环 workflow：
 - 根据项目的优先级和风险分配资源
 - 根据变更动态地管理工作
- 每次迭代过程结束时应建立并发布一个已检测过的基线

控制软件变更能够...

- 定义需求变更的工作流，并不断重复这个工作流
- 变更请求促进了更加清晰的交流
- 为开发人员建立独立的工作空间，减少平行工作之间的干扰
- 变更率为客观评价项目状况提供了很好的度量
- 工作空间包括了所有制品，有益于保持一致性
- 变更的产生和传播是可评价和可控制的
- 可以在一个健壮的定制系统中维护变更

1.10 Rational统一过程

- 软件开发过程的四个行为：
 - 为开发小组的活动顺序提供向导
 - 详细说明哪些制品将被开发，以及什么时候开发
 - 指导每一个开发人员和整个开发组的工作
 - 为监控和度量项目的产品和活动提供准则
- **Rational统一过程**将**6大**最佳实践活动以一种适当的形式结合起来，从而适应了广泛的项目和开发组织。



第2章 Rational统一过程



目录

- **2.1 什么是Rational统一过程**
- **2.2 作为一个产品的Rational统一过程**
- **2.3 二维过程结构**
- **2.4 Rational统一过程中软件的最佳实践**
- **2.5 Rational统一过程的其他重要特征**
- **2.6 Rational统一过程的简要历史**
- **2.7 小结**

第2章章节关系

什么是Rational统一过程

作为一个产品的Rational统一过程

二维过程结构

动态时间

第四章、第七章

静态内容

第三章

承接第1章

Rational统一过程中软件的最佳实践

第3-13章详细讲述实现这些最佳实践的方式

Rational统一过程的其他重要特征

Rational统一过程的简要历史

2.1 什么是Rational统一过程

- **Rational统一过程简称为RUP**

- 是一个软件开发过程
- 是一个过程产品
- 是一个过程框架
- 包含了**6**个最佳软件开发实践

- 为所有方面和层次的程序开发提供指导、模板和工具指南的支持。

Provides guidelines, templates and tool mentors for the effective implementation of key best practices

RUP是一个软件开发过程

- 软件开发过程是一个将用户需求转化为软件系统所需要的活动的集合。
 - 按照预先制定的时间计划和经费预算，开发出满足用户需求的高质量的软件产品



软件开发过程的四个行为

- 为开发小组的活动顺序提供向导
- 详细说明哪些制品将被开发，以及何时被开发
- 指导每一个开发人员和整个开发组的工作
- 为监控和度量项目的产品和活动提供准则

Bed Process or Good Process

- **Bed:** 每个成员都以自己的方式在工作，开发成功与否取决于是否有几个具有献身精神的人愿意付出他们的最大努力。
- **Good:** 最佳实践活动被编入过程中，使得开发团队以一种可预测的循环方式来开发复杂系统，开发团队的能力也将获得稳定提高。

RUP是一个过程产品 (process product)

- **Rational**软件公司开发并维护这个产品
- 该产品与**Rational**公司的一系列软件开发工具集成到一起

RUP是一个过程框架

- **RUP**把开发中面向过程的方面（例如定义的阶段，技术和实践）和开发制品（例如文档，模型，手册以及代码等等）整合在一个统一的框架内。
- 这个框架可以被改造和扩展以适应采纳此方法的组织

2.2 作为一个产品的Rational统一过程

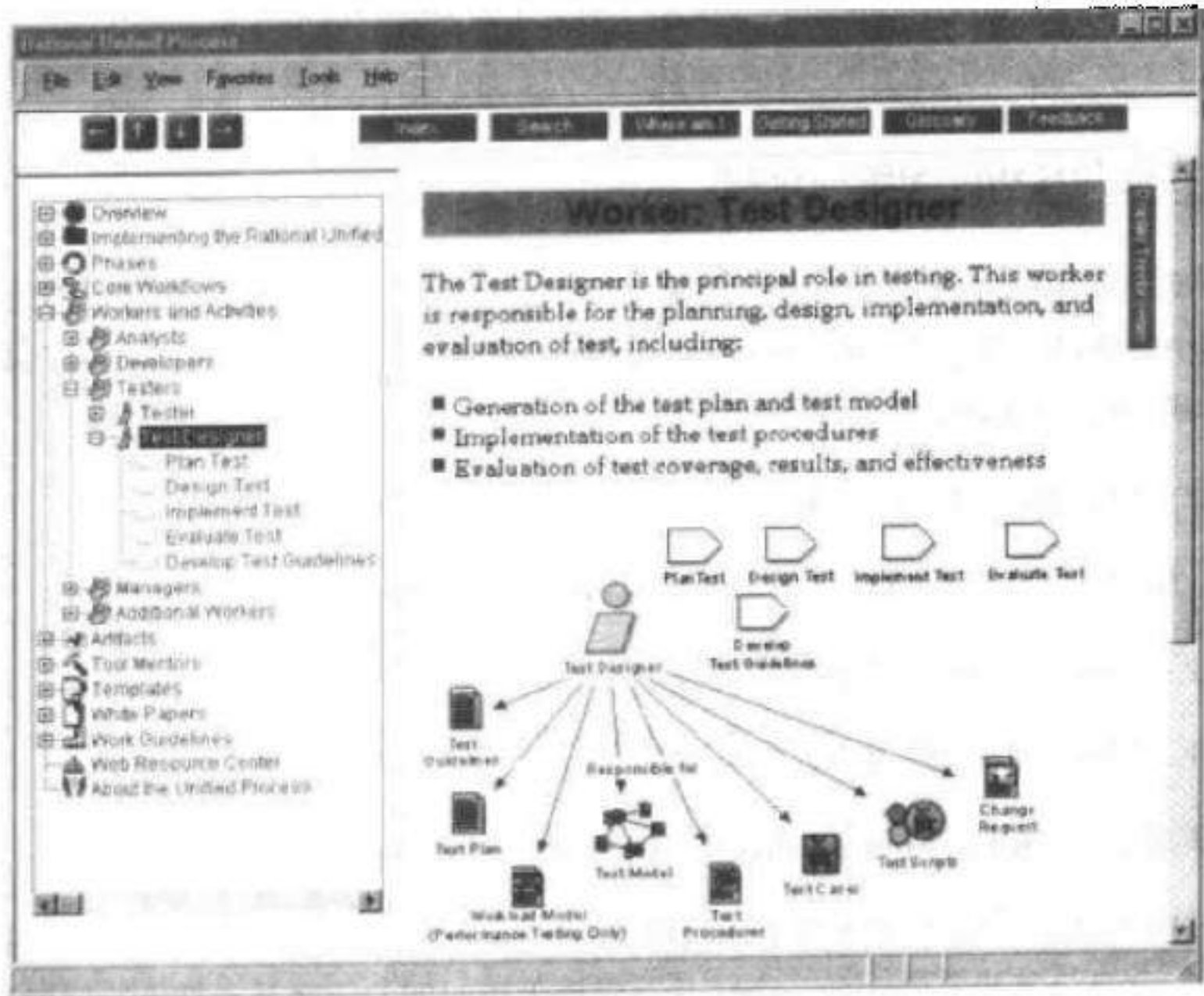
- 软件过程也是软件
- **RUP**有着软件产品的共同特征：
 - 发布升级版本
 - 通过网络技术在线移交**RUP**
 - 可以根据开发的需要量体裁衣
 - 与多种**Rational**软件开发工具集成，如**Rational Rose**

将这种过程作为一种软件产品有以下好处

- 过程从来不会过时,每隔一段时间,公司就会获得一个版本
- 所有项目成员可以从公司内部网上获得过程的最新版本
-

Rational 统一过程产品包括以下内容:

- 基本简介,即这本书
- **RUP**的在线版本,即电子版教程
 - 广泛的超链接
 - 图形导航
 - 分层树形浏览器
 - 详尽的索引
 - 内置的搜索引擎
 - 详尽的位置图
 - 工具指南
 - 主要过程制品的模板
 - 关于简单项目制品的示例



电子版教程网页的快照

360导航_一个主页, 整个世界 x rup现状_360搜索 x ibm_百度搜索 x IBM Browsing RUP with Mozilla Firefox x rup - 淘宝巴巴 x

https://www-01.ibm.com/support/docview.wss?uid=swg21190076

IBM

Search

IBM Support Offerings My support Downloads Documents Cases Communities Training Other

Browsing RUP with Mozilla FireFox.

Technote (troubleshooting)

Problem(Abstract)

This technote explains that to browse RUP® with Mozilla™ FireFox, the attached zip file needs to be downloaded. Without this fix, the RUP applet is too small and the height cannot be adjusted.

Cause

RUP 2003.06.13 does not support Mozilla FireFox out of the box. The supported browsers are mentioned in ...\\Program Files\\Rational\\doc\\rup_readme_sr.html:

Windows:

- Microsoft Internet Explorer 5.01 with SP2 or later
- Netscape Navigator versions 4.72 to 4.77 and 7.0

UNIX

- Netscape Navigator 4.72 to 4.77 or later

Resolving the problem

正在传输来自 static.hotjar.com 的数据...

对该页面打分

☆☆☆☆☆

平均评分 (0 用户)

文档信息

More support for: Rational Unified Process

软件版本: 2003.06.13

操作系统: Platform Independent

参考号: 1190076

修改日期: 13 一月 2005

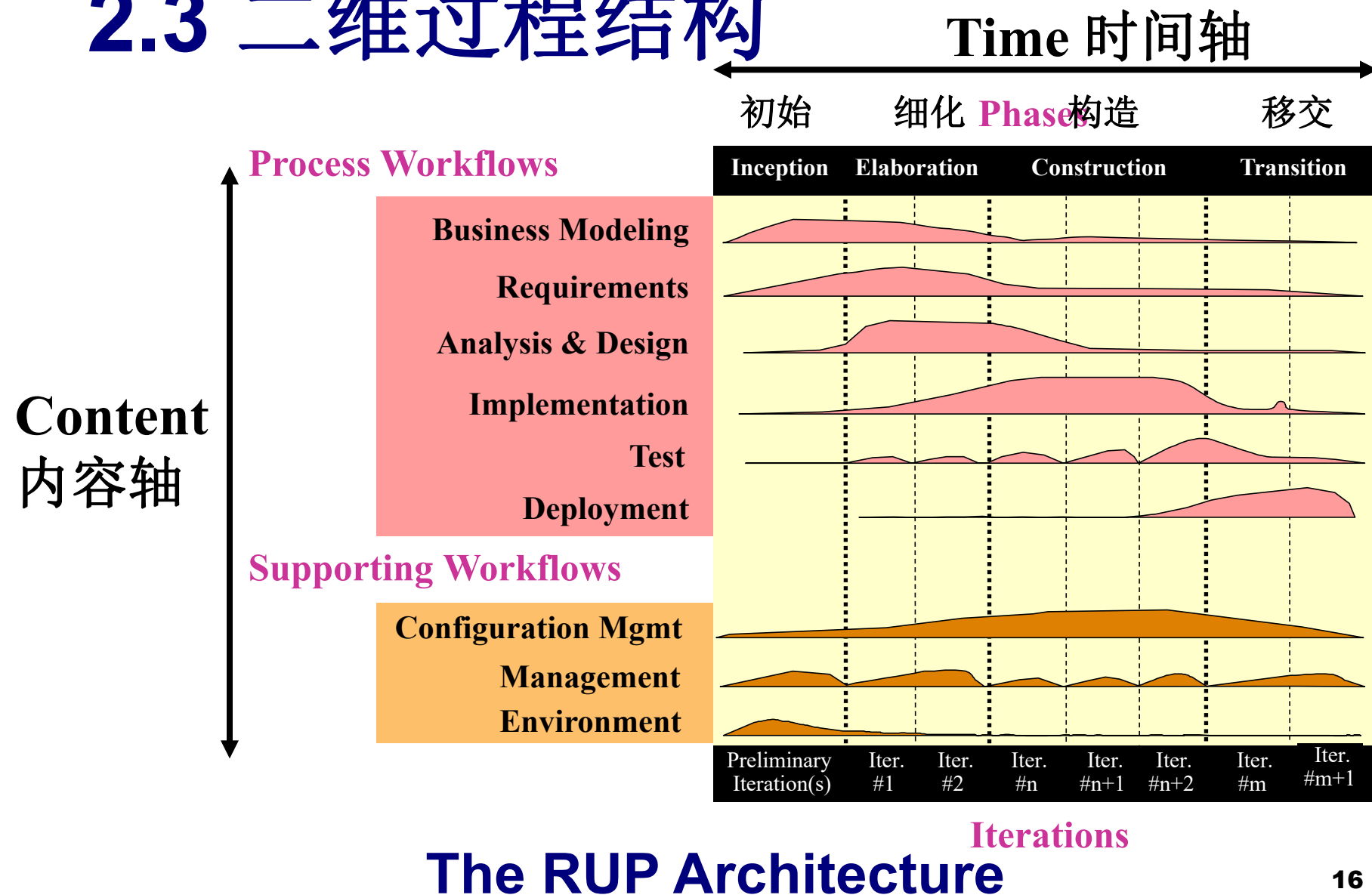
Contact and feedback

■ <https://www-01.ibm.com/support/docview.wss?uid=swg21190076>

RUP作为一个过程产品,包含的主要过程制品的模板如下

- 微软的**Word**模板和**Adobe**的**FrameMaker**模板, 用来写大部分普通文档和报告
- **Rational SoDA**模板, 将各种资源自动集成文档
- 微软的**Project**模板, 用来帮助在**RUP**的基础上计划迭代项目
- **HTML**模板, 用于扩展在线过程本身
-

2.3 二维过程结构



■ 时间轴:过程被激活时动态的一面

- 通过周期、阶段、迭代和里程碑来表示
- 第4章 “动态结构：迭代开发”
- 第7章 “项目管理 workflow”

■ 内容轴:过程静态的一面

- 通过过程构件、活动、workflow、制品和工作人员描述过程。
- 第3章 “静态结构：过程描述”

2.4 Rational统一过程中软件的最佳实践

- 迭代开发
- 需求管理
- 构架和构件的使用
- 建模和**UML**
- 过程质量和产品质量
- 配置管理和变更管理

上述实践能够为使用者带来诸多好处

迭代开发的优点

第4章、第7章

- 可以使开发者考虑到需求的改变
- 集成工作是逐渐完成的，而不是在最后结尾时的“大爆炸”
- 可以在早期降低风险
- 当产品做出战略性改变时，迭代开发提供了相应的变更管理方法
- 迭代开发可以促进重用
- 构架在连续迭代过程中获得不断修正，会变得更加坚固
- 开发人员可以更早的参与到过程中，并不断地学习
- 开发过程本身也在不断提高和精化

需求管理的优点

第9章、第13章

- 更好地控制复杂项目
- 提高软件质量和客户满意度
- 降低项目成本和延迟
- 增强群组中的交流

构架和构件的使用

第5章

- 软件构件(**Component**)是软件、模块、包或子系统的一个重要部分，这部分具有一个明确的功能和界限，并且能够集成为一个定义好的构架。

构件开发的不同目的

- 为了定义一个模块化的构架。
 - 要确定、分离、设计、开发和测试已形成的构件。这些构件可以被逐步集成，最终成为一个完整的系统。
- 为了重用解决方案。
 - 他们构成了重用基础，提高了整个软件的生产能力和质量。
- 为了构造一个支持软件构件概念的基础平台。
 - **CORBAR、ActiveX、JavaBeans**等在不同领域支持开发人员使用这些现成构件。
 - 他们允许开发人员购买并集成构件，而无需自己开发。

■ RUP支持以构件为基础的开发，具体体现在以下几个方面：

- 迭代方法允许开发人员**逐渐确定构件**，决定哪些构件需要开发、哪些构件可以重用、哪些构件要去购买
- 将注意力**集中在软件构架上**，构架列举了构件和集成它们的方法，同时也描述了他们交互作用的基础机制和模式
- 在分析设计中**用到如包、子系统和层这样的概念**来组织构件和确定接口
- **一开始对独立模块进行测试**，而后逐渐扩大到对更大的集成构件进行测试

建模和UML

第6章，第8章，
第10章，第12章

- 模型帮助使用者理解并找到问题及问题的解决方案。
- 模型是现实的简化，帮助使用者掌握复杂系统
- **RUP**在开发过程中开发和维护系统模型
- **UML**是一种图形语言，用来将软件密集型系统的制品可视化、具体化，并构造、记录这些制品
- **UML**是描述不同模型的通用语言，但它不能告诉我们如何开发软件
- **RUP**指导使用者如何有效地使用**UML**建模。
- **RUP**告诉使用者需要什么样的模型，以及如何建造这样的模型。

产品质量和过程质量

第12章

■ 产品质量

- 生产出的主要产品(如软件或系统)及产品中包含的所有元素(如构件、子系统、构架等)的质量

■ 过程质量

- 指产品制造过程中，过程被执行并坚持到的程度。
- 还关注支撑主要产品的制品质量(如迭代计划、测试计划、用例实现、设计模型等)

配置管理和变更管理

第13章

- 在迭代开发中开发计划和执行过程都允许变更，因此要跟踪这种变更，确保每一个人、每一件事的同步进行。
- 配置管理处理产品结构，重要的制品要受到版本控制。
- 变更管理处理过程结构，一个变更请求是一份需要变更一个或多个制品的文档化建议。

2.5 Rational统一过程的其它重要特征

- 用例对开发的驱动作用
- 作为过程框架，**RUP**可以根据需要被裁剪和扩充
- 需要软件开发工具来支持过程

用例驱动

第6章

- **Rational**统一过程是一个用例驱动方法，这意味着为系统定义的用例是开发过程的基础。
- 用例是业务建模的关键。
- 用例在过程工作流程中扮演着重要的角色，尤其是在确定需求、设计、测试和管理工作流程中。

过程配置

第3章、第17章

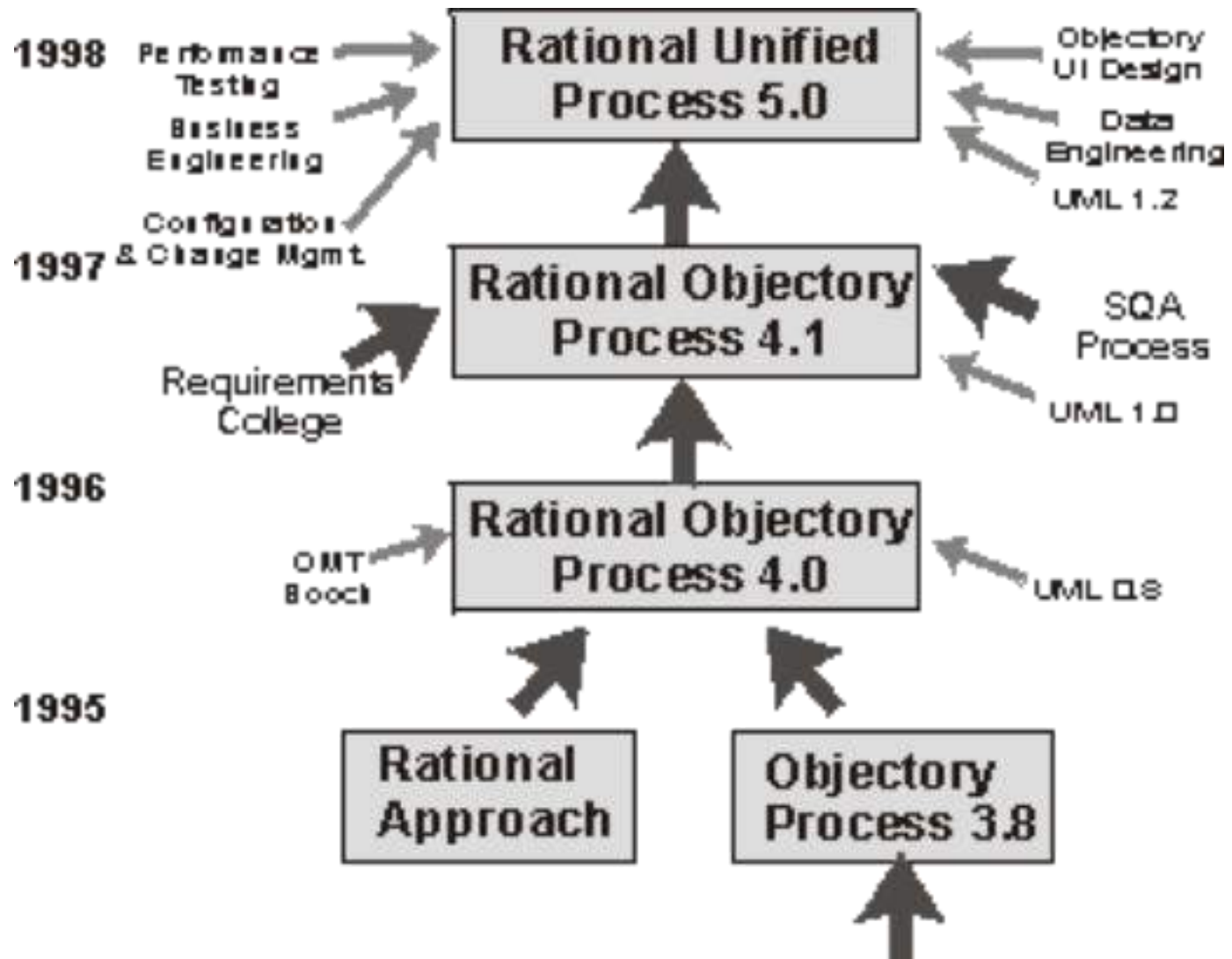
- **RUP**是一个过程框架，开发人员可以对其进行**修改、调整和扩展**，以适应自己特殊的需求、特征、约束等。
 - 开发人员不能盲目地遵循一个过程，以至于生产出无附加值的制品。而要**尽可能地精简过程**，同时很好地完成任务，以最快的速度生产出期望的高质量软件产品。
 - **很多过程元素有可能会被修改、客户化、增加或压缩**，其中包括制品、活动、工作人员和工作流，以及指南和制品模板。

工具支持

第3章

- 支持**RUP**的工具有很多，这些工具能自动完成许多活动中的步骤。
- 这些工具用于生产和维护软件工程过程中的不同制品，包括可视化建模、编程和测试。
- 这些工具对于记录每个迭代过程中的配置管理和变更管理起着重要作用。

2.6 RUP的简要历史



2.7 小结

■ RUP是一个

- 适用于整个软件开发周期的软件开发过程
- 过程产品，以电子版教材的形式为开发人员提供知识
- 嵌入了现代化技术和方法的指导。如对象技术、基于构件的开发、建模及**UML**、构架和迭代开发等

■ RUP是

- 活动的，总被维护和不断更新的
- 建立在坚固的过程构架基础上的，允许开发团队自主配置和剪裁

■ RUP支持六项最佳软件开发实践

■ Rational软件公司开发的很多工具都支持RUP

■ RUP二维过程结构

- 时间轴：第4章、第7章
- 内容轴：第3章

■ RUP中软件的最佳实践

- 迭代开发：第4章、第7章
- 需求管理：第9章、第13章
- 构架和构件的使用：第5章
- 建模和UML：第6章，第8章，第10章，第12章
- 过程质量和产品质量：第12章
- 配置管理和变更管理：第13章

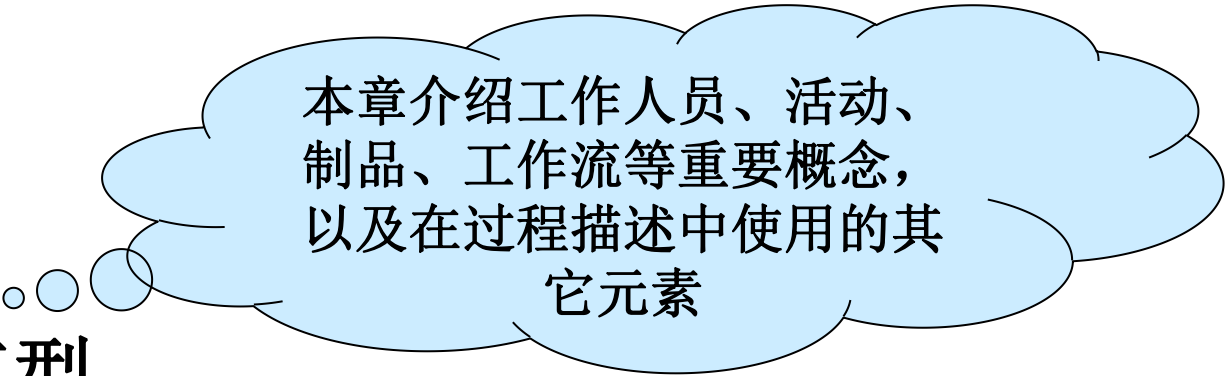
■ RUP的其它重要特征

- 用例驱动：第6章
- 过程配置：第3章、第17章
- 工具支持：第3章



第3章 静态结构：过程描述

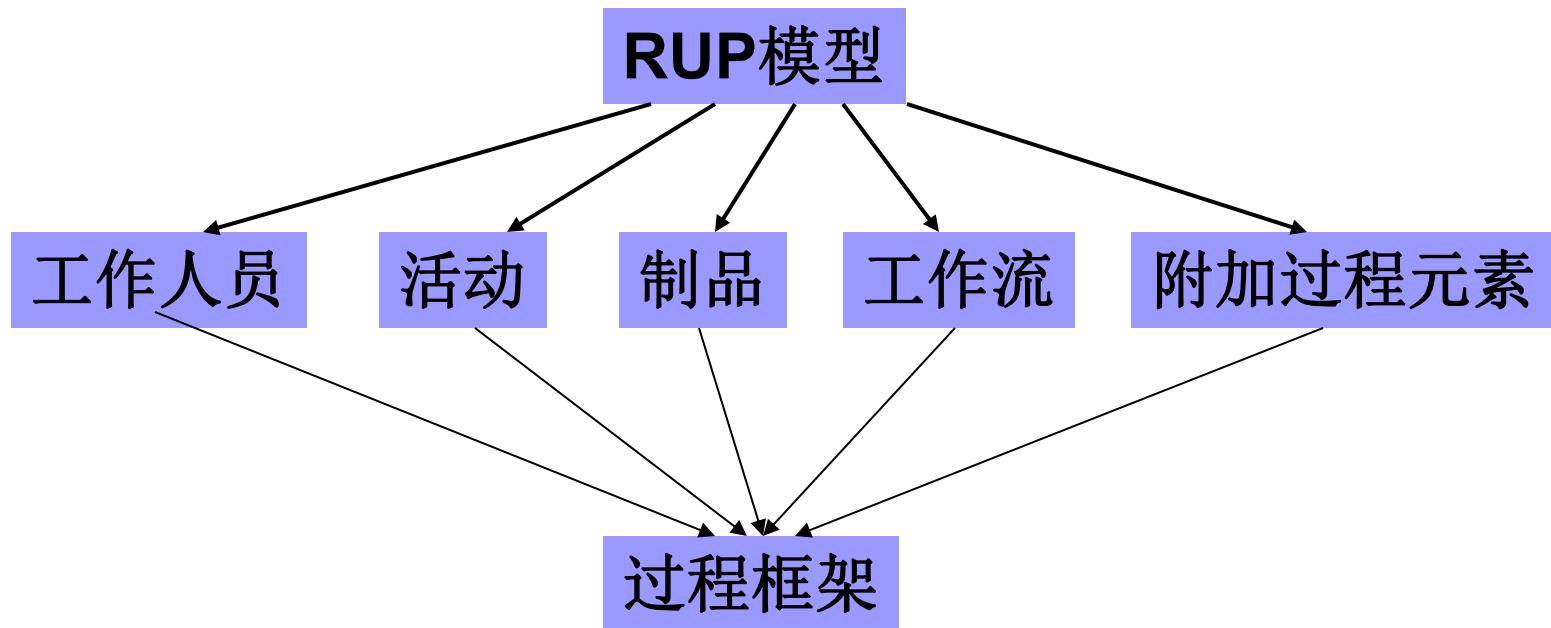
目录



本章介绍工作人员、活动、制品、 workflow 等重要概念，以及在过程描述中使用的其它元素

- **3.1 RUP模型**
- **3.2 工作人员**
- **3.3 活动**
- **3.4 制品**
- **3.5 workflow**
- **3.6 附加过程元素**
- **3.7 过程框架**
- **3.8 小结**

章节关系



3.1 RUP模型

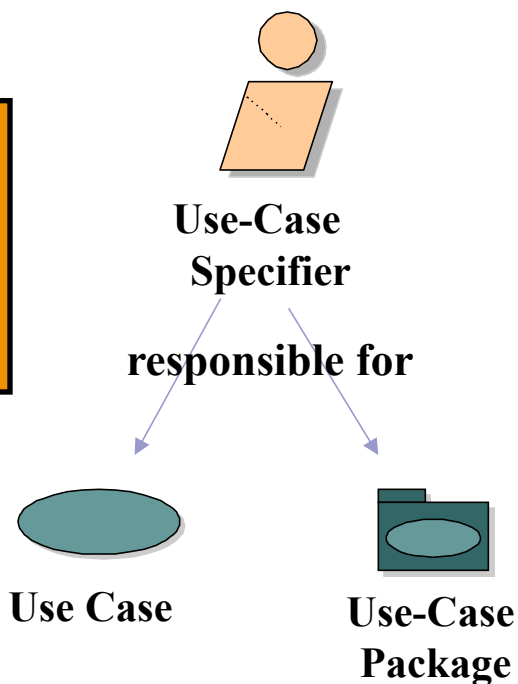
- 一个模型描述了谁来做、做什么、怎么做和什么时候做。
- **RUP**应用了四种重要的模型元素。
 - 工作人员(**worker**): 谁来做
 - 活动(**activity**): 怎么做
 - 制品(**artifact**): 做什么
 - workflow(**workflow**): 什么时候做

工作人员、活动和制品

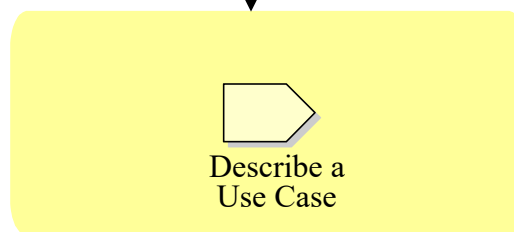
描述一个需要某个
worker执行的工作

Worker

表示一个角色，
可以由一个人充当，也
可以由一个团队来充当



Activity



Artifact

表示过程产生、修改或
使用到的一种信息

3.2 工作人员

- 工作人员(**worker**)定义了个人或一个工作组的行为和职责。
- **行为**用活动(**activity**)表示，每一个“**worker**”都与一组“内聚”的活动相联系。
 - 内聚是指这些活动最好由一个人来完成。
- **职责**的表示通常与某一特定制品(**artifact**)相联系，这些制品由**worker**制造、修改和控制。

常见的工作人员

■ 系统分析员

- 概述系统功能和界定系统
- 引导和协调需求，用例建模

■ 设计师

- 定义一个或多个类的职责、操作、属性和它们之间的关系
- 决定如何调整类以适应实现环境

■ 测试设计师

- 计划、设计、实现和评估测试，包括产生测试计划和测试模型，执行测试规程，评估测试覆盖度、测试结果和测试效率

worker与个人之间的关系

- “worker”是角色，每个人可以充当多种“worker”，一个“worker”也可能由若干个人充当。
- 从个人到“worker”的映射由项目经理来指定



3.3 活动

- 活动(**activity**)定义了**worker**执行的工作
- 活动有明确的目的，能够产生有意义的结果。通常是生产制品或更新制品(如模型、类或计划)
- 活动时间可以是几个小时到几天
- 一个活动通常涉及一个**worker**，影响到一个或几个制品
- 多个活动也可能重复作用于一个制品，如从一个迭代过程到另一个迭代过程时

常见的活动

- 计划迭代过程
 - 项目经理完成
- 寻找用例和参与者
 - 系统分析员完成
- 评审设计
 - 设计评审员完成
- 执行性能测试
 - 性能测试人员完成

活动步骤可以分解为三步

■ 思考步骤(thinking step)

- 理解任务的性质
- 收集和检验输入制品
- 阐明输出

■ 执行步骤(performing step)

- 生产或更新制品

■ 评审步骤(reviewing step)

- 根据一些标准检验结果

一个活动步骤的例子

- 活动：找到用例和参与者 说明：思考与执行在本例中没有强调和明显区分。

☐ 1) 寻找参与者

☐ 2) 寻找用例

☐ 3) 描述参与者和用例之间的交互作用

☐ 4) 将用例和参与者打包

☐ 5) 用用例图表示用例模型

☐ 6) 开发一个用例模型综述

☐ 7) 评价结果

查找(思考)

执行

评价

3.4 制品

- 制品(**artifact**)是由过程生产、修改或使用的有形产品
- 项目在生产出最终软件产品的过程中生产或使用它们
- 根据与活动的关系，制品分为输入制品和输出制品

常见的制品

- 模型

- 如用例模型和设计模型

- 模型元素

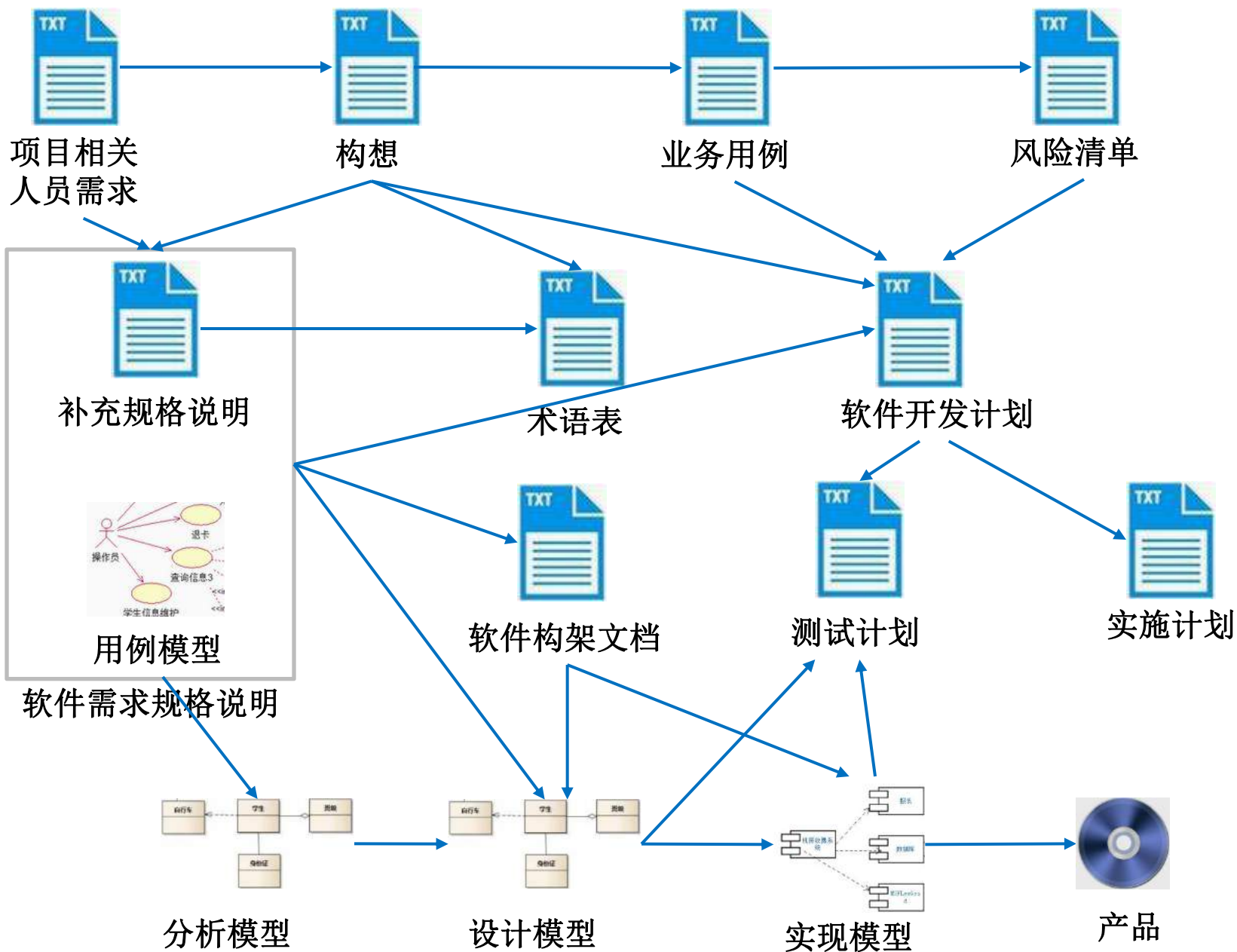
- 如一个用例、类、子系统

- 文档

- 如一个业务用例、软件架构文档

- 源代码

- 可执行文件



制品不是文档，文档是一种制品

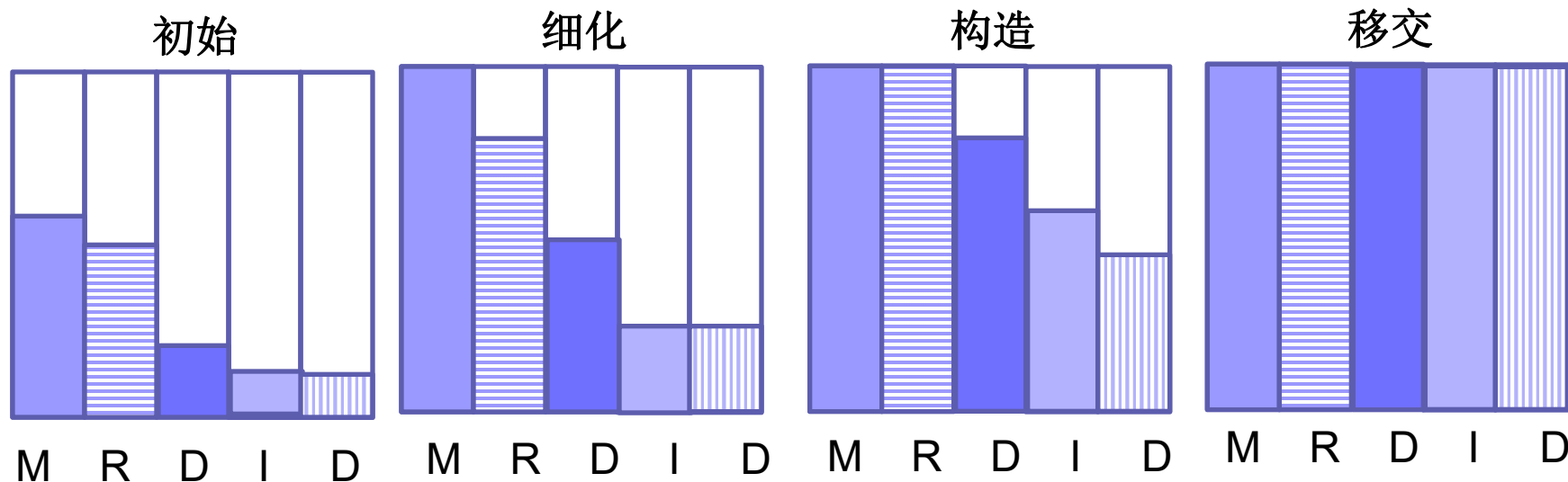
- **RUP**不鼓励系统地生产纸质文档，最有效和最实用的管理项目制品的方法是使用适当的生产和管理这些制品的工具来维护这些制品。
- 关于制品的具体示例：
 - 存储在**Rational Rose**中的设计模型
 - 存储在**Microsoft Project**中的项目计划
 - 存储在**ClearQuest**中的缺陷

制品集

■ RUP中的制品分为五个信息集(information set)

- 管理集：计划制品、版本说明、状态评估等
- 需求集：构想文档、业务模型、用例模型等
- 设计集：设计模型、构架描述、测试模型等
- 实现集：源代码和可执行程序、相关数据文件等
- 实施集：安装资料、用户文档、培训资料。

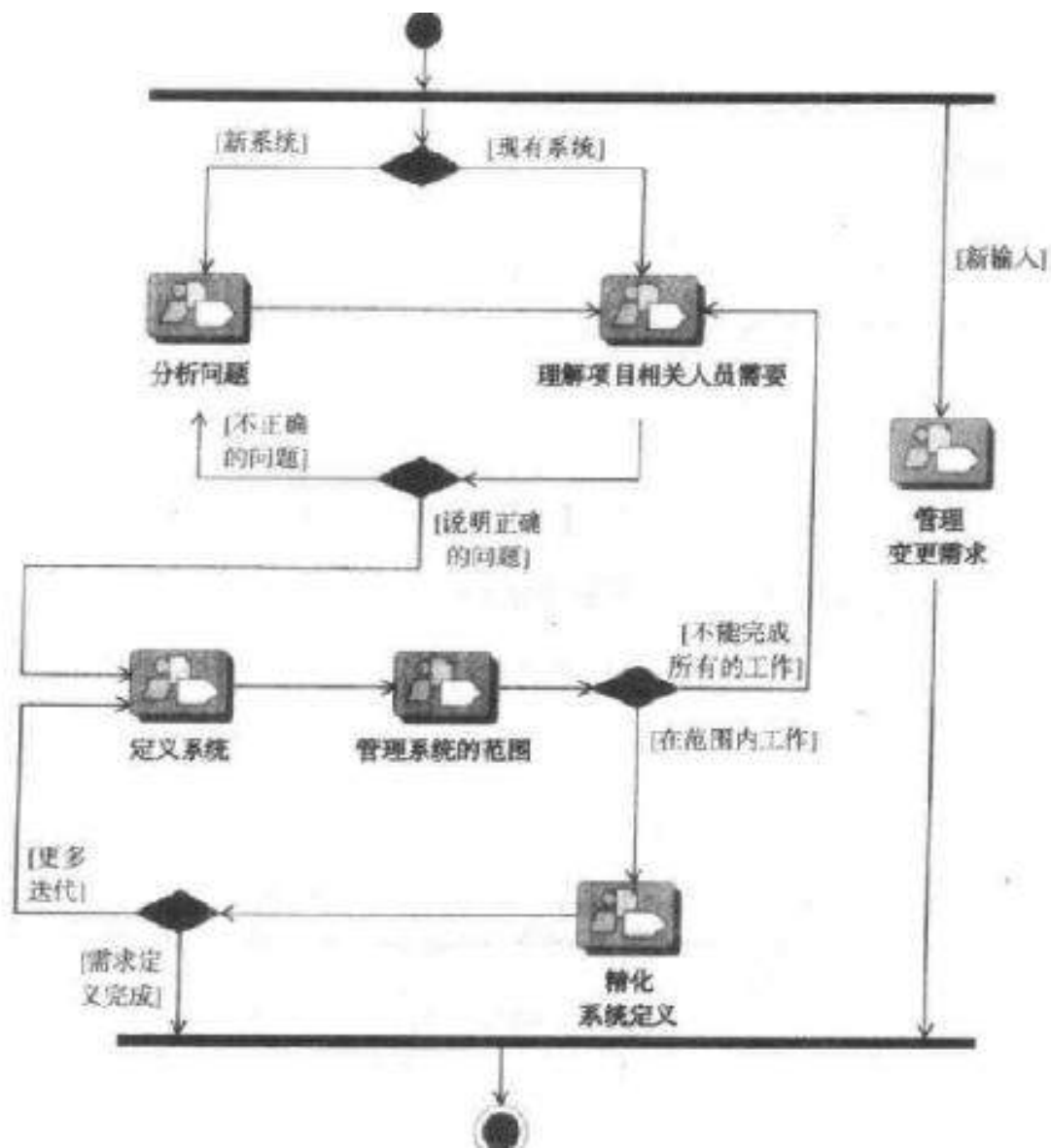
不断成长的信息集



M: 管理集
R: 需求集
D: 设计集
I: 实现集
D: 实施集

3.5 workflows

- workflow描述能够产生有用成果的有重要意义的活动序列并表示出worker之间的交互作用



一个需求工作流的示例

核心 workflows

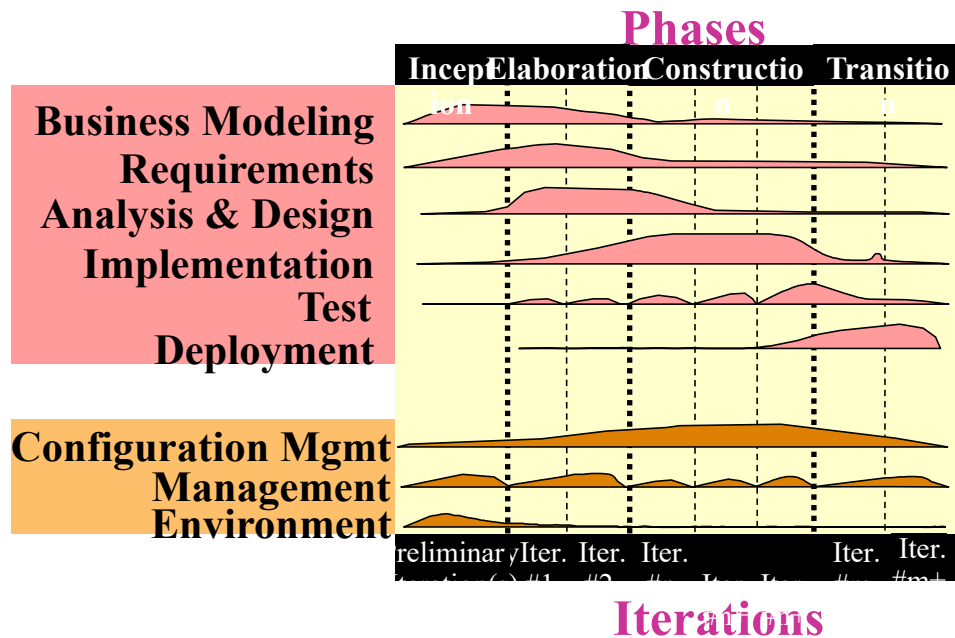
- **RUP**将所有工作人员和活动按涉及的领域进行逻辑分组, 划分为九个核心 workflows

- 核心过程 workflows

- 业务建模、需求、分析和设计、实现、测试、实施

- 核心支持 workflows

- 项目管理、配置和变更管理、环境



3.6 附加过程元素

- 指南(guideline)
- 模板(template)
- 工具指南(tool mentor)
- 概念

指南

■ 工作指南

- 评审工作指南
- 用例研讨会工作指南

■ 制品指南

- 建模指南
- 编程指南
- 用户接口指南

模板、工具指南和概念

■ 模板

- 用于制造出相应的制品
- 微软的**word**模板，用于写文档和一些报告
- 微软的**Project**模板，用于项目计划
- ...

