第二章 软件生命周期过程
统一软件过程模型（UP)

主讲：张雷
zlei@bupt.edu.cn;
北京邮电大学计算机学院

提纲

1、UP/USDP概述

2、UP的基本结构

3、UP的阶段

4、核心工作流

5、最佳实践

6、UP工件

§1 UP/USDP概述

- 软件开发模型的出发点
  - 如何更快（效率）更好（质量）地满足需求
  - 使得开发过程在一种受控的方式下运行
  - 过程←活动←任务
  - 还需要涉及：项目、人员、工件
- UP（Unified Process）是权衡30年的软件开发实践形成的产物。

§1 UP/USDP概述

- **UP对于如何运用UML的概念进行软件开发提供了详细的指导，包括：**
  - 安排开发活动的次序，
  - 为开发团队指定任务，
  - 规范需开发的工件，
  - 提供对工件、活动进行监控与度量的准则。

§1 UP/USDP概述

- **UP的重要因素：人员、项目、过程和工具**
  - 人员：总设计师/开发者/测试者/管理者/用户/关注者等；
  - 项目：在生命周期内面向机器、开发人员、关注者的所有“工件”的集合。项目的成果是一个最终的发布产品。
  - 过程：将用户需求转变为产品所需的“活动集”
  - 工具：过程定义的活动进行自动化的软件。

§1 UP/USDP概述

§1 UP/USDP概述

- **UP是以用例为驱动、以体系结构为中心的迭代增量的过程。**
  - 是一个软件开发过程的框架
  - 拥抱变化：用户反馈和适应调整逐步满足用户需求；
  - 迭代增量式开发
  - 用例驱动整个开发过程
  - 提倡基于构件的软件体系结构为中心展开开发活动

§1 UP/USDP概述

1. 以用例为驱动：

含义：产品开发的各阶段都可回溯到用户需求。

作用：

- 驱动开发过程：以用例为单位指定计划、分配任务、监控执行和进行测试。
- 由用例可得到“体系结构”和其它工件。
- 可导出测试用例、测试规程。
- 由用例可估算系统的性能、硬件需求和可用性。

§1 UP/USDP概述

2. 以体系结构为中心：

含义：从不同的角度描述要构造的系统，注重系统的静态和动态结构，描述：子系统、依赖关系、接口、协作、节点和主动类。

作用：理解系统，控制开发、鼓励复用、演进系统。

构造过程：（简述）

1. 构建“基线”—“小的、皮包骨架” 的系统；系统框架
2. 使用架构模式（风格）
3. 描述架构，包括：

1. 用例模型的架构视图
2. 设计模型的架构视图
3. 实施模型的架构视图
4. 实现模型的架构试图

§1 UP/USDP概述

3. 迭代和增量：

含义：将整个项目划分为一系列微项目，对每个微项目都进行一次需求规约、分析、设计、实现和测试的迭代。

作用：

降低风险，

获得健壮的骨架，

处理不断变化的需求，

允许灵活的改变

实现持续的集成

尽早得到相关的知识

特点：

风险驱动

一次迭代产生一个增量结果

进化模型

提纲

1、UP/USDP概述

2、UP的基本结构

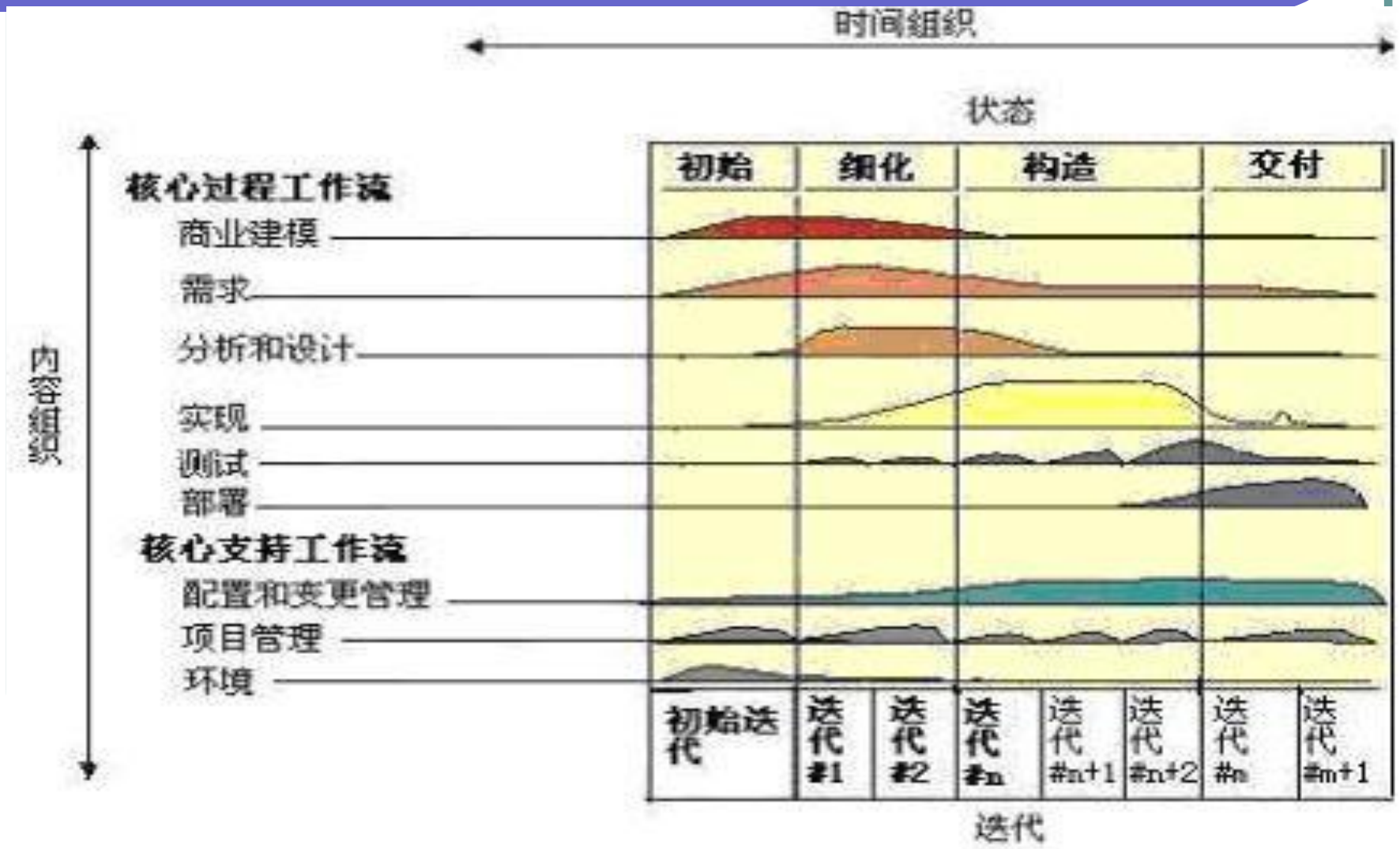
3、UP的阶段

4、核心工作流

5、最佳实践

6、UP工件

§2 UP的基本结构


时间组织
状态
初始
细化
构造
交付
核心过程工作流
商业建模
需求
分析和设计
实现
测试
部署
核心支持工作流
配置和变更管理
项目管理
环境
内容组织
初始迭代
迭代#1
迭代#2
迭代#n
迭代#n+1
迭代#n+2
迭代#n
迭代#m+1
迭代

§2 UP的基本结构

- **阶段**是软件开发过程按时间顺序的划分。
  - **初始**阶段确立系统的可行性；-**启动项目**
  - **细化**阶段关注“做的能力”；-**稳定的架构**
  - **构造**阶段建造系统；-**具备基本的可操作能力**
  - **移交**阶段转移到用户环境；-**产品发布**
- 每个阶段结束于一个主要的里程碑(Major Milestones)