■ 工具指南

- 介绍如何使用特定软件工具来完成每个步骤

■ 概念

- 一些关键概念：迭代、阶段、制品、风险、性能测试

3.7 过程框架

- 在静态结构中，**RUP**建立了过程框架(**framework**)。工作人员、制品、活动指南、概念和工具指南都是可以增加或替代的元素，从而满足开发团队的需要。

小结

- **RUP**建立在三个基础实体上：工作人员、活动和制品
- 在产出有价值成果的工作流序列中，工作流与活动、工作人员密切关联
- 指南、模板和工具指南给实践者提供了详细的指导，是对过程描述的有益补充
- **RUP**是过程框架，允许对静态结构进行配置



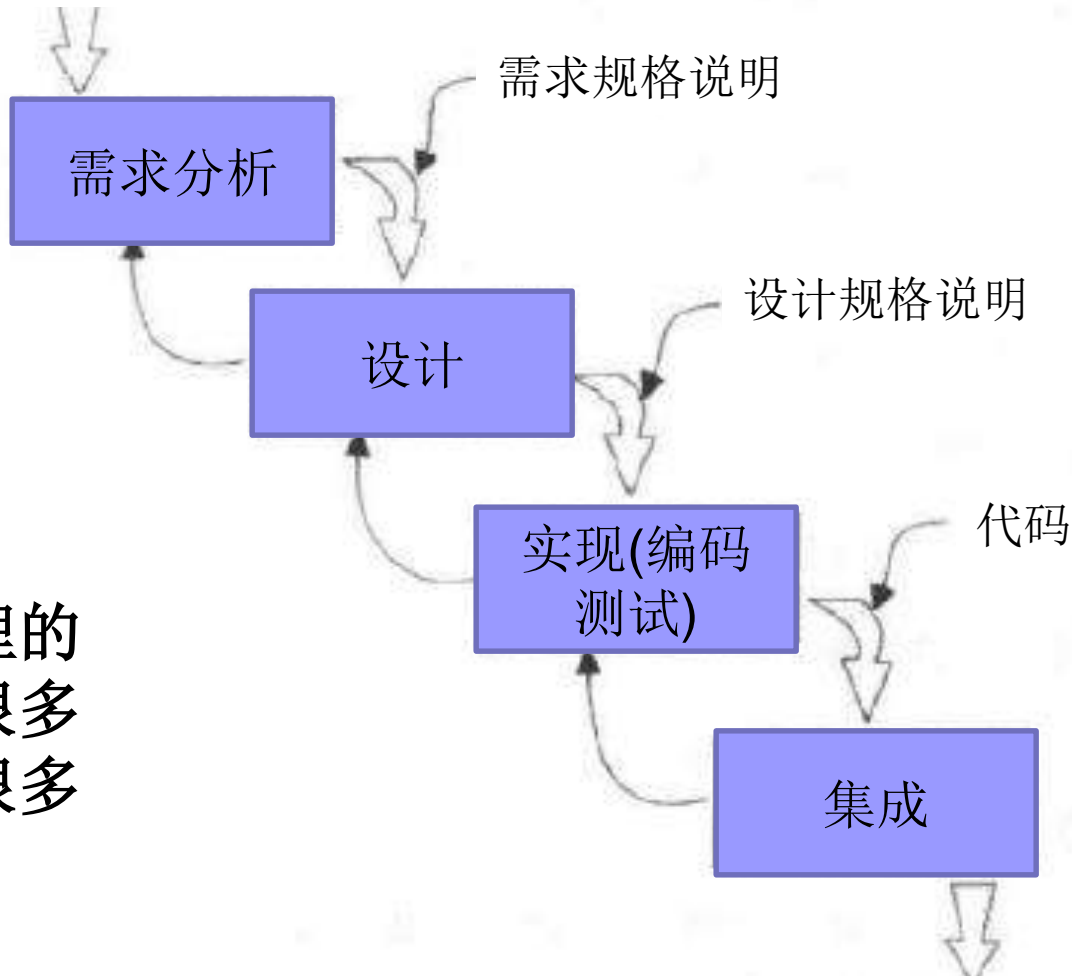
第四章 动态结构： 迭代开发

目录

- 4.1 顺序开发过程
- 4.2 克服困难的方法：迭代式开发
- 4.3 通过阶段和里程碑对项目进行控制
- 4.4 生命周期中焦点的转移
- 4.5 阶段重访
- 4.6 迭代方法的好处
- 4.7 小结

4.1 顺序开发过程

用户的需求



一个看起来很合理的开发过程，却被很多人指责其中存在很多问题与失败之处。

顺序过程运用失败的原因

- 错误的假设
- 软件开发的语境与其它工程规则不同
- 没有考虑到一些人为因素
- 开发环境的承担能力低于人们的预期
- 当前仍停留在软件工程的探索阶段

顺序过程导致软件开发失败的两个致命错误假设

- 需求是固定的
 - 用户会改变
 - 问题会改变
 - 技术基础会改变
 - 市场会改变
 - 不可能得到足够详细和精确的需求
- 在进行实际开发之前做出正确的设计

顺序过程不愿改变需求和无法关注最终产品的原因

■ 在开发周期长的项目中

- 顺序过程中的每一步骤(除了最后一步)的目标是产生一个中间制品(通常是一些文档)。
- 当这些中间制品准备作为下一个步骤的初始材料时，要求是确定的、被认可的、冻结了的。
- 对中间制品的过分重视导致缺乏对以前各阶段工作的反馈，使得工作人员无法系统地考虑。

顺序过程的适用项目

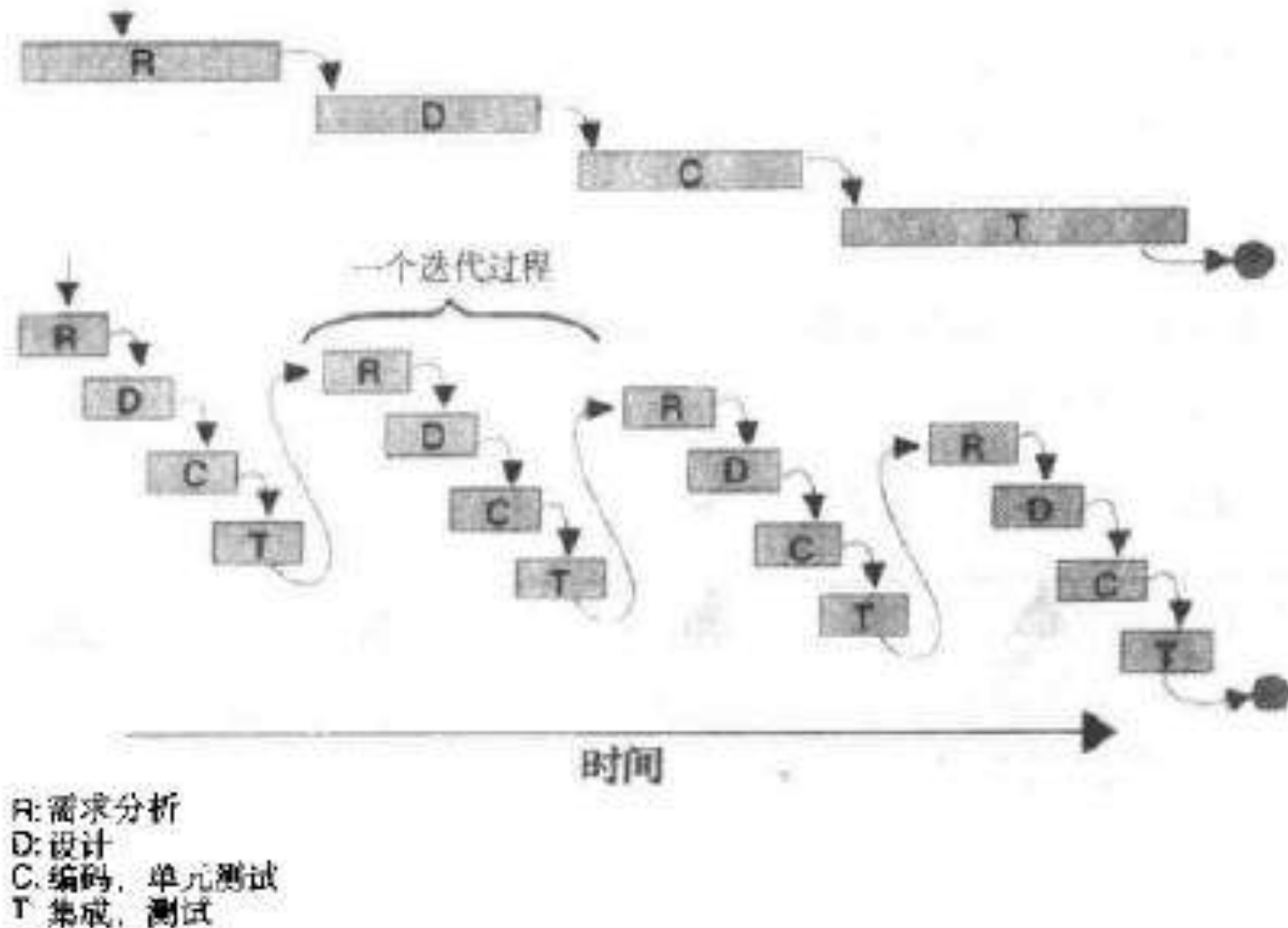
- 为期几星期到几个月的小型项目
- 能非常清楚地预测事情发展的项目
- 能比较透彻地了解所有困难方面的项目
- 没有新意的项目

假如在项目中要用到新技术、新开发工具或起用新人、顺序过程不能给你任何进行学习或自我提高的空间

顺序过程对基于规模的计划 和基于时间的计划

- 基于规模的计划：用**9**个月的时间做出所有我们想做的事
- 基于时间的计划：在前**3**个月做计划表中的前**3**件事，接下来的**3**个月中做后两件事，以此类推。
- 顺序过程不适用于基于时间的计划，而适用于基于规模的计划

4.2 克服困难的方法：迭代式开发



迭代技术的问题

- 迭代技术容易说明，但不容易做到：
 - 如何将各个迭代阶段的成果聚合成一个产品
 - 如何避免在混乱中开始一个周期
 - 每个迭代开发周期中你要选择做什么
 - 每个周期中应考虑哪些需求以及有什么样的风险
 - 迭代方法如何解决我们以前提出的主要问题
- 本章剩下的部分及第7章中给出答案

4.3 通过阶段和里程碑对项目进行控制

■ 里程碑(milestone)

- 是指开发过程中决定继续、取消还是改变迭代过程的控制点。

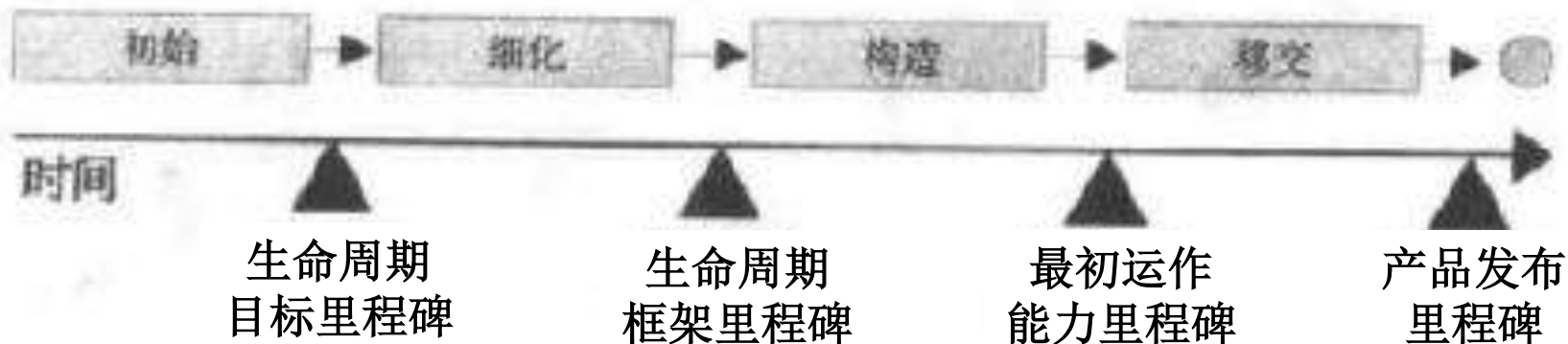
■ 迭代开发次序

- 要根据特定的短期目标来分割和组织。

■ 项目进展

- 要根据用例完成的个数、特性实现的个数、测试用例通过的个数、性能需求是否令客户满意及风险是否降低来度量。

- 迭代过程一般分为四个阶段：初始、细化、构造和移交，简称为I,E,C和T。每个阶段以一个重要的里程碑结束。



四个阶段的任务(一)

■ 初始(Inception)阶段

- 确定最终产品的构想及其业务用例、并定义项目范围
- 初始阶段以生命周期目标(LCO)里程碑为结束点

■ 细化(Elaboration)阶段

- 计划出必须完成的活动和需要的资源；详细说明产品特性并设计架构
- 细化阶段以生命周期构架(LCA)里程碑为结束点

四个阶段的任务(二)

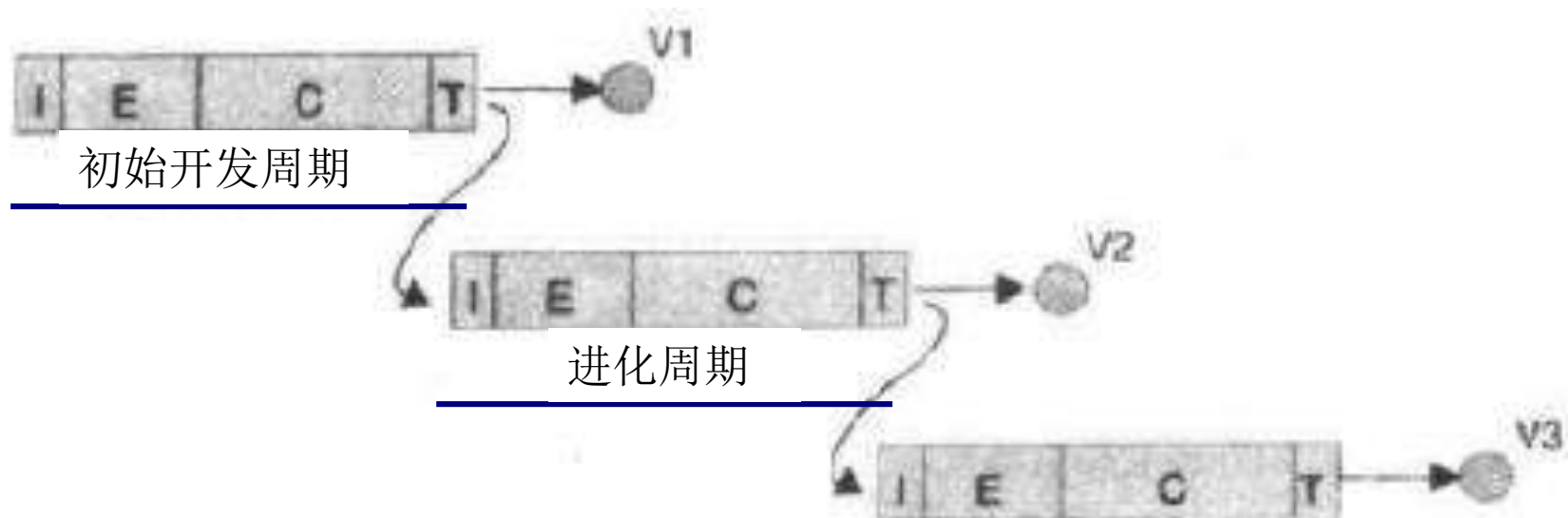
■ 构造(Construction)阶段

- 构造整个产品，逐步完善视图、构架和计划，直到产品(完整的构想)已完全准备好交付给用户。
- 构造阶段以最初运作能力(IOC)里程碑为结束点。

■ 移交(Transition)阶段

- 移交产品给用户，包括制造、交付、培训、支持及维护产品，直至用户满意。
- 移交阶段以产品发布版本里程碑为结束点，这也是整个周期的结束点。

- **I,E,C,T**这四个阶段构成了开发周期，当周期结束时产生一代软件产品。
- 软件产品重复**IECT**这个过程，从而发展成为下一代产品。
- 在不同的周期产品开发的重点是不同的。



周期是可以重叠的

- 一个产品经历了几个进化周期后，新一代的产品就产生了。
- 用户需求的升级及改变、基础技术的变化及竞争都可以激发进化周期。
- 周期与周期之间是可以存在时间重叠的。一个周期的初始与细化阶段与上一个周期的移交阶段可能是同时进行的。

- 每个阶段所经历的时间不一定是等同的
- 每个阶段的目标和里程碑是最重要的



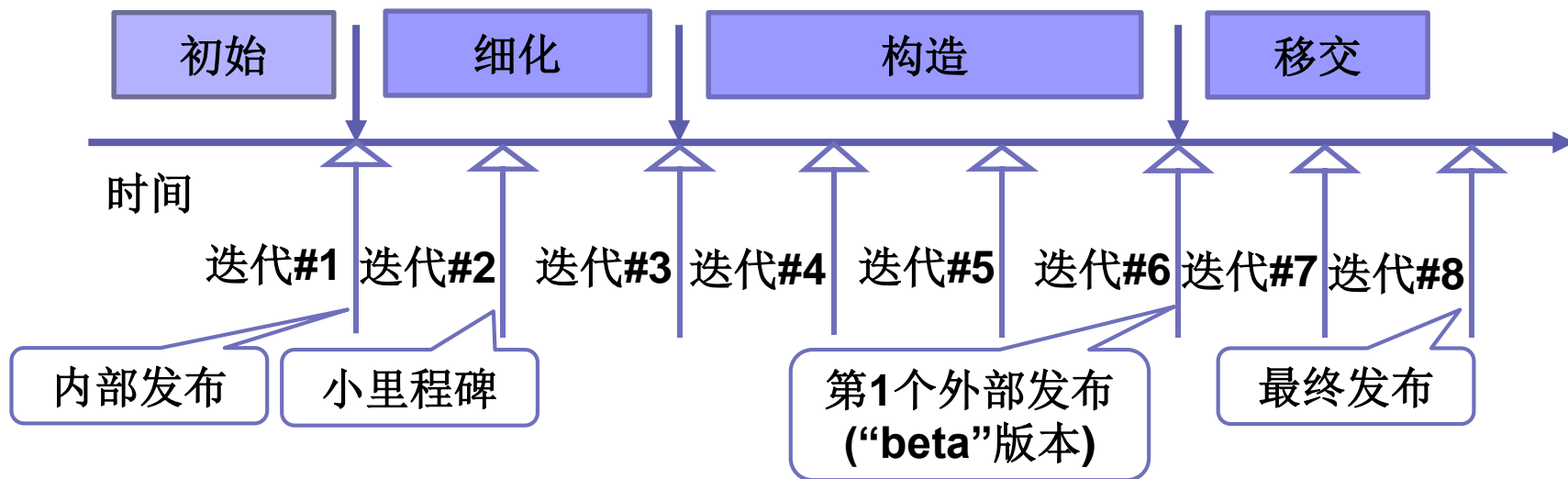
典型的初始开发周期时间线

一个为期两年的项目的阶段分配

- **2.5个月的初始阶段**
- **7个月的细化阶段**
- **12个月的构造阶段**
- **2.5个月的移交阶段**

阶段与迭代

- 每个阶段都可以迭代地进行开发，每个阶段都包含了一个或几个迭代过程。

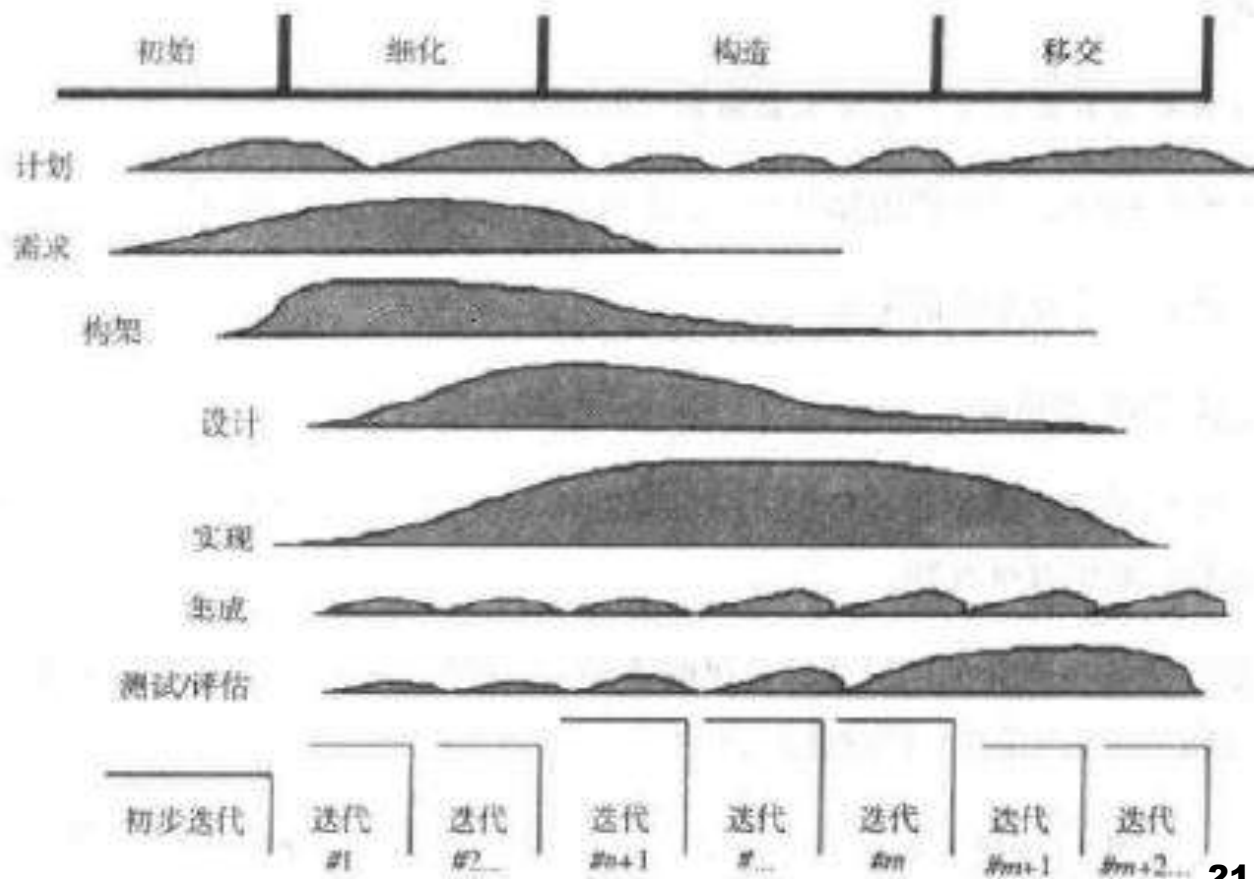


4.4 生命周期中焦点的转移

- 从一个迭代过程到另一个迭代过程，从一个阶段到另一个阶段，重点关注的活动是不同的
 - 初始阶段，焦点主要放在理解所有的需求并决定开发工作量的范围上。
 - 细化阶段，焦点放在需求上。一些软件设计和实施的目标是开发构架原型，用来缓解特定技术风险并学习运用特定的工具和技术。此构架原型将作为下一阶段的基线。

基线是一个经过审批通过的制品版本，可以构成进一步演化或开发的一致基础，并只能通过正规的过程(如改变和配置控制)进行变更。

- 构造阶段，焦点放在**产品的设计与实现**上。这时，可以在最初的构架原型基础上生产出第一个可以运行的产品。
- 移交阶段，焦点是确保系统**保持预期的质量水平；修复缺陷、培训用户、调整特性、增加遗漏掉的元素**。至此，交付最终产品。



4.5 阶段重访

- 本节将详细介绍阶段目标和每个重要里程碑的评价准则
 - 初始阶段
 - 里程碑：生命周期目标
 - 细化阶段
 - 里程碑：生命周期构架
 - 构造阶段
 - 里程碑：最初运作能力
 - 移交阶段
 - 里程碑：产品发布

初始阶段

- 主要任务
- 主要目标
- 主要活动
- 成果和制品

■ 主要任务：

- 令所有的项目相关人员对生命周期目标达成一致意见。

■ 主要目标：

- 确定项目的软件范围和边界条件，包括一个可操作的概念、可接受的准则和什么是产品目标、以及什么不是产品目标的详尽描述
- 识别系统的关键用例——即主要行为情景
- 展示或者示范至少一个针对主要情景的候选构架
- 估计整个项目需要的费用和时间表，并对下一阶段给出评估
- 评估风险(不确定性的来源)

■ 主要活动

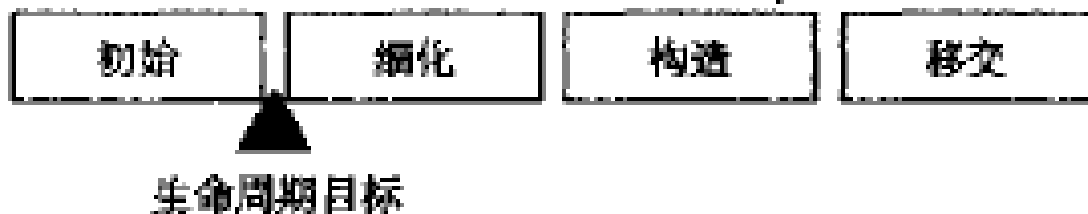
- 系统化地明确表述项目的范围——即捕获项目的语境、需求及限制。
- 计划并准备业务用例，评价风险管理、人员配备、项目计划，权衡成本、进度和利润。
- 合成一个候选构架，评价有关设计因素，评估制作/买进/重用的对策，以估算成本、进度和需要的资源

■ 成果和制品

- 一个构想文档——即关于项目核心需求、关键特性和主要限制的构想
- 一个关于用例模型的调查，包括所有在此阶段可以确定的用例和参与者
- 初期的项目术语
- 一个初始的业务用例，包括
 - 业务环境
 - 关于是否成功的评价标准(项目收入、市场认识等)
 - 经济预测
- 一个早期的风险评估
- 一个可以显示阶段和迭代的项目计划

- 初始阶段可能还会产生以下制品
 - 完成**10%-20%**的用例模型
 - 一个比术语表更详细的领域模型
 - 一个业务模型
 - 一个初步的开发案例描述，用于详细说明将用到的过程
 - 一个或几个原型

生命周期目标里程碑



- 生命周期目标里程碑评价初始阶段的准则如下：
 - 是否定义了项目范围，估计了成本和制定了进度安排。
 - 评估用例对需求的理解是否正确
 - 评估成本、进度安排、优先权、风险和开发过程的可信度
 - 实际成本与计划成本的对比

细化阶段

- 主要任务
- 主要目标
- 主要活动
- 成果和制品

细化阶段在四个阶段中最关键：项目从不稳定的、灵活的低风险运作阶段进入到高投入、具有巨大的惯性的高风险运作阶段。

■ 主要任务

- 分析问题领域、建立合理的构架基础、确定项目计划、评价项目最有可能出现的风险因素。

■ 主要目标

- 以实际所能达到的最快速度定义、确认构架，并将其基线化
- 设置构想的基线
- 为构造阶段的高可信度计划设定基线
- 考虑构想的基线的合理性，即可以在合理的时间范围内以合理的费用来完成

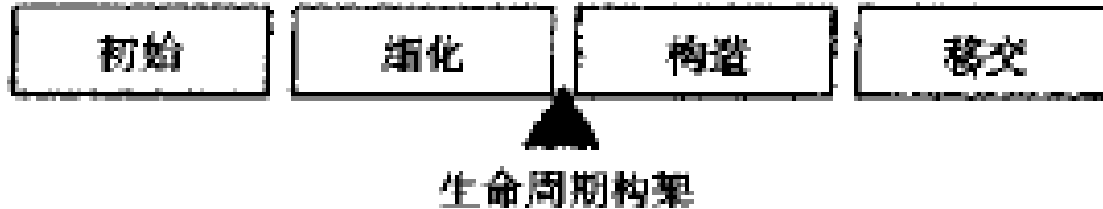
■ 主要活动

- 细化构想，完成和明确大部分的关键用例，这些用例将驱动做出最终构架和决定计划。
- 细化过程、基础设施和开发环境，过程、工具和自动支持也都各就各位。
- 细化构架并选择组件。

■ 成果和制品

- 用例模型至少完成**80%**，包括定义了在用例模型调查中识别出的所有用例、参与者和大部分用例描述。
- 补充需求分析，包括非功能性需求
- 一个软件构架描述
- 一个可执行的构架原型
- 一个修改过的风险清单和业务用例
- 一个整个项目的开发计划，计划中显示了迭代过程和每个迭代过程的评价准则。
- 一个已更新的开发案例，其中详细描述了将要用到的过程。
- 一个初步的用户手册(可选的)。

生命周期构架里程碑



- 生命周期构架里程碑检验细化的系统目标和范围、构架的选择以及主要风险的解决方案。

■ 评价准则回答以下问题：

- ☐ 产品构想是否稳定？
- ☐ 构架是否稳定？
- ☐ 可执行的演示中，主要风险因素是否已经确实处理并解决？
- ☐ 构造阶段计划是否足够详细精确？估计是否建立在可信的基础上？
- ☐ 后继开发人员是否会认可当前的构想？
- ☐ 实际的资源消耗相对于计划消耗来说是否可以接受

构造阶段

- 主要任务
- 主要目标
- 主要活动
- 成果和制品

■ 主要任务

- 所有保留下来的构件和应用程序特征将被开发并集成以形成产品，而后所有的特征将被彻底测试。

■ 主要目标

- 优化资源和避免不必要的浪费和返工，降低开发成本。
- 以实际最快的速度提高产品质量
- 以实际最快的速度生产一个可用的版本(beta)

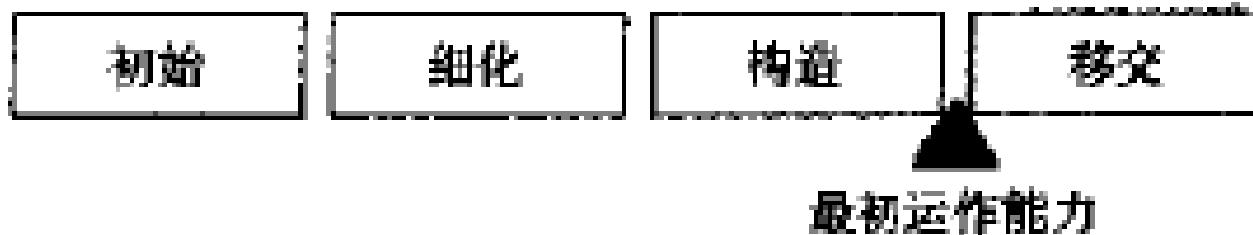
■ 主要活动

- 资源管理、资源控制和过程优化
- 完成构件开发并根据已定义的评价准则进行测试
- 针对根据构想制定的接受准则对发布的产品进行评估

■ 成果和制品

- 在适当的平台上集成的软件产品
- 用户手册
- 对当前版本的描述

最初运作能力里程碑



- 最初运作能力里程碑要明确软件、场地、用户是否都准备好正常运行而没有将项目暴露在高风险下，这个版本通常被称为**beta**版本。
- 评价准则回答以下问题：
 - 这个产品版本是否足够成熟稳定，可在用户群中实施？
 - 是否所有项目相关人员都准备好将产品向用户群推广？
 - 实际的资源消耗相对于计划消耗是否可以被接受？

移交阶段

- 主要任务
- 主要目标
- 主要活动

■ 主要任务

- 将软件产品移交给用户群，具体包括：
 - **Beta**测试，确认系统与用户的期望是否一致
 - 平行操作项目将要替代的遗留系统(选作)
 - 转换运行的数据库(选作)
 - 培训用户和维护人员
 - 产品的首次展示

■ 主要目标

- 达到用户自我可支持的程度
- 完成实施基线，该基线满足构想的评价准则
- 以实际可能的最快速度和最高效益达到最终的产品基线

■ 主要活动

- 完成实施工作：结尾工作、商业包装和生产、销售展示以及训练专业人员
- 调整活动，包括修复缺陷以及为了提高性能和可使用性而作的添加
- 根据产品构想的接受准则评估实施基线

产品发布里程碑

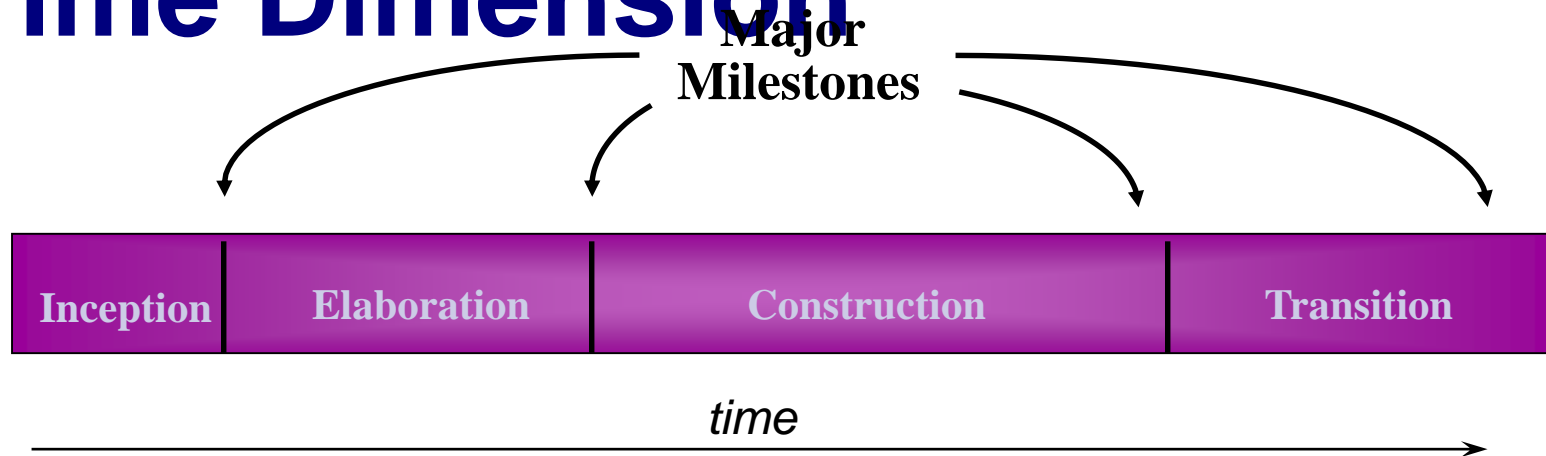


- 产品发布里程碑判断本周期目标是否达到，并决定是否要进入下一个新的开发周期。
- 有些情况下，这个里程碑会与下一个周期的初始阶段的结束点同时发生。
- 评价准则回答以下问题：
 - 用户是否满意？
 - 实际的资源消耗相对于计划消耗是否可以被接受

总结与归纳

- 一个周期的四个阶段
- 初始阶段
- 细化阶段
- 构造阶段
- 移交阶段
- 资源需要

Phases and Iterations - The Time Dimension



RUP有四个阶段:

- 初始 – 定义项目范围
- 精化(细化) – 制定项目计划、确定项目特性、基线化体系结构
- 构建(构造) – 项目编码开发
- 移交 – 将项目移交给最终用户

Inception Phase

□ Purpose

- 为开发一个新系统或者变更一个旧系统建立一个商业用例(**the business case**)
- 确定项目范围(**the project scope**)

□ Outcome

- 项目需求总览
 - 10-20%的初始用例模型和领域模型
- 初始商业用例
 - 成功结项标准 (e.g., 收益预计)
 - 初始风险评估
 - 所需资源评估

□ Lifecycle Objectives Milestone(生命周期目标里程碑)⁴⁵

Elaboration Phase

□ Purpose

- 分析问题领域
- 搭建一个稳定的体系结构平台
- 确定项目中的最高风险要素
- 制定一个用于指导项目怎样完成的综合性计划

□ Outcome

- 80%的初始用例模型和领域模型
- 一个可执行的架构体系和相应的技术文档
- 一个被修订的商业用例，包括修订风险评估
- 一个覆盖整个项目的开发计划

□ Lifecycle Architecture Milestone(生命周期架构里程碑)

Construction Phase

□ Purpose

- 增量式开发一个准备移交给用户的完整的软件产品

□ Products

- 全部的用例和设计模型
- 增量式发布各个功能的可执行版本
- 用户手册
- 部署文档
- 每次迭代的评估标准
- 当前版本的产品说明书，包括质检结果
- 变更后的开发计划

□ Initial Operational Capability Milestone(初始功能里程碑)

Transition Phase

□ Purpose

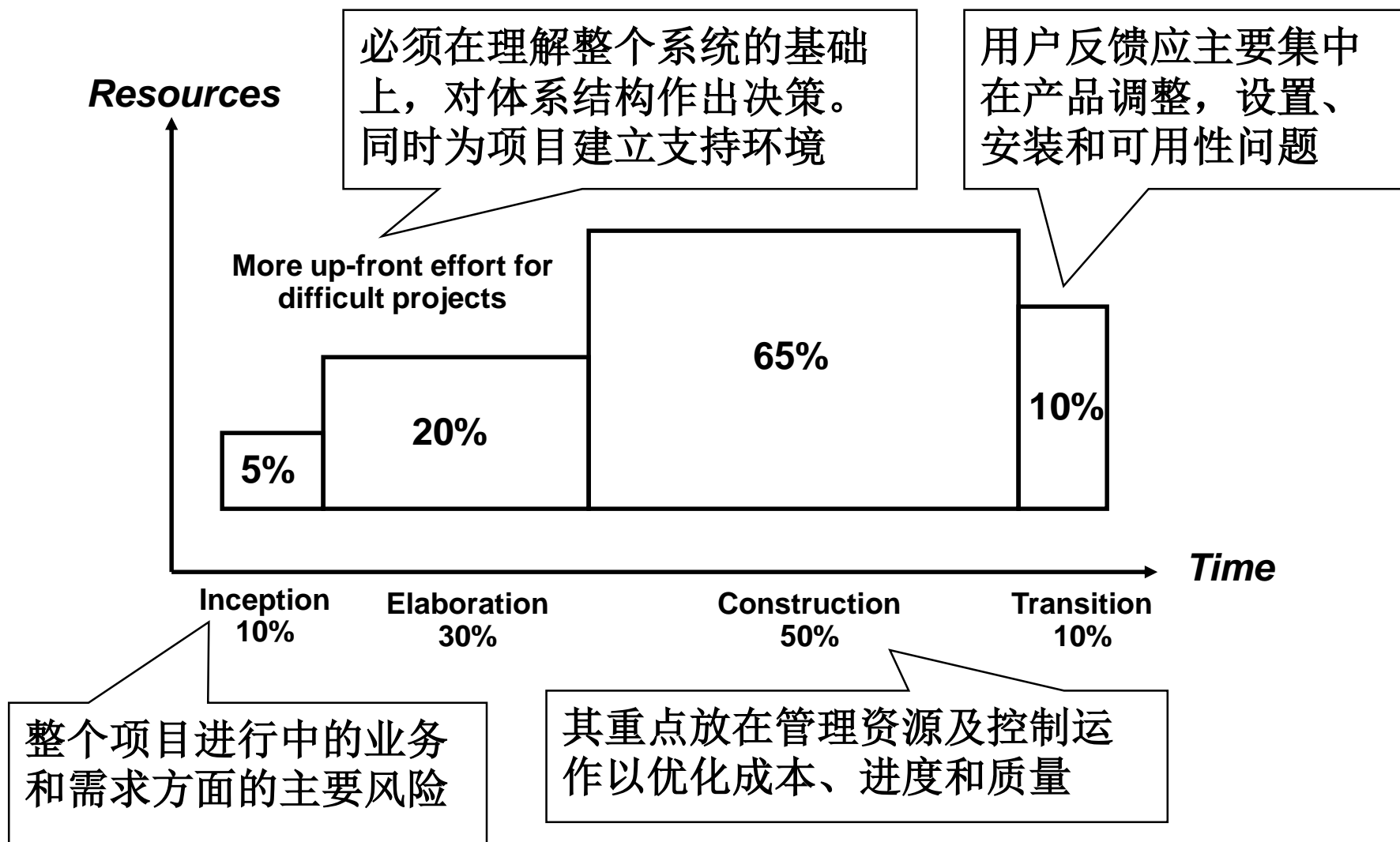
- 将软件产品交付给用户

□ Products

- 可执行的发布版本(releases)
- 变更后的系统模型
- 每次迭代的评估标准
- 当前版本的产品说明书，包括质检结果
- 变更后的用户使用手册
- 变更后的部署文档
- 项目总结分析

□ Product Release Milestone(产品发布里程碑)

Resource Needs



4.6 迭代方法的好处

- 与传统的瀑布模式相比较，迭代方法有以下优点：
 - 风险在早期被降低
 - 易管理变更
 - 提高了重用的程度
 - 在整个过程中项目群组可以不断地进行学习
 - 产品整体质量提高

- **缓解风险：**在众多风险当中，有两种风险可以通过迭代方法在早期降低
 - 集成风险
 - 构架风险
- **适应变更：**迭代过程可以更好地适应变更
 - 需求的变更
 - 策略的变更
 - 技术的变更
- 在工作过程中学习
- 增加重用机会
- 更好的整体品质

4.7 小结

- 顺序过程即瀑布式开发模式对于没有多大风险、并且使用众所周知的技术和领域的小项目非常适用，但是不适用于开发周期长、新意大、风险高的项目。
- 迭代过程将开发周期分解成连续的迭代周期。每个迭代过程看起来好像一个小的瀑布式模型，其中包括了需求分析、设计、实现和评估等各项目活动

- 为了控制整个项目并明确每个迭代过程适当的焦点，开发周期被分成四个连续的阶段，即初始、细化、构造和移交阶段，其中每个阶段又划分成若干个迭代过程。
- 迭代方法可以适应需求和实现策略中的变更，可以使我们尽早地面对并缓解风险。开发团队还可以通过这种方法不断成长、学习和提高。迭代方法中注重真实的、明确的目标。



第5章 以构架为中心的过程



目录

- 5.1 模型的重要性
- 5.2 构架
- 5.3 构架的重要性
- 5.4 构架的定义
- 5.5 构架的表示
- 5.6 以构架为中心的过程
- 5.7 构架的目标
- 5.8 基于构架的开发
- 5.9 其他的构架概念
- 5.10 小结

5.1 模型的重要性

- **Rational**统一过程有很大一部分内容是关于建模的。
- 模型是现实的简化，它能帮助我们整体把握不易理解的大型复杂系统。
- 选择用什么模型和哪项技术来表示复杂系统对于我们考虑问题和确定解决方案有着至关重要的影响。
- 需要多重模型来表示软件开发的不同方面。

5.2 构架

- 构架就是当你去掉任何部分，就无法使其他人理解整个系统和解释它是如何工作的系统描述。
- 构架要让设计师、程序员、使用者和经理能完成以下任务：
 - 理解这个系统是做什么的
 - 理解系统是怎样工作的
 - 能够完成系统的一部分工作
 - 扩展系统
 - 对系统部分重用，从而建立另一系统。

5.3 构架的重要性

- 一个大型、复杂的软件系统需要一个构架设计师，以便开发人员向着一个共同的目标努力。
- 开发团队需要构架的原因如下：
 - 理解系统
 - 组织开发
 - 鼓励重用
 - 进化系统

■ 对于一个以构架为中心的开发组织，需要做到以下三件事：

□ 理解构架

为什么架构如此重要？我们从中得到什么好处？
应如何开发构架？

□ 构架的表示

确定一个统一的表示构架的形式，从而使构架具体化。这样就可以系统地就构架进行交流、评审和改进。

□ 构架的过程

如何建立并验证一个能够满足项目需求的构架？
谁来做这件事？这个任务的制品和质量属性是什么？

5.4 构架的定义

Rational统一过程按如下方式定义构架

- 构架要对以下四个方面作出决策：
 - 软件系统的组织
 - 选择构成系统的结构元素和它们之间的接口，以及当这些元素相互协作时所体现出的行为。
 - 如何组合这些元素，使它们逐渐成为更大的子系统。
 - 构架风格，它将指导系统组织及其元素、它们之间的接口、协作和构成。

对构架概念的解释(1)

- 软件构架不仅仅注重软件本身的结构和行为，还注重其他特性：使用性、功能性、性能、弹性、重用、可理解性、经济和技术限制及折中方案、美学等。
- 构架是设计的一部分，它决定了如何建立系统。但它不是全部的设计，它只涉及一些重要元素——那些对系统质量有着普遍的深远影响的元素。
- 构架是关于结构和组织的，但是也处理行为。

对构架概念的解释(2)


- 构架不仅仅关心系统内部，还着眼于系统的两个外部语境：操作语境(它的最终用户)和开发语境(开发系统的技术人员)。它不仅包含系统的技术问题，还包含经济和社会问题。
- 构架同时还重视一些“软”问题，如风格和美学。

5.5 构架的表示

- 不同参与者使用构架的目的不同，关心构架的方面也不同。因此，一个完整的构架是多维的，而不是平面的。
- 为了使不同的参与者能够交流、讨论和推断构架，必须以参与者能够理解的形式表示构架。
- 构架和构架的表示不是同一事物。

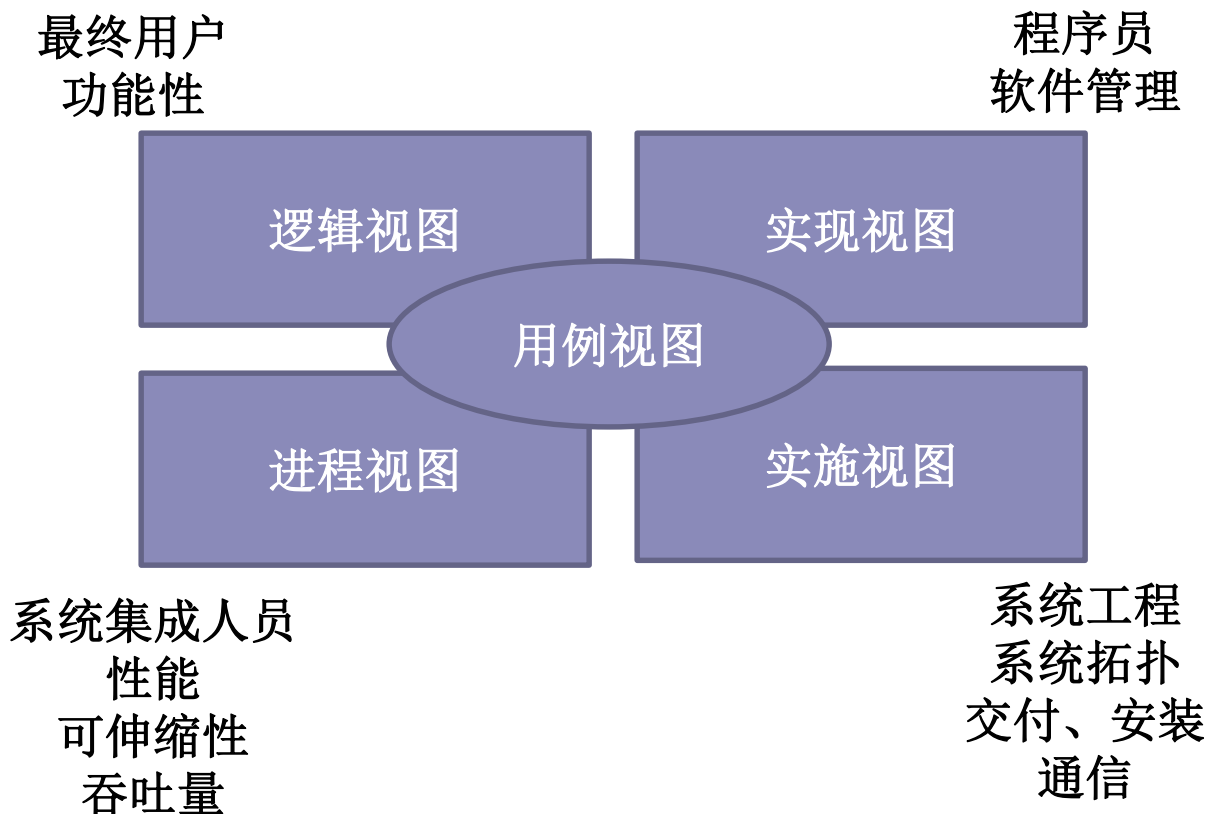
多重视图

- 在软件密集型系统构架中，不同目的需要不同的蓝图。即我们常说的构架视图(**architecture view**)。
- 一个构架视图是对于从某一视角看到的系统所做的简化描述，描述涵盖了系统的某一特定方面，省略了与此方面无关的实体。

- 
- 对于每一个视图，都要明确以下问题：
 - 视图出发点——从哪类项目相关人员的角度出发，关心哪一方面
 - 在视图中要捕获并表达哪些元素，以及它们之间的关系是什么
 - 用来构成视图的组织原则
 - 这个视图的元素以一种什么方式与其他视图的元素相连
 - 用什么样的过程创建视图最好

构架的4+1视图模型

- 逻辑视图
- 实现(开发)视图
- 进程视图
- 实施视图
- 用例视图



■ 模型和构架视图的区别：

- 模型是系统的完整表达，不同的需求要创建不同的模型来表述
- 构架视图只关注对构架重要的方面
- 不是所有的设计都是构架

■ 用例视图：包含几个关键情景或者用例。这个视图在初始和细化阶段用来驱动构架的挖掘和设计，之后将被用于验证不同的视图。

- 这几个为数不多的情景在软件构架文档中用来阐明其他视图是如何工作的。

- **逻辑视图**：着重描述系统的功能性需求，即这个系统能为它的最终用户做些什么。

- 逻辑视图是设计模型的抽象，确定了重要的设计包、子系统和类。

- **实现视图**：从打包、分层、配置管理(所有权、版本等)的角度描述了处于开发环境中的静态软件模型(源代码、数据文件、构件、可执行程序和其他伴随的制品)的组织结构。

- 实现视图着重讨论了如何使开发工作更简易，以及如何管理软件资产、重用、转包合同和现成的构件。

- **进程视图**：表述了系统在运行时的并发性——任务、线程、过程及它们之间的交互作用。
 - 进程视图讨论了并发性和并行性、系统启动和关机、容错性和对象分布等问题，处理了如死锁、应答时间、吞吐量以及功能和故障的隔绝问题等。它主要关心系统的可升级性。
- **实施视图**：展示了不同的可执行程序和其他运行时间构件是如何映射到底层平台或计算节点上的。
 - 实施视图融合了软件工程和系统工程，它讨论了如实施、安装系统和系统性能等问题。

- 模型是系统的完整表达，不同的需求要创建不同的模型来表述
- 构架视图只关注对构架重要的方面

- 构架视图好像是从不同模型中切下来的薄片，说明的只是这些模型中最重要的和最有意义的元素。

模式和视图的关系

模 型	构 架 视 图
设计模型	逻辑视图
设计模型*	进程视图
实现模型	实现视图
实施模型	实施视图
用例模型	用例视图

* 或一个复杂系统的过程模型

对构架重要的方面

- 是指系统的结构和性能、健壮性、可发展性和伸缩性方面
- 构架中对这些方面有重大影响的重要元素有：
 - 主要的类，尤其是那些为主要业务实体建模的类
 - 将行为赋予类的构架机制，如一致性机制和交流机制
 - 模式和框架
 - 层和子系统
 - 接口
 - 对于主要过程或线程的控制

构架不仅仅是一个蓝图

- 除了软件构架描述之外，与构架相关的最重要的制品就是构架原型。
- 这些原型做出了最重要的设计决定，足以用来验证构架，即测试和度量构架。
- 这些原型将贯穿构造阶段，并发展成为最终的系统。

5.6 以构架为中心的过程

- **Rational**统一过程定义了两个关于构架的主要制品：
 - 软件构架描述(**SAD**)，它描述了与项目有关的构架视图
 - 构架原型，它用于验证构架并充当开发系统其余部分的基线
- 这两个关键制品是其他三个制品的基础：
 - 设计指南，由所做的构架选择决定，反映了模式和习惯用语的使用
 - 在开发环境中基于实现视图的产品结构
 - 基于实现视图结构的开发团队结构

5.7 构架的目标

■ 智能控制

- 构架可以使得整个项目得到并保持智能控制，从而有效管理项目的复杂性以保持系统的完整性
- 一个复杂的系统不是各个部分的简单叠加，也不是独立的明智决定的集成。它必须有一个统一、聚合的结构，能够将各个部分系统化地组织起来。
- 同时还要提供精确的规则，以确保不要开发出一个过于复杂而使人们理解不了的系统。

■ 重用

- 构架明确了主要构件和它们之间的重要接口，从而可以考虑重用。它明确了系统中的公共部分，从而可以协助内部重用；它结合了现成的商用构件，从而也可以协助外部重用。
- 框架还可以协助更大规模的重用：在同一领域不同功能的系列产品中，构架本身可以进行重用。
- 构架师需要理解对于可缩放性的限制，明确类型相同的部分，定义变化点。

■ 开发的基础

- 构架为项目管理提供了基础。
- 制定的计划和人员配备沿着主要构件线（层和子系统）进行组织。构架组决定了最基本的结构，但并不发布这个决定。
- 整个开发工作被分配给各个开发组，每个组负责系统的一个或几个部分。不同的开发组要明确哪些接口可以由他们支配，并明确对这些接口能做的变更限制。
- 构架在细化阶段所做的工作为进一步开发提供了基础，包括设计指南、原则、风格、模式和重用机制。

5.8 基于构件的开发

- 基于构件的开发是指创建和实施由构件装配而成的软件密集型系统。
- 基于构件的开发主张应用各种零件而不是手工开发每个单个元素。
- 构件的定义必须足够广泛，要包含一个良好的构架的所有可能制品。

- 构件是一个系统中非常重要、独立而可替换的部分，在定义良好的构架语境中完成明确的功能。
- 一个构件要符合并提供一系列接口的物理实现。
- 构件和构架是两个相互缠绕的概念
 - 构架定义了构件、构件的接口和构件在多维中的交互作用。
 - 构件只有在与某一构架相联系时才存在。

从不同的角度看待构件

■ 运行构件

- 交付、安装之后可以运行，例如可执行程序、过程和动态链接库(DLL)

■ 开发构件

- 开发构件即实现子系统，具有高内聚合力和低外耦合度，可以被其他开发人员重用。

■ 业务构件

- 一系列联系紧密的运行构件(或开发构件)，执行业务级别的功能

5.9 其他的构架概念

- **构架风格**：限定构架中可能出现的形式，并给构架的统一性设定了某个确定的级别。例如：客户与服务、事件驱动风格等。
- **构架机制**：是用来给普遍问题提供普遍解决方案的一个类、一组类或一个模式。机制主要体现在构架的底层或中层。例如：数据库管理系统、事件广播系统。
- **构架模式**：讨论并提出一个普遍解决方案，用于解决设计过程中某种特定设计情况下重复发生的设计问题。模式记录了已经存在的而且经过验证的设计经验。例如 **MVC(模型视图控制器)**模式和对象请求代理模式。

5.10 小结

- 系统构架在**Rational**统一过程中是一个主要制品，可以在开发过程中概念化、构造、管理和进化系统
- 构架是一个复杂的概念，最好通过多重而协调一致的构架视图来表示
- 构架视图是一个模型的抽象，它关注于模型的结构和重要元素



第6章 用例驱动的过程



目录

- 6.1 定义
- 6.2 识别用例
- 6.3 用例的进化
- 6.4 用例的组织
- 6.5 在过程中使用用例
- 6.6 小结

概述

- 选择什么样的模型和选择什么样的技术来表述模型, 对于我们考虑问题和解决问题有着重大的影响.
- 问题模型与解决方案的模型要保持一致性.
- 本章将介绍用例、参与者、情景等概念, 同时还介绍如何在生命周期中将用例作为许多活动的驱动来使用, 并在模型之间传递信息和保持模型的一致性。

6.1 定义

- 6.1.1 用例和参与者
- 6.1.2 事件流
- 6.1.3 情景
- 6.1.4 用例模型

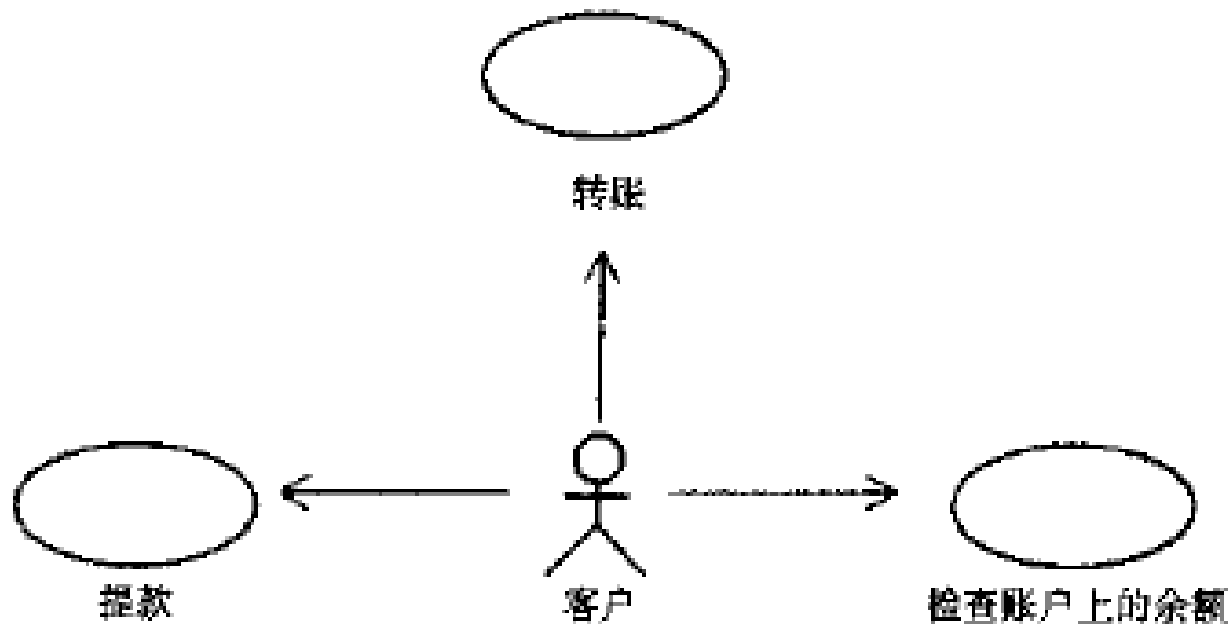
6.1.1 用例和参与者

- 用例(**use case**)是一个系统执行的**动作序列**，对特定**参与者**产生**有价值的可见结果**。
- **参与者(actor)**是系统之外与系统能产生交互作用的人或物
- **动作(action)**是一个计算程序或算法程序，在参与者给出系统信号或系统得到一个时间事件时被调用。动作是原子的，一个动作要么全部执行，要么根本不执行。

- **动作序列(a sequence of actions)**是指贯穿于系统的特定事件流。
- **有价值的可见结果(an observable result of value)**:动作序列一定要产生对系统的参与者有价值的可观测到的结果。
- 用例的定义描述了执行用例时系统要做些什么。一组用例就可以定义系统的功能，每一个用例代表了一个特定事件流。
- 用例的作用是从用户角度获取系统的功能，当客户和系统开发人员进行交流时，充当共同的语言。

用例和参与者的示例

- 银行客户可以通过自动取款机(ATM)提款、转帐或检查帐户上的余额。



6.1.2 事件流

- 事件流(**flow of events**)描述了参与者和系统之间的动作序列。
- 事件流用自然用语写成，或者用含有精确术语的短语写成。
- 一个关于事件流的示例： 《提款》 见下一页

1. 用例开始于客户将卡插入**ATM**机。系统读取并验证卡上的信息。
2. 系统提示输入个人标识号(**PIN**)。客户输入**PIN**。系统进行验证。
3. 系统询问客户需要哪种操作。客户选择“提款”。
4. 系统询问提款数量。客户输入要提取的金额数。
5. 系统询问账户种类。客户选择账户种类。
6. 系统与**ATM**网络交换信息，以验证账户**ID**、**PIN**以及剩余金额。
7. 系统提问客户是否要收据。这一步只有在可以用纸打印收据时才执行。
8. 系统提示用户取走卡。用户将卡取走。
9. 系统给出客户申请的现金。
10. 假如客户需要，系统打出收据，用例结束。

- 用例事件流最终要描述所有可能的过程，要考虑所有可相互替换的路线和不同的情景。
- 选择什么样的路径依赖于以下几点：
 - 参与者的输入(input from an actor)
 - 系统内部状况(the internal state of the system)
 - 超时或出错(time-outs and errors)

6.1.3 情景

- 情景(scenario)可以理解为是用例的实例
- 用例的实例是一个特定的事件流或一个特定的路径。
- 情景的作用
 - 在过程中使用情景的目的是提取并强调一个独特的贯穿于用例的动作序列或“线程”。在定义测试用例时尤其有用。
 - 在项目早期寻找用例时，最容易的方法是从一个特定的情景开始，而后将它扩展到更多的事件流，最后总结为成熟的用例。

6.1.4 用例模型

- 用例模型由整个系统或系统某一部分的所有用例及与这些用例交互的参与者组成，以此来描述整个系统的功能。
- 用例模型提供了系统预期功能模型和外部环境模型，可以将用例模型作为开发人员和客户之间的契约。
- **Rational**统一过程中使用用例图和活动图将用例模型可视化，包括用例之间可能的关系。

用例建模阶段是否考虑并发性等非功能需求

- 在用例建模阶段，假设所有情景都可以同时运行而不会产生任何问题。在设计阶段时，要确保能正确处理所有非功能性需求。

6.2 识别用例

- 思考方式1：研究系统提供给参与者的价值结果，并把产生这些价值结果的动作序列分组
- 思考方式2：认为用例实现了参与者的某一特定目标，而这个目标也正是系统要完成的。

6.3 用例的进化

- 在研究细节之前要先开发一个用例概要
- 用例概要不需要对事件流进行详尽地说明，只要相关人员的理解能够达成一致即可
- 在细化阶段的早期迭代过程中，只有少数用例(对构架非常重要的用例)在概括描述之后进行了详尽的说明
- 在细化阶段结束时，要完成对那些必要用例的详尽描述

6.4 用例的组织

- 包：相关的用例放在一个包中

- 用例之间关系：

- 包含

- 扩展

- 泛化/特化

不要过度使用这
三种关系

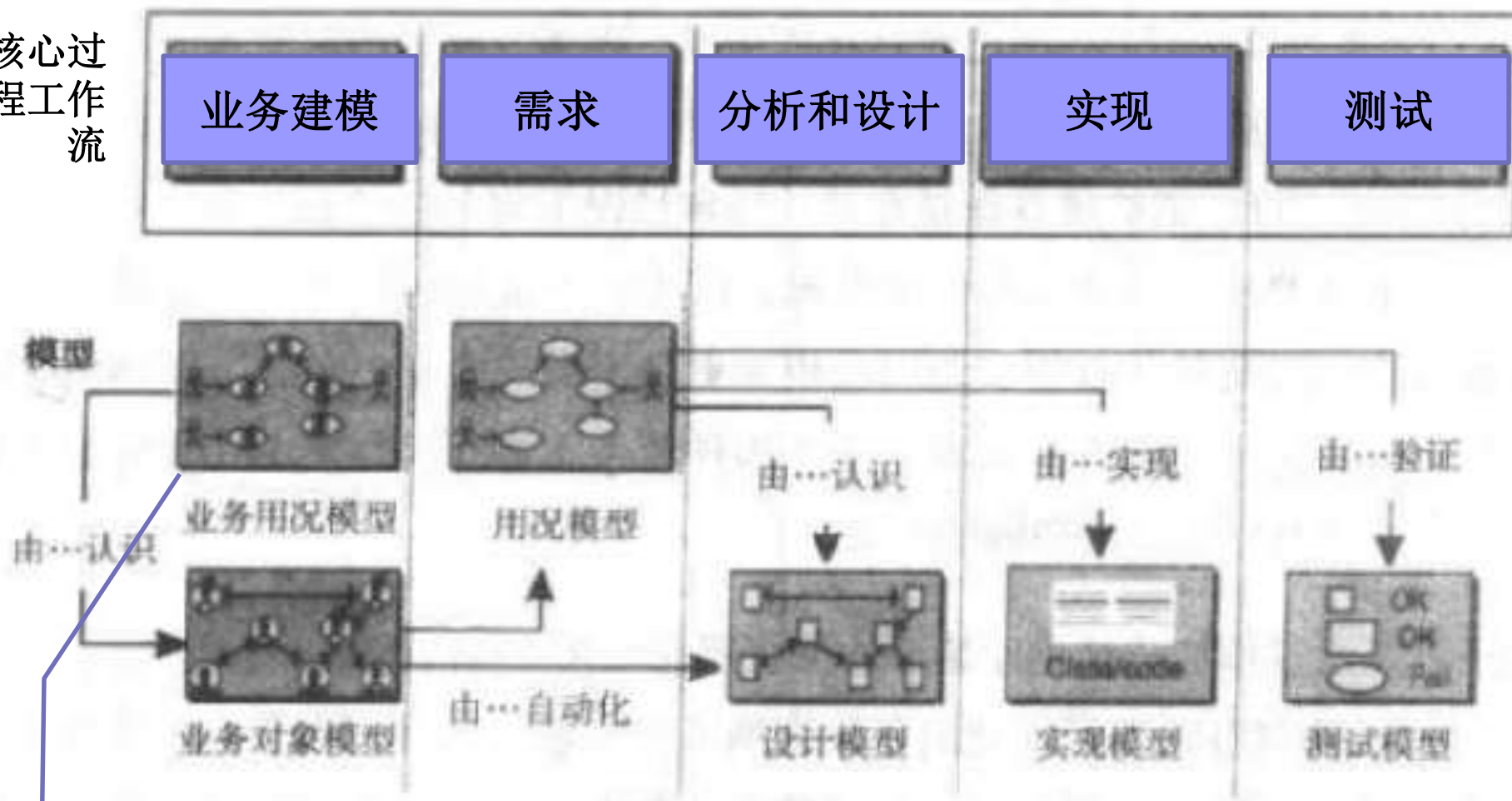
- 包含和扩展的区别：

- 包含关系中，基本事件流在不包含时是**不完整**的；扩展关系中，基本事件流在不进行扩展之前是**完整**的

6.5 在过程中使用用例

- 用例模型是需求工作流的结果
- 在分析和设计阶段，用例是将需求活动 and 设计活动连接起来的桥梁，在用例实现中充当基础。通过遍历用例，开发人员会找到对象和类。
- 在实现阶段，设计模型是实现规格说明。用例是设计模型的基础，我们根据设计类实现用例。在设计模型中实现用例可以理解系统的动态性，并确定在哪里来优化系统性能。
- 在测试阶段，用例构成了确定测试用例和测试规程的基础。每一个用例都用来验证系统。

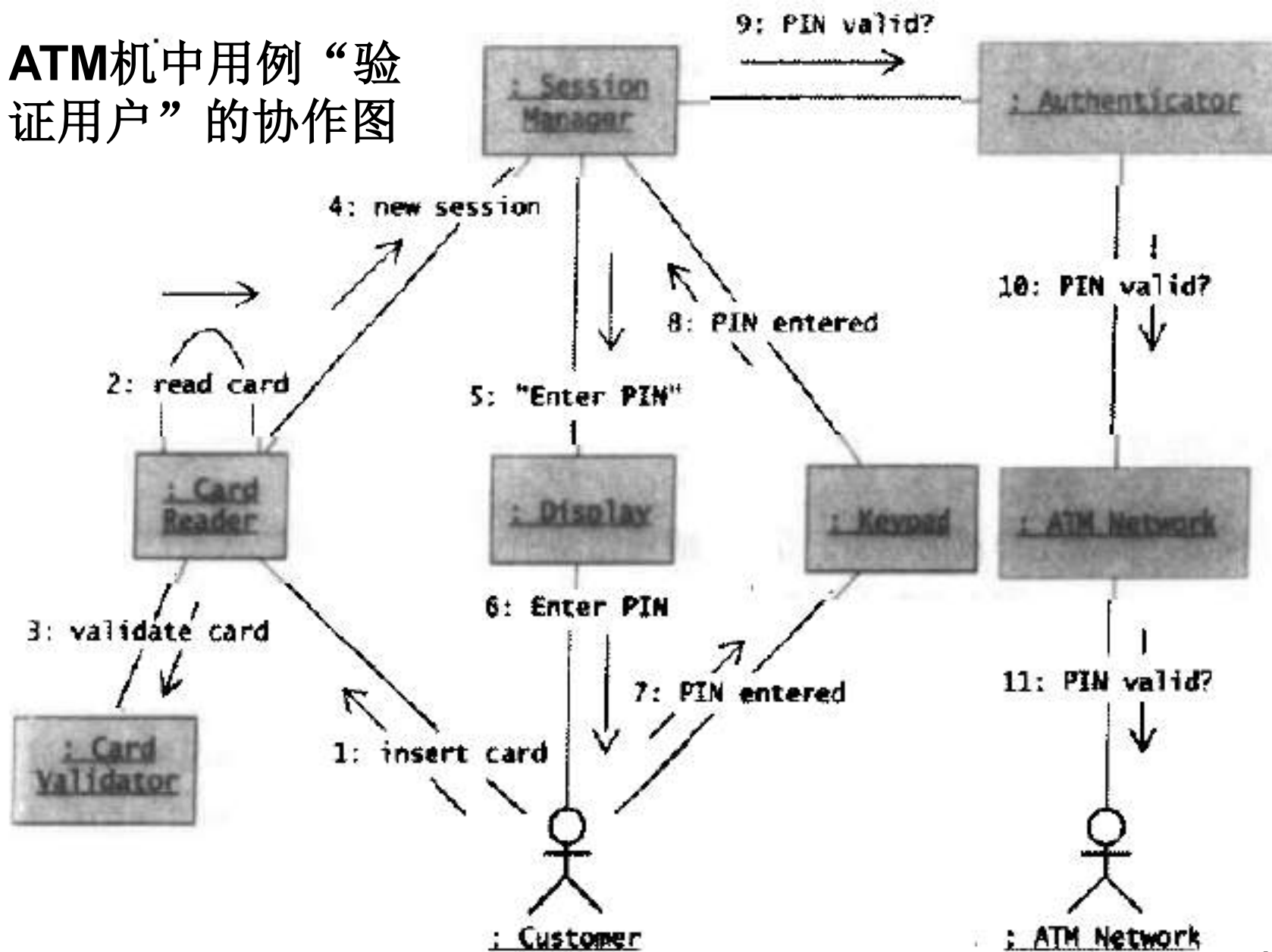
核心过
程工作
流



业务模型考虑的是整个业务而不仅仅是一个系统

用例与不同模型之间的关系

ATM机中用例“验证用户”的协作图



- 在实施阶段，用例包可以用来计划阶段性实施，还可以定义系统变量
- 用户界面的定义和原型设计也来源于以用例记事板形式出现的用例
- 在业务建模中也使用用例概念。但是业务模型考虑的是整个业务而不仅仅是一个系统(第8章)

6.6 小结

- 用例是表达系统功能性需求的方法
- 用例是用简洁的语言写成的，可以被每一个项目相关人员理解
- 用例可以协调不同模型的内容，使其同步
- 用例在整个开发过程中是作为一个单一整体进行管理的
- 用例可以用于业务建模，提供了系统开发的语境
- 用例模型将用例组织起来，揭示了用例之间的关系
- 情景是用例实例的描述

在Rational统一过程中，用例驱动了多项活动：

- 创建和验证设计模型
- 在测试模型中定义测试用例和测试规程
- 计划迭代
- 创建用户手册
- 实施系统



第7章 项目管理 workflow



目录

- 7.1 目的
- 7.2 计划迭代项目
- 7.3 风险的概念
- 7.4 度量元的概念
- 7.5 度量元的类型
- 7.6 工作人员和制品
- 7.7 workflow
- 7.8 制定一个迭代计划
- 7.9 小结

7.1 目的

- 软件项目管理是一门平衡竞争对象、管理风险和克服约束的艺术
- 软件项目管理最终目的是交付一个满足客户(付账人)和最终用户需求的产品
- 项目管理 workflow 有以下三个目的：
 - 为管理软件密集型项目提供框架
 - 为计划、执行、监督项目和分配人员提供实际的指南
 - 为管理风险提供框架

- 项目管理 workflow 不覆盖项目管理的所有方面
- 项目管理 workflow 不包含以下问题：
 - 管理人员：雇用、培训、指导
 - 管理预算：详细说明和分配
 - 管理供应商和客户之间的契约
- 项目管理 workflow 主要关注迭代开发过程的某些方面：
 - 制定一个整个生命周期的迭代项目计划或某个特定迭代的计划
 - 风险管理
 - 监督迭代项目和度量元的进展情况

7.2 计划迭代项目

■ 项目经理的困惑：

- 需要多少个迭代？
- 每个迭代要有多长？
- 如何确定一个迭代的内容和对象？
- 如何跟踪迭代的进展？

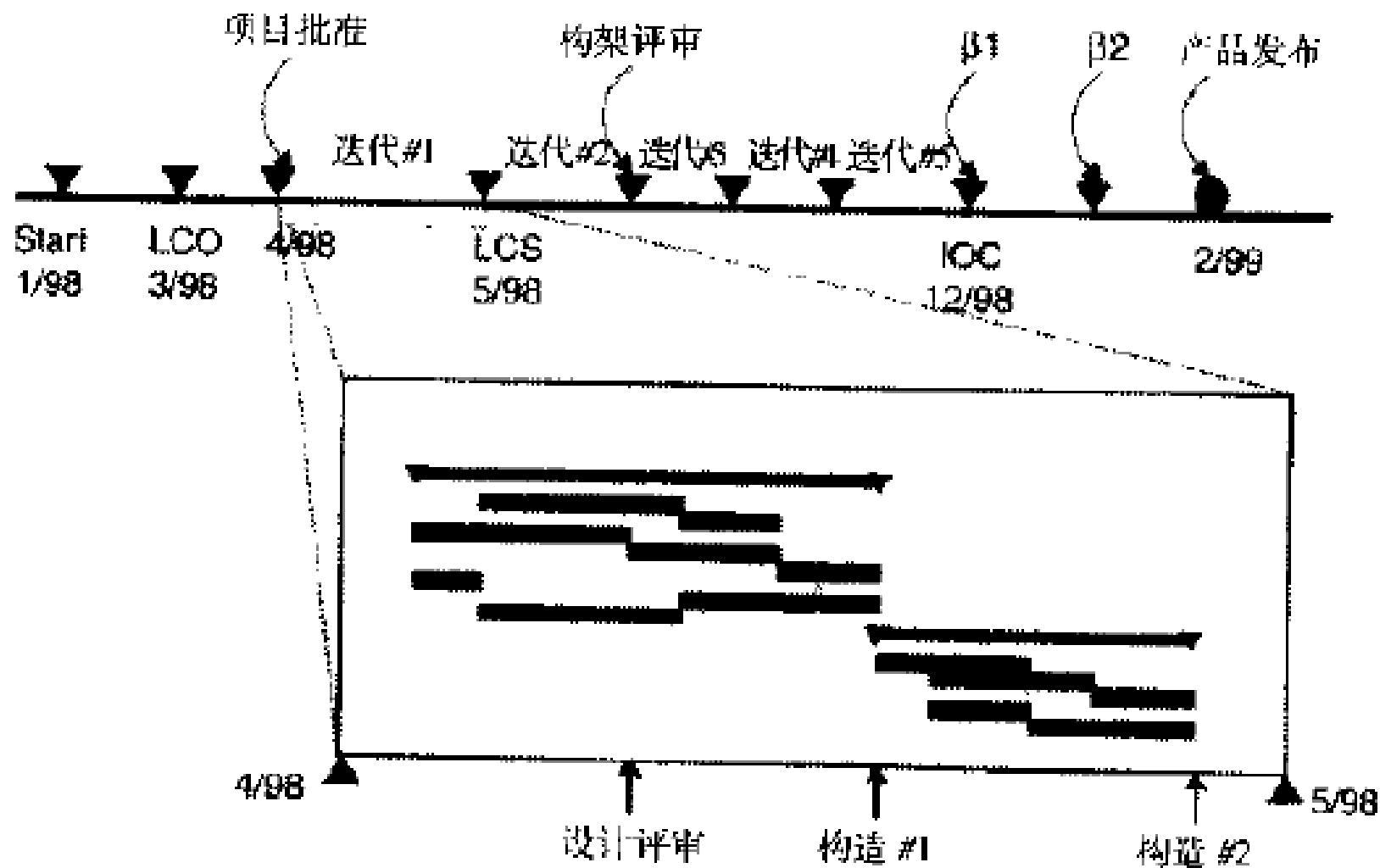
■ 项目计划的目标：

- 在一段时间内给一个团队中的人员分配任务和职责
- 监督计划进展状况，在项目展开过程中发现潜在的问题

计划的两个级别

- 一个粗粒度的计划：阶段计划
 - 阶段计划在初始阶段的早期完成，只在需要进行更新
- 一组细粒度的计划：迭代计划
 - 迭代计划仿佛是一个面向阶段计划的放大镜