提纲

1、UP/USDP概述
2、UP的基本结构
3、UP的阶段
4、核心工作流
5、最佳实践
6、UP工件

§3 UP的阶段（初始阶段，inception）

- 初始阶段的目标是为系统建立初始业务案例和确定项目的边界。

项目边界的确定

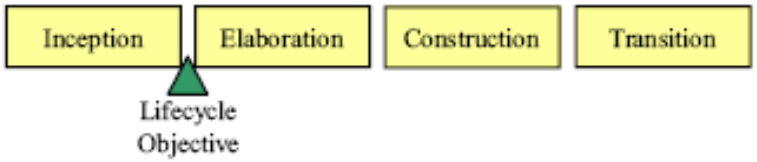
1、识别外部角色，识别用例，描述主要用例；（系统应该为不同的用户提供什么？）

2、用户提出的非功能性要求描述。

3、系统的整体架构划分（子系统的划分），与外界环境的交互关系等。

- 初始业务案例（business case）---标书
  - 使用资源估计，包括项目的支撑环境；
  - 估计潜在的风险；
  - 对整个项目做最初的项目成本和日程估计
  - 项目验收规范。
  - 投资回报估算

§3 UP的阶段（初始阶段，inception）



- **初始阶段主要目标：**
  - 明确软件系统的范围和边界条件，包括从功能角度的构想(vision)分析、产品验收标准和哪些做与哪些不做的相关决定；
  - 明确区分系统的关键用例和主要的功能场景；
  - 展现或者演示至少一种符合主要场景要求的候选软件体系结构；
  - 对整个项目做最初的项目成本和日程估计（更详细的估计将在随后的细化阶段中做出）；
  - 估计出潜在的业务风险（主要指各种不确定因素造成的潜在业务风险）；
  - 准备好项目的支持环境。
- **评审标准：**
  - 风险承担者就范围定义、成本/日程估计达成共识；
  - 以客观的主要用例证实对需求的理解；
  - 成本/日程、优先级、业务风险和开发过程的可信度；

§3 UP的阶段（初始阶段，inception）

- **初始阶段的产出：**
  - 构想文档：核心项目需求、关键特征、主要约束的总体构想；
  - 原始用例模型（完成10%～20%）；
  - 原始项目术语表（可能部分表达为业务模型）；
  - 初始业务案例，包括业务背景、验收规范、成本预计、投资回报等；
  - 初始的业务风险评估；
  - 一个或多个概念证明原型

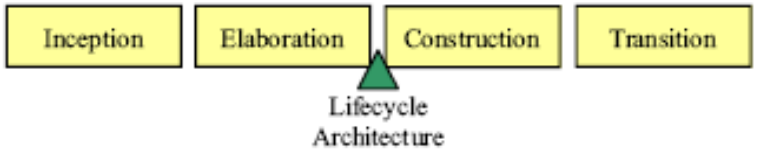
§3 UP的阶段（初始阶段，inception）

- 第一阶段的里程碑：
  - 已经阐明了初始的架构（候选架构）；
  - 已经识别出最严重的风险，进行可行性研究；
  - 已经建立足够描述信息的“初始业务案例”

§3 UP的阶段（细化阶段，elaboration）

- 细化阶段的主要目标是分析问题领域，建立健全的体系结构基础，编制项目计划，淘汰项目中最高风险的元素。
  - 确保软件结构、需求、计划足够稳定，确保项目技术风险已经降低到能够预计完成整个项目的成本和日程的程度；
  - 针对项目的软件结构上的主要技术风险已经解决或处理完成；
  - 通过完成软件结构上的主要场景建立软件体系结构的基线；
  - 建立一个包含高质量组件的可演化的产品原型；
  - 说明基线化的软件体系结构可以保障系统需求可以控制在合理的成本和时间范围内；
  - 建立好产品的支持环境。

§3 UP的阶段（细化阶段，elaboration）



- 评审标准：
  - 产品的构想是否稳定？
  - 体系结构是否稳定？
  - 可执行的演示版是否显示技术风险要素已被处理和可靠的解决；
  - 构建阶段的计划是否足够详细和精确？是否被可靠的审核基础支持？
  - 如果当前计划在现有的体系结构环境中被执行而开发出完整系统，是否所有的风险承担人同意该构想是可实现的？
  - 实际的费用开支与计划开支是否可以接受？

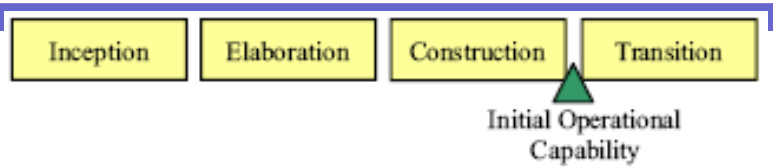
§3 UP的阶段（细化阶段，elaboration）

- 细化阶段的产出：
  - 用例模型（完成至少80%）……所有用例均被识别，大多数用例描述被开发；
  - 补充捕获非功能性要求和未关联于特定用例要求的需求（补充规范）
  - 软件体系结构描述，含：用例、分析、设计、实施、实现模型的各种视图。
  - 可执行的软件原型
  - 经修订过的技术风险清单和**完整的**业务案例
  - 总体项目的开发计划，包括粗粒度的项目计划，显示迭代过程和对应的审核标准；
  - 用户手册的初始版本（可选）

§3 UP的阶段（构造阶段，construction）

- 构造阶段：所有剩余的构件和应用程序功能被开发并集成为产品，所有的功能被详尽的测试。
  - 通过优化资源和避免不必要的返工达到开发成本的最小化；
  - 根据实际需要达到适当的质量目标；
  - 根据实际需要形成各个版本（α，β和release）
  - 对所有必须的功能完成分析、设计、开发和测试工作；
  - 采用循环渐进的方式开发出一个可以提交给最终用户的完整产品；
  - 确定软件、站点、用户都为产品的最终部署做好了相关准备；
  - 达成一定程度上的并行开发机制。

§3 UP的阶段（构造阶段，construction）

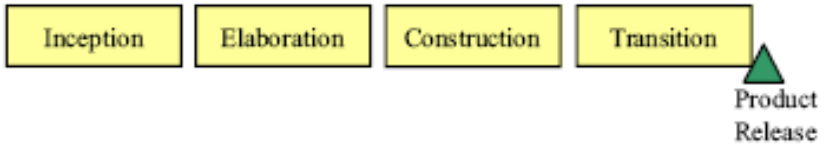


- 审核标准：
  - 产品是否足够稳定和成熟地发布给用户？
  - 是否所有的风险承担人准备好向用户移交？
  - 实际费用与计划费用的比较是否仍可被接受？
- 构造阶段的产出：
  - 特定平台上的集成产品；
  - 用户手册；
  - 当前版本的描述。

§3 UP的阶段（移交阶段，transition）

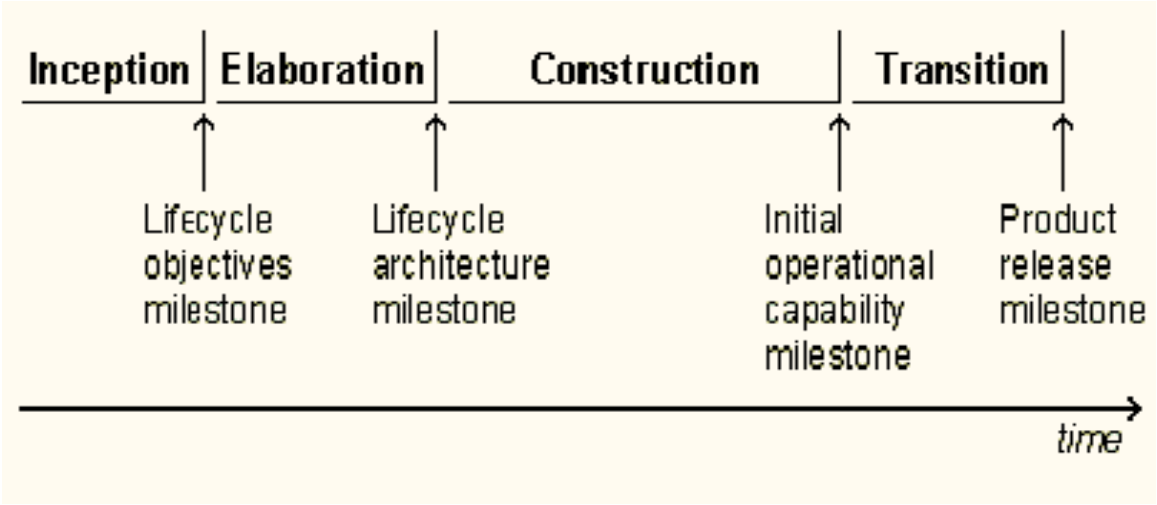
- 移交阶段的主要目标：确保软件产品可以提交给最终用户。
  - 进行β测试以期达到最终用户的需要；
  - β测试版本和旧系统的并轨；
  - 转换功能数据库；
  - 对最终用户和产品支持人员的培训；
  - 提交给市场和产品销售部门；
  - 和具体部署相关的工程活动；
  - 协调bug修订、改进性能和可用性(usability）等工作；
  - 基于完整的构想和产品验收标准对最终部署做出评估；
  - 达到用户要求的满意度；
  - 达成各风险承担人对产品部署基线已经完成的共识；
  - 达成各风险承担人对产品部署符合构想中标准的共识