粗粒度计划：阶段计划




细粒度计划：迭代计划

阶段计划

- 确定主要里程碑的日期
 - 生命周期目标(在初始阶段结束时设定项目范围并准备好足够的资金)
 - 生命周期构架(在细化阶段结束时完成构架)
 - 最初运作能力(在构造阶段结束时完成第一个**beta**版本)
 - 产品发布(在移交阶段和周期结束时完成相应的产品发布)
- 人员配备概括
- 确定次里程碑的日期： 预测每个迭代结束的时间和它的主要目标

迭代计划

- 一个项目通常有两个活跃的迭代计划
 - 当前迭代计划(为当前迭代制定的计划)，用来跟踪进展情况
 - 下一个迭代计划(为即将开始的迭代制定的计划)，针对当前迭代后半部分情况所做，在当前迭代结束时准备好
- 迭代计划中包括重要的日期，如主要部件完成时间、从其他开发组织中得到构件的时间以及主要评审时间

- 
- 迭代是动态的，当目标和战略改变，迭代就随之改变，所以根本不用花过多的时间做一个超越当前计划范围的详细计划。
 - 必须覆盖正确的时间范围，有一个正确的度量元，才能做出好的详尽的迭代计划。

7.3 风险的概念

- 软件开发过程主要关心软件开发中“已知”的方面。你只能精确地描述、计划、分配和评审你所知道的事情。
- 风险管理关注“未知”的方面。很多组织应用“否决风险”的模式工作：计划和评定整个过程，就好像已经知道了所有的可变因素

什么是风险

- 简单地说，风险就是未知和不确定
 - 成功则是指达到了所有人对项目期望的需求和约束
- 风险可以量化为直接风险和间接风险
 - 直接风险(**direct risk**):项目对其控制得很好的风险
 - 间接风险(**indirect risk**):项目对其几乎没有什么控制的风险
- 风险的属性：风险量级指示
 - 考虑发生的可能性和对项目的影响(严重性)
 - 共分为五个级别：最高、较高、适中、较低、最低

如何处理风险

■ 风险管理的关键思想

- 在决定如何处理风险之前，不能被动地等待风险的来临(或项目的失败)。对任何一个可预测的风险，必须提前采取行动

■ 处理风险有以下几种方法：

- 避免风险：重新组织项目，从而避免受到风险的影响
- 转移风险：重新组织项目，从而使某些人和某些事情可以忍受风险的存在(如客户、卖方、银行或其他因素)
- 接受风险：把风险当做一个偶然状况允许其存在。检测风险的征兆，决定风险发生时要做些什么。



■ 当接受风险时，需要做两件事：

- 缓解风险：采取直接的、提前的步骤减少风险出现的可能性和它带来的影响
- 制定应急计划：当风险成为一个实际问题时，应采取的行动步骤；换句话说，就是制定一个“后备计划”

7.4 度量元的概念

- 度量的主要目的是更好地控制项目从而管理好项目。
- 度量的作用：
 - 察看距离目标计划还有多远
 - 为项目工作量、项目质量和成本制定一个更好的计划
 - 明确变更带来的影响

7.5 度量元的类型

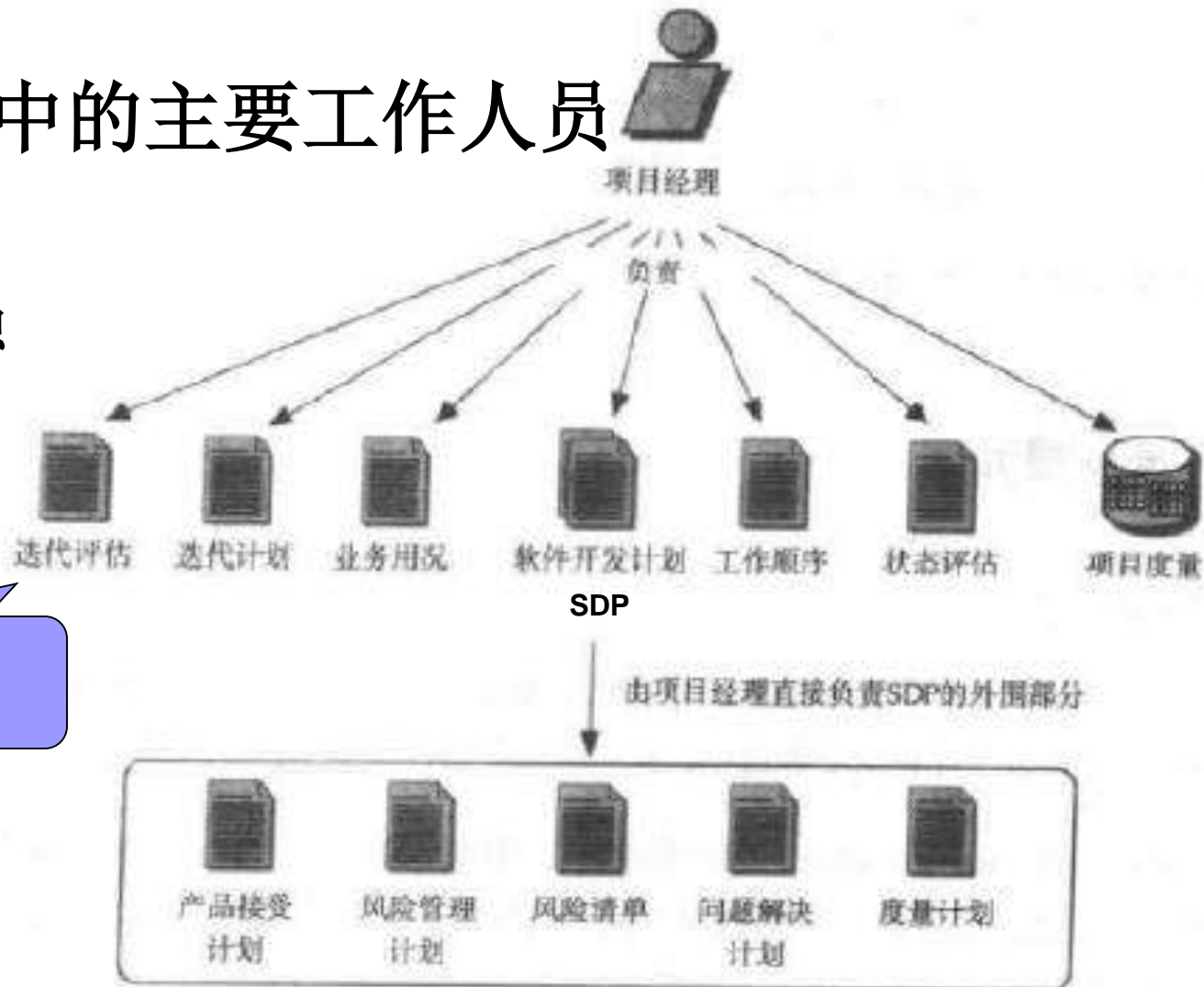
■ 度量元有两种类型

- 度量元：表示实体可度量的属性。例如，完成一个项目所需要的工作量是对项目规模的度量。为了计算这个度量元，需要将所有用在这个项目上的时间片登记相加
- 原始度量元：是一组数据，用来计算一个度量元。时间片登记就是原始度量元。它不能被独立解释。
- 每个度量元都是一个或多个原始度量元的集合，每个原始度量元都必须明确区分。

7.6 工作人员和制品

■ 项目管理流中的主要工作人员

- 项目经理
- 项目评审员



参见下一页



■ 项目经理负责的关键制品是

□ 软件开发计划(SDP)

- 产品接受计划
- 风险管理计划(和风险清单)
- 问题解决方案
- 度量计划

□ 业务用例

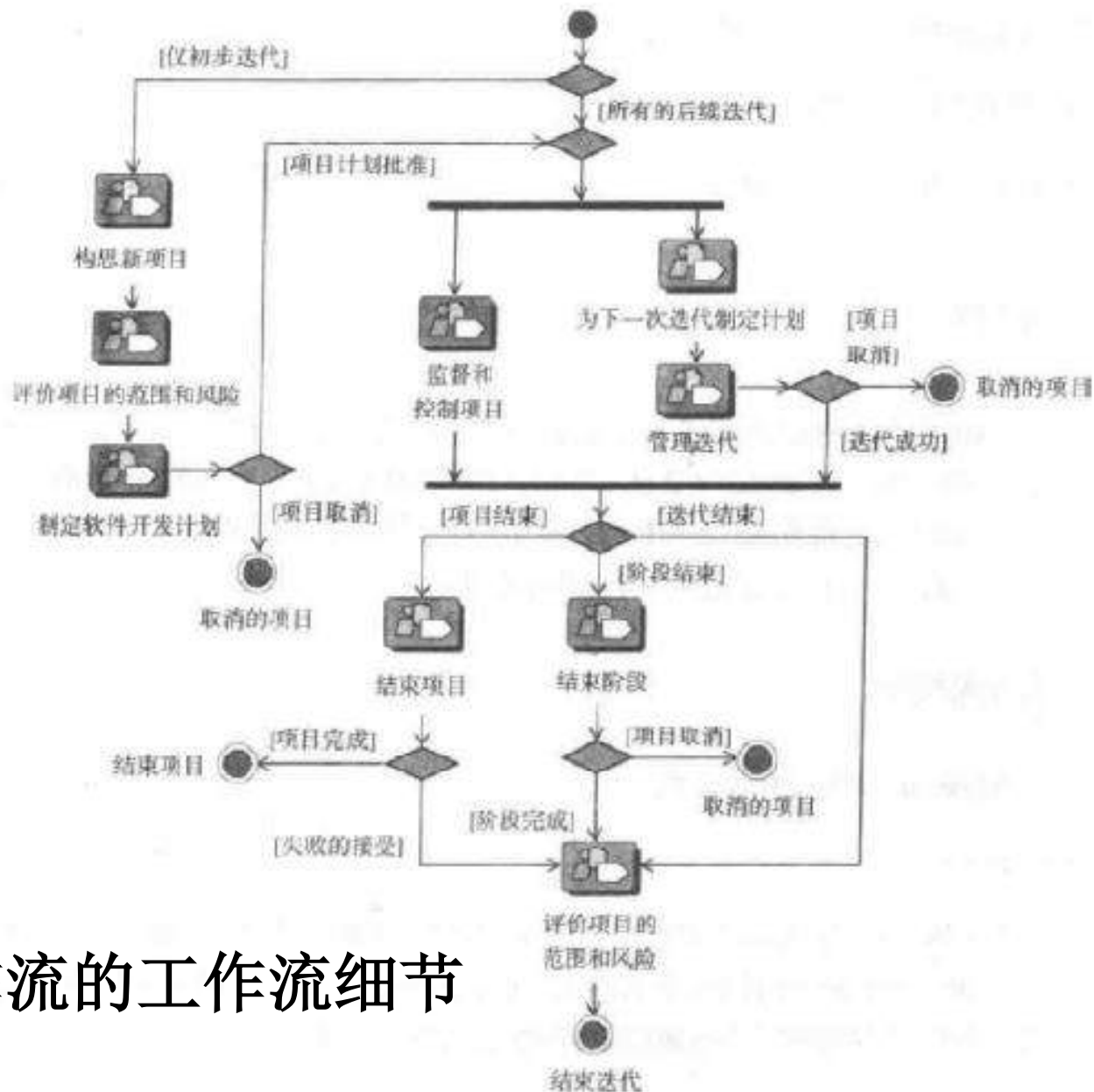
□ 迭代计划(每个迭代一个迭代计划)

□ 迭代评估

□ 工作顺序

□ 项目度量数据库

7.7 workflows



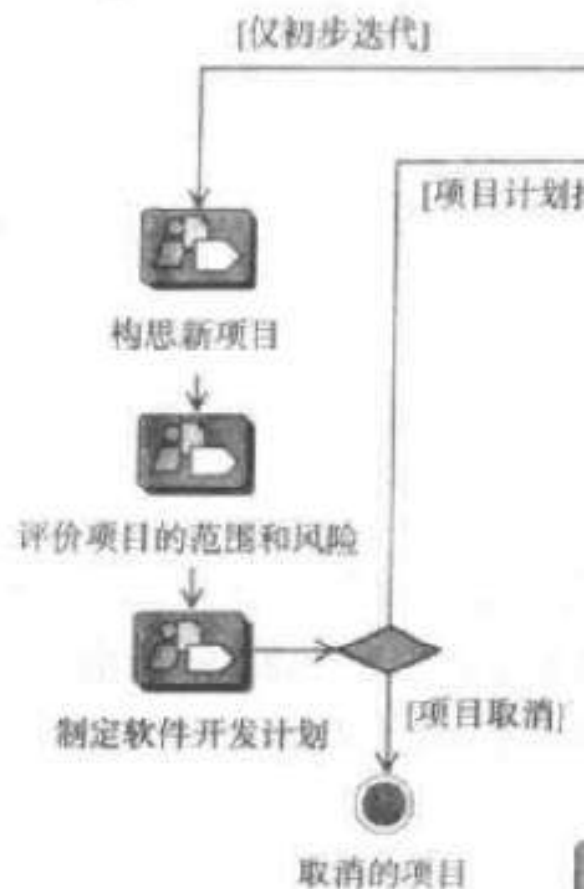
■ 项目管理 workflows 的工作流细节

1 构思新项目

■ 包括以下几项活动

- 由项目经理执行的**标识和评估风险、开发业务用例、初始化项目活动**
- 由项目评审员执行的**项目批准评审活动**

■ 该 workflow 细节的目的是将项目从仅仅是一个构想发展到可以合理地作出是继续还是抛弃项目的决定

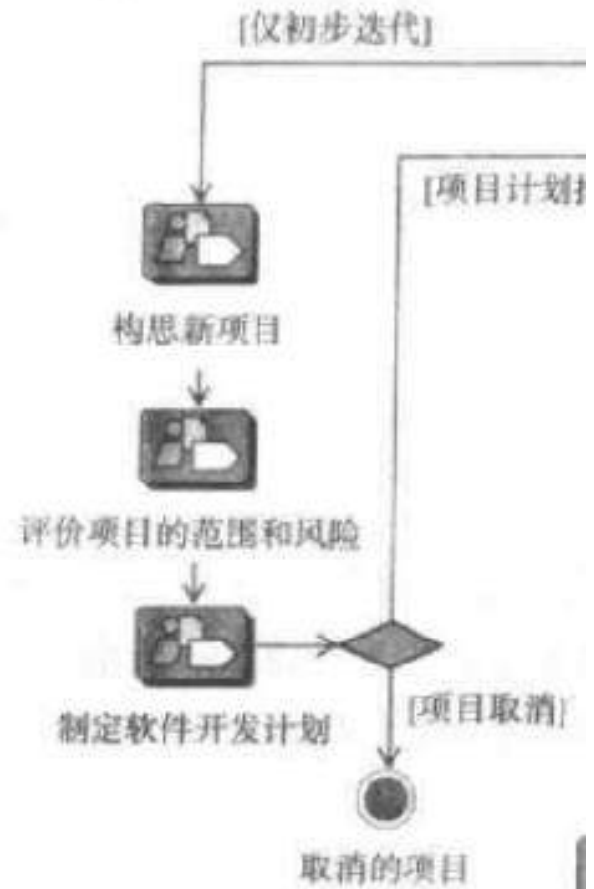


2 评价项目的范围和风险

- 包括以下几项活动

- 由项目经理执行的**标识和评估风险、开发业务用例活动**

- 该 workflow 细节的目的是重新评估项目的预期能力和特性以及在达到这个目的时可能遇到的风险

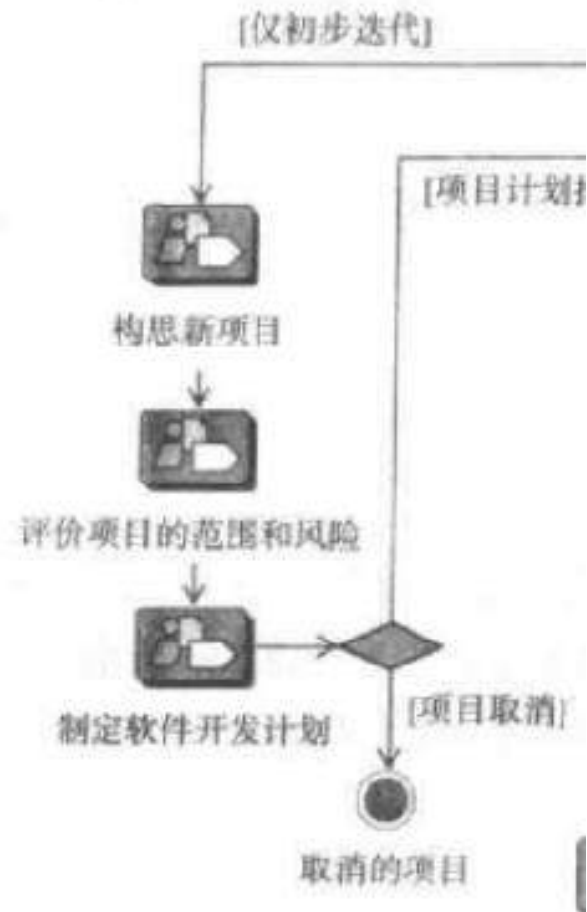


3 制定软件开发计划

■ 包括以下几个活动

项目经理执行

- 开发度量计划
- 制定风险管理计划
- 制定产品认同计划
- 制定问题解决方案
- 定义项目组织和人员配备
- 定义监督和控制过程
- 计划阶段和迭代
- 汇总软件开发计划



项目评审员执行

□ 项目计划评审活动

- 该 workflow 细节的目的是制定软件开发计划的构件及其附件，然后针对可行性和项目相关人员的可接受性，对这个计划进行正式评审，并将它作为制定下一个迭代的精细计划的基础
- 制作这些制品所需工作量主要集中在初始阶段早期；而后，在每个迭代开始时激发这个 workflow 细节，根据已完成的迭代中获得的经验，结合下一个迭代的计划，修改软件开发计划

4 计划下一个迭代

■ 包括以下活动

项目经理执行

□ 开发迭代计划

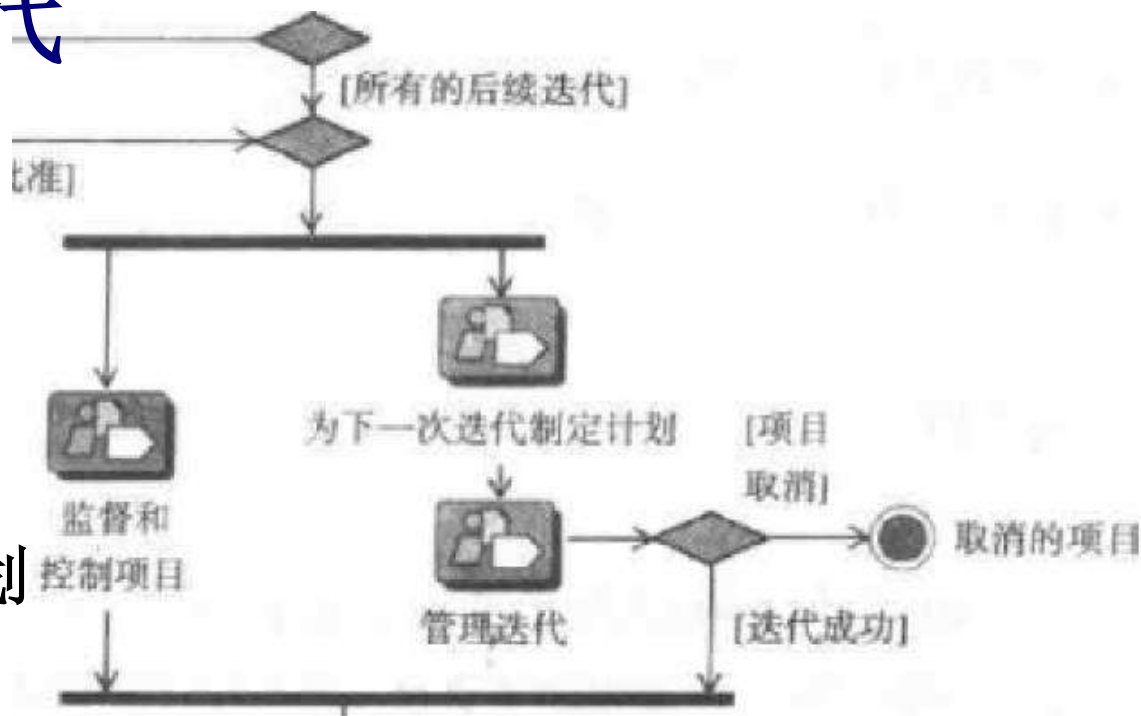
□ 开发业务用例

□ 制定软件开发计划

项目评审员执行

□ 迭代计划评审活动

- ### ■ 这个 workflow 细节的目的是创建一个详细的迭代计划，用来指导下一个迭代。在制定了这个计划之后，可能需要调整软件开发计划和业务用例



5 监督和控制项目

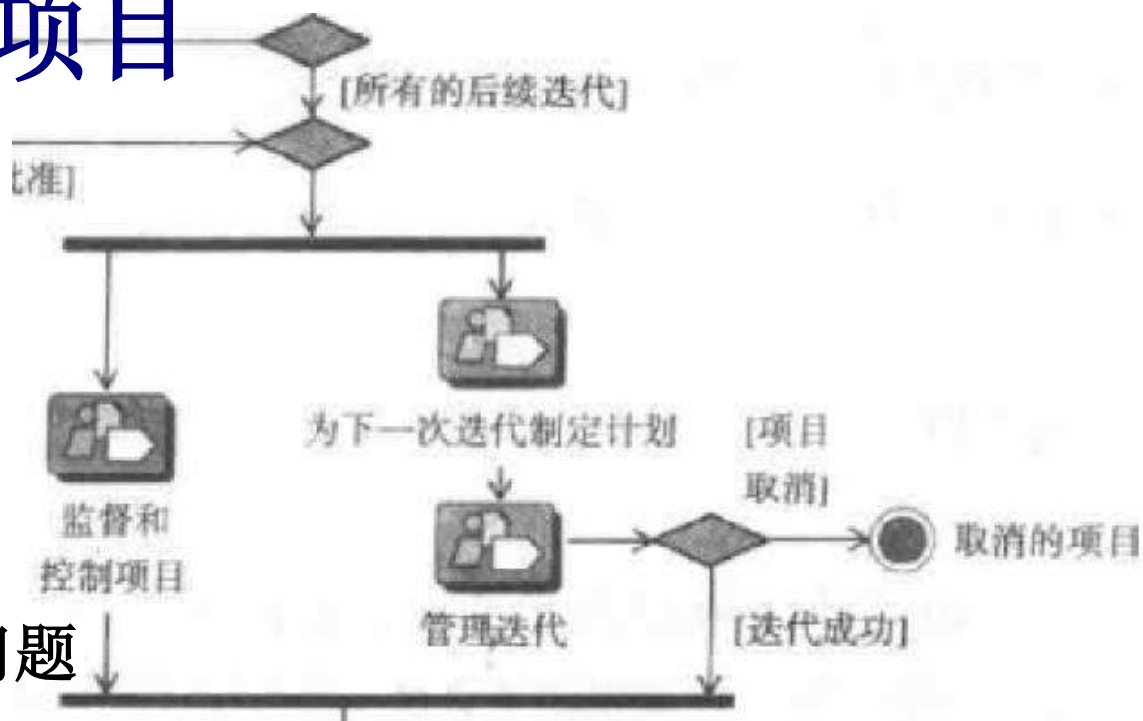
■ 包括以下活动


项目经理执行

- 计划和分配工作
- 监督项目状态
- 处理异常事件和问题
- 报告状态

项目评审员执行

- 项目评审当局评审





■ 这个 workflow 细节包含了项目经理的日常工作

- 处理变更请求，确定变更时间
- 对进展情况和质量进行度量，持续地监督项目
- 定期报告当前的项目状况
- 当出现问题时，根据问题解决方案计划来处理

6 管理迭代

■ 包括以下活动

项目经理执行

□ 安排工作人员

□ 初始化迭代

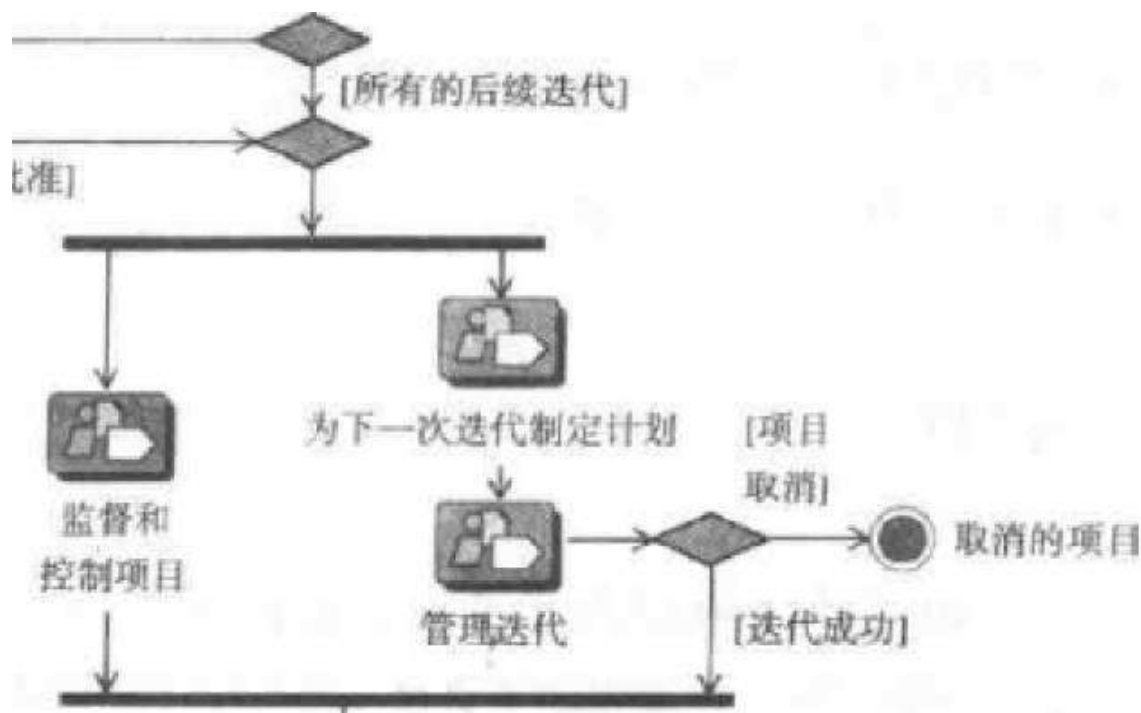
□ 评估迭代

项目评审员执行

□ 迭代评价准则评估

□ 迭代接受评审

■ 这个 workflow 细节包括开始、结束、评审迭代等活动。目的是获取执行迭代所必需的资源，分配将要完成的工作和评审迭代的结果



7 结束阶段

■ 包括以下活动

项目经理执行

□ 阶段结束准备

项目评审员

□ 生命周期里程碑评审

■ 在这个 workflow 细节中，项目经理明确了以下问题后结束整个阶段

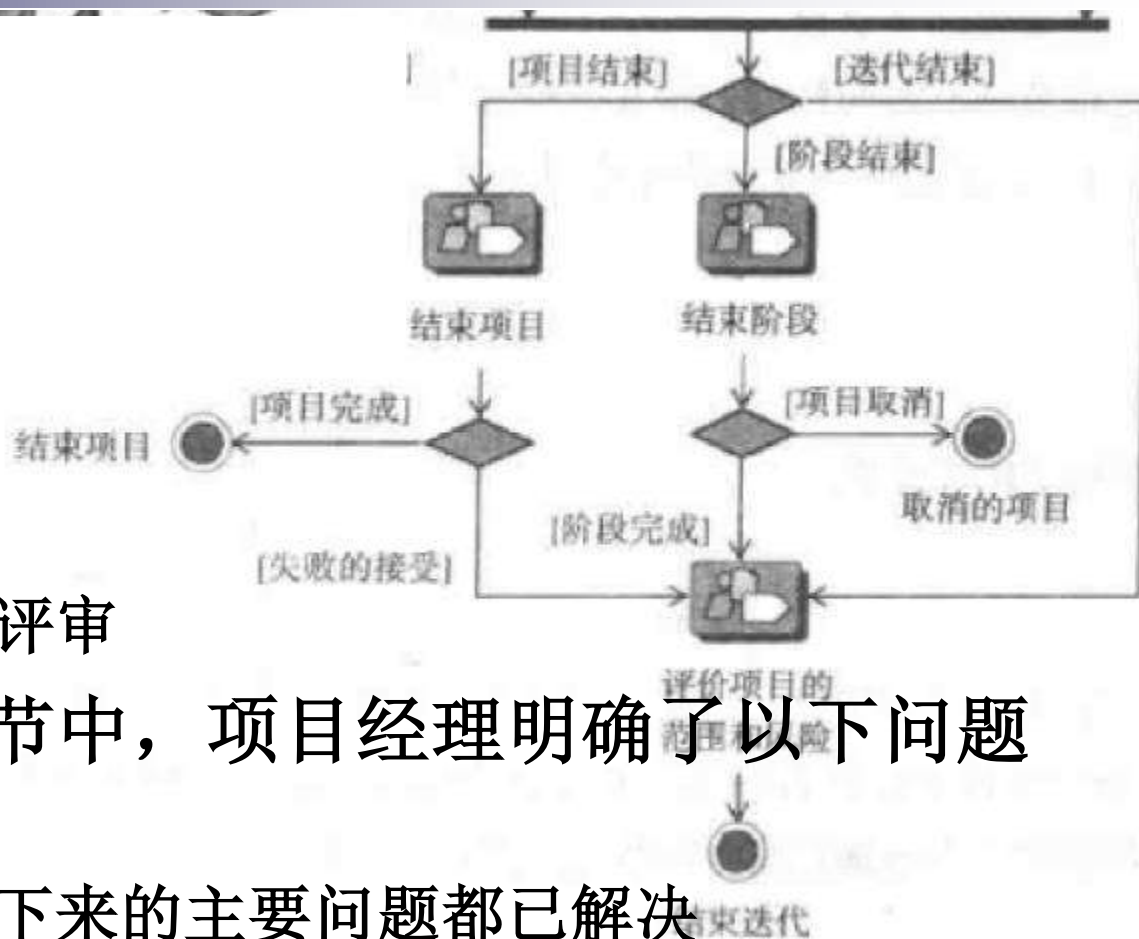
□ 上一个迭代遗留下来的主要问题都已解决

□ 所有制品的状态都已知道

□ 项目相关人员都已得到需要的制品

□ 解决了所有实施问题(例如安装、移交、培训)

□ 当前契约结束后，下一步项目资金要到位



8 结束项目

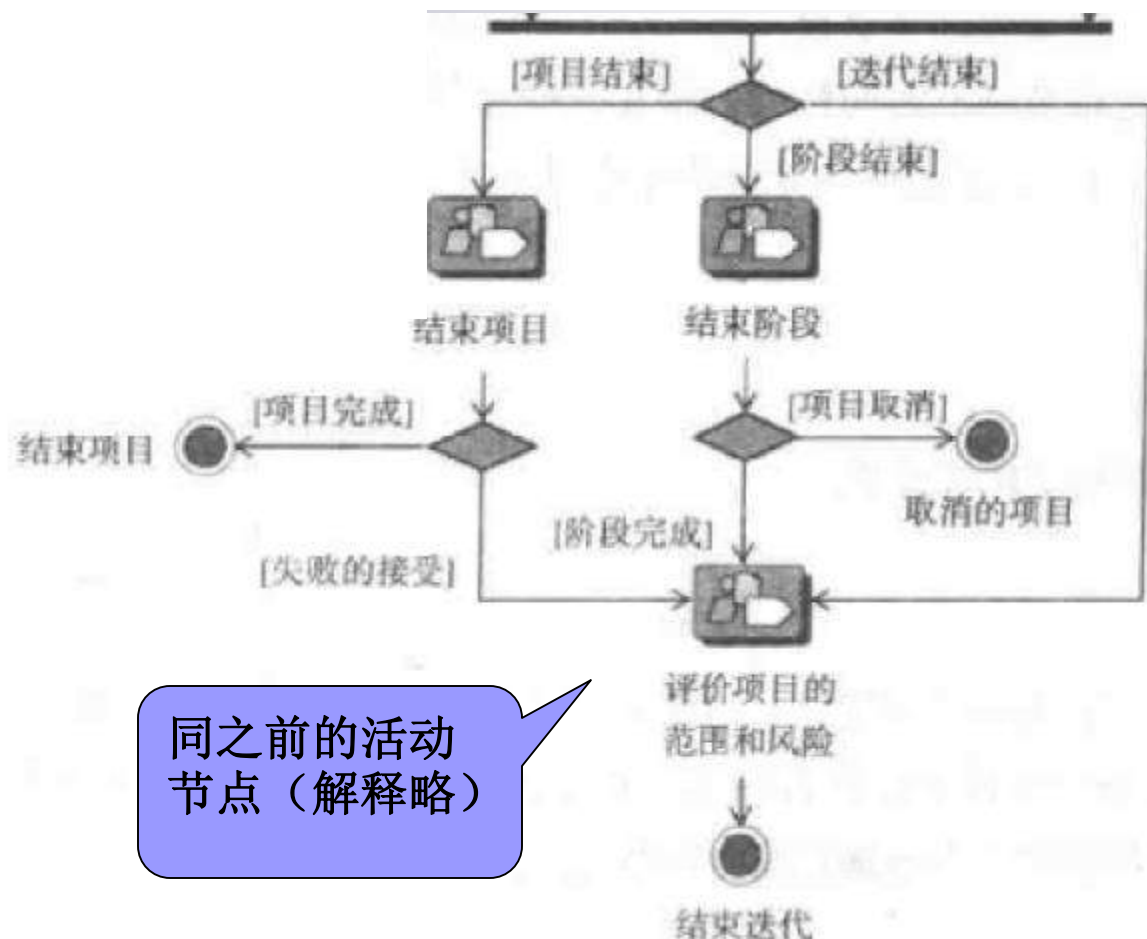
■ 包括以下活动

项目经理执行

□ 项目结束准备
项目评审员

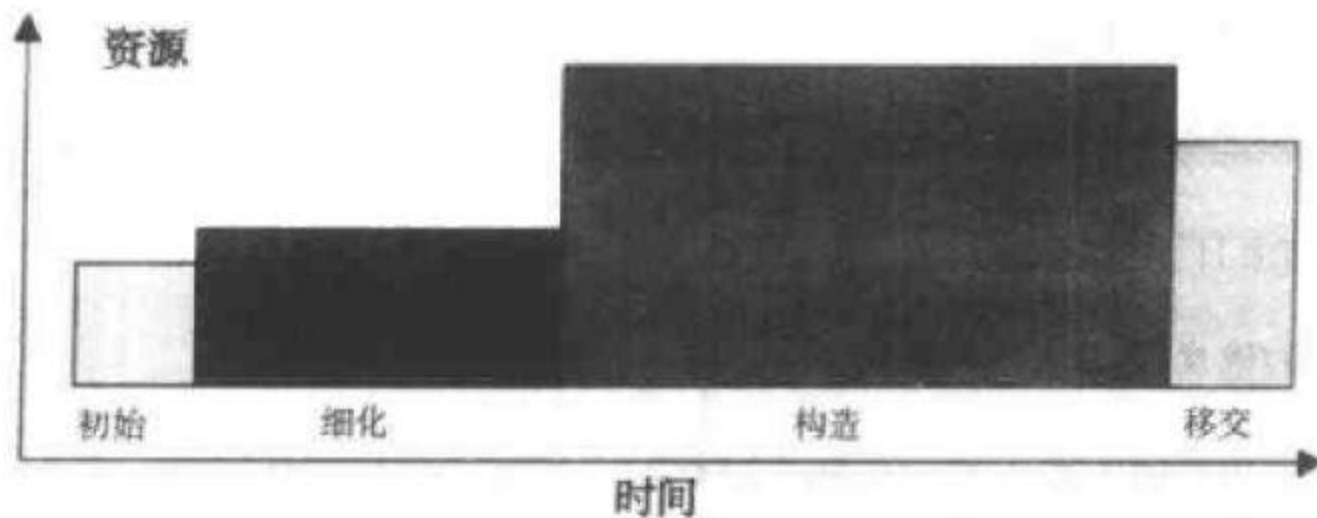
□ 项目接受评审

- 评估项目是否可以接受，如果可以接受，标记出在什么状况下客户正式地接受产品所有权。而后，项目经理在处理了剩余资产并重新分配了剩余工作人员的任务后，就可以将项目结束了



如何制定一个阶段计划

- 需要对项目有一个大致了解
 - 工作量
 - 最终版本的发布日期
- 从结束日期往前推移，计划每个重要里程碑的假设日期
 - 权衡人员配备、进度和范围
 - 橡皮泥剖面图



- 典型的项目剖面图显示了每个阶段大致持续时间和工作量

表7-1 一个典型项目各个阶段持续时间和工作量的相对比例

阶 段	持 续 时 间	工 作 量
初始阶段	10%	5%
细化阶段	30%	20%
构造阶段	50%	65%
移交阶段	10%	10%

上述分配比例适用的项目特点

- 中等规模、中等工作量
- 处于最初开发周期
- 没有事先已存在的构架
- 仅存在很少的风险和未知因素

迭代的个数

■ 在初始化阶段

- 目标是尽快得到所有问题的答案
- 没有实际的迭代，没有软件被开发出来，仅仅是制定计划和做一些市场调查
- 只在要完成下列任务时需要使用迭代方法
 - 建立原型，为了验证设想的正确性
 - 建立原型，为了缓解主要技术风险
 - 通过工具和过程使开发组织加快开发速度

■ 初始化阶段有一个迭代，或者根本没有



■ 在细化阶段

- 至少要计划一个迭代
- 如果没有现成的构架，而且有很多新的因素要协调(新员工、新工具、新技术、新平台)，那么就要计划**2到3**个迭代用于处理风险
 - 向用户展示原型
 - 更改架构中的缺陷

■ 细化阶段需要**1到3**个迭代

■ 在构造阶段

- 至少要计划一个迭代
- 为了更好的集成和测试工作，**2个迭代**更合理
- 自动化水平较高和过程较成熟的情况下，为了提高项目水平，也可以**3个或更多迭代**

■ 构造阶段需要**1到3个迭代**

■ 在移交阶段

- 至少要计划一个迭代
 - 在**beta**版本之后发布最终版本
- 市场的现实情况或最初版本的质量问题会导致更多迭代

■ 这个阶段需要**1到2个迭代**

在整个开发周期中[IECT],可以分成3种水平

- 低水平:3个迭代[0,1,1,1]
- 一般情况:6个迭代[1,2,2,1]
- 高水平:9个迭代[1,3,3,2]
- “正常的”项目需要 6 ± 3 个迭代,称之为“6加减3”经验

7.8 制定一个迭代计划

- 迭代计划要精确地描述迭代中每个时间段上都有什么事情发生，将活动分配给工作人员(**worker**)，再将个人分配给工作人员
- 要将迭代的长度和为它分配的资源看作是边界框
- 要根据时间制定计划，而不是根据工作量制定计划

制定一个迭代计划必须遵循以下步骤

- 定义迭代成功的目标准则
- 明确要开发或更新哪些具体的、可度量的制品以及要完成的活动
- 从一个典型的迭代工作分解结构开始，并考虑当前必定要发生的实际活动
- 估计每个活动的持续时间和工作量，注意要控制在预算的资源范围之内

细化阶段的迭代


- 定义细化阶段的迭代目标时，有**3**个主要驱动：
 - 风险(最重要)
 - 尽早缓解或消除风险
 - 覆盖
 - 确保构架涵盖了要开发的软件的所有方面
 - 关键程度
 - 确保系统包含了关键功能和服务，即使没有可察觉到的风险与它们相联系

- 对于大多数破坏性的风险，在一个用例中确定一个情景就可以让开发团队注意到这个风险：

1) 假设有一个集成风险，如“数据库D可以很好地在操作系统Y上工作”，即使这是一个合适的数据库，也要确保用况中包括了一个涉及到与数据库交互的情景。

2) 假设有一个性能风险，如“计算飞行器轨道所需的时间过多”。你要确保用况中包含了涉及到这个计算过程的情景，至少包括那些最明显最频繁发生的情况。

- 对于关键程度，要确保用例包含了系统需要提供的最基本的功能和服务。从用例中选择一些情景代表最普遍或发生频率最高的服务和性能的形式

- 
- 对于覆盖，到细化结束时，需要开发那些触及到已知领域的情景，即使这些情景既不关键也没有风险
 - 要注意，避免过早出现过于“厚”的情景，即那些试图覆盖很多方面、不同变量和错误用例的情景
 - 在细化阶段还要记住，评审风险可以缓解风险，但是同时带来的很多改变也会破坏原有的稳定性

细化阶段迭代的例子

- 在客户工作站创建一个电话用户的记录，并将该记录存储在服务器端的数据库中。
 - 该情景涉及关键功能和集成风险(数据库和服务端系统)
- 确保至少能创建**20000**个电话用户，并且访问一个电话用户耗时不超过**200**毫秒。
 - 该场景处理了主要性能问题（数据量和相应事件），假如没有达到这些要求，可能会严重影响构架
- 撤销对一个电话用户地址的变更。
 - 这个情景选择了一个简单的功能，促使你去思考所有“撤销”功能的涉及。然后与最终用户流
- 完成与供应链管理相关的所有用例。
 - 细化阶段的目标之一就是捕获需求

构造阶段的迭代

- 构造阶段里，风险仍旧是关键驱动，尤其是那些还未发现的新风险和悬而未决的风险。
- 用例的完成也是驱动。可以一个特性一个特性地计划迭代，尽早地完成比较关键的特性，这样就可以在不只一个迭代中对它进行彻底地测试。
- 构造结束时，主要目标是确保覆盖了所有的用例

构造阶段迭代的例子

- 1) 实现所有呼叫转接的变体，包括错误的变体。这是一系列相互联系的特性。
- 2) 完成所有电话接线员特性，除了夜间服务。这属于另一组特性。
- 3) 在两台计算机设备上达到每小时5000次交换。这将提高在前面迭代中达到的实际性能值（2357次交换/小时）。
- 4) 集成地理信息系统的新版本。这个目标对构架是一个适当的修改，也有可能是由于早先发现的问题做出的必要修改。
- 5) 修改所有1级和2级缺陷。修复在前面迭代测试中发现而没有立即改正的缺陷。

移交阶段的迭代

- 此阶段的主要目标是完成这一代产品。
- 每个迭代的目标表述为修复了哪些缺陷，提高了哪些性能或可用性
- 假如之前为按时结束构造阶段曾抛下某个特性，在对当前已完成部分没有危害的情况下，现在可以完成或启动这些特性

移交阶段迭代的例子

- 1) 修改所有在beta客户站点中发现的严重性为I级的问题。这个目标从质量的角度出发，将关系到产品在市场中的信誉度。
- 2) 消除所有由于数据不匹配产生的系统启动故障。这是另一个从质量角度出发的目标。
- 3) 达到每分钟2000次交换。这属于性能上的调节和优化：涉及到数据结构的改变、高速缓冲和更聪明的算法。
- 4) 将不同的对话框的数量减少30%。通过减少视觉混乱以提高可使用性。
- 5) 完成德文和日文版本。Beta版本仅仅是给英语用户使用的，原因是没有时间完成其它语言版本，也是为了减少对beta版本的重写。

迭代中的工作细节

- 当选择好将要开发的情景或成熟的用例(加上要修复的缺陷)并将他们大致描述后，要明确哪些制品将要受到影响：
 - 哪些类将要重新访问
 - 哪个子系统会受到影响或要新建哪个子系统
 - 哪些界面可能要修改
 - 哪些文档必须更新

■ 确定在Rational统一过程中涉及到的活动并将它们包含在计划中。

- 有些活动要在每个迭代中执行1次(如制定1个迭代计划)，还有些活动则要在每个类、每个用例或每个子系统都执行1次(如设计1个类)
- 大多数过程中描述的活动都小到可以在几个小时或几天之内由一个人或一个团队来完成
- 当发现迭代没有足够的时间完成预定工作时，不要扩展迭代(这样会拖延移交时间或减少迭代数量)，而要降低迭代的目标。根据当时处于什么阶段，简化情景或减少功能

7.9 小结

- **Rational**统一过程中的项目管理 workflow 对于平衡相互竞争的目标、管理风险和克服障碍以交付一个满足客户和最终用户需求的产品是非常有用的
- 在迭代中，开发应该以阶段计划和一系列迭代计划为基础
- 风险是计划的驱动器
- 度量是用于控制项目的关键技术
- 在制定阶段计划时，必须顾及人员配备、进度和项目范围之间的权衡关系
- 每个阶段对于迭代范围定义的准则是不同的



第8章 业务建模 workflow



目录

- 8.1 目的
- 8.2 为什么要进行业务建模
- 8.3 在业务建模中使用软件建模技术
- 8.4 业务建模情景
- 8.5 工作人员和制品
- 8.6 workflow
- 8.7 从业务模型到系统
- 8.8 小结

8.1 目的

■ 业务建模的目的如下

- 理解将要实施的系统的组织(目标组织)结构和动态特性
- 理解当前在目标组织中的问题，并明确改进的潜力
- 确保客户、最终用户和开发人员对目标组织有统一的理解
- 获取用于支持目标组织的系统需求



■ 为了达到以上这些目的

- 业务建模工作流程介绍了如何开发一个新目标组织的构想
- 根据这个构想定义业务模型中组织的过程、角色和职责
- 这个模型包括一个业务用例模型和一个业务目标模型

8.2 为什么要进行业务建模

- 软件应用程序开发要在进行软件工程项目之前或者同时就要了解业务领域
- 业务模型和系统定义不仅仅是IT行业人员的兴趣所在，还是所有涉及业务的开发人员的焦点
- 并不是每一个软件工程工作都需要进行业务建模。当有很多人直接参与使用系统，并且有很多信息需要系统处理时，业务模型比较实用

8.3 在业务建模中使用软件建模技术

- 在业务建模中使用与软件工程技术相似的建模技术会带来许多便利，主要的优势是语言相通。
- 有助于理解在业务领域中描述的事物是如何与软件领域中的事物相联系的
- 它还可以简化对于业务模型中的制品和对应的系统模型中的制品之间关系的描述

业务建模的注意事项

- 业务用户(客户、销售者或合作伙伴)表示为业务参与者
- 业务过程表示为业务用例和业务用例实现
- 人们在组织中扮演的角色表示为业务工作人员
- 组织管理或制造的“东西”表示为业务实体



业务过程
分析员



业务术语表



业务规则



业务用况模型



业务对象模型



目标组织评估



业务构想



业务构架文档



补充业务
规格说明



业务设计师



业务用况



业务参与者



业务用况实现



组织单元



业务实体



业务工作人员

■ 业务建模 workflow 中的工作人员和制品

8.4 业务建模情景

- 业务工程的工作量由于内容和需求的不同而不同
- 情景1：组织图
- 情景2：领域建模
- 情景3：一个业务，多个系统
- 情景4：普通业务建模
- 情景5：新的业务
- 情景6：修订

情景1： 组织图

- 希望建立一个组织及其过程的简单图，以对正在设计的应用程序的需求有一个更好的理解
- 这时，业务建模就是软件工程项目的一部分，它主要是在初始阶段执行

情景2：领域建模

- 希望创建一个主要目的是管理和表示信息的应用程序(如一个订购管理系统或银行系统)
- 这时可以在业务一级上为这些信息建立一个模型，并不用考虑业务 workflow，这就是领域建模
- 领域建模通常是软件工程项目的一部分，在工程的初始和细化阶段执行

情景3：一个业务，多个系统

- 希望建立一个大的系统，或者一个应用程序族，那么创建一个业务模型就可以充当好几个软件工程项目输入
- 业务模型将帮助找到业务需求，同时还可以作为应用程序族构架的输入
- 在这种情况下，业务建模工作通常被视为一个独立的项目来处理

情景4：普通业务建模

- 希望创建一个供几个组织使用的应用程序(例如，一个支持销售的程序或一个记账程序)，就可以根据各个组织处理业务的方法，在建模时将它们结合起来，从而避免系统过于复杂
- 如果无法将组织结合起来，建模工作可以帮助理解和管理组织使用应用程序的不同方法，并能简化应用程序功能优先级的排列

情景5：新的业务

- 一个组织决定开发一个全新的业务线，并且建立一个信息系统来支持它，这时就需要执行业务建模。
- 在这种情况下，建模的目的不仅仅是找到系统需求，还要决定新的业务线的特性。
- 此时，通常把业务建模视作一个独立的项目来处理。

情景6：修订

- 一个组织决定彻底修订处理业务的方法(业务过程再设计)，业务建模本身通常就是几个项目之一。
- 业务过程再设计通常要经过以下几个阶段：设想新的业务，推翻已存在的业务设计，建立新的业务设计，安装新的业务

8.5 工作人员和制品

业务建模中涉及到的主要工作人员包括：

■ 业务过程分析师：

- 通过概述和限定被建模的组织，引导和调整业务用例建模。例如，建立一个新业务构想，包括业务参与者和业务用例及它们之间是如何交互作用的

■ 业务设计师：

- 通过描述一个或几个业务用例来详述部分功能。他们决定实现业务用例需要哪些业务工作人员和业务实体，以及如何实现业务用例
- 定义一个或多个业务工作人员和业务实体的职责、操作、属性和关系

其他人员

- 项目相关人员：
 - 代表组织的不同部分，并提供输入和评审
- 业务评审员：
 - 评审制品的结果

业务建模中的关键制品

- 业务构想文档：定义了业务建模工作的目标
- 业务用例模型：业务预期功能的模型，用做确定组织中的角色和可交付的功能的输入
 - 业务用例模型包括业务参与者和业务用例
 - 参与者是业务外部的角色(例如客户)
 - 业务用例是过程
- 业务对象模型：描述业务用例实现的对象模型
 - 业务对象模型从业务工作人员和业务实体交互作用的角度说明了业务用例是如何执行的
 - 业务工作人员和业务实体可能被分组为不同的组织单位

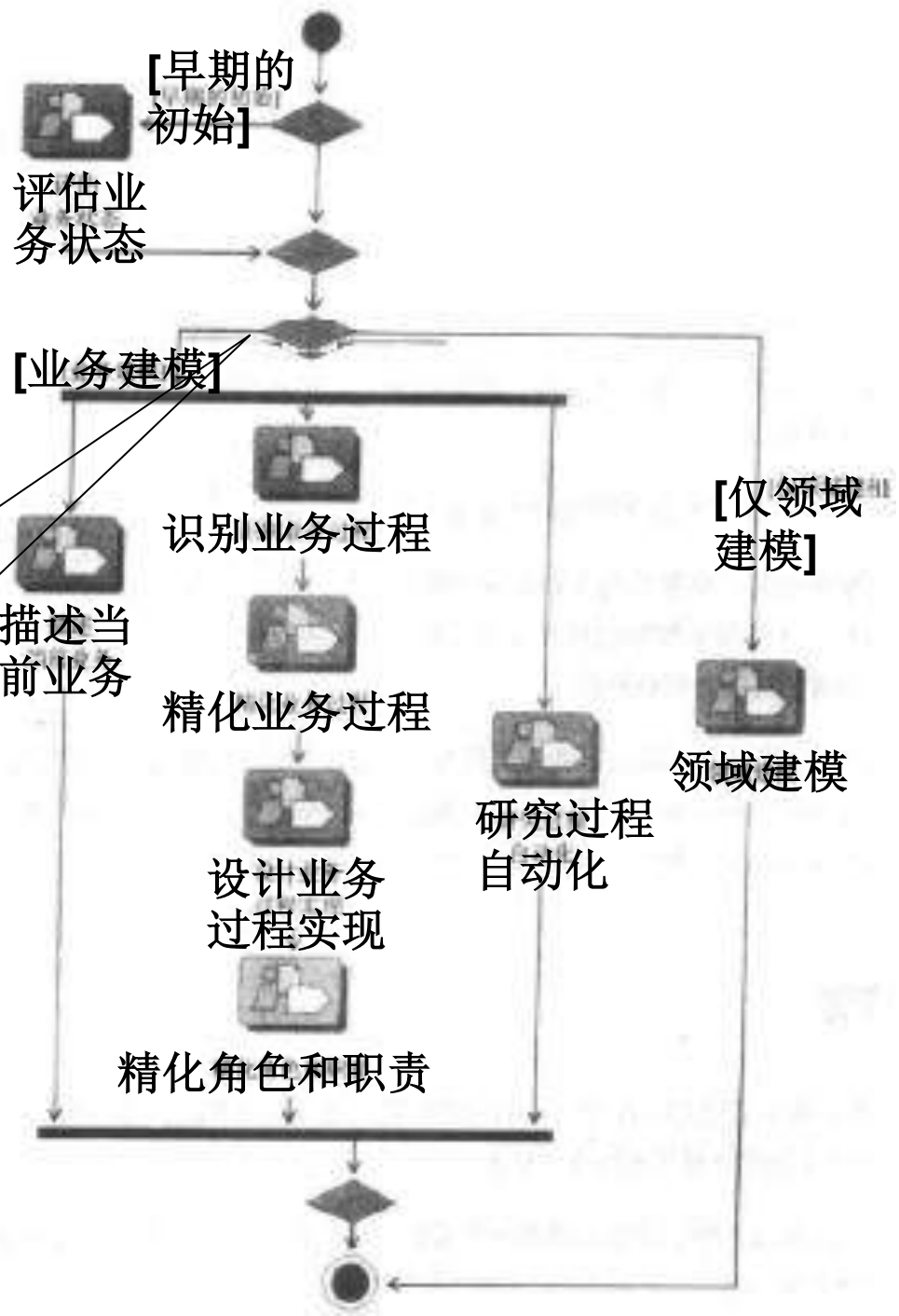
其他制品

- 目标组织评估：描述了准备实施系统的组织的当前状态
- 业务规则：对相关政策的声明或必须满足的条件
- 补充业务规格说明：业务定义文档，它提出了在业务用例模型和业务对象模型中没有定义的那些业务的定义
- 业务术语表：定义了业务中用到的重要术语

8.6 workflows

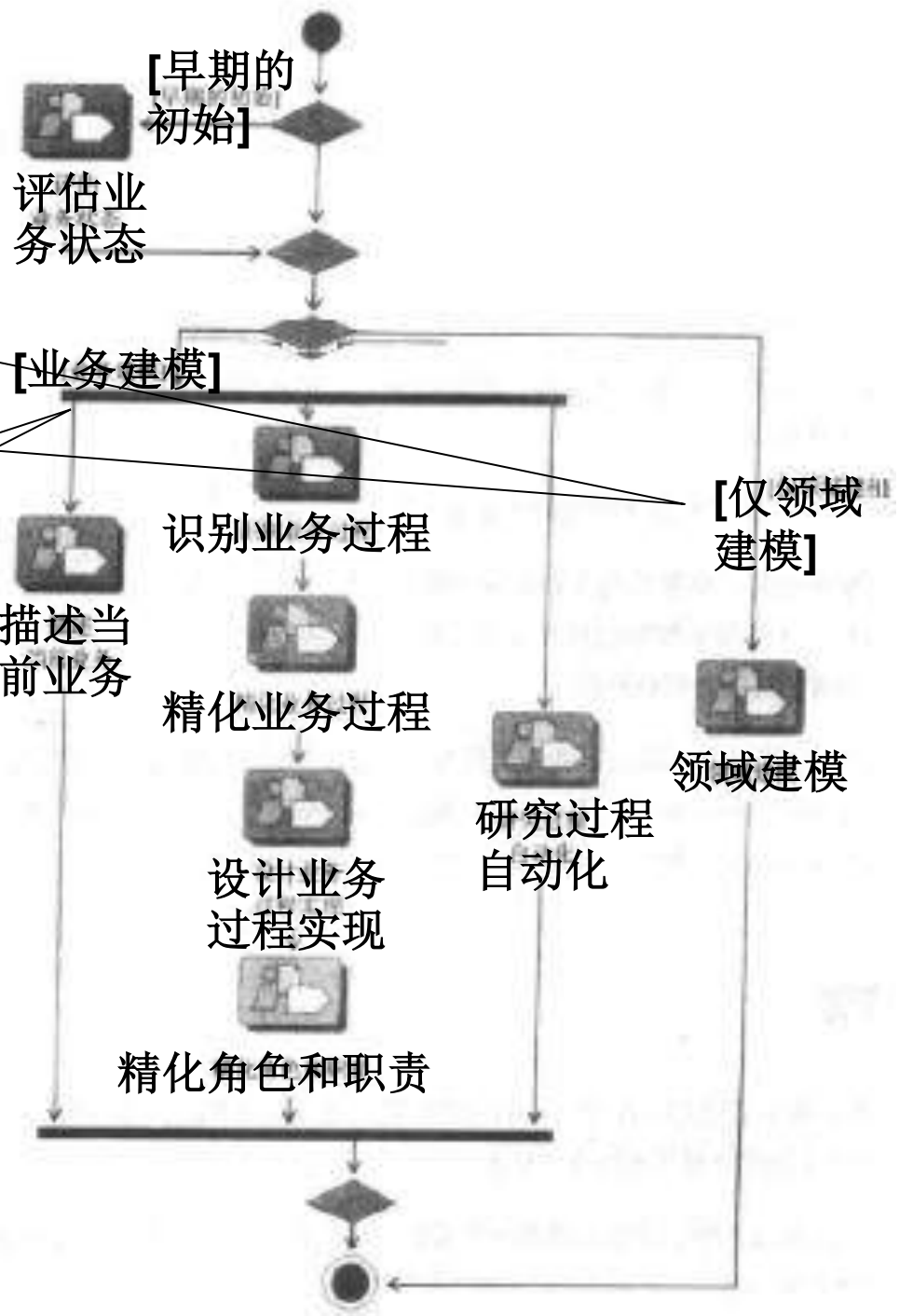
在第一个迭代过程中，评定要实施最终系统的组织的状态。这时产生的主要制品是目标组织评估和业务构想。

在评估结果的基础上，决定遵循哪个业务建模情景。然后决定如何继续这个迭代以及接下来如何工作。



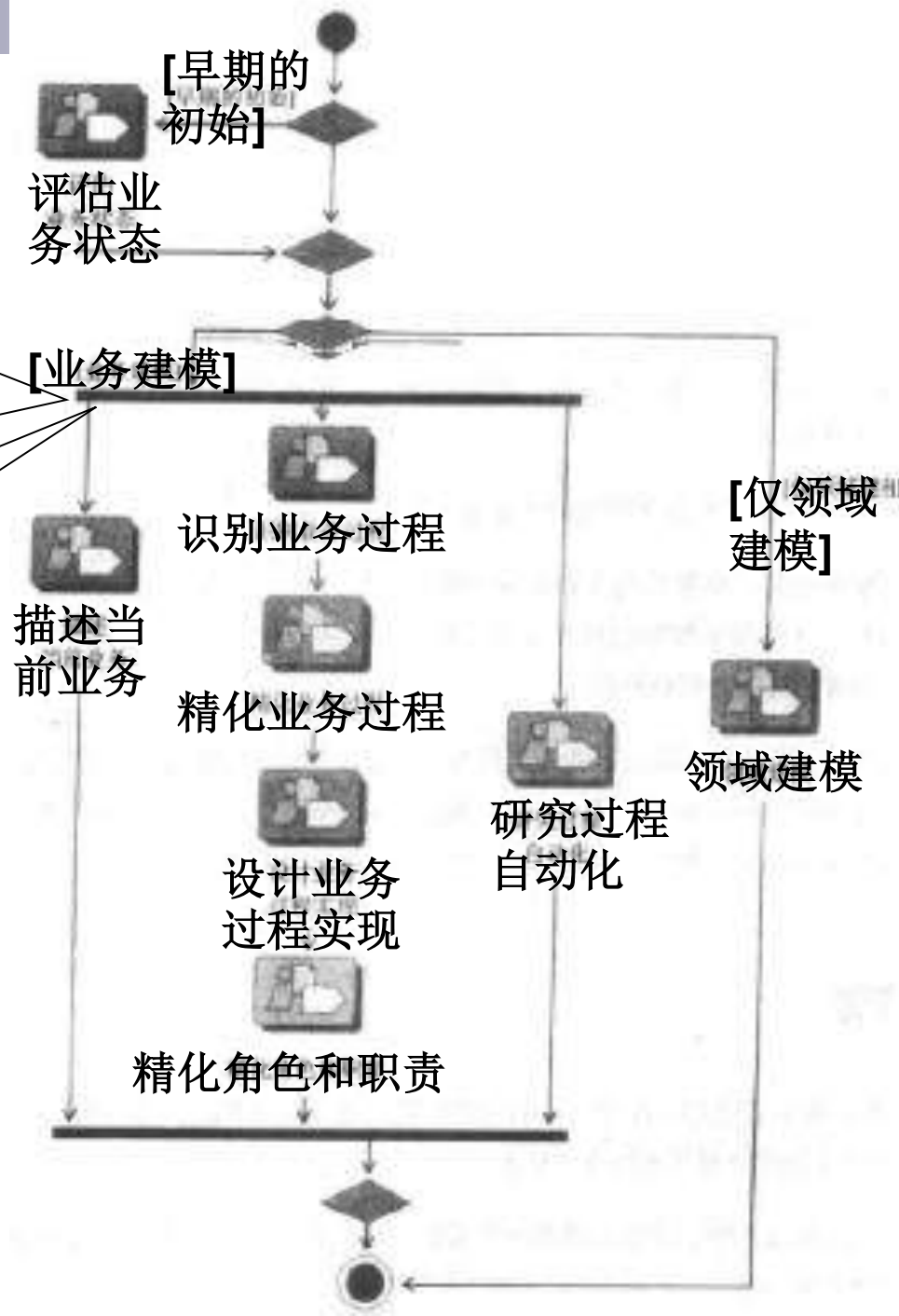
仅需要领域建模，则走此路径，领域建模以及模型中的业务实体被认为是业务对象模型的子集

如果没有必要对业务过程进行重要的改动，只需绘制这些已经存在的过程，并从中寻找系统需求(情景1)。有时可以跳过“描述当前业务”



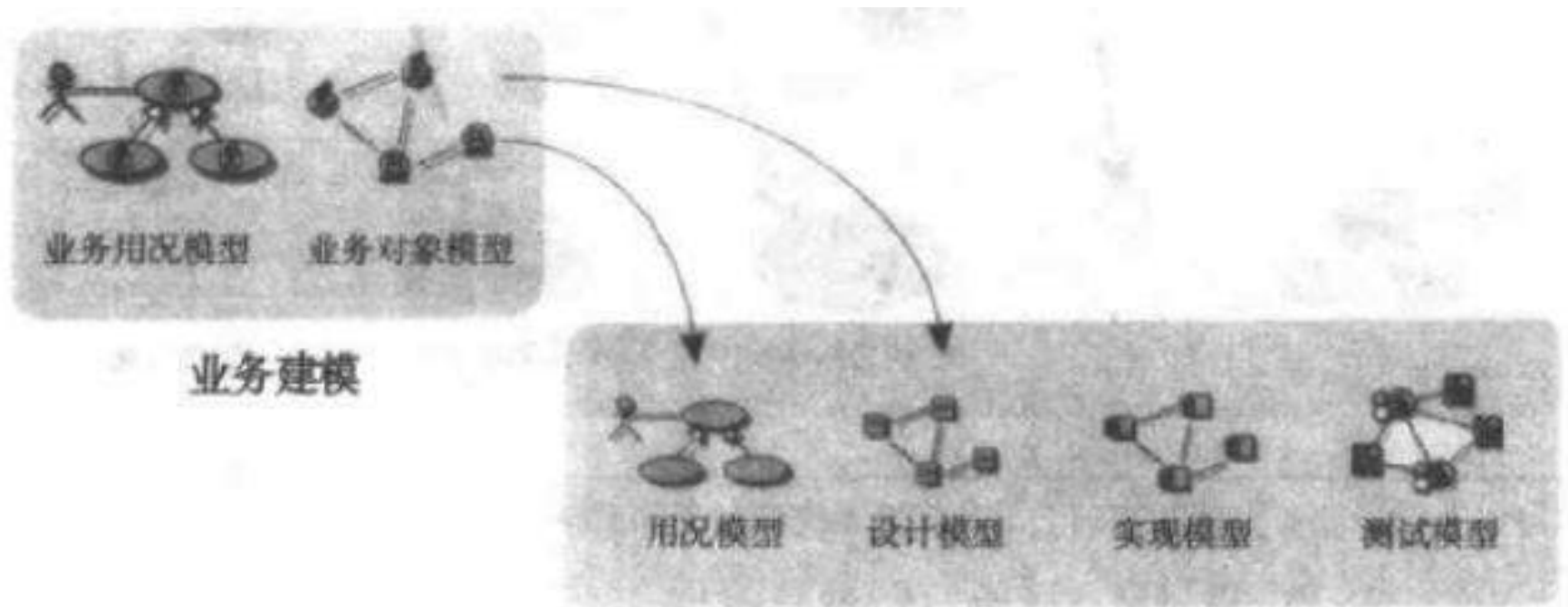
假如建立业务模型的目的是试图提高或重新设计某个已存在的业务(情景3,4,6),可以同时给当前业务和新业务建模

假如建立业务模型的目的是在一个草稿的基础上开发一个或多或少的
新业务(情景5),可以预想新的业务并给新业务建模,而跳过“描述当前业务”



8.7 从业务建模到系统

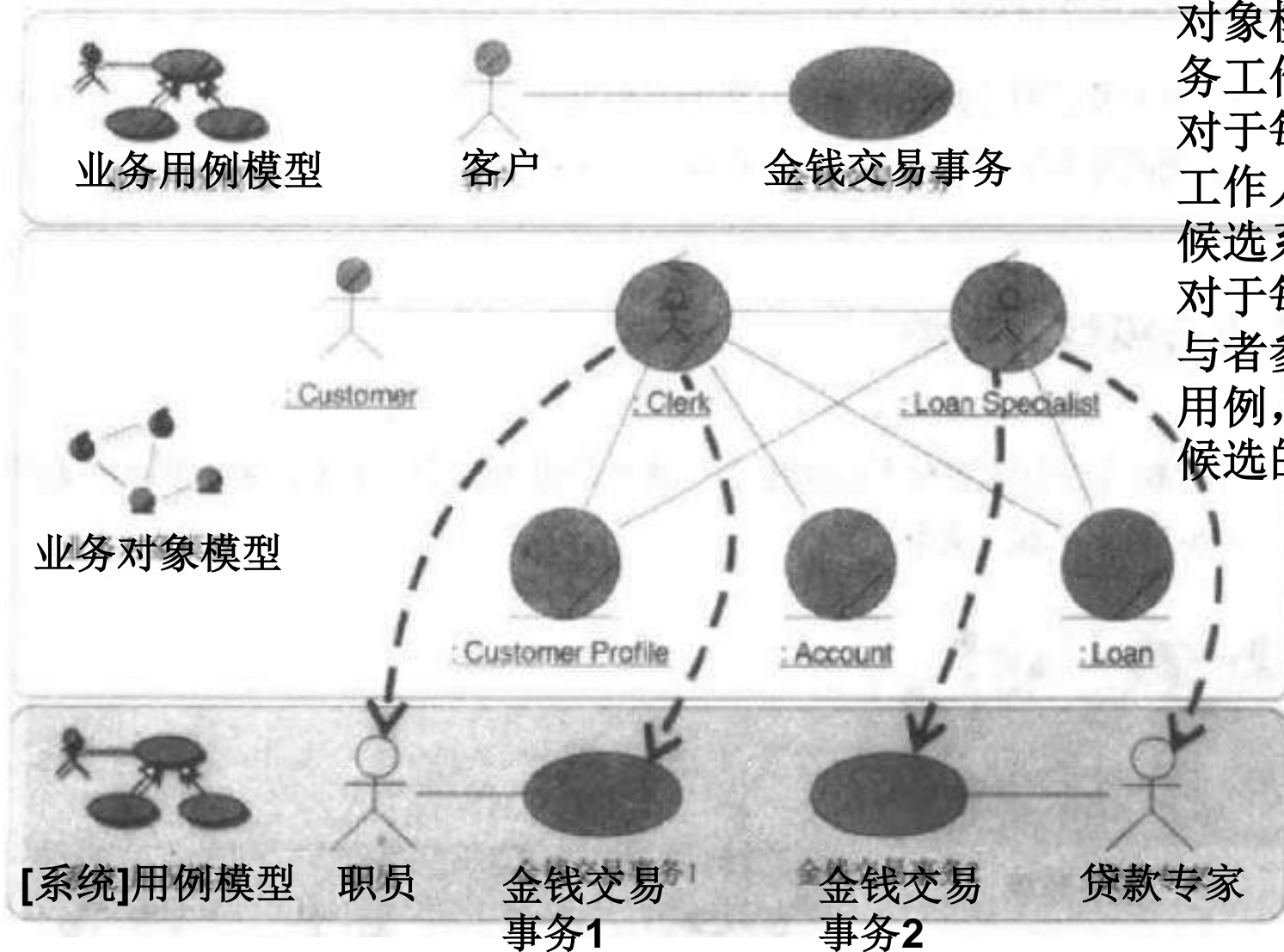
- 业务建模使用一种非常明确而简洁的方法表示业务和系统模型之间的关系



从业务模型到系统模型

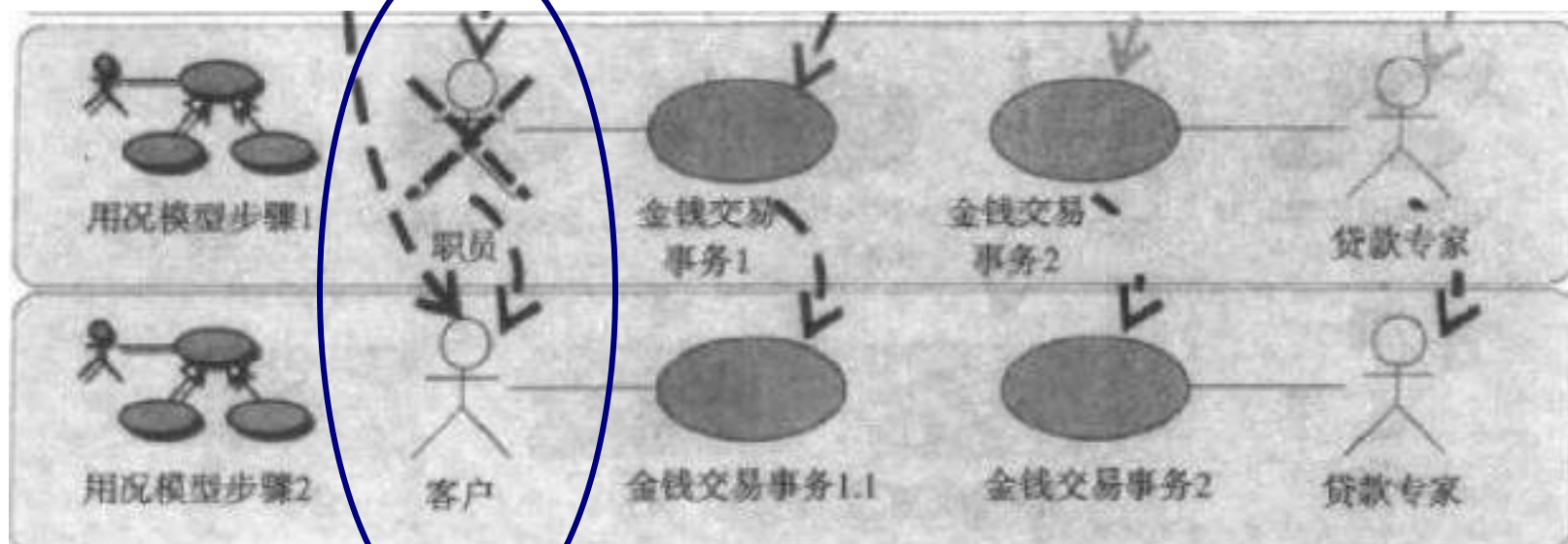
业务模型和系统参与者

为了确定信息系统用例，从业务对象模型中的业务工作人员入手。对于每一个业务工作人员，驱动候选系统参与者。对于每个业务参与者参与的业务用例，建立一个候选的系统用例。



自动业务工作人员

- 假如要建立一个可以使一组业务过程完全自动化的系统，例如电子商务应用系统。这时的业务工作人员的职责将移交给业务参与者,即从职员变为客户



系统需要的其他资源

- 在业务模型之外有很多关于信息系统的知识资源和对信息系统的需求，例如
 - 在业务模型中没有显示出来的用户，如系统经理，他是信息系统的用户，但是不在业务模型中表示出来
 - 业务作为整体决定的策略，如重用，兼容性和质量
 - 每个遗留下来的考虑因素
 - 调整以与其它项目同步和一致

业务模型和系统构架

- 从构架的角度看，假如要构造一个以下类型的系统，建立业务模型非常有用：
 - 为某一特定行业中的一个或多个公司(例如银行、保险公司)定制的系统
 - 为开放的市场开发的应用程序族(如订单处理系统、开账单系统、空中交通管制系统)
- 业务模型可以给用例视图和逻辑视图提供输入

Rational Rose提供了可视化模型,也存在其他**Rational**软件设计工具支持业务模型。

8.8 小结

- 业务建模可以帮助理解业务的结构和动态特性，并确保所有的项目相关人员对于问题有着相同的理解
- 可以对软件工程技术进行翻译并用于业务建模
- 软件需求大部分来源于业务模型



第9章 需求 workflow




目录

- 9.1 目的
- 9.2 什么是需求
- 9.3 需求中的种类
- 9.4 捕获需求和管理需求
- 9.5 设计以用户为中心的界面
- 9.6 需求工作流程
- 9.7 需求分析中的工作人员
- 9.8 需求分析中使用的制品
- 9.9 工具支持
- 9.10 小结

9.1 目的

- 在系统将做些什么以及为什么这样做的问题上与客户和其他项目相关人员达成协议,并维护这个协议
- 给系统开发人员提供对系统需求更好的理解
- 定义系统边界
- 为计划迭代过程中的技术内容提供基础
- 为估计开发系统需要的成本和时间提供基础
- 定义系统的用户界面,主要关注用户需求和目标

- 
- 为达成这些目的,需求 workflow 描述了如何定义系统构想并将这个构想转化成用例模型
 - 这个用例模型和补充规格说明书定义了详细的系统软件需求
 - 需求 workflow 还描述了如何使用需求属性来帮助管理系统范围和需求的变更

9.2 什么是需求

- 需求：系统必须满足的条件或具备的能力。
- 软件系统**质量属性**分为(FURPS)
 - 功能性(Functionality)
 - 使用性(Usability)
 - 可靠性(Reliability)
 - 性能(Performance)
 - 可支持性(Supportability)

功能性
需求

非功能
性需求

功能性需求

- 功能性需求描述了系统必须有能力执行的动作
- 功能性需求通过详细说明所期望的输入和输出条件来描述系统行为

非功能性需求

- 非功能需求使最终用户获得期望的系统质量，它们对于用户群来说和功能性需求一样重要
- 使用性(usability)
 - 使用性需求是指人为因素(审美学、易学性、易用性等)和用户界面、用户文档、培训资料的一致性。
- 可靠性(reliability)
 - 可靠性需求是指失败的频率和失败严重性、可恢复性、可预测性和准确性

■ 性能(performance)

- 性能需求是指在功能性需求上施加的条件，例如，需求详述了交换率、速度、有效性、准确性、响应时间、恢复时间和内存使用，同时还加上了必须执行某个活动的条件。

■ 可支持性(supportability)

- 可支持性需求是指易测性、可维护性和其他在系统发布后维持系统更新需要的质量。

- 实际上，争论一个特定需求是使用性需求还是可支持性需求没有任何价值。
- 定义需求类型的目的是为找到需求和评估对需求理解的完整性提供一个模板
- 一旦理解了这些需求的分类，那么在投入到开发工作之前就能够对关键性需求有了深刻的理解
- 某个特定的可靠性需求可能暗含于某个功能性需求中，而发现这个可靠性需求与发现这个功能性需求同样重要
- 没有满足隐含的可靠性需求或性能需求的系统与没有满足明显的功能性需求的系统一样不好

9.3 需求的种类

- 项目相关人员的需求
- 系统特性的需求
- 软件需求

项目相关人员的需求

- 项目相关人员：是指对一个工程的结果有投入或有既定兴趣的任何人或组织的代表
 - 如最终用户、购买者、订阅人、开发人员、项目经理
 - 任何足够关心这个项目的人
 - 项目需要满足其需求的人
- 如何获取这类需求？
 - 早期通过访谈、调查问卷和专题讨论会等形式收集
 - 收集变更请求、升级请求和缺陷报告
 - 这类需求为产品需求提供了关键输入，由此决定什么是系统行为和它为什么是系统行为

系统特性的需求

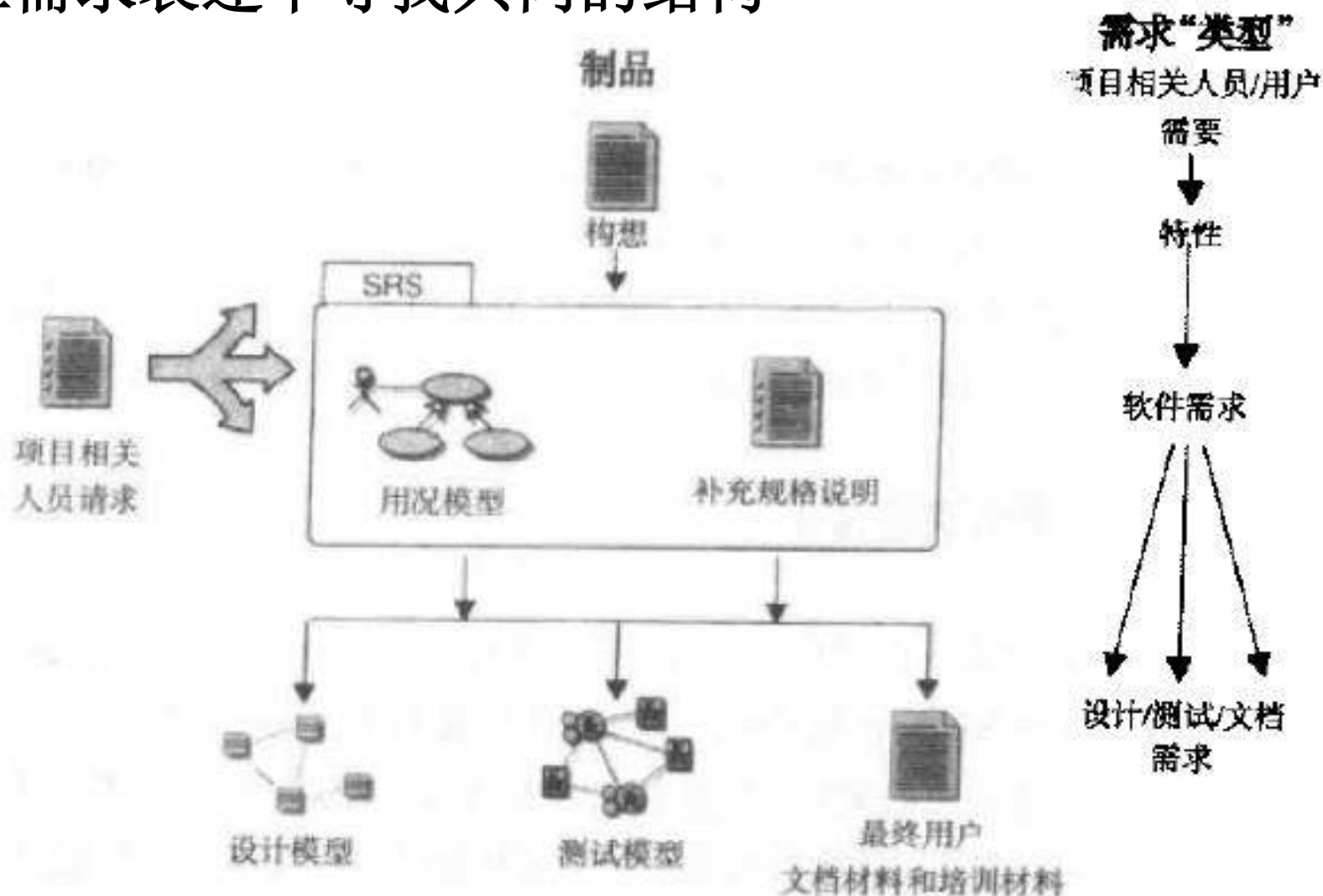
- 系统特性：是指对系统行为高度抽象的描述
 - 抽象描述：我希望我的车有自动防滑系统
 - 具体描述：我希望我的车能有一个可以控制每个轮子的计算机控制系统
 - 抽象描述：需要一个“e-mail自动提示”
 - 具体描述：我需要和Joe保持实时交流
- 从技术上说，特性是系统提供的能直接满足用户需求的服务
- 特性的获取要通过自然语言讨论并同项目相关人员进行交流
- 特性上要附加不同的属性——例如风险、工作量、客户优先级，从而增加了对系统的理解和认识