§3 UP的阶段（移交阶段，transition）

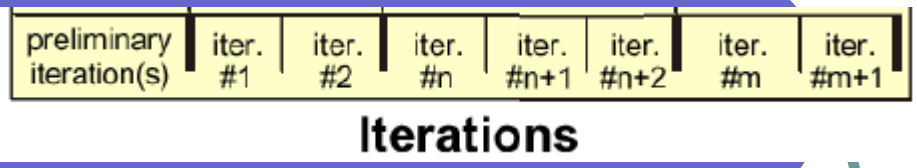


- 评审标准：
  - 用户是否满意？
  - 实际费用与计划费用的比较是否仍可被接受？

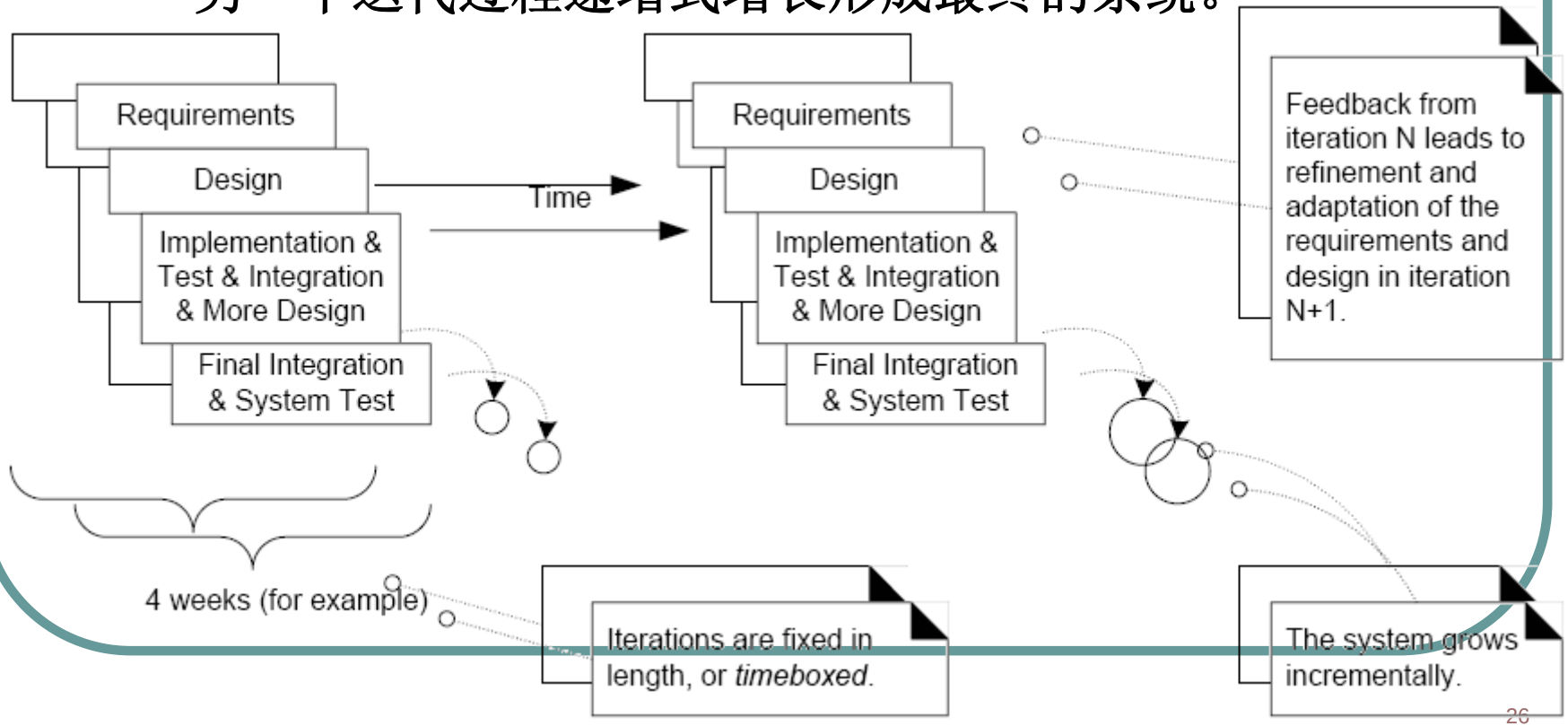
总结：



§3 迭代增量式开发

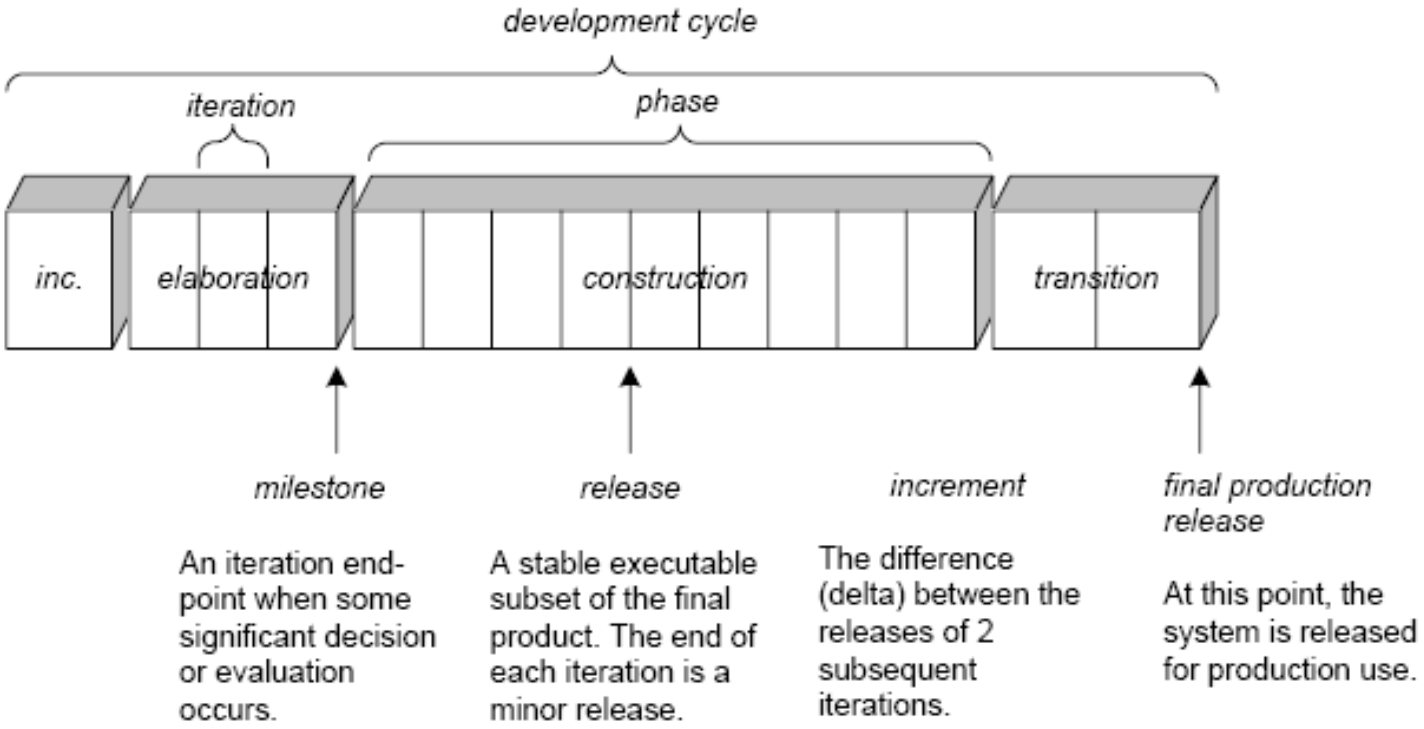


- UP的每个阶段可以进一步分为迭代过程。
  - **迭代过程是导致可执行产品版本（内部和外部）的完整开发循环，是最终产品的一个子集，从一个迭代过程到另一个迭代过程递增式增长形成最终的系统。**