软件需求

- 仅仅理解项目相关人员的需求和系统特性都不足以与开发人员交流系统究竟应该做些什么
- 还需要一个附加规格说明，将这些需求和特性翻译成为可以设计、实现和测试的详细说明，从而验证系统的一致性
- 规格说明中必须处理系统的功能性需求和非功能性需求

9.4 捕获需求和管理需求

■ 在需求表述中寻找共同的结构





■ 1 收集项目相关人员的请求

- 包括从最终用户、客户和市场及其他项目相关人员处得到的所有的请求和希望清单

■ 2 开发构想文档

- 构想文档中要包括项目相关人员和用户的关键需要, 以及系统的高级特性
- 构想文档中包括的特性取决于对于实现特性所需成本的分析以及对于这项投入所得利润的预测

■ 3 细化软件需求

- 我们可以在用例模型和补充规格说明中捕获详细需求，补充规格说明中应捕获那些不适用于用例的其它需求
- 详细需求在设计模型和最终用户文档中实现，并在测试模型中验证

- 需求 workflows 要以迭代的形式不断重复，直至明确了所有的需求，也考虑了所有反馈，并做出了必然的变更

9.5设计以用户为中心的界面

- 用户界面设计指的是至少以下两件事之一：
 - 可视化地确定用户界面,以使它可以处理不同使用需求
 - 从设计类(和构件,例如**ActiveX**类和**JavaBeans**)角度进行用户界面设计, 这些设计类与其他处理业务逻辑、一致性等问题的设计类相关联, 这样的设计会引导最终用户界面的实现
- 在需求工作流程中, 我们在以上定义的第1种情况下进行操作, 力图完成一个以用例为中心的设计

9.6 需求 workflow

分析问题

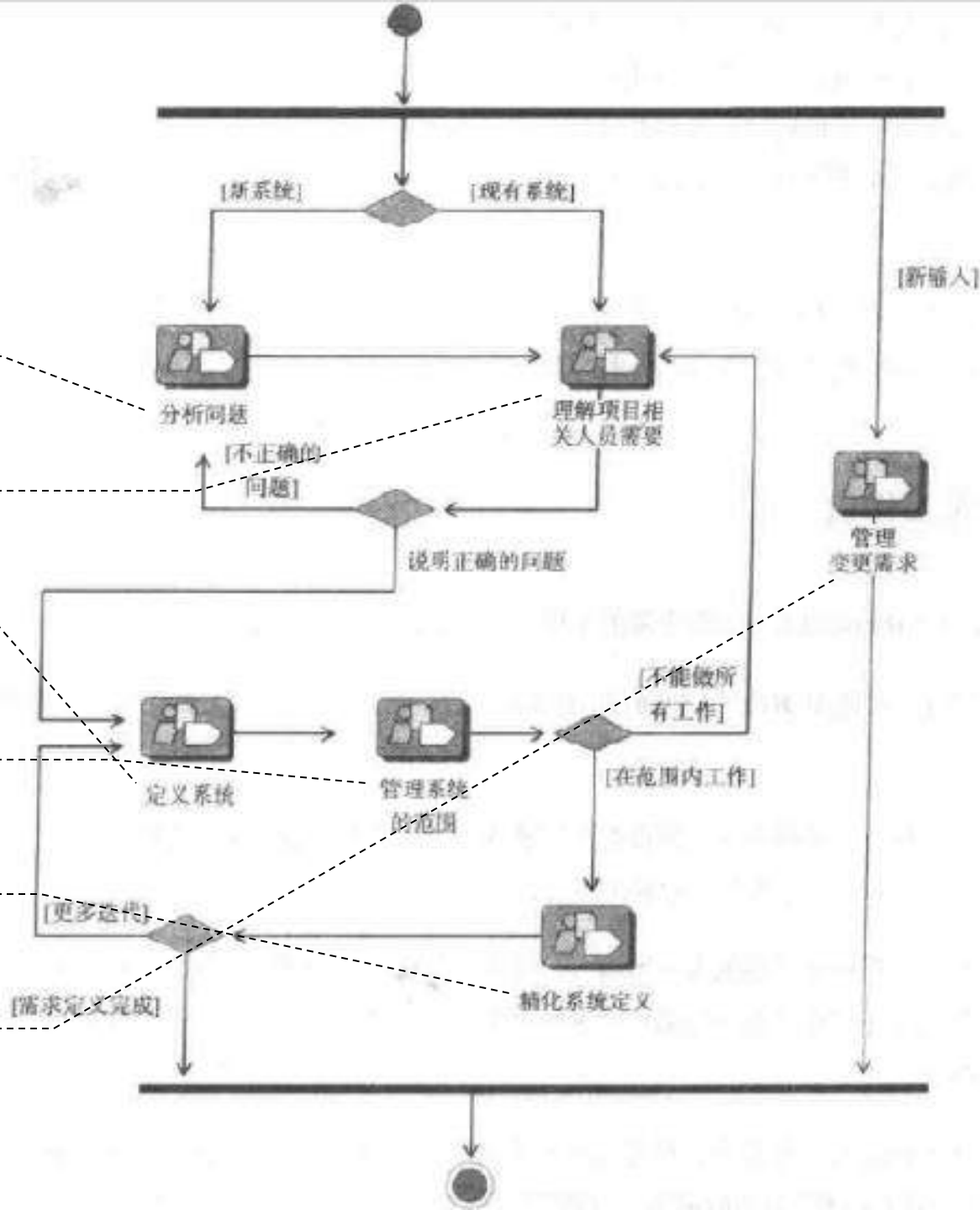
理解项目相关人员的需要

定义系统

管理系统的范围

精化系统定义

管理变更需求





■ 分析问题

- 对需要解决的问题陈述达成一致；
- 确定项目相关人员；
- 确定系统的边界和约束

■ 理解项目相关人员的需求

- 使用不同的抽取技术，收集项目相关人员的请求
- 清楚地理解系统用户和项目相关人员的真正需求

■ 定义系统

- 根据项目相关人员的输入，明确要交付的一系列系统特性
- 确定用于制定优先级的准则，并与项目相关人员合作来预计要交付的特性
- 确定每一个关键特性需要的参与者和用例。

■ 管理系统的范围

- 收集项目相关人员掌握的项目重要信息，将它们作为需求属性，同需求一起进行维护，从而可以根据客户的期望按时、按预算交付产品。

■ 精化系统定义

- 通过用例模型细化系统软件需求，就系统需要具备什么样的功能这一问题上与客户达成一致
- 捕获其他重要需求，如非功能性需求、设计约束等。

■ 管理变更需求

- 应用需求属性和可跟踪性来评估变更需求所产生的影响
- 应用一个中央控制机制控制需求的变更
- 维持与客户达成的一致，并预计哪些特性要交付

9.7 需求分析中的工作人员





■ 系统分析员

- 大致描述系统的功能和界定系统，领导并协调需求抽取和用例建模

■ 用例描述员

- 通过一个或多个用例描述需求，细化系统的所有功能或部分功能

■ 用户界面设计师

- 负责挑选用例，以显示用户和系统之间的基本交互作用。同用户一起工作，将这些用例开发为用例情景板，并开发一个用例界面原型以帮助确定系统的最终需求

9.8 需求分析中使用的制品

- 构想文档
- 用例模型
- 补充规格说明
- 术语表
- 用例情景板
- 用户界面原型

后三个制品是对前三个制品的有益补充

构想文档

- 详细说明主要的需求和特性
 - 构想文档是从客户角度写成的，关注系统基本特性和可接受的质量水平
 - 同时说明操作能力、用户概况以及与系统外部实体交互作用的可应用接口
- 为要开发的软件系统提供了完整的构想，并支持资金筹备方和开发组织之间的契约
 - 构想文档提供给项目人员需求可视化的契约性基础

用例模型

- 充当交流媒介，在客户、用户和系统开发人员之间充当关于系统功能的契约
 - 使客户和用户确定系统向他们所期望的方向发展
 - 使系统开发人员开发出所期望的系统
- 用例模型由用例和参与者组成
 - 用例显示了系统是如何与参与者进行交互的，以及系统在用例中做了什么
 - 在系统分析、设计、实现和测试中还会使用同样的用例模型

补充规格说明

- 是用例模型的重要补充，两者相加才能捕获所有需要描述的系统需求(功能性需求和非功能性需求)，才能充当完整的软件需求说明

9.9 工具支持

- **Rational RequisitePro**提供了需求捕获和需求管理支持，以含有重要属性的文档和需求知识库的形式出现，用于管理需求范围和需求变更
- **Rational Rose**在用例模型、用例情景板和边界类中给参与者和用例提供了自动支持
- **Rational SoDA**是一种自动生成文档工具。

9.10 小结

- 需求管理需要项目相关人员和开发团队在系统应该做什么的问题上达成协议，并由专门团队进行维护
- 对于典型的项目，要考虑几种类型的需求，包括高级特性、详细的功能性需求和非功能性需求以及用例
- 维护需求属性和可跟踪性是在整个项目生命周期中有效地管理项目范围和处理变更需求的关键
- 用户界面设计关注关键需求和用户目标，以建立一个以用户为中心的系统并满足使用性需求
- **Rational**工具支持捕获需求、可视化建模、需求管理及其属性和可跟踪性



第10章 分析和设计 workflow

目录

- 10.1 目的
- 10.2 分析与设计
- 10.3 到底要设计到什么程度
- 10.4 工作人员和制品
- 10.5 设计模式
- 10.6 分析模型
- 10.7 接口扮演的角色
- 10.8 实时系统的制品
- 10.9 基于构件的设计
- 10.10 workflow
- 10.11 工具支持
- 10.12 小结

10.1 目的

- 分析和设计工作流的目的是将需求翻译为规格说明,它描述如何实现系统。
- 即将系统需求转变为一组类和子系统的形式

在项目的早期, 要建立一个强壮的框架, 从而可以设计一个易于理解、开发和进化的系统。然后调整设计, 使之适应实现环境, 并设定它的性能、强壮性、可升级性、可测试性以及其他质量特性。

10.2 分析与设计

- 分析关注的是要确保处理系统的功能性需求。
- 为了简化，在分析时忽略了许多非功能性需求，以及实现环境的约束。
- 分析最终展现的是一幅近乎理想的系统画面
- 设计的目的是使分析的结果适应于非功能性需求、实现环境等带来的约束。
- 设计是分析的精化。
- 设计关注的是要确保完全覆盖需求以及系统设计的优化。

10.3 到底要设计到什么程度

- 设计必须正好足够定义该系统，才能无歧义地实现该系统
 - “足够”随着项目的不同而不同，随着公司的不同而不同
- 设计可能会细化到使代码完全依赖设计而实现
- 设计也可能仅仅勾画一个草图，即细化到使实现人员了解需要开发哪些构件

■ 规格说明的详细程度

- 取决于实现人员的专业知识和技能、设计的复杂程度，以及由于可能误解设计而带来的风险

■ 双向工程

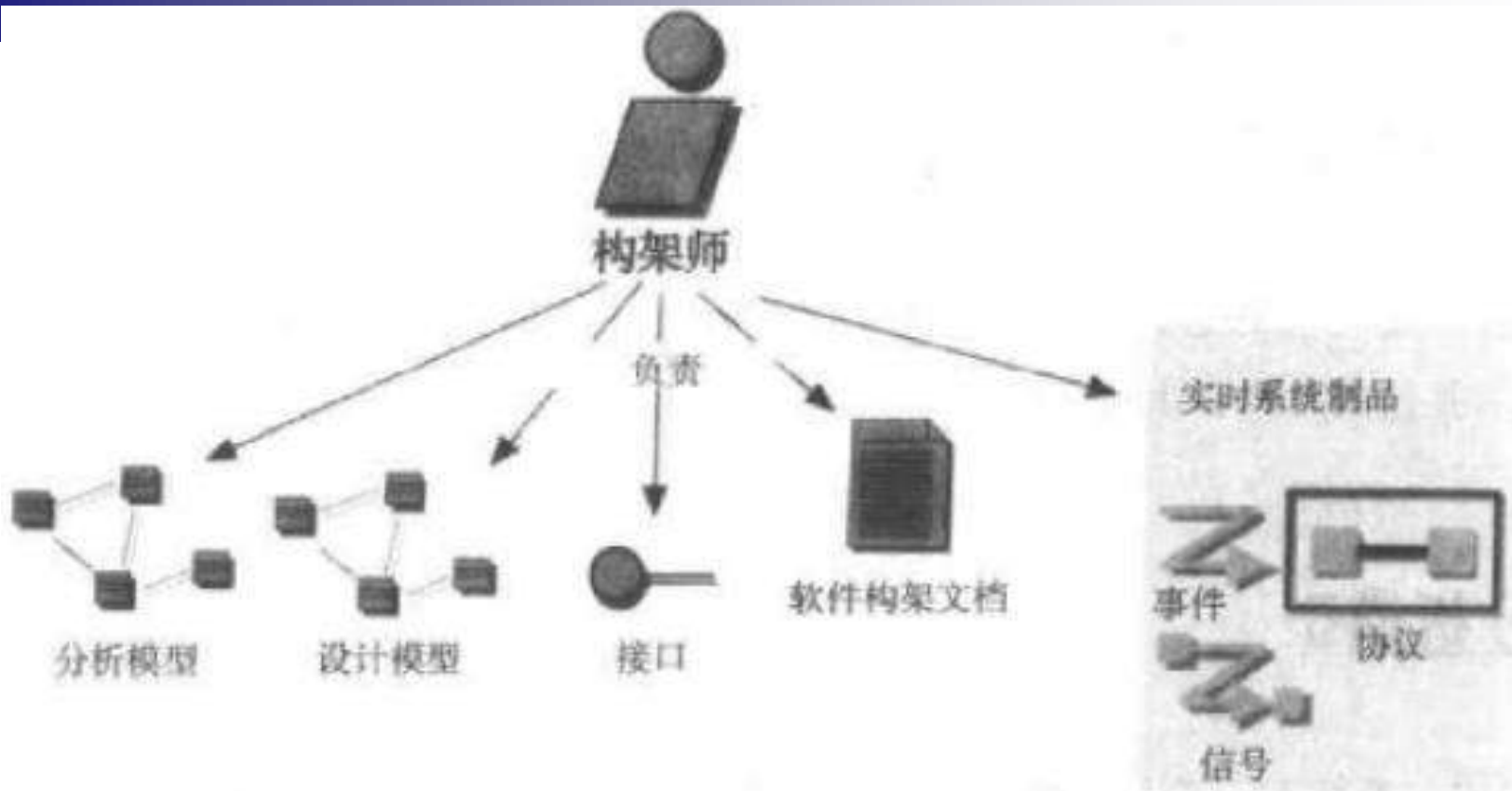
- 如果设计非常精确，代码就可以完全依赖于设计，就可以采用双向工程的方法使代码与设计保持同步，从而避免了一个转换步骤和一个潜在的错误来源

设计是可执行的

- 如果设计的完整程度和精确程度都非常高，以至于可以通过解释设计或者从设计快速生成少量代码来直接执行变换，使这种变换对于设计师几乎是不可见的，从而使设计看起来就是“可执行的”

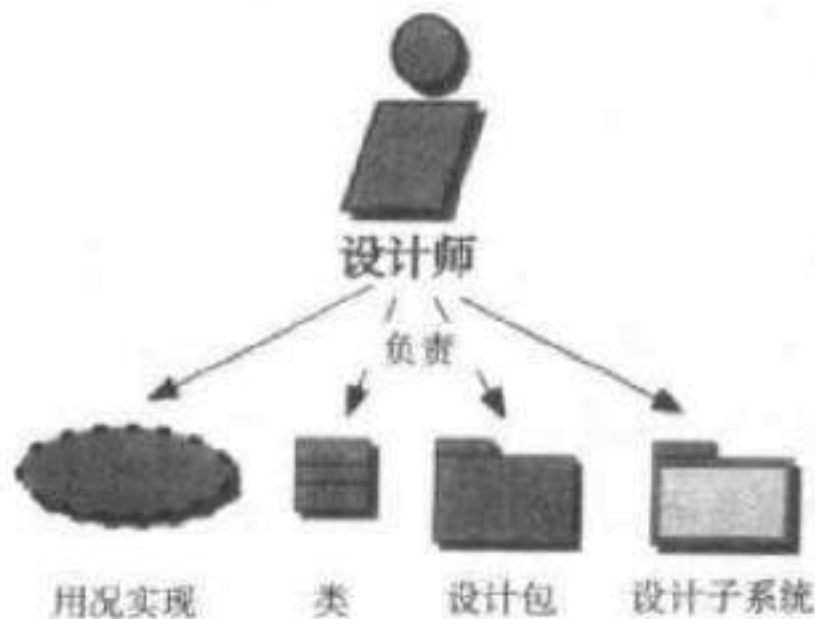
10.4 工作人员和制品

- 分析和设计工作流程中涉及到的主要工作人员如下：
 - 构架师
 - 设计师
- 还可能包括以下工作人员：
 - 数据库设计师
 - 封装设计师
 - 构架评审员和设计评审员



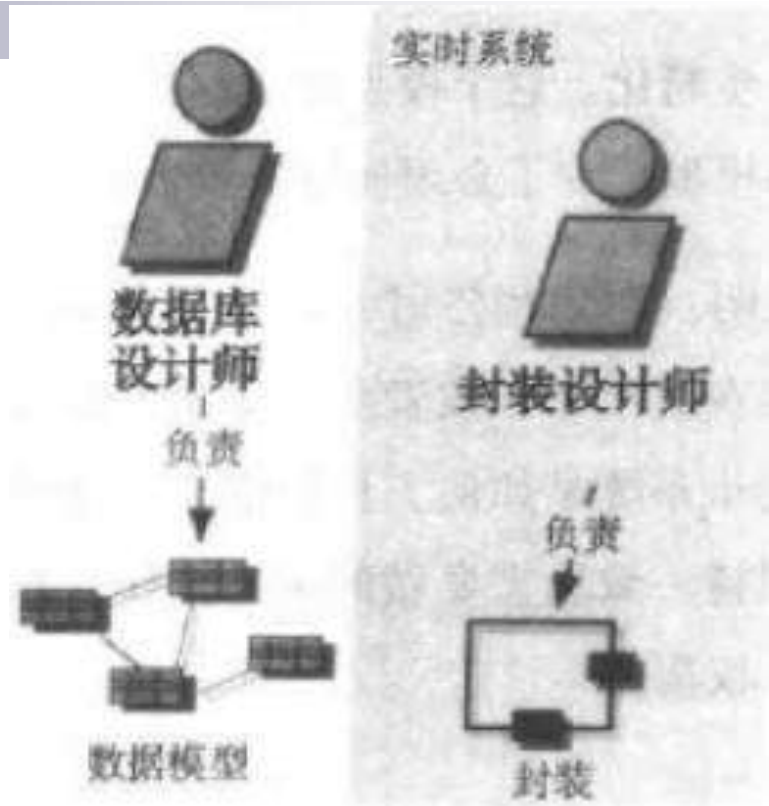
■ 构架师

- 构架师在整个项目中领导和协调技术活动和制品。
- 构架师为每个构架视图建立总体结构：视图的分解，元素的分组，以及各主要分组之间的接口的视图
- 构架师的视图要更注重宽度而不是深度



■ 设计师

- 设计师定义了职责、操作、属性、一个或多个类之间的关系，并决定如何调整它们使之适应实现环境。
- 设计师还要负责一个或多个设计包或设计子系统，包括所有属于这些包或这些子系统的类。



■ 数据库设计师

- 当系统中包含数据库时，就需要数据库设计师

■ 封装设计师

- 主要任务是通过适当地使用当前的设计工具，确保系统可以及时地对事件做出反应

■ 构架评审员和设计评审员

- 负责评审在分析和设计工作流程中生产的关键制品

分析和设计工作流程中的关键制品包括

- 设计模型

- 是在构造阶段的主要系统蓝图

- 软件构架文档

- 涵盖了系统的不同构架视图

10.5 设计模型

- 设计模型主要由类的协作组成，这些类的协作可能集成为包和子系统，以帮助组织模型并提供模型中的组合构造块
 - 类是一组共享相同职责、关系、操作、属性和语义的对象描述
 - 包是对类的逻辑分组，这是为了组织起来方便，以减少系统的复杂程度
 - 子系统是包的一种，由作为独立单元、提供特定系统行为的一组类组成

10.6 分析模型

- 分析模型描述了系统针对应用的方面，是一个设计的抽象或泛化
- 分析模型忽略系统是如何工作的细节，提供系统功能的概况
- 分析模型和设计模型要保持一致
- 对于某些公司，如果其系统要存在数十年，或者要有很多不同的系统，那么有一个独立的分析模型非常有用

10.7 接口扮演的角色

- 接口以一种独立于实现行为的方式描述由类、子系统和构件提供的行为
- 接口描述了一系列由模型元素执行的操作，包括返回的类型和参数的类型和数目
- 提供同样接口的模型元素可以互换
- 接口可以降低系统各部分之间的依赖性，改善设计的灵活性

10.8 实时系统的制品

- 实时系统是那些对时间或响应有特殊需求、并通常对最终期限有约束的系统
- **RUP**介绍了以下几个专门为满足开发实时系统需要的附加制品：
 - 封装
 - 协议
 - 事件
 - 信号

10.9 基于构件的设计

- 构件是实现模型中的元素，它用来表达通过一系列较小的“块”实现系统的方法
- 构件提供了一个或多个接口，并且仅仅依赖于其它构件的接口
- 与构件相对应的设计模型元素是子系统：子系统是独立的功能单元，允许独立设计和实现系统的不同部分。子系统提供了接口，并依赖于其它模型元素的接口

10.10 workflows

分析和设计中的工作流

定义候选构架

精化构架

分析行为

设计构件

设计实时构件

设计数据库



定义候选构件

- 由构架师执行的**构架分析**和由设计师执行的**用例分析**等活动组成。它的目的是：
 - 建立一个初始的系统构架草图
 - 从重要的构架用例中确定分析类
 - 确定分析类之间的交互作用，更新用例实现

精化构架

- 由构架师执行的识别设计机制、识别设计元素、合并已有设计元素、描述运行构架和描述分布等活动以及由构架评审员执行的评审构架活动组成。它的目的是：
 - 提供从分析活动到设计活动的自然转移
 - 保持构架的一致性和完整性
 - 描述系统运行的组织和实施构架
 - 组织实现模型以便在设计 and 实现之间实现无缝转移

分析行为

- 由设计师执行的用例分析活动、由构架师执行的识别设计元素活动以及由设计评审员执行的评审设计活动等组成。它的目的是：
 - 将用例提供的行为描述转变为一系列可作为设计基础的元素
- 在分析行为时，较少关心系统的非功能性需求，而更加注重如何交付需要的应用能力

设计构件

- 由设计师执行的用例设计、类设计和子系统设计活动以及由设计评审员执行的评审设计活动所组成。它的目的是：
 - 找到设计元素实现要求的行为细节，以精化关于设计元素的定义
 - 基于新的设计元素精化并更新用例实现，以确保用例实现总是最新的
 - 当设计进化后，评审该设计

设计实时构件

- 这个细节应用在实时或交互系统的语境中，它使用封装制品作为主要的设计元素，还包括封装设计活动。
- 它的目的与设计构件相似。

设计数据库

- 由数据库设计师执行的**设计数据库活动**、设计师执行的**类设计活动**以及设计评审员执行的**评审设计活动**组成。它的目的是：
 - 在该设计中标识永久类
 - 设计合适的数据库结构以存储永久类
 - 定义一种存储和检索永久数据的机制和策略，以满足该系统的性能准则

10.11 工具支持

- 用于捕获、管理、展示这些模型的工具是 **Rational Rose**。
- **RUP**还提供了以下工具指南以指导设计师使用**UML**和**Rose**:
 - **Rose RealTime**可以直接执行设计模型
 - **SoDA**可以自动生成文档和报告

10.12 小结

- 分析和设计是需求和实现之间的桥梁。分析和设计 workflow 使用用例来标识一系列后来被精化成为类、子系统和包的对象
- 分析和设计中的职责分配给了构架师(处理蓝图问题)、设计师(处理细节问题)和数据库设计师
- 分析和设计产生设计模型，它可以抽象为3个构架视图。
 - 逻辑试图将系统分解为一序列逻辑元素(类、子系统、包和协作)。
 - 过程视图将这些元素转变为系统的过程和线程。
 - 实施视图将这些过程转变为一组节点
- 在某些情况下，一个独立的分析模型对于表示系统的概况或者系统的抽象是非常有用的



第11章 实现 workflow



目录

- 11.1 目的
- 11.2 构造
- 11.3 集成
- 11.4 原型
- 11.5 工作人员和制品
- 11.6 workflow
- 11.7 工具支持(略)
- 11.8 小结

11.1 目的

- 实现工作流有以下四个目的:
 - 为了从层次组织实现子系统的角度定义代码组织
 - 为了从构件(源文件、二进制文件、可执行文件以及其他文件)的角度实现类和对象
 - 为了将要开发的构件看作一个单元进行测试
 - 为了将个人或开发团队开发的结果与可执行系统进行集成

实现 workflows 与测试 workflows

- 实现 workflows 中的测试范围限制在单个构件的单元测试上。
- 系统测试和集成测试工作则在测试 workflows(第12章)中描述。
- 为了解释RUP中的实现 workflows，下面介绍以下三个关键概念：
 - 构造(build)
 - 集成(integration)
 - 原型(prototype)

11.2 构造(build)

- 构造是系统的可操作版本或者是该系统的一部分。
- 每个构造提供一个早期评审点，并帮助在集成问题出现时及时发现问题。
- 构造展示了迄今为止开发完成的功能。
- 假如新加入的功能导致系统的破坏,或者威胁到构造的整体性,这时就需要回到以前的版本,所以每个构造都处于配置控制下。
- 通常,项目每隔一定间隔产生一个构造

11.3 集成(Integration)

- 集成是指将几个独立的软件构件结合成一个整体的软件开发活动
- 集成是在实现的不同水平和不同阶段上进行的，其目的如下：
 - 在将子系统交给系统集成人员之前，要将在同一个子系统上工作的开发组里的工作成果集成起来
 - 将子系统集成为一个完整的系统
- 在每一个迭代中要进行的集成是非常重要的。迭代计划中不仅定义了要设计的用例和要实现的类，还定义了集成策略
 - 集成策略关注实现和集成类的顺序。

集成的种类

■ 增量式集成

- 一小块一小块地编写代码并测试代码，然后通过每一次增加一小块将他们结合成一个整体
- **RUP**的集成方法

■ 阶段式集成

- 是指同时集成多个(新的和已变更的)构件
- 缺点是很难对错误进行定位

增量式集成的优点

- 容易定位故障
- 更加充分地测试构件
- 系统的某些部分提前运行

11.4 原型

- 原型用于以一种直接的方式减少风险。
- 原型可以通过给用户、客户和经理展现一些具体的、可执行的实体，帮助建立对该产品的支持
- 原型必须在它的整个生命周期中保持非常明确的性质和目的
- 为展现用户接口而设计的行为原型很少能发展成为一个强壮的、有弹性的产品

原型的种类

- 根据原型要探测什么，可以将其分为：
 - 行为型原型：探测系统的某一特定行为
 - 结构型原型：探测构架或者技术上关注的问题
- 根据原型的进化，或者说进化的结果，可以将其分为：
 - 探测型原型：也叫抛弃型原型，在完成之后要将它抛弃
 - 进化型原型：将进化成为最终系统

行为型原型

- 这类原型趋向于成为探测型原型，它不是待开发系统构架的复制
- 主要关注于用户看到的系统将要做什么
- 这类原型属于“快速研制但并不精巧”，它并不是按照项目的标准制作的

结构型原型

- 这类原型趋向于成为进化型原型；它们更有可能使用最终系统的基础设施(“骨架”), 并且很有可能发展成为最终系统
- 这类原型可以帮助测试开发环境和帮助相关人员熟悉新的工具和规程
- **RUP**支持在整个细化阶段使用一个进化的结构型原型，同时伴随任意数量的探测型原型

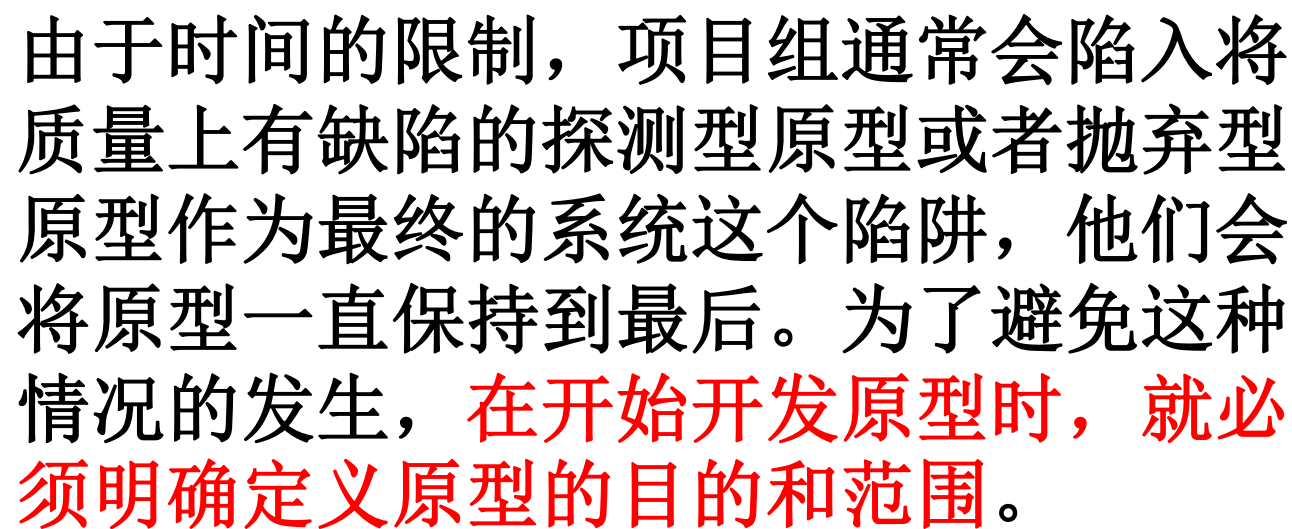

探测型原型与进化型原型

■ 探测型原型

- 这类原型用于测试关于功能或技术或两者兼而有之的一个关键假设
- 探测型原型可能非常小，例如一个用来测试软件的关键构件的几百行小程序
- 这类原型也可以用来弄清楚需求(行为型测试)

■ 进化型原型

- 这类原型是从一个迭代到另一个迭代中进化形成的

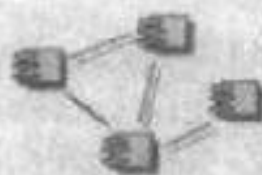


由于时间的限制，项目组通常会陷入将质量上有缺陷的探测型原型或者抛弃型原型作为最终的系统这个陷阱，他们会将原型一直保持到最后。为了避免这种情况的发生，**在开始开发原型时，就必须明确定义原型的目的和范围。**

11.5 工作人员和制品



构架师



实现模型



系统集成人员



集成构造计划



代码评审员



实现人员



构件

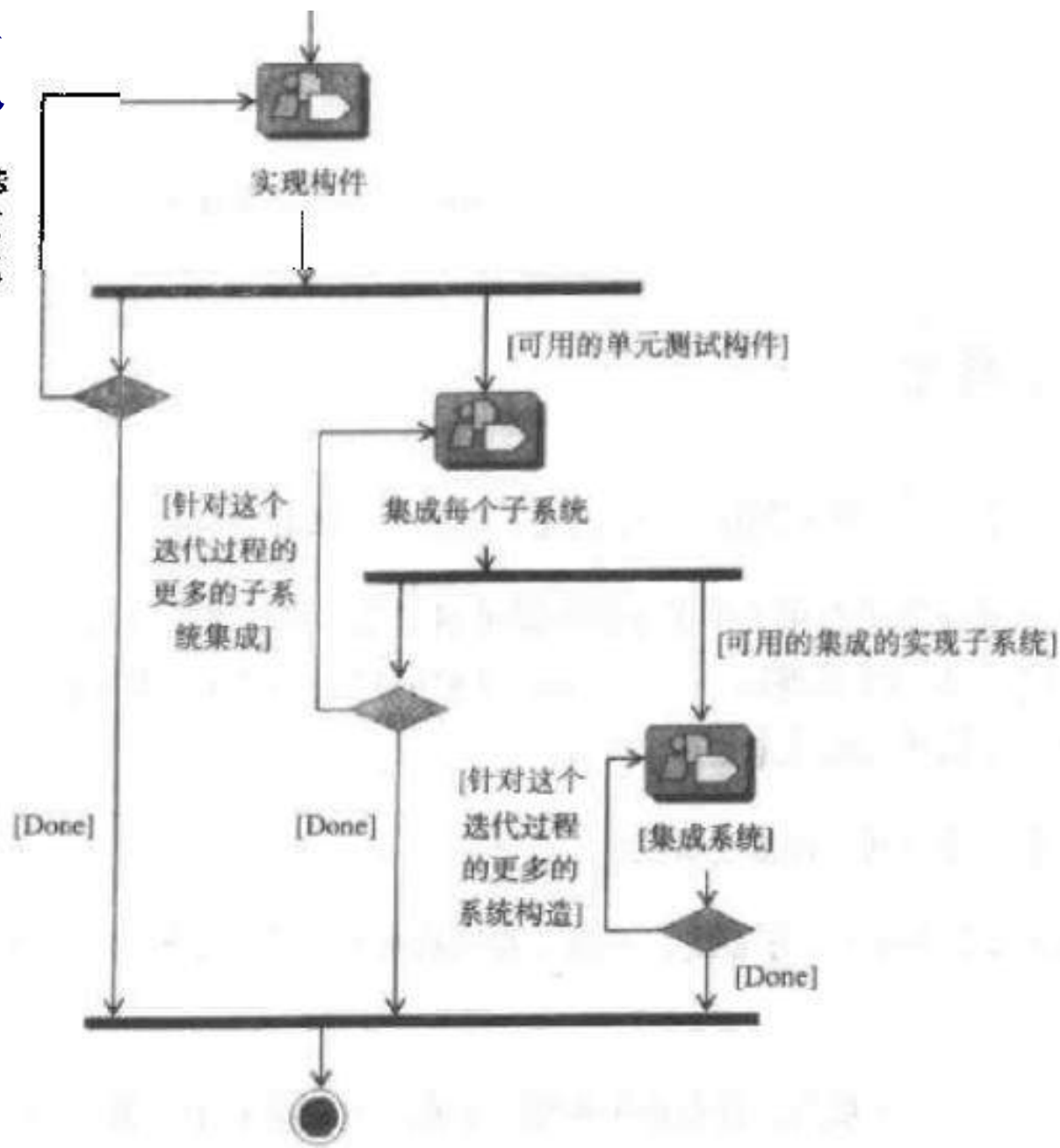


实现子系统

11.6 workflow



[针对这个迭代过程的更多的要实现的构件]



构造实现模型

- 构造实现模型的主要工作在细化阶段早期完成。
目标是尽可能以一种开发构件和构造过程无冲突的方式来组织实现模型
- 一个组织良好的模型可以预防配置管理问题的出现，并允许不断用更大规模的集成构造来建造该产品

对于每一个迭代

- 确定要实现哪一个子系统，并确定在当前迭代中子系统集成的顺序
- 对于每一个子系统，要计划子系统的集成，也就是要定义实现这些类的顺序
- 实现类和对象。如果发现缺陷，要向设计递交返工反馈
- 修改有缺陷的代码，执行单元测试以验证这些变更，而后对代码进行评审

- 若多个实现人员实现同一个子系统，则要有一个
人负责把来自其他实现人员的新的和已变更的构
件集成为一个新的实现子系统版本。
 - 这种集成产生了一系列构造，每个构造都由集成测试
人员进行测试。所发布的最新版本子系统是系统集
成的输入
- 根据集成构造计划，系统集成人员将发布的那些
子系统加入到集成领域并创建一些构造来集成系
统。
 - 在最后一个增量交付之后，该构造就是系统，并由系
统测试人员进行测试

11.8 小结

- **RUP**的关键特性是在整个生命周期中进行增量式集成
- 在构造阶段，建立一个能够成为最终系统的进化型和结构型原型
- 并行地建立几个可抛弃的行为型原型，以探测一些例如用户界面方面的问题



第12章 测试 workflow

目录

本章介绍关于质量的概念，描述测试 workflow，然后讨论在过程中质量、测试和其他 workflow 之间的关系。

- 12.1 目的
- 12.2 质量
- 12.3 在迭代生命周期中进行测试
- 12.4 测试的层面
- 12.5 测试模型
- 12.6 工作人员和制品
- 12.7 工作流
- 12.8 工具支持(略)
- 12.9 小结

12.1 目的

- 测试的目的是评估产品质量。
- 测试不仅仅针对最终产品，它开始于项目早期对构架的评估，并一直延续到将最终产品交付给客户
- 测试 workflow 涉及以下几点：
 - 验证构件之间的交互作用
 - 验证构件集成是否适当
 - 验证所有的需求是否已经正确地实现
 - 确定并保证所有已发现的缺陷在软件实施之前已经修复

12.2 质量

- 质量是指没有缺陷，并达到了预期的目的
- 测试的最终目标是评估最终产品的质量。为此，还要评估组成这个产品的构件的质量以及构架的质量
- 评估的意义是：根据事先定义好的并被接受的度量 and 准则进行评估，以确保产品满足或者超出了事先定义并被普遍接受的需求
- 质量评估通常既要考虑直接产品质量，也要考虑过程质量和组织因素

产品质量所有权

- 质量要在一开始就设计和建立在产品之中
- 测试的角色不是保证质量，而是评估质量，并及时地提供反馈，以便及时而又经济地解决质量问题
- 测试人员的角色是评估质量并提供反馈
- 项目组的角色是生产满足需求和质量期望的制品或工作产品

12.3 在迭代生命周期中进行测试

- 测试不是单一的活动，也不是项目中的一个评估质量阶段
- 开发人员要在整个生命周期中进行测试
 - 测试早期原型的主要功能
 - 测试构架的稳定性、覆盖率和性能
 - 修复发现的缺陷
 - 测试最终产品

12.4 测试的层面

- 为了评估产品质量，需要不同类型的测试，每种测试都有不同的重点。这些测试可以分为以下几个角度：
 - 质量层面：关注的是主要的质量特征或属性
 - 测试阶段：在生命周期中执行测试的点
 - 测试类型：单独测试的特定测试目标，通常限制在一个单一的质量层面上

质量层面

- 每一个产品都要对以下几个方面进行评估
 - 可靠性。软件对执行失败时所具有的抵抗能力：没有崩溃、不会死机、不会内存泄露等
 - 功能性。软件执行了需要的用例或者预期的行为
 - 性能。当操作负载、操作强度和操作时间过长时，软件和系统要及时地执行和响应并能继续以用户可接受的方式完成任务
- 此外，还包括可维护性、可延伸性、灵活性等

测试阶段

- 测试在软件开发的不同阶段具有不同的测试目标。包括以下四个阶段：
 - **单元测试**：逐一测试系统最小的可测试元素，通常是在实现这些元素的同时测试这些元素
 - **集成测试**：测试集成单元、构件或子系统
 - **系统测试**：测试整个应用程序和系统(一个或多个应用程序)
 - **接受测试**：由最终用户测试整个应用程序(或系统)，其目的是确定是否已经作好实施的准备

在生命周期的初始化阶段和细化阶段也要进行测试工作

- 初始化阶段要对用于评估产品构想生存能力的早期概念原型进行接受测试
- 细化阶段要对开发的构架原型进行集成测试和系统测试，以确认构架的整体性和关键构架元素的性能

测试类型

- 每一种测试类型关注一个特定的测试目标，并且只测试软件的一个特性或属性。
 - **基准测试**：将测试目标的性能和一个已知标准进行比较
 - **配置测试**：验证测试目标的功能在不同配置下(软件或硬件)是否都可以接受
 - **功能测试**：验证测试目标功能是否适当地执行了所需要的、期望的用例
 - **安装测试**：验证测试目标的安装是否合适，是否可以在不同的配置和不同条件下成功地安装

- **整体性测试**：验证测试目标的可靠性、健壮性和在执行过程中的抗失败能力
- **加载测试**：验证测试目标的性能在配置不变、操作条件不同(如用户数量、处理的业务数量)的情况下的可接受性
- **性能测试**：验证测试目标的性能在操作条件不变、而使用不同配置的情况下的可接受性
- **强度测试**：验证测试目标的性能当遇到不正常或极端条件时(如减少可用资源或用户数量极多等)的可接受性

回归测试

- 回归测试是一种测试策略，是对测试目标的新版本再一次执行以前执行过的测试，以确保目标的质量在增加了新的能力的情况下并没有倒退
- 回归测试的目的是保证
 - 在上一个测试执行中发现的缺陷已经更正
 - 代码的改变没有引入新的缺陷，也没有再次引入老的缺陷

12.5 测试模型

- **测试模型:**描述了将要测试什么以及将要如何测试
 - 测试模型描述了测试工作本身以及与测试工作量有关的测试目标的各个方面
 - 包括测试用例、测试规程、测试脚本和期望的测试结果以及关于它们之间关系的描述

- **测试用例:**为了一个特定的测试目标开发的测试数据、执行条件和期望的测试结果的集合

- 一个测试用例可以由一个或多个测试规程实现

- **测试规程:**用于测试用例的启动、执行和测试结果的评估的详细指令的集合

- 一个测试规程实现一个或多个测试用例，也有可能只实现测试用例的一部分

- **测试脚本:**自动执行测试规程的计算机可读指令
 - 一个测试脚本自动执行一个或多个测试规程的整体或一部分
- **测试类和构件:**实现测试设计的类和构件
- **测试协作:**用协作图或时序图表示, 描述了在测试中构件和测试目标之间产生的时间顺序消息流
- **注释:**描述了测试模型中的约束和附加信息的文本信息

测试用例

测试规程

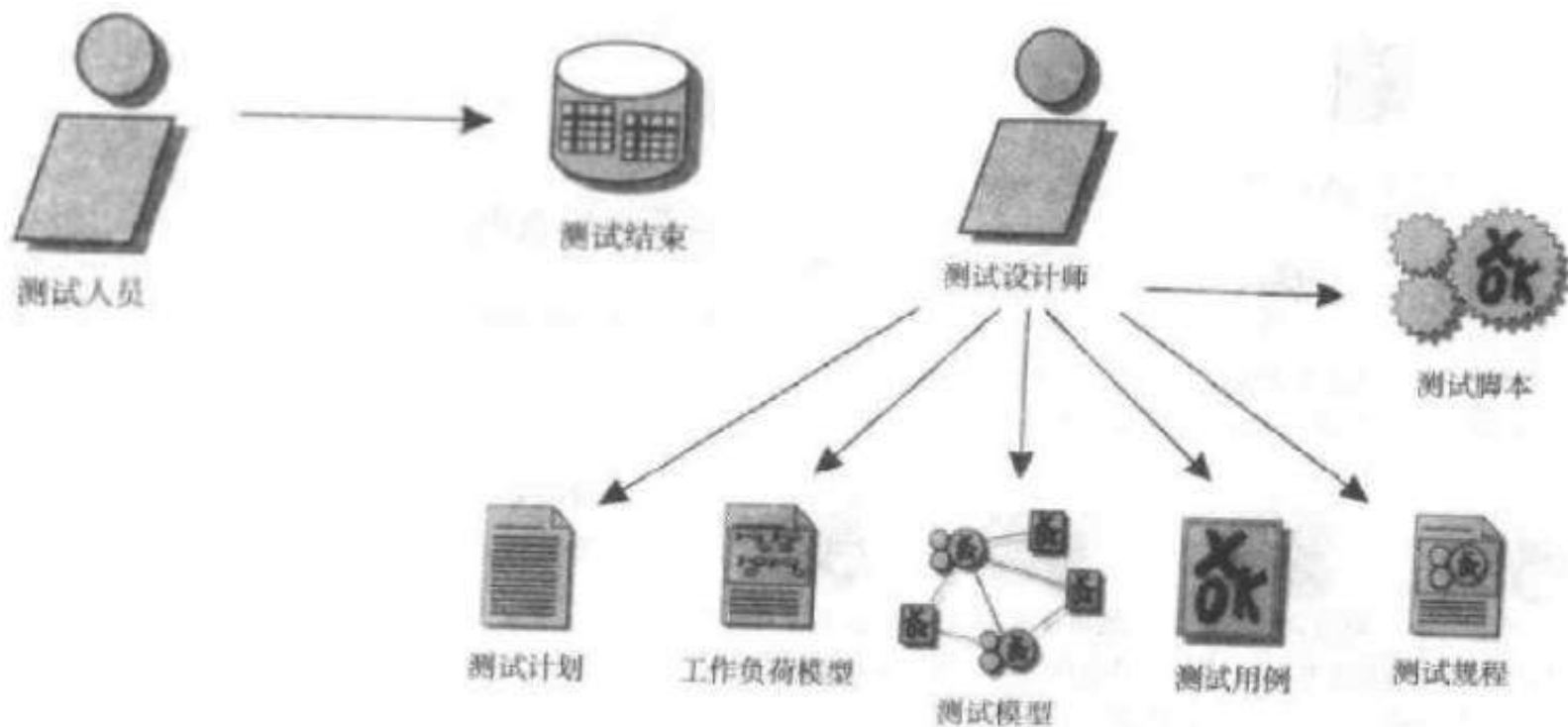
测试脚本
(个人)

测试脚本
(测试套件)



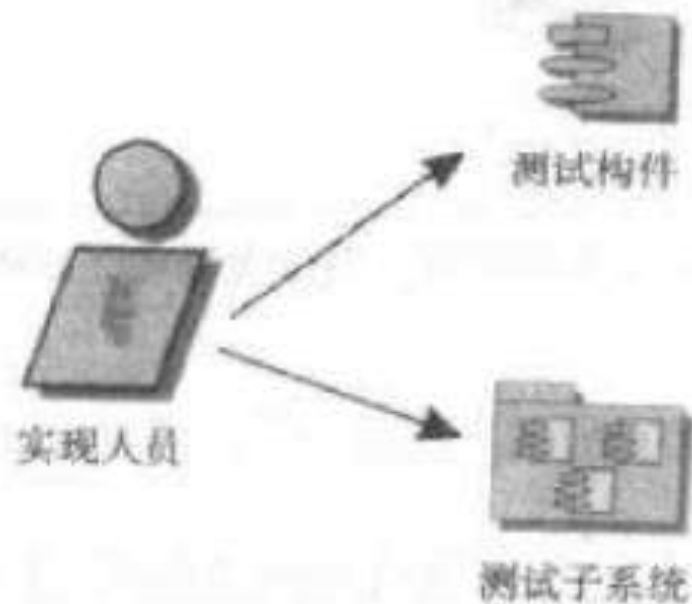
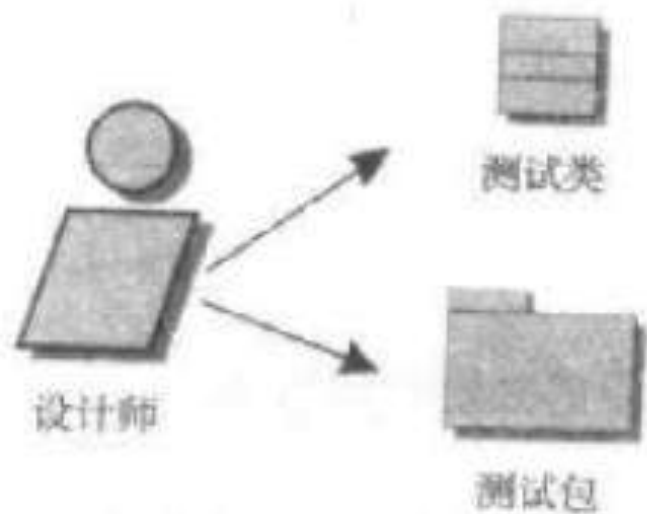
12.6 工作人员和制品

- 测试工作流程中涉及两个主要的工作人员
 - 测试设计师：负责计划、设计、实现和评价测试
 - 测试人员：负责执行系统测试



其他工作人员

- 设计师
- 实现人员

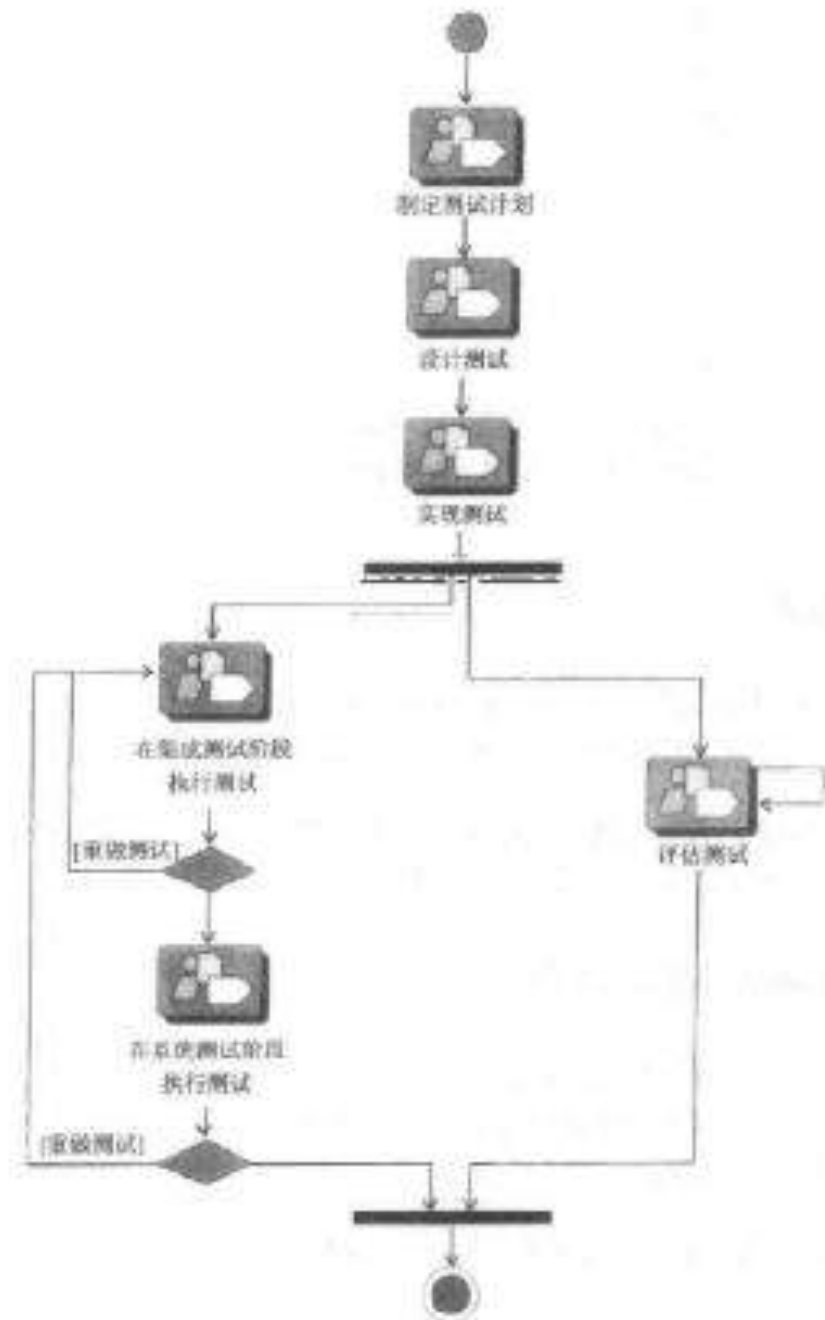


关键制品

- 测试计划
- 测试模型
- 测试结果性能测试的工作负荷模型
- 缺陷
- 当必须开发测试专用的代码时会产生以下制品
 - 测试包和测试类
 - 测试子系统和测试构件

12.7 workflow

- 制定测试计划
- 设计测试
- 实现测试
- 在集成测试阶段执行测试
- 在系统测试阶段执行测试
- 评估测试



■ 制定测试计划：目的是确定并描述要实现和执行的测试。

- 可以开发一个包含要实现和执行的各种测试类型的单一的测试计划，或者可以针对每种测试类型开发一个测试计划

■ 设计测试：目的是确定、描述和生成测试模型以及报告中提到的制品

- 确保测试软件满足需求，并有良好的结构和组织。测试设计将用例转变为可接受的单元测试用例、集成测试用例和系统测试用例

■ 实现测试：目的是实现在设计测试中定义的测试规程

- 测试规程通常在测试自动工具环境或程序设计环境中产生。生成出来的制品是计算机可读的测试规程版本，叫做测试脚本
- 如果需要特定的测试代码，则需要设计师和实现人员与测试设计师共同合作以设计和实现测试代码

- **在集成测试阶段执行测试：** 目的是确保在装配过程中系统地构建像预期的那样合作，同时增加的部分具有正确的行为
 - 系统测试人员增量式地编译并连接系统。对于每一个增量，都要测试增加的功能，执行回归测试，产生测试结果
- **在系统测试阶段执行测试：** 目的是确保整个系统的功能都像预期的那样完成
- **评价测试：** 目的是交付并评估可以计量的测试度量，以确定测试目标的质量和测试过程的质量

12.9 小结

- 测试是对于所生产的产品质量的评估
- 在生命周期各个阶段的迭代过程中都要进行测试,以便在设计和构造过程中就获得关于产品质量的早期反馈,以改善产品质量
- 质量是每一个人的职责。质量不是由质量保证或测试组织产生的。在迭代过程中,测试及时地为开发组织提供了关于质量度量的反馈,从而使开发组织提高系统的质量
- 测试工作流为项目提供了反馈机制,可以度量质量并标识缺陷。



第13章 配置和变更 管理 workflow



目录

- **13.1 目的**
- **13.2 CCM立方体**
- **13.3 工作人员和制品**
- **13.4 workflow**
- **13.5 工具(略)**
- **13.6 小结**

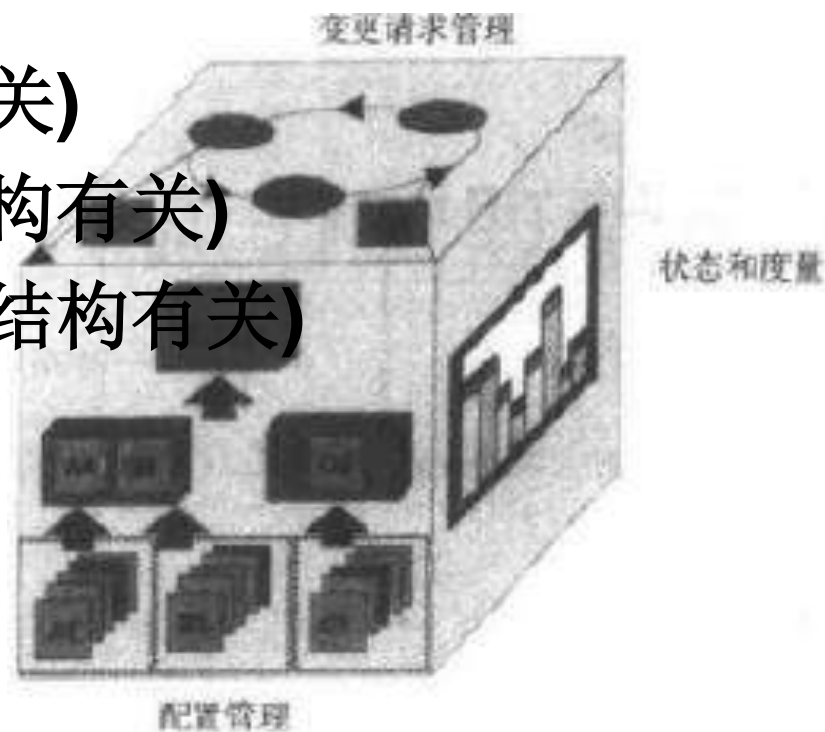
13.1 目的

- 配置和变更管理工作流的目的是跟踪并维护不断进化的项目资产的整体性
- 项目组必须能够标识和定位制品,选择合适的制品版本,察看它的历史,以理解它当前的状况和它变更的原因,并确定当前对它负责的工作人员
- 项目组必须跟踪产品的进化,捕获并管理对于变更的请求,然后通过一系列制品以一致的方式实现变更

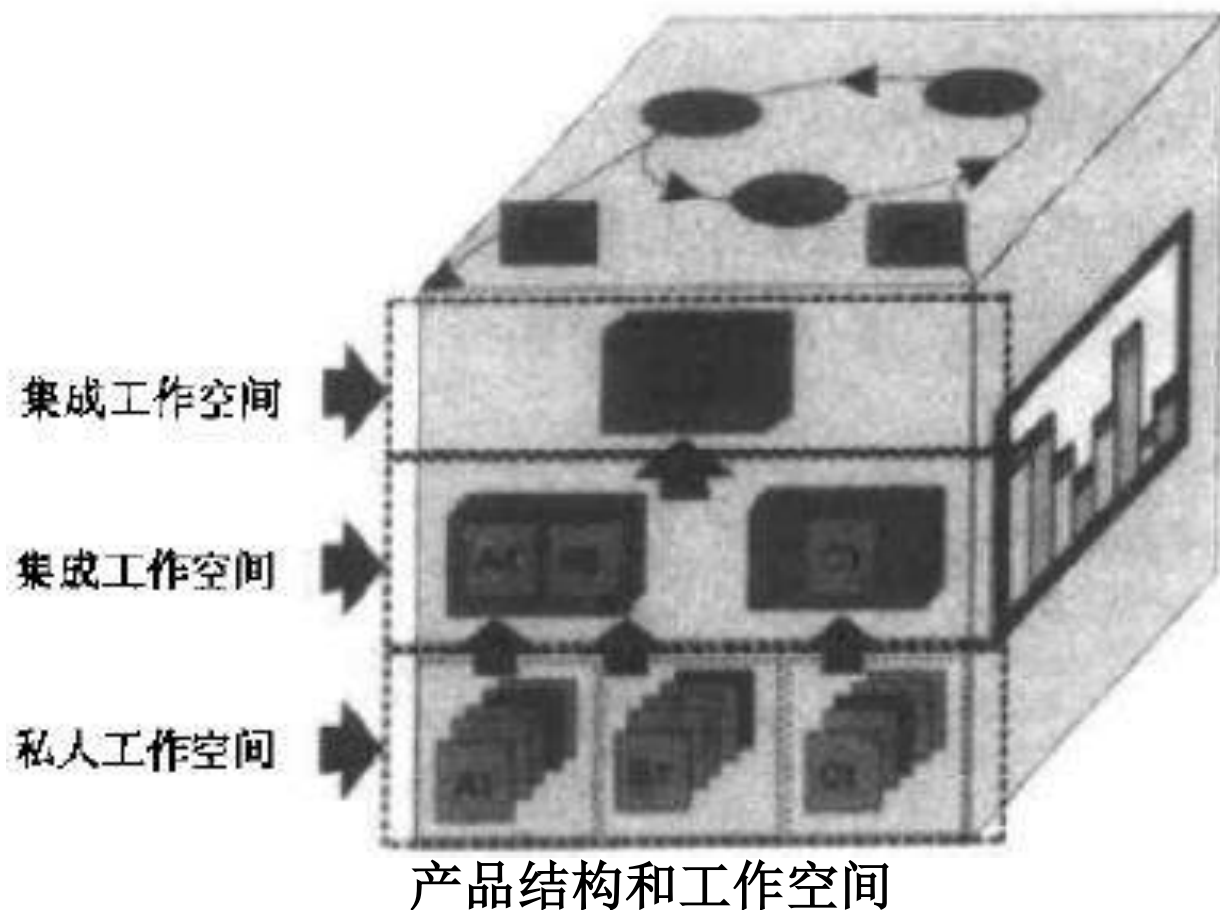
13.2 CCM立方体

■ 配置和变更管理(CCM)包含3种相互依赖的功能

- 配置管理(与产品结构有关)
- 变更请求管理(与过程结构有关)
- 状态和度量(与项目控制结构有关)



配置管理

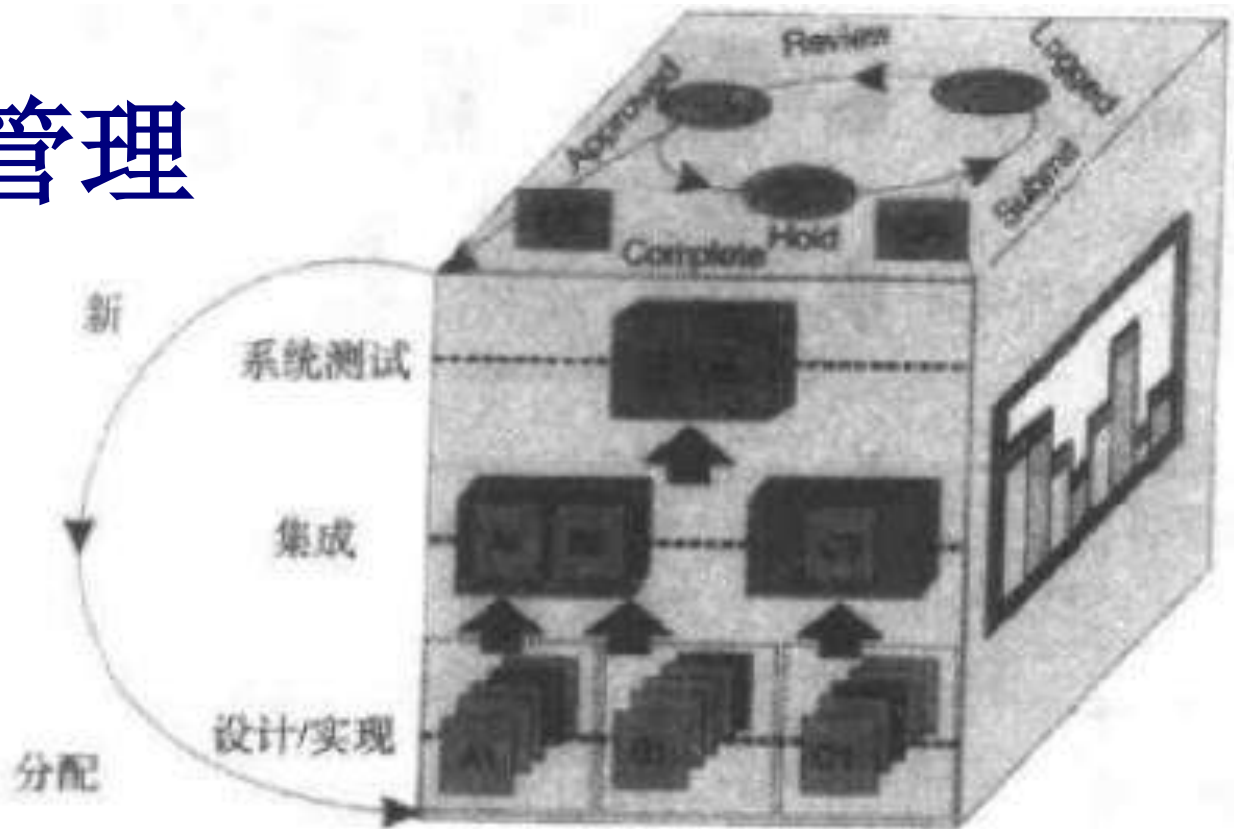


- 重要的制品要受到版本控制。当一个制品进化时，就出现了多重版本，必须标识制品、制品的版本以及制品的变更历史

■ 配置管理处理产品结构。

- 包括制品的标识和版本、制品之间的相互依赖关系以及配置的标识等问题；
- 还处理提供个人和群组工作空间的问题
- 建立管理的目的是确保组成构造(**Build**)的构件是可用的，并且根据这些构件之间的依赖层次以正确的顺序集成为这个构造

变更请求管理

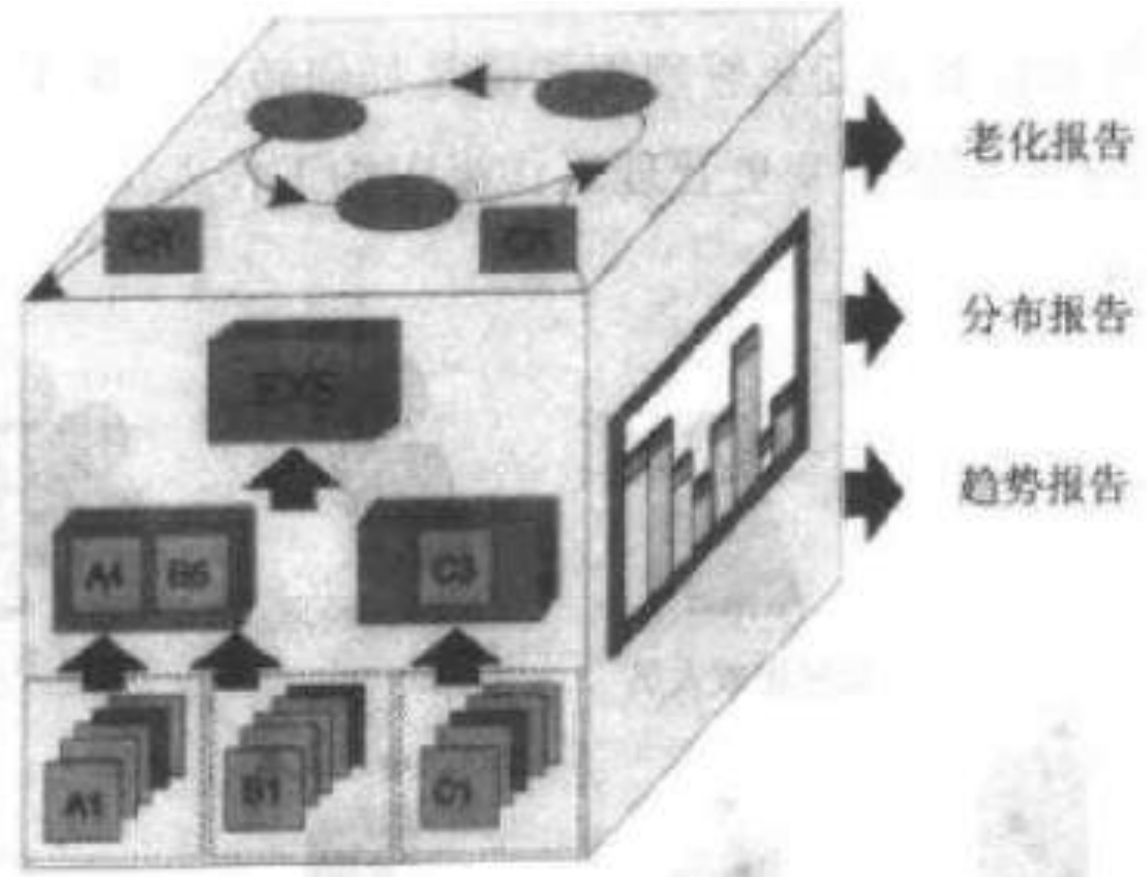


- 变更请求可以用简单的状态机表示它的生命，状态分为新的、已记录的、已批准的、已赋值的和完成了的等不同状态。
- 当变更请求从一个状态转移到另一个状态，关于变更的信息就会增加。

■ 变更需求管理处理过程结构。

- 一个变更请求是一份需要变更的一个或多个制品的文档化的建议
- 变更请求可能来源于不同原因：修复缺陷、提高产品质量、增加需求，或记录在一个迭代和下一个迭代之间的增量
- 变更需求管理还对变更带来的潜在影响进行分析，并跟踪变更请求带来的影响，直至变更结束

状态和度量



- 变更请求具有状态，也具有其他如根本原因、种类(例如缺陷或添加)、严重性、优先级和受影响的领域(如层或子系统)等属性。变更请求的不同状态提供了有用的标记，可以作为度量元报告的基础

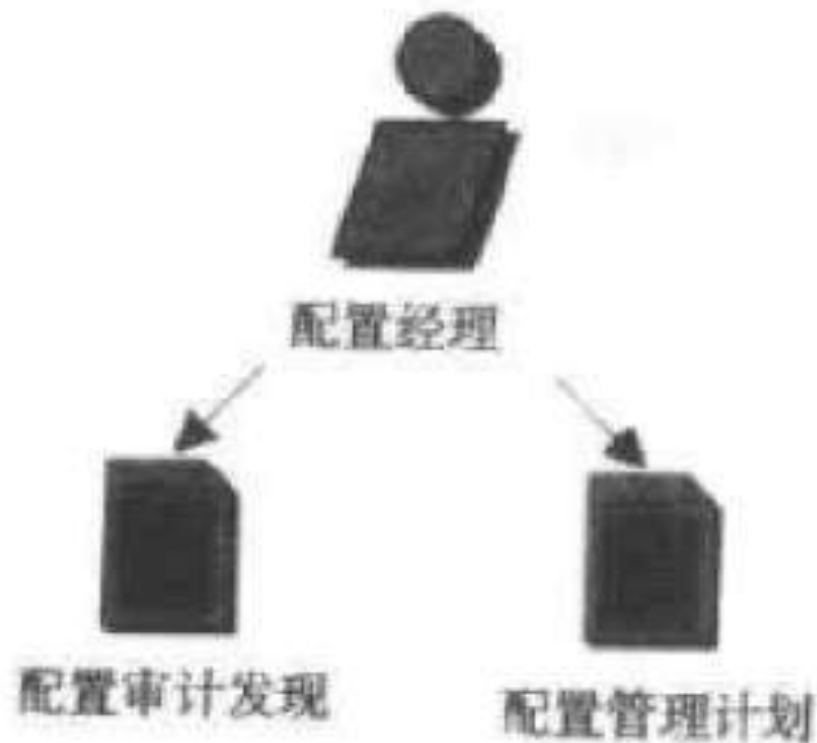
■ 状态和度量处理项目控制结构。

□ 状态和度量抽取与项目管理有关的信息，便于质量评估：

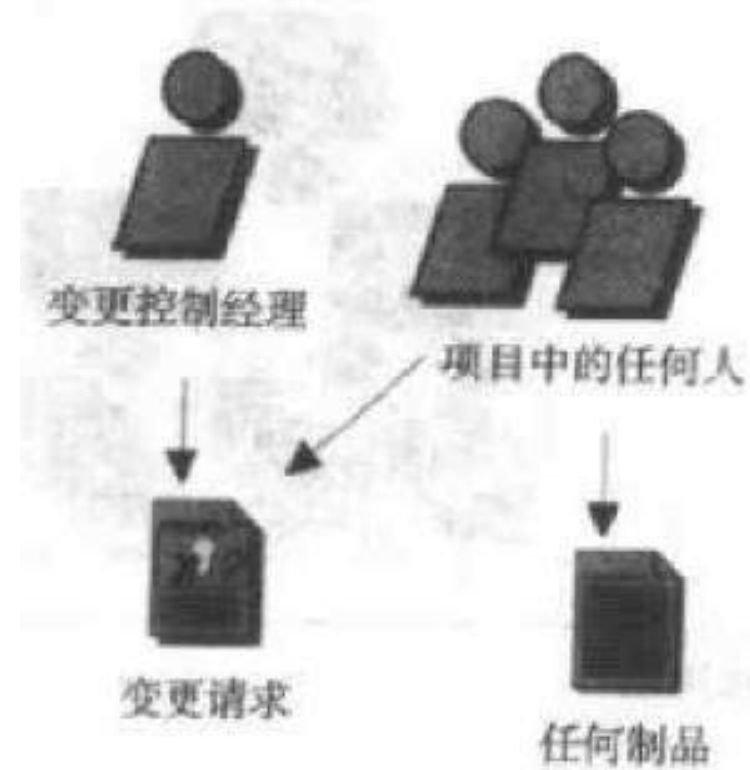
- 产品状态、进展、趋向和质量
- 完成了什么，没有完成什么
- 花费
- 需要关注的问题领域

13.3 工作人员和制品

- 配置经理负责建立产品结构，供开发人员定义和分配工作空间以及在集成阶段进行相应的操作。配置经理还要向项目经理提交适当的状态和度量元报告



- 变更控制经理负责监控变更控制过程。还负责定义变更请求管理过程

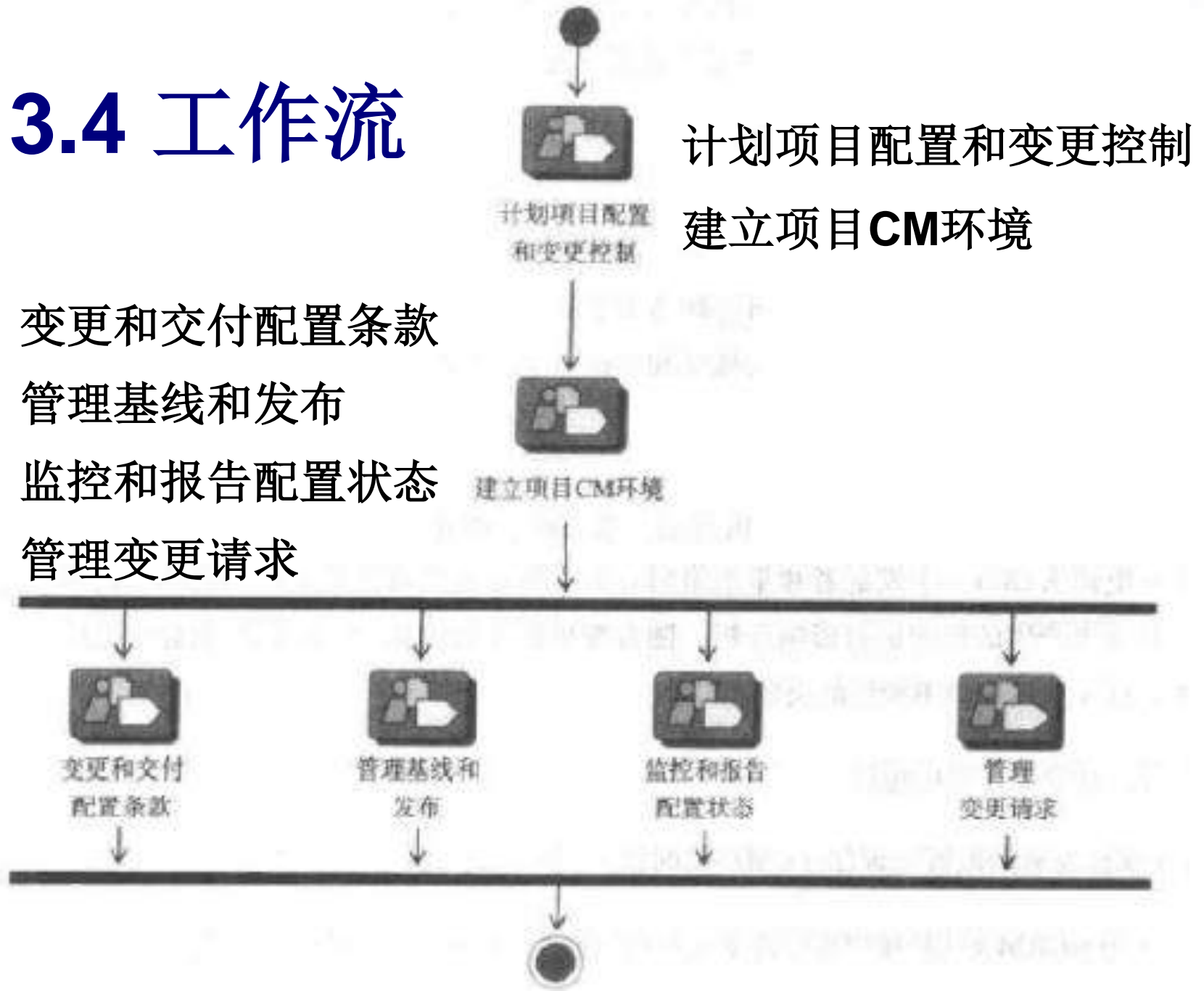


- 其他工作人员

- 实现人员：访问他们所需要的合适的工作空间和制品
- 集成人员：在集成工作空间中验收变更并建立产品
- 任何工作人员：都可以提交变更请求
- 构架师：根据实现视图提供产品结构的输入

13.4 workflow

变更和交付配置条款
管理基线和发布
监控和报告配置状态
管理变更请求



13.6 小结

- 配置和变更管理工作流的目的是项目制品在响应变更请求而不断进化的情况下维持他们的整体性
- 配置管理处理产品结构，标识元素、版本、有效的配置元素和工作空间
- 变更请求管理包括以一致的方式修改制品的过程
- 从配置和变更管理信息中可以提取状况和度量元，以协助状态评估



第14章 环境 workflow



目录

- 14.1 目的
- 14.2 工作人员和制品
- 14.3 工作流
- 14.4 小结

14.1 目的

- 环境工作流的目的是通过使用过程和工具支持开发组织
 - 工具的选择和获取
 - 安装工具和配置工具、使其适应开发组织
 - 过程配置
 - 过程改进
 - 提供技术服务以支持过程：信息技术基础设施、帐户管理和备份工作等