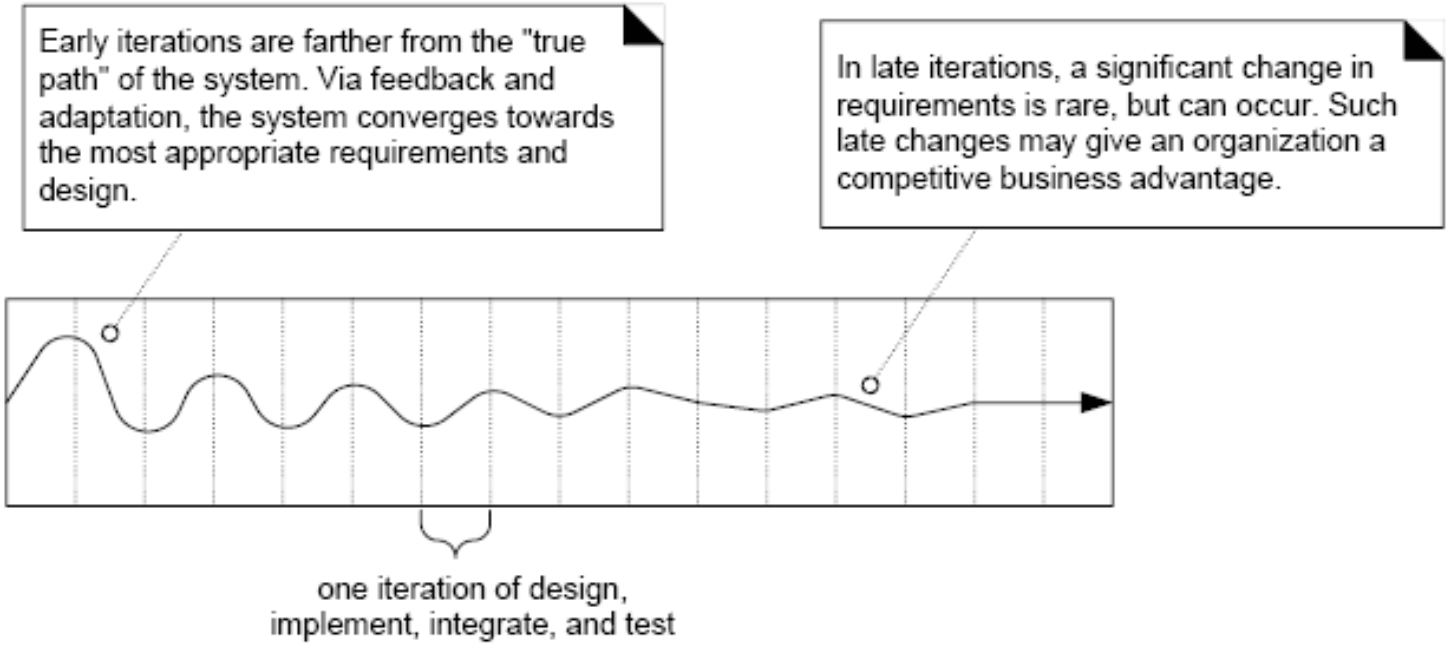
§3 迭代增量式开发

- UP中的迭代增量式开发（风险驱动）

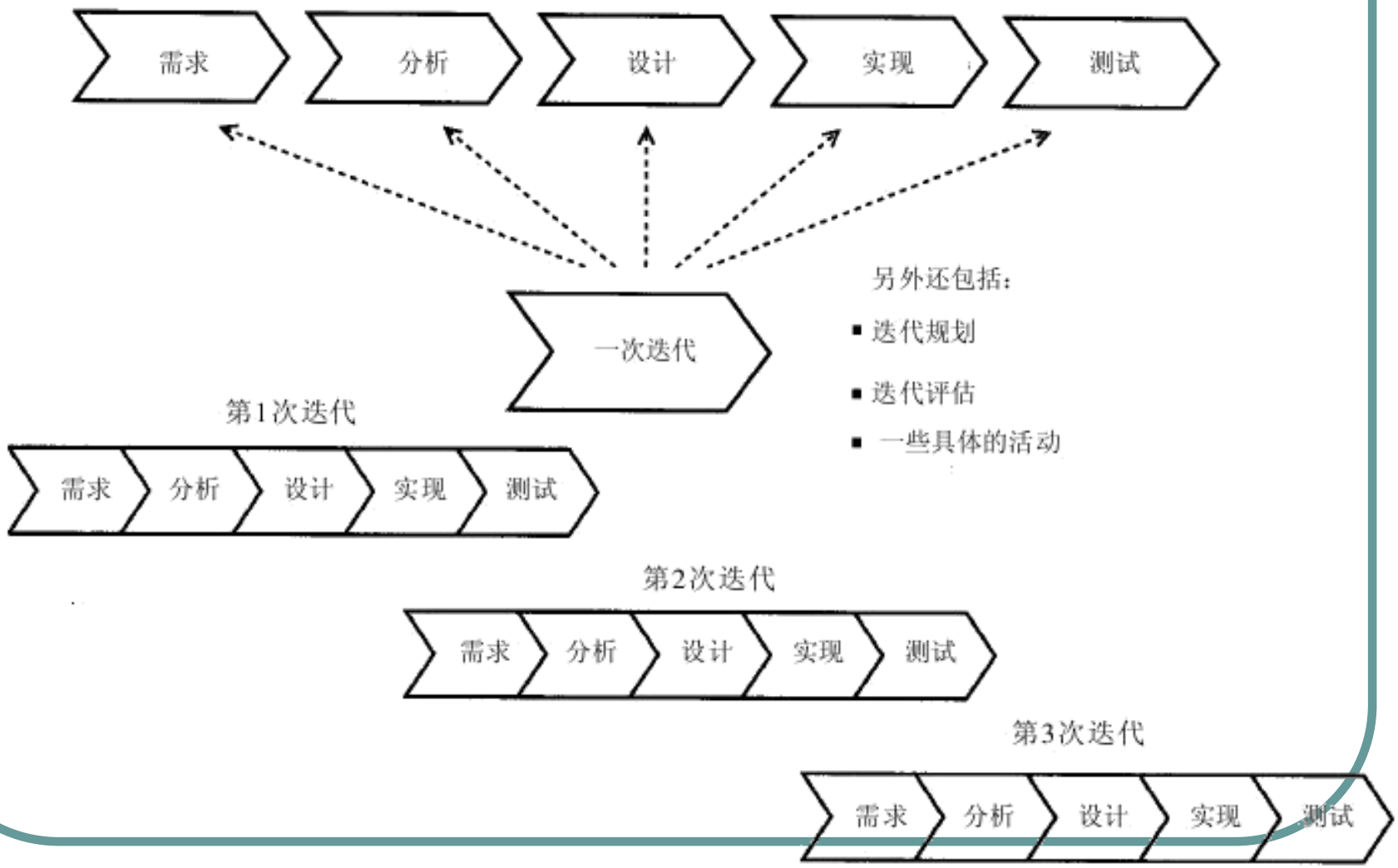


§3 迭代增量式开发

- 项目的主要风险集中在前两个阶段：
  - 在细化阶段中经过几次迭代后，为系统建立一个稳定的架构，之后在实现更多的系统需求时，不再对该架构进行修改。
  - 同时，在细化阶段中，通过迭代来不断地收集用户的需求反馈，便得系统的需求逐步地明确和完整。

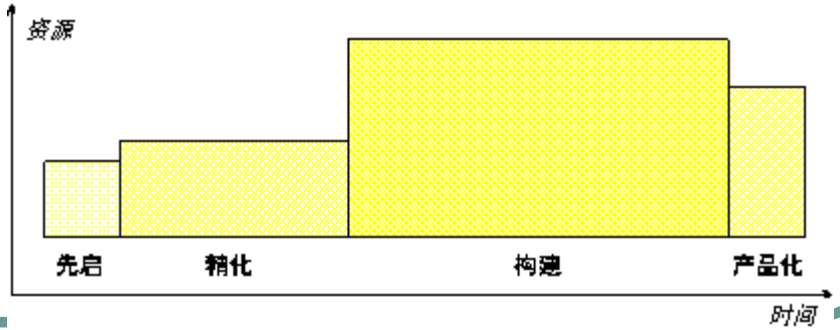


§3 迭代增量式开发


需求
分析
设计
实现
测试
一次迭代
另外还包括：
迭代规划
迭代评估
一些具体的活动
第1次迭代
需求
分析
设计
实现
测试
第2次迭代
需求
分析
设计
实现
测试
第3次迭代
需求
分析
设计
实现
测试

§3 迭代增量式开发

- 开发计划的组织
  - 项目计划
    确定整个项目的开发目标和进度安排，包括每一个阶段的起止时间段。
  - 阶段计划
    当前阶段中包含有几个迭代，每一次迭代要达到的目标以及进度安排。
  - 迭代计划
    针对当前迭代的详细开发计划，包括开发活动以及相关资源的分配。



§3 迭代增量式开发

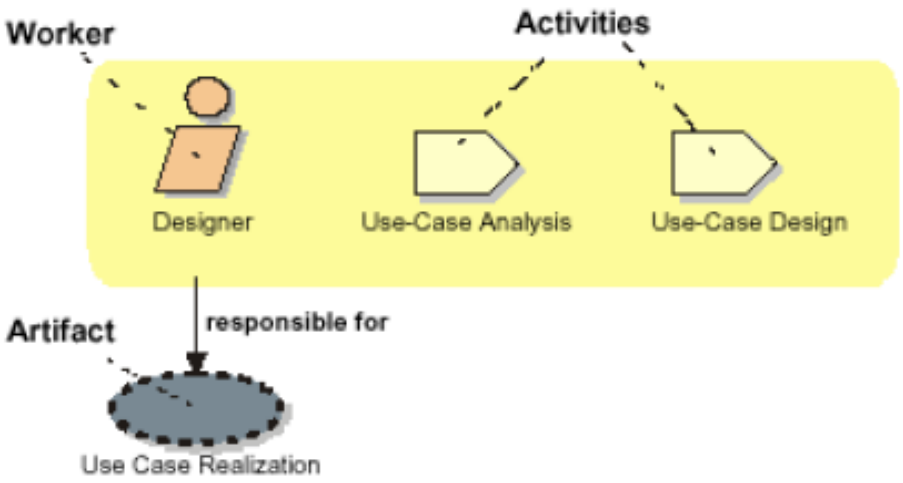
- 项目开发计划也是完全体现迭代化的思想：
  - 每次迭代中项目经理都会根据项目情况来不断地调整和细化项目开发计划。
  - 迭代计划是在对上一次迭代结果进行评估的基础上制定的，如果上一次迭代达到了预定的目标，那么当前迭代只需要解决剩下的问题；如果上一次迭代中留有一些问题还没有解决，则当前迭代还需要继续去解决这些问题。
  - 所以必须注意，迭代是不能重叠的，即当还没有完成当前迭代时，决不能进入下一迭代，因为下一次迭代的计划是根据当前迭代的结果而制定的。

提纲

1、UP/USDP概述
2、UP的基本结构
3、UP的阶段
4、核心工作流
5、最佳实践
6、UP工件

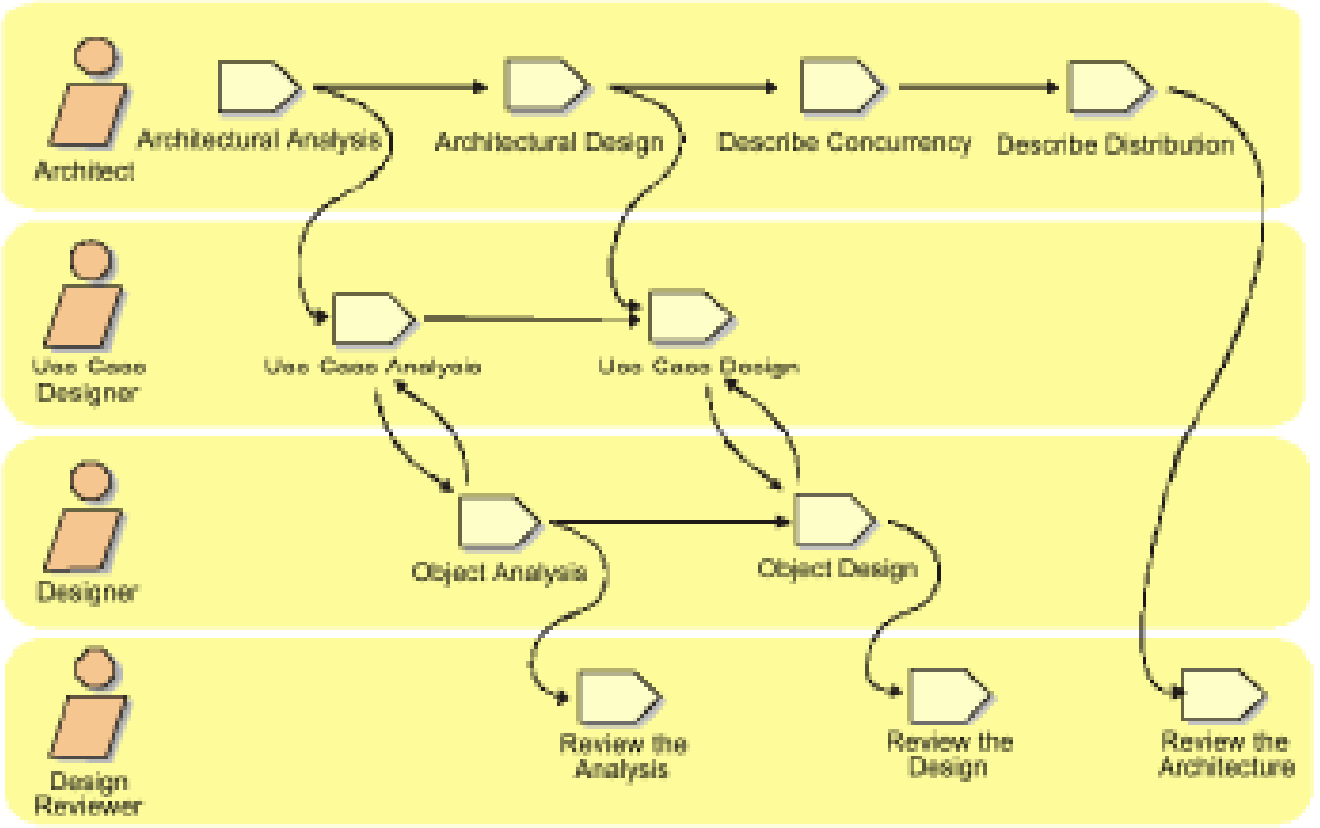
§4 核心工作流

- 软件开发流程定义了“谁”、“何时”、“如何”做“某事”。四种主要的建模元素被用来表达：
  - 角色（worker）“谁”
  - 活动（activity）“如何”
  - 工件（artifact）“某事”
  - 工作流（workflow，