14.2 工作人员和制品





测试设计师



测试
指南



构架师



设计
指南



程序设计
指南



工具专家



工具



工具支持
评估



工具
指南



系统管理员



支持环境



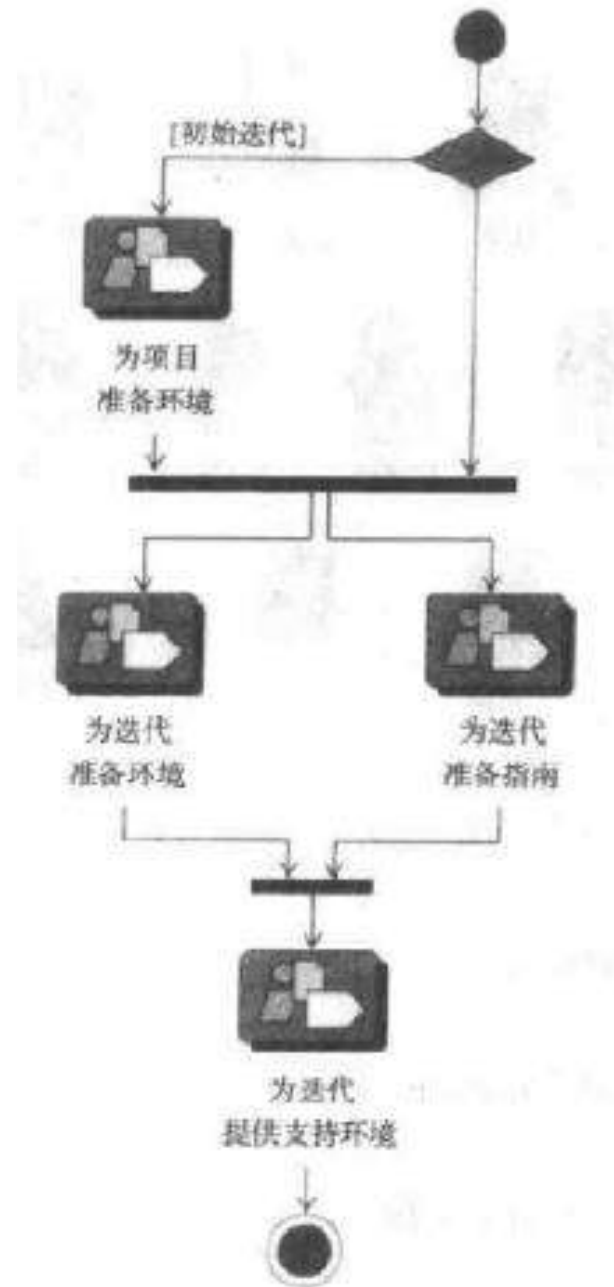
用户界面
设计师



用户界面
指南

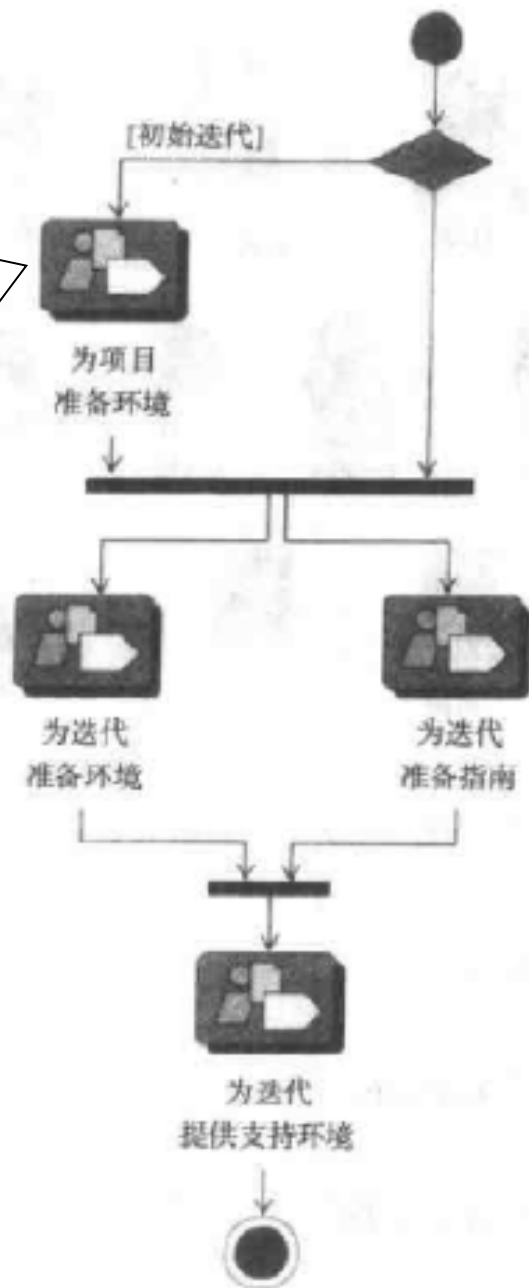
14.3 workflow

- 为项目准备环境
- 为迭代准备环境
- 为迭代准备指南
- 为迭代提供支持环境



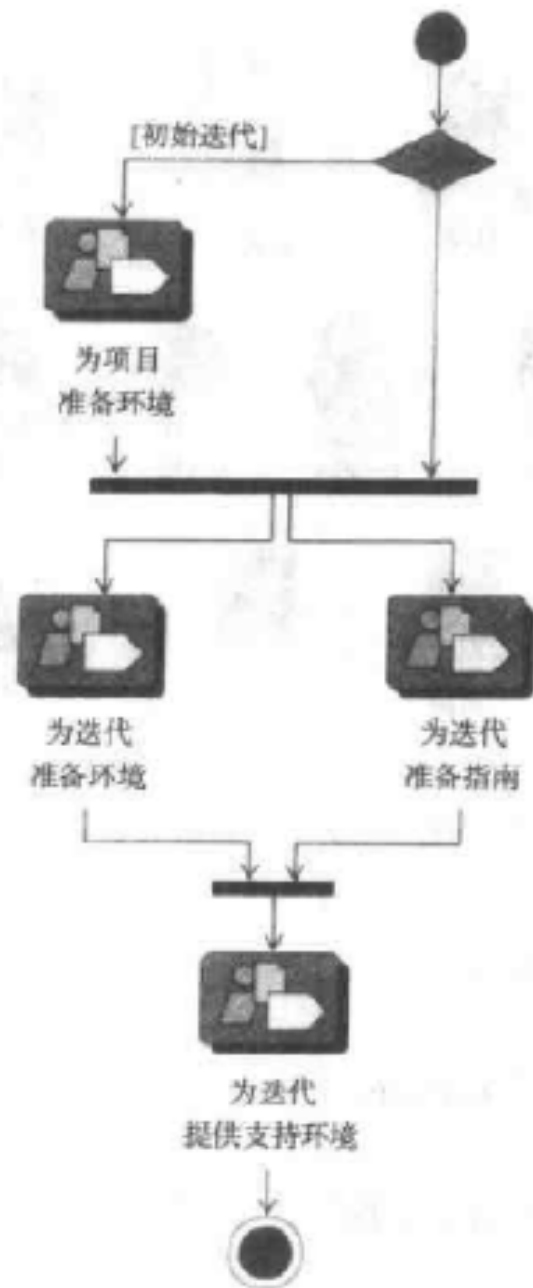
为项目准备环境的目的是：

- 评估当前的开发组织
- 评估当前的工具支持
- 制定开发案例的第一个草案
- 列出一个用于开发的候选工具清单
- 列出关键制品的候选项目专用模板清单



为迭代准备环境的目的是：

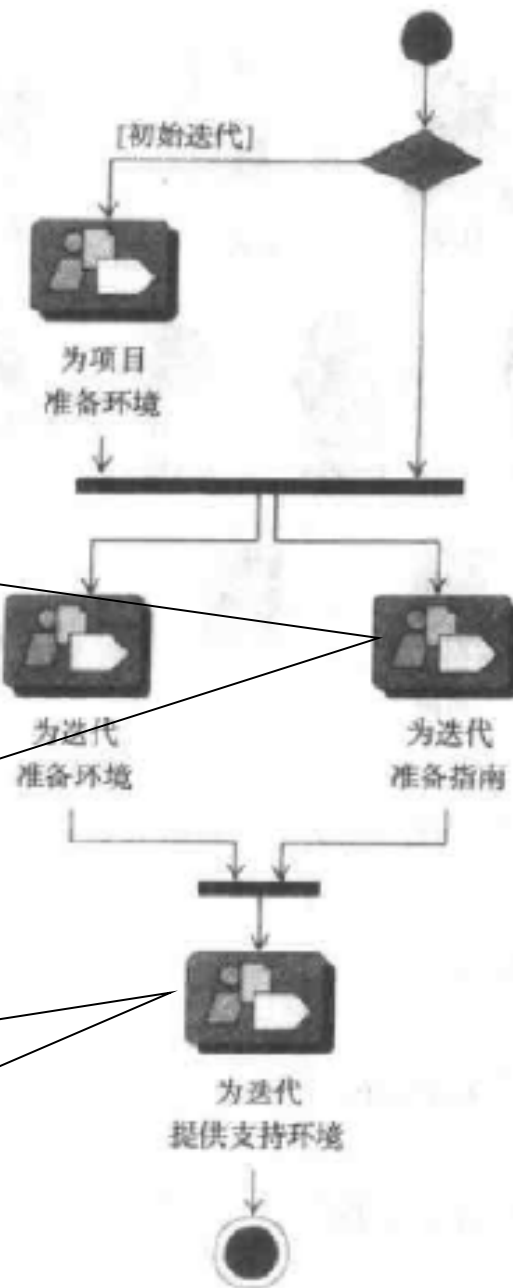
- 完成开发案例,为迭代做好准备
- 准备在迭代中使用的工具,需要时将其客户化
- 验证工具是否已经正确地配置和安装
- 生产一个在迭代中使用的项目专用模板集
- 培训人员,使他们能够使用并理解工具和开发案例



为迭代准备指南是非常必要的，
具体内容包括：

- 业务建模
- 用例建模
- 设计
- 程序设计
- 手册风格指南
- 用户界面
- 测试
- 工具

在迭代期间，开发人员需要在
他们使用工具和过程方面提供
支持



14.4 小结

- 环境工作流的目标是为开发组织在工具、过程和方法上提供适当的支持



第15章 实施 workflow



目录

- **15.1 目的**
- **15.2 工作人员和制品**
- **15.3 工作流**
- **15.4 小结**

15.1 目的

- 实施的目的是将完成了的产品交给用户。
- 实施 workflow 设计了以下类型的活动：
 - 在软件的最终运行环境中测试软件(**beta**测试)
 - 包装所要交付的软件
 - 分布软件
 - 安装软件
 - 培训最终用户和销售队伍(产品大量生产的情况下)
 - 移植现有软件或转换数据库

实施模式包括三种类型

■ 在用户定制系统中的软件实施

- 用户定制系统通常是独一无二的，有时甚至与用户定制的硬件相关。例如运输行业和电讯行业、嵌入式系统、银行和航运公司等

■ 压缩打包软件的实施

■ 可在互联网上下载的软件实施

- 安装压缩打包的软件和从网上下载的软件并没有什么区别。安装人员在安装指南的帮助下可以容易安装和运行产品软件。两者只是交付机制不同

实施的时间选择

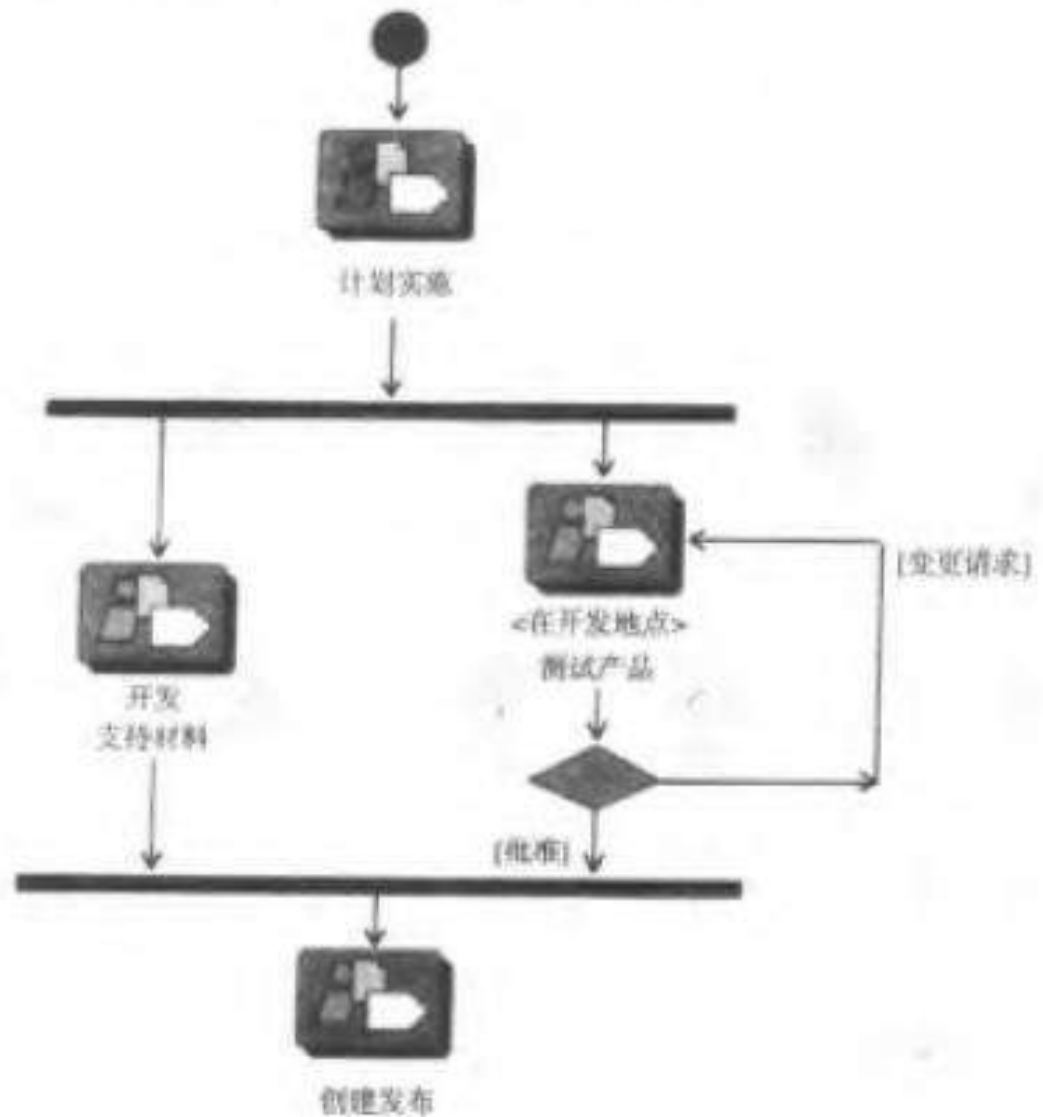
- 实施活动主要集中在移交阶段，实施是否成功以及整个开发工作是否成功，要由客户是否愿意使用这个新的软件来判定
- 实施计划和其他项目计划活动一样，可以在项目生命周期的早期开始，以考虑整体实施、客户准备策略、交付测试产品需要的资源以及最终用户支持材料
- 在细化阶段结束时，一旦构架和需求已经稳定，就可以尽早开始对实施制品(如用户手册和培训手册)进行工作

15.2 工作人员和制品

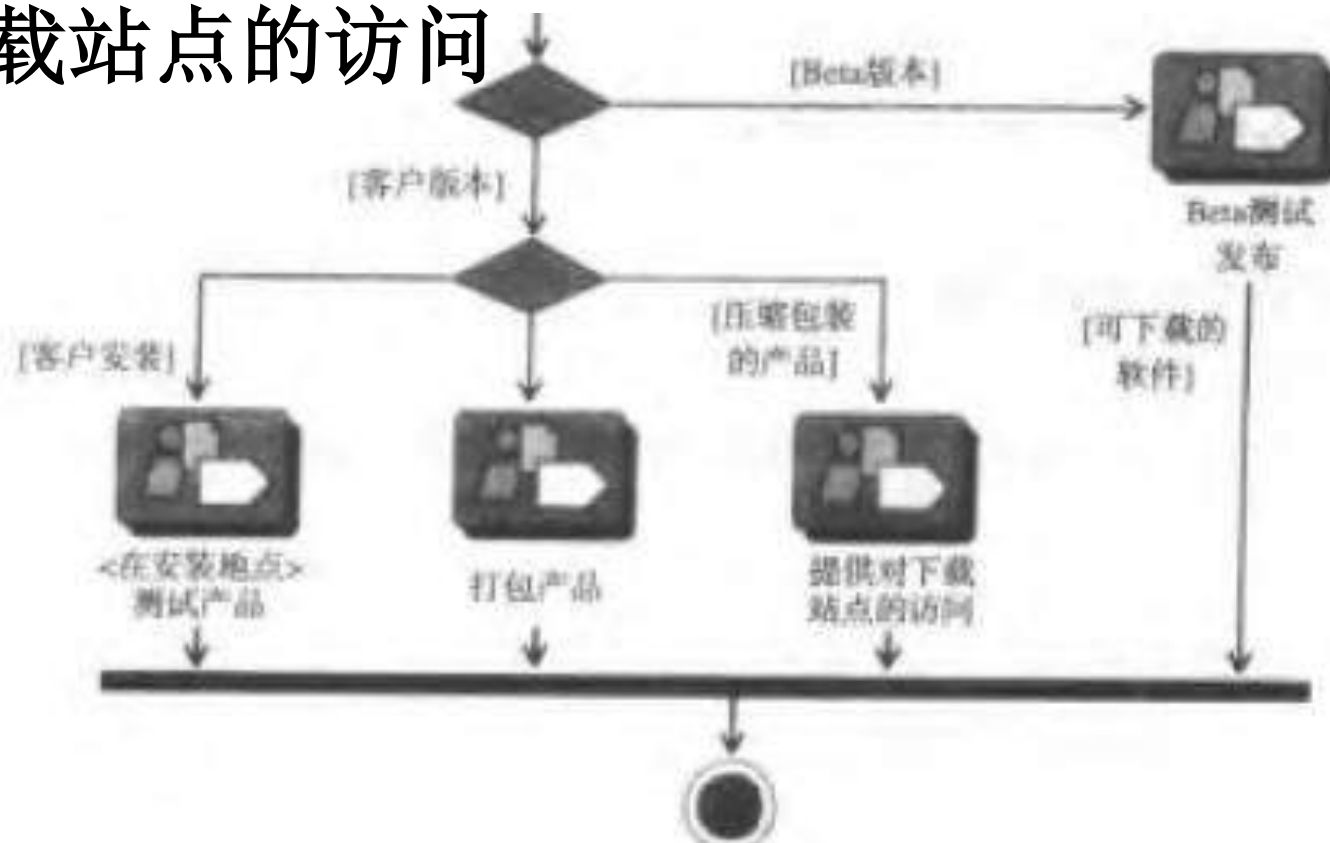


15.3 workflow

- 计划实施
- 开发支持材料
- <在开发地点>测试产品
- 创建发布

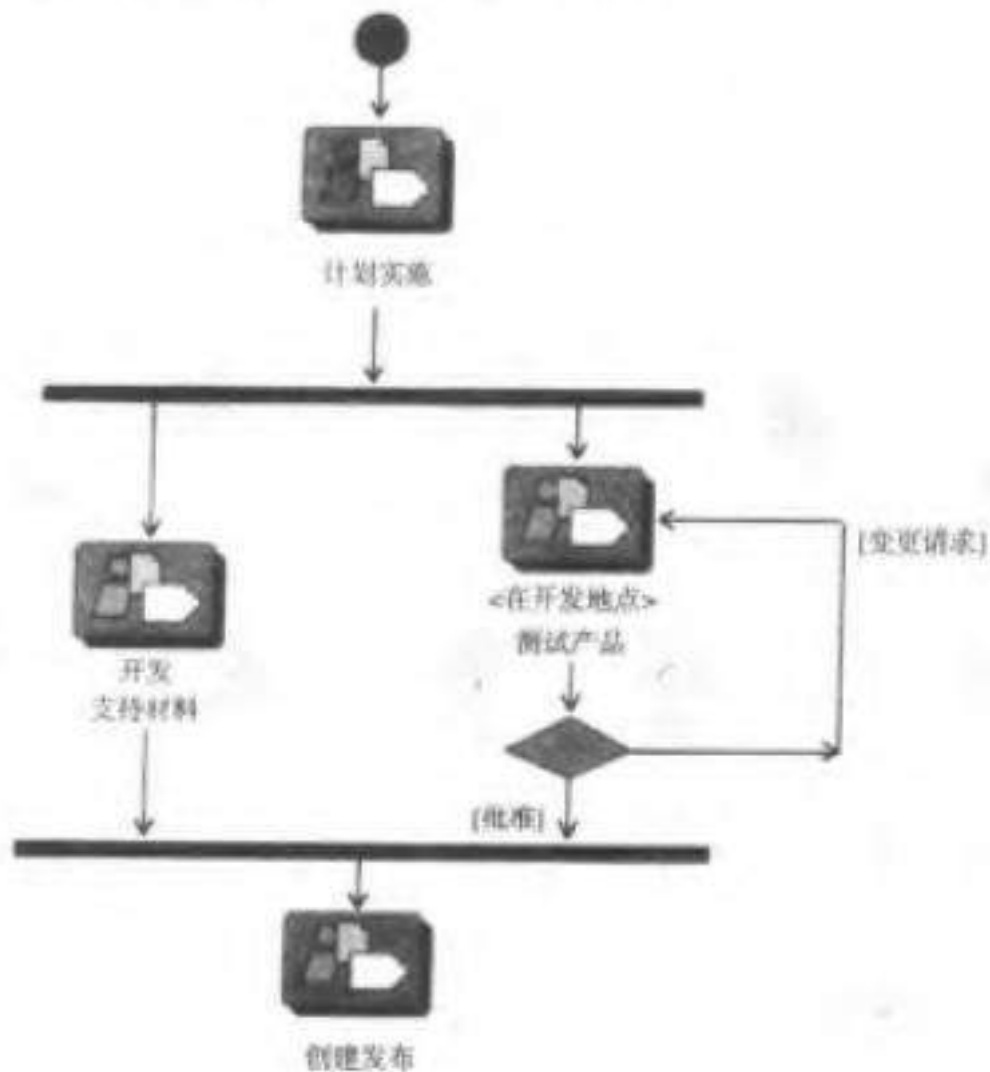


- Beta测试发布
- <在安装地点>测试产品
- 打包产品
- 提供对下载站点的访问



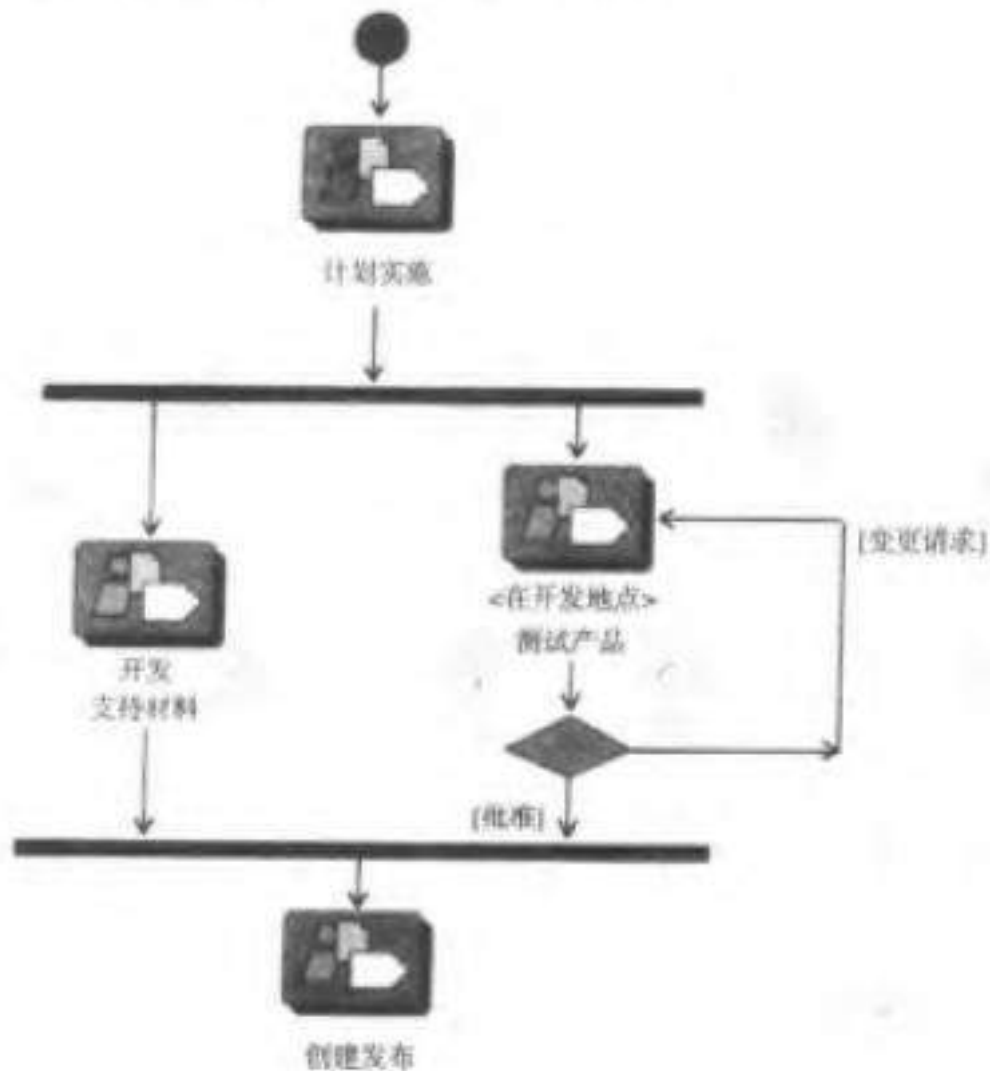
■ 计划实施

- 制定实施计划要考虑什么时候以及如何开发完整的交付清单，还要确保最终用户可以获得关于新软件交付的必要信息
- 整个系统的实施计划需要客户的高度协作和准备



■ 开发支持材料

- 支持材料包括最终用户进行安装、操作、使用和维护系统所需要的各种信息。还包括对于各种岗位的培训材料，这些材料是高效使用新的系统所需要的。

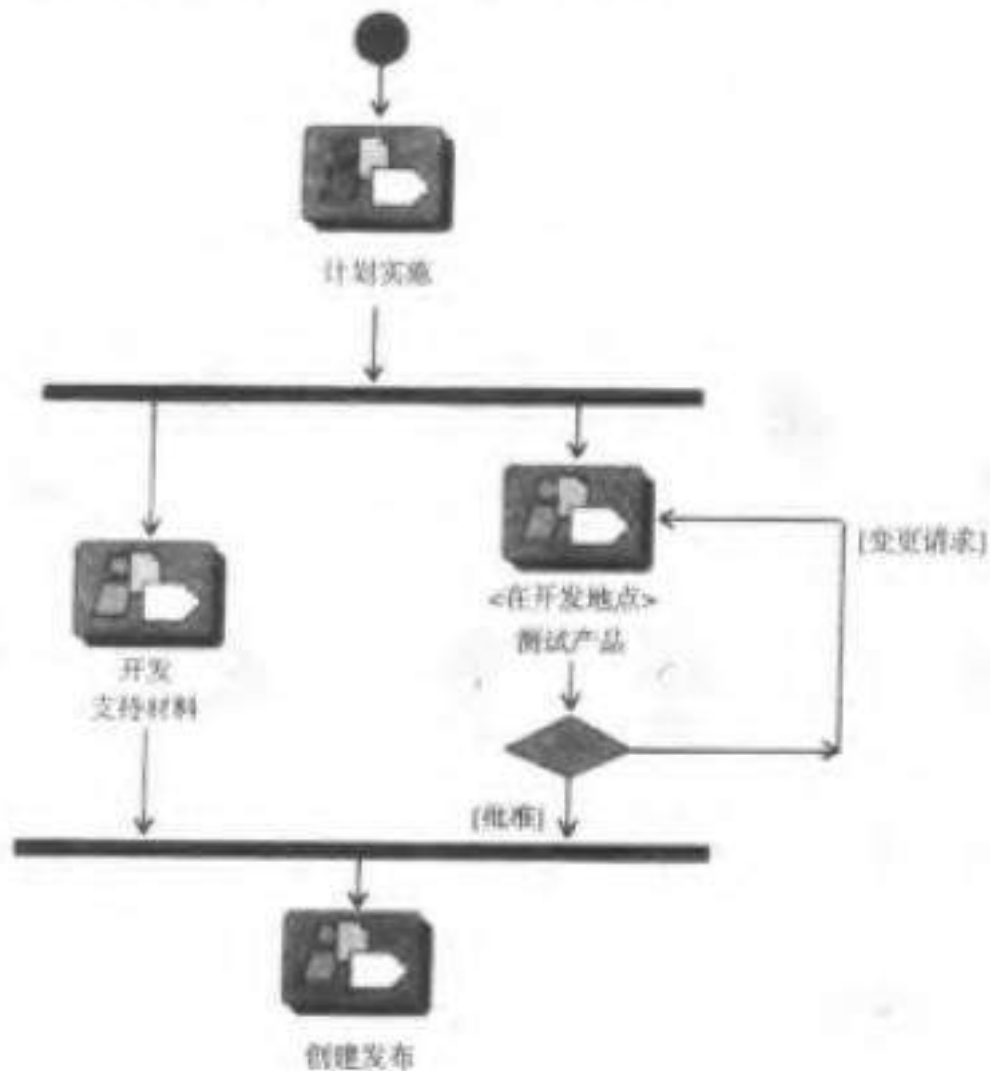


■ 在开发地点测试产品

- 在开发地点测试产品可以帮助确定产品发布是否足够成熟。
- 测试产品活动在开发现场和目标现场中应该是相同的

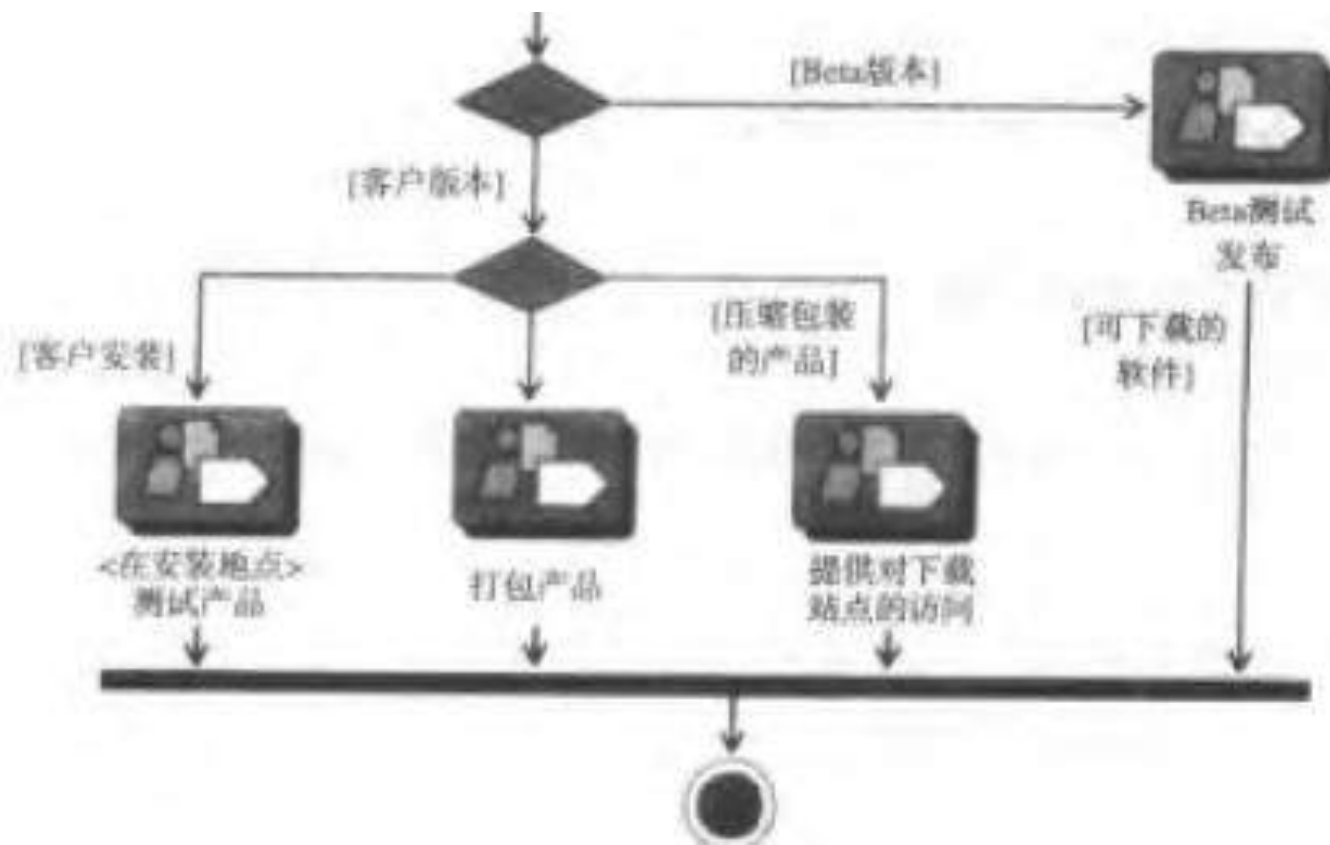
■ 创建发布

- 目的是确保产品已经完全准备好交付给客户



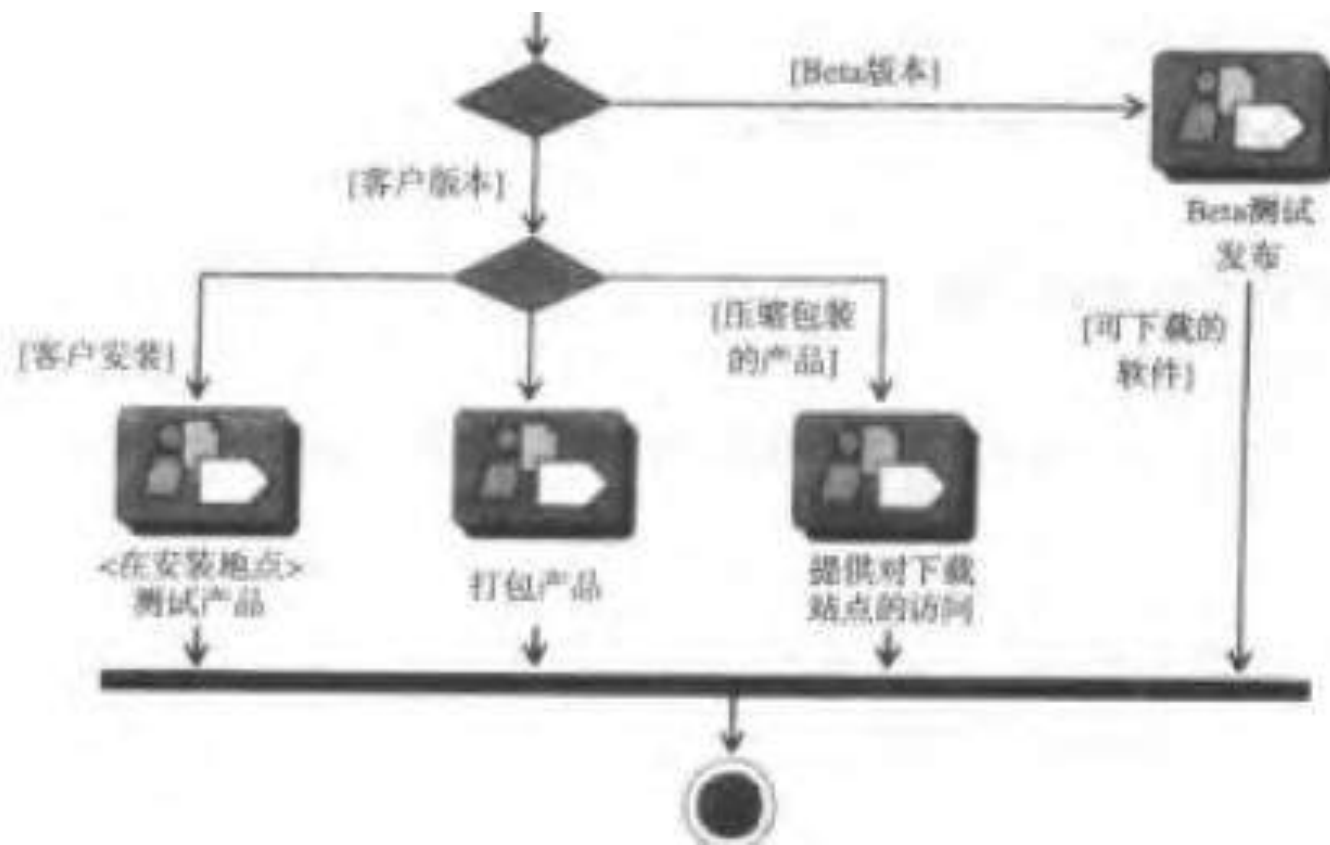
Beta测试发布

- Beta测试提供对于软件性能和可使用性的反馈
- 在迭代开发的语境中，**beta**测试的目标是确保产品满足了客户的期望，并要确保下一个迭代开发考虑了重要的用户反馈因素



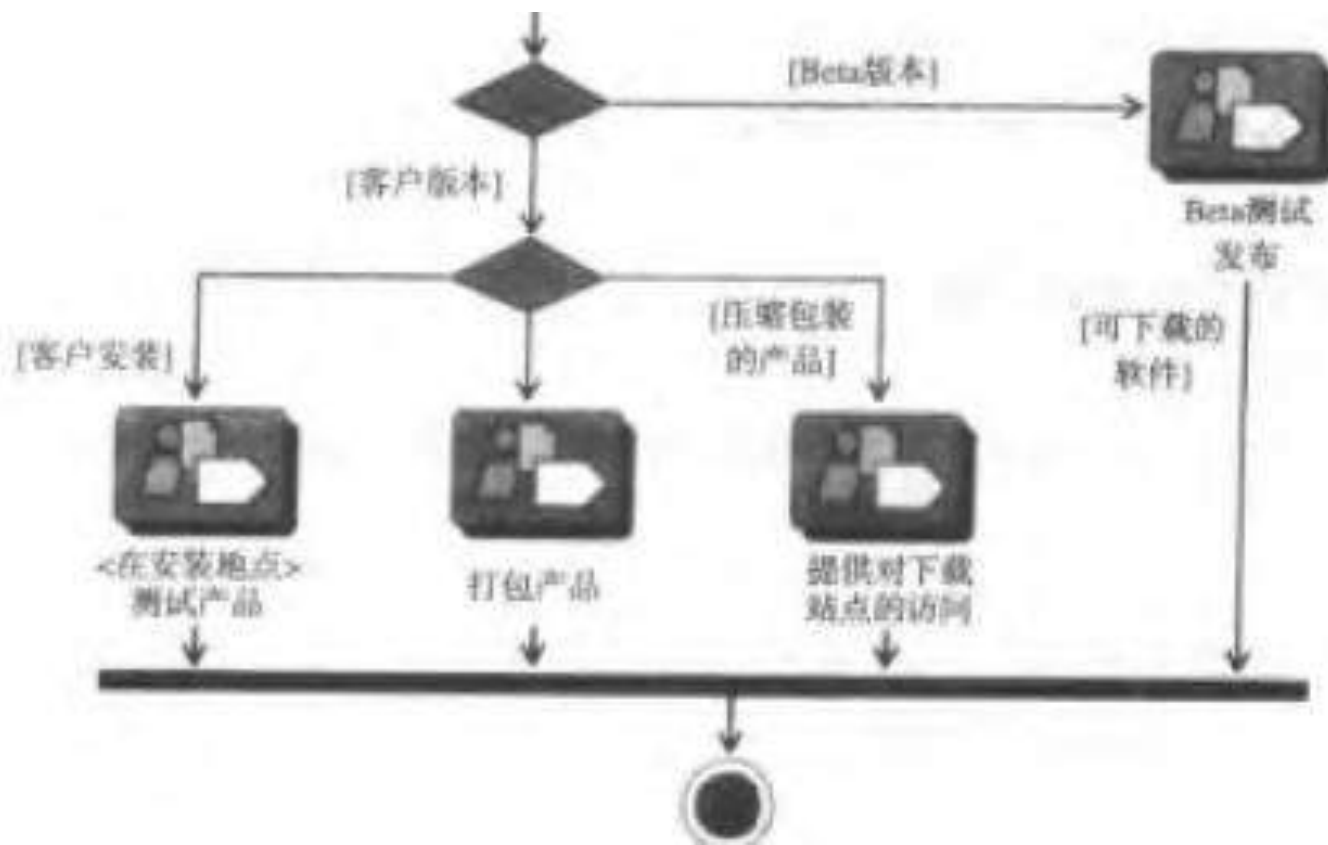
■ 在安装现场测试产品

- 产品进行了充分的内部测试和**Beta**测试后，就可以在现场进行安装，并由客户进行测试。
- 基于已经完成了迭代测试并有客户参与，最终现场测试的目的是为了让客户接受该系统



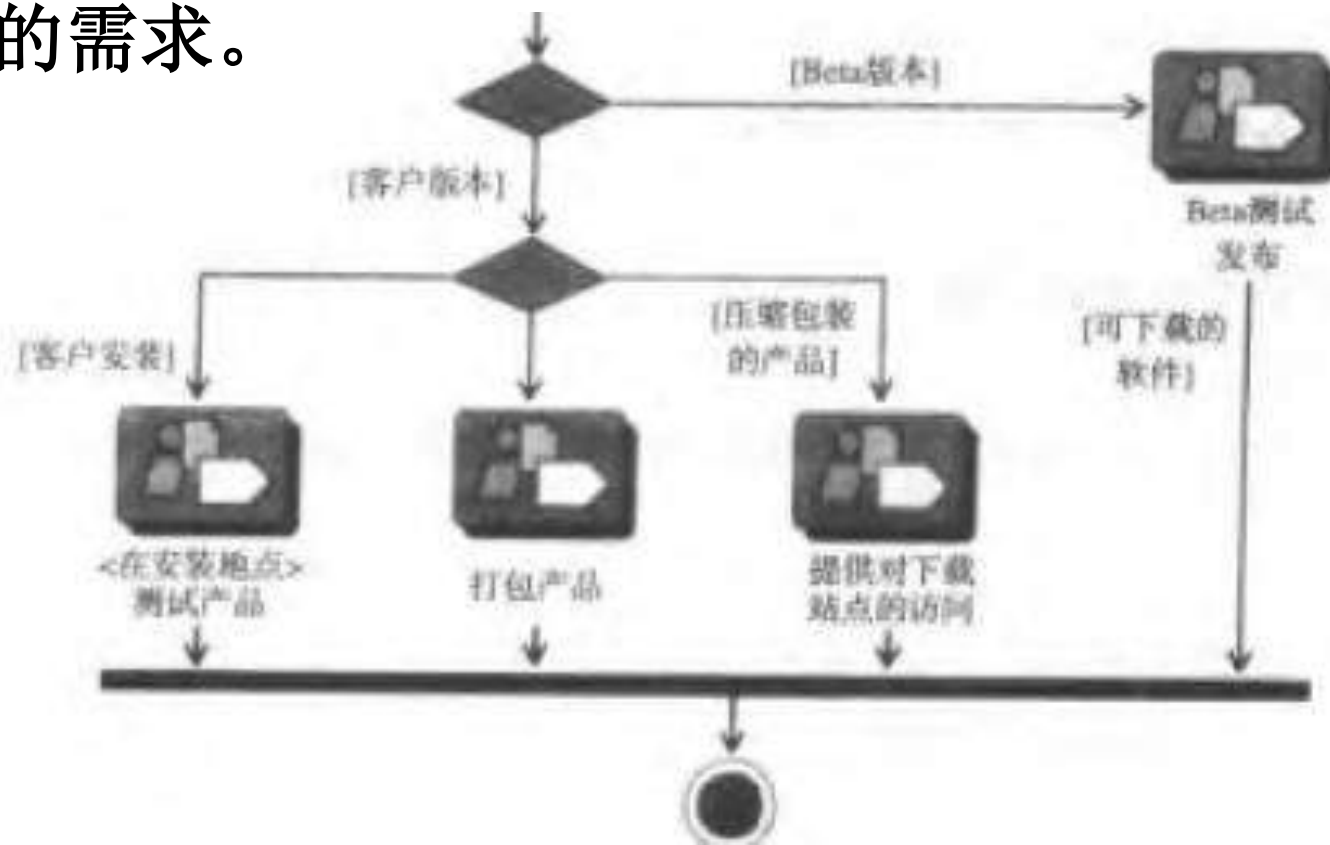
打包产品

- 是一组选择性的活动，描述了生产“打包软件”产品时要完成哪些工作。
- 在这种情况下，发布作为批量生产的母板保存，然后将产品与物料账单一起打包装箱，并交付给客户。



■ 提供对下载站点的访问

- 要确保全球市场在任意时间内都可以获得这个产品。
- 实施经理需要知道如何将产品加入到网站提供的产品清单中，并要知道如何在网上交流以满足购买和交付的需求。



15.4 小结

- 实施 workflow 必须照顾到将要交付给最终用户或客户以及支持组织的所有制品
- 实施 workflow 描述了与 **Beta** 测试和交付可安装软件相关的活动。安装可以由以下人员来完成
 - 卖主：如果是复杂的分布式系统
 - 用户：如果软件是打包产品或是通过互联网下载的产品
- 这个 workflow 与开发产品的种类和业务语境联系紧密