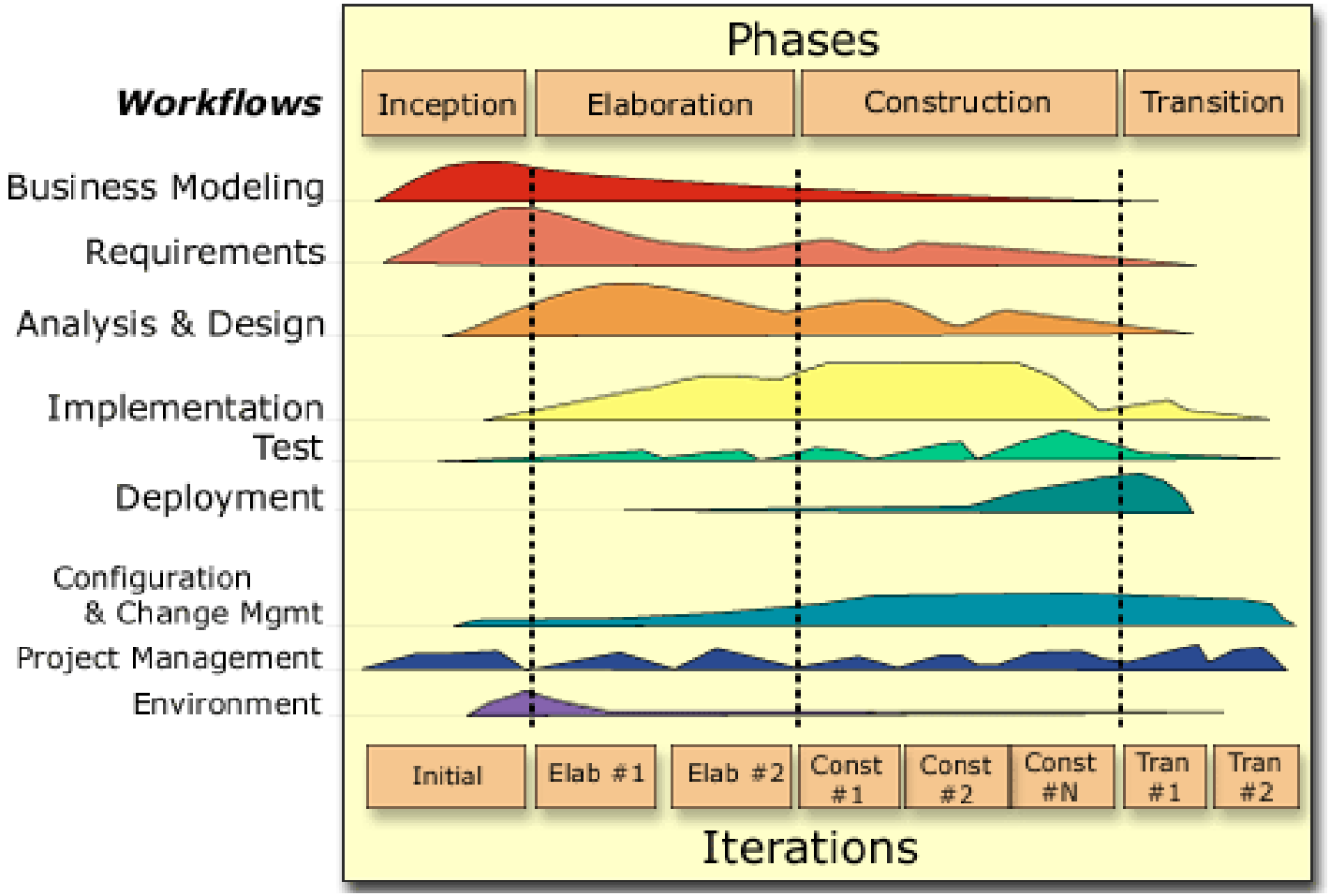


§4 核心工作流

- 工作流是产生具有可观察结果的活动序列



§4 核心工作流


Phases
Workflows
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration
& Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #N
Tran #1
Tran #2
Iterations

§4 核心工作流

- 业务建模
  - 问题是软件工程人员和业务人员之间不能正确地交流，业务工程的产出没有作为软件开发输入而正确地被使用，反之亦然。业务需求理解偏差
  - 在业务建模中使用业务用例来文档化业务过程，从而确保了组织中所有业务过程人员达到共识。
  - 业务用例被分析以理解业务过程如何被业务支持，而这些在业务对象模型中被核实。
  - 许多项目可能不进行业务建模。

业务模型是软件工程人员与用户交流的语言

§4 核心工作流

- 需求
  - 是描述系统应“做什么”，并允许开发人员和用户就该描述达成共识。
    - 创建构想
    - 建立用例模型
      - 识别actor
      - 识别use case
      - 描述use case
    - 其他功能和非功能性需求在补充规范中说明。

Use case起到贯穿整个系统的开发周期线索的作用，相同的用例模型在需求捕获阶段、分析/设计阶段和测试阶段中使用。

1、捕获需求

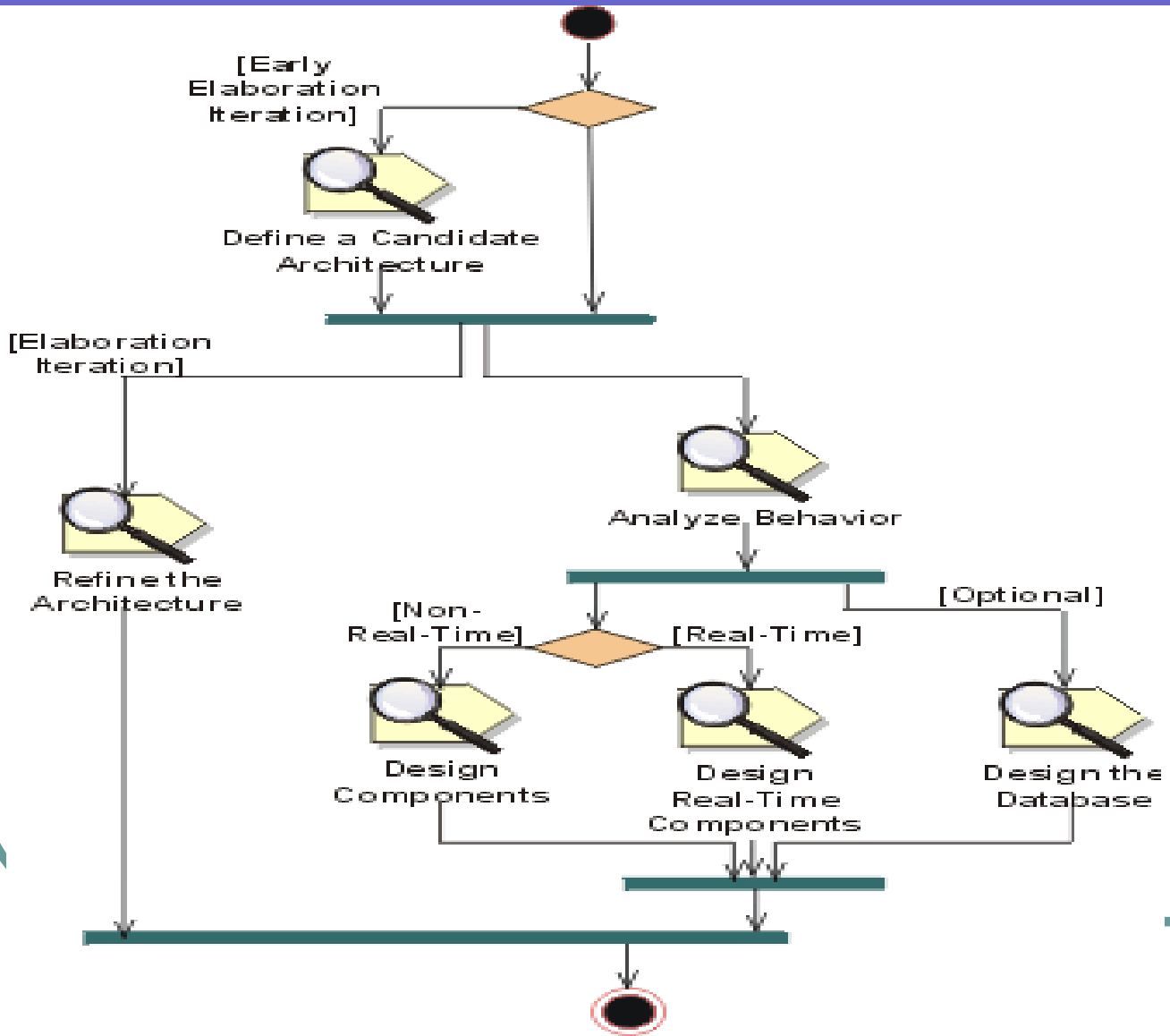
序号	输入	活动	执行者	输出
1	领域/业务模型， 补充需求，特征表	找出参与者 找出用例	系统分析员 客户/用户 其他分析员	用例模型（概述） 术语表
2	用例模型、补充需求、术语表	赋予用例优先级	体系结构设计者	体系结构描述（用例模型角度）
3	用例模型、补充需求、术语表	细化用例	用例描述者	用例（详述）
4	用例（详述）、用例模型、补充需求、术语表	人机接口原型	人机接口设计者	人机接口原型
5	用例（详述）、用例模型、补充需求、术语表	构造用例模型	系统分析员	用例模型（详述）

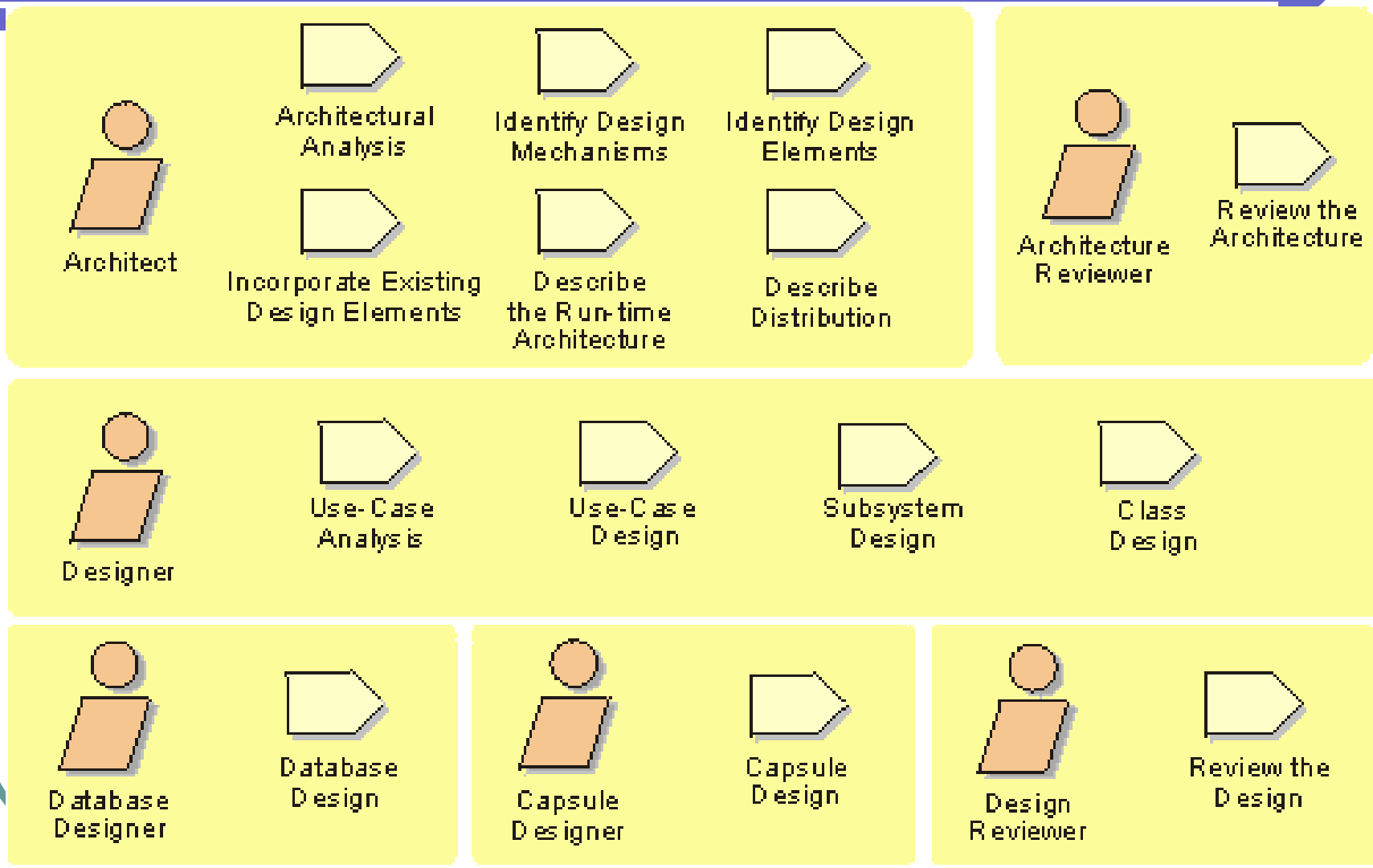
§4 核心工作流（分析和设计）

- 分析和设计
  - 显示系统“如何”在实现阶段被实现的，达到下列目标：
    - 在特定的实现环境中完成用例描述中指定的任务和功能
    - 满足了所有的需求
    - 健壮地被建造（如果功能需求发生变化，应该易于更改）
  - 分析设计结果是一个设计模型和可选的分析模型：
    - 设计模型由设计类和一些描述组成：
      - 设计类被组织成具有良好接口的设计包和设计子系统
      - 描述则体现了类的对象如何协同工作实现用例的功能
    - 设计模型是源代码的抽象

设计活动以体系结构设计为中心

系统分析、设计工作流


[Early Elaboration Iteration]
Define a Candidate Architecture
[Elaboration Iteration]
Refine the Architecture
Analyze Behavior
[Non-Real-Time]
[Real-Time]
[Optional]
Design Components
Design Real-Time Components
Design the Database


Architect
Architectural Analysis
Identify Design Mechanisms
Identify Design Elements
Incorporate Existing Design Elements
Describe the Run-time Architecture
Describe Distribution
Architecture Reviewer
Review the Architecture
Designer
Use-Case Analysis
Use-Case Design
Subsystem Design
Class Design
Database Designer
Database Design
Capsule Designer
Capsule Design
Design Reviewer
Review the Design

2、分析

序号	输入	活动	执行者	输出
1	用例模型、补充需求，业务/领域模型、体系结构描述（用例模型角度）	体系结构分析	体系结构设计者	分析包（概述） 分析类（概述） 体系结构描述（分析模型角度）
2	用例模型、补充需求，业务/领域模型、体系结构描述（分析模型角度）	分析用例	用例工程师	用例（实现-分析） 分析类（概述）
3	用例（实现-分析） 分析类（概述）	对类分析	构件工程师	分析类（完成）
4	分析包（概述） 体系结构描述（分析模型角度）	对包进行分析	构件工程师	分析包（完成）

3、设计

序号	输入	活动	执行者	输出
1	用例模型、 补充需求， 分析模型、 体系结构描述（分析模型角度）	体系结构设计	体系结构设计者	子系统（概述） 接口（概述） 设计类（概述） 部署模型（概述） 体系结构描述（设计/部署角度）
2	用例模型、补充需求， 分析模型、设计模型 部署模型	设计用例	用例工程师	用例（实现-设计） 设计类（概述） 子系统（概述） 接口（概述）
3	用例（实现-设计） 设计类（概述） 接口（概述） 分析类（完成）	对类设计	构件工程师	设计类（完成）
4	体系结构描述（设计模型角度）、 子系统（概述） 接口（概述）	设计子系统	构件工程师	子系统（完成） 接口（完成）

§4 核心工作流（实现）

- 实现
  - 目的：
    - 定义代码的组织结构——以层次化方式组织实施子系统；
    - 实现类和对象——以构件的形式（源文件、二进制文件、可执行文件等）；
    - 将开发出的构件作为单元进行测试；
    - 将由单个实现者或小组产生的结果集成为可执行的系统。

4、实现

序号	输入	活动	执行者	输出
1	设计模型、 部署模型、 体系结构描述（设计/部署模型角度）	实现体系结构	体系结构设计者	构件（概述） 体系结构描述（实现/部署角度）
2	用例模型、补充需求， 设计模型、 实现模型（当前建造）	集成系统	系统集成者	集成建造计划 实现模型（连续建造）
3	集成建造计划 体系结构描述（实现模型角度） 设计子系统（已设计） 接口（已设计）	实现子系统	构件工程师	实现子系统（完成） 接口（完成）
4	设计类（已设计） 接口（由设计类提供）	实现类	构件工程师	构件（完成）
5	构件（完成） 接口	完成单元测试	构件工程师	构件（完成单元测试）

§4 核心工作流（测试）

- 测试
  - 目的
    - 验证对象间的交互作用；
    - 验证软件构件的正确集成；
    - 验证所有需求被正确的实现；
    - 识别并确保在软件发布之前缺陷被处理。

■UP提出了迭代的方法，意味着在整个项目中进行测试，从而允许尽可能早地发现缺陷，从根本上降低了修改缺陷的成本；
■测试生命周期的几个阶段：计划、设计、实现、执行和审核。

§4 核心工作流（发布）

- 发布
  - 目标是成功地生成版本，将软件分发给最终用户。包含的活动：
    - 生成软件本身外的产品；
    - 软件打包
    - 安装软件
    - 给用户提供帮助

许多情况下还包括如下的活动：

- 计划和进行β测试版
- 移植现有的软件或数据
- 正式验收

§4 核心工作流（配置和变更管理）

- 配置和变更管理
  - 完成建立并管理基线的任务。
    - 基线：已经通过正式复审和批准的某规约或产品，它因此可以作为进一步开发的基础，并且只能通过正式的变更控制过程进行改变。
    - 配置项：置于配置和变更管理控制之下的工件。
  - 提供了管理系统演化中的多个变体、跟踪软件版本的准则；
  - 描述了如何管理并行开发、分布式开发，如何自动化创建工程；
  - 涵盖了需求变更管理，即：如何报告缺陷？如何管理缺陷？及如何使用缺陷来跟踪进展和发展的倾向？

§4 核心工作流（项目管理、环境）

- 项目管理
  - 集中在迭代开发过程的组织管理方面
  - 目标是提供以下的事物来使该任务更简单：
    - 管理项目的框架；
    - 计划、配备、执行、监控项目的实践准则；
    - 管理风险的框架。

§4 核心工作流（项目管理、环境）

- 环境
  - 目的是给软件开发组织提供软件开发环境（过程和工具），软件开发队伍需要它们的支持；
  - 集中在项目环境中配置过程的活动，同样着重支持项目的开发规范的活动，提供了逐步指导手册，介绍了如何在组织中实现过程；
  - 还包含了定制流程所必须的准则、模板、工具的开发工具箱。

提纲

1、UP/USDP概述
2、UP的基本结构
3、UP的阶段
4、核心工作流
5、最佳实践
6、UP工件

§5 最佳实践

- 短时间分区式的迭代：2～6周，不鼓励时间推迟
- 适应性开发：小步骤、快速反馈和调整
- 在早期迭代中解决高风险和高价值的问题
- 不断地让用户参与评估、反馈和需求；
- 在早期迭代中建立内聚的核心架构
- 不断地验证质量；提早、经常和实际地测试；
- 使用用例：获取需求、制定计划、进行设计、测试、编写终端用户文档的驱动力量。
- 可视化软件建模（使用UML）
- 仔细地管理需求（需求提出、记录、等级划分、追踪和生命周期跟踪。）
- 实行变更请求和配置管理。

提纲

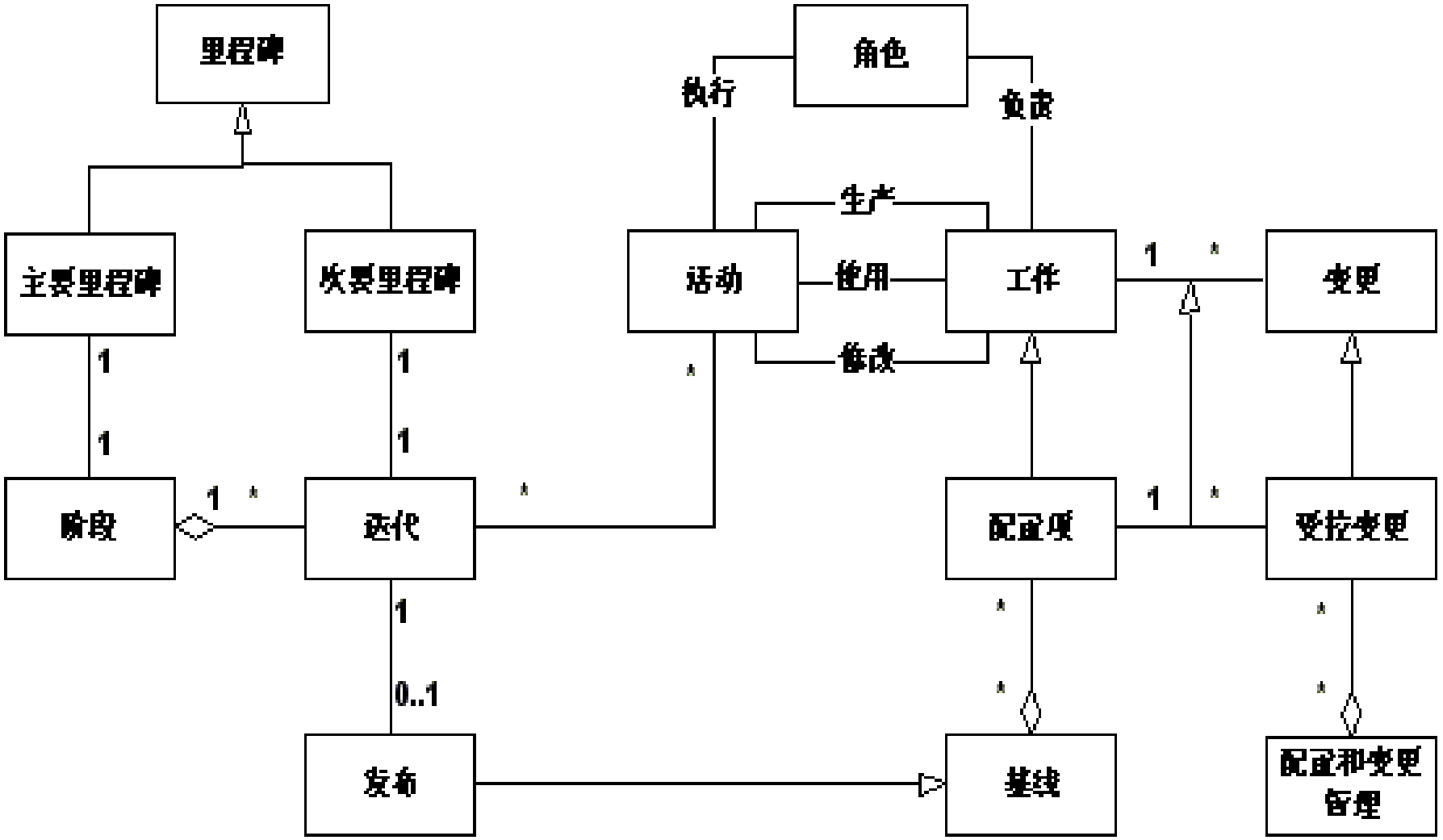
1、UP/USDP概述
2、UP的基本结构
3、UP的阶段
4、核心工作流
5、最佳实践
6、UP工件

§6 UP工件

流程	工件	初始 I1	细化 E1..En	构造 C1..Cn	移交 T1..Tn
业务建模	领域模型		S		
需求	用例模型	S	R		
	构想	S	R		
	补充规范	S	R		
	术语表	S	R		
设计	设计模型		S	R	
	软件架构文档		S		
	数据模型		S	R	
实现	实现模型		S	R	R
项目管理	软件开发计划	S	R	R	R
测试	测试模型		S	R	
环境	开发案例	S	R		

S表示开始（Start），R表示提炼（Refine）

总结


里程碑
角色
执行
负责
生产
主要里程碑
次要里程碑
活动
使用
工作
变更
修改
1
1
1
1
1
*
*
1
*
阶段
迭代
*
1
*
配置项
受控变更
1
0..1
*
*
*
*
发布
基线
配置和变更管理