统一过程

内容

- 概述
- 过程模型
- 特点
- 最佳实践
- 剪裁

内容

- 概述
- 过程模型
- 特点
- 最佳实践
- 剪裁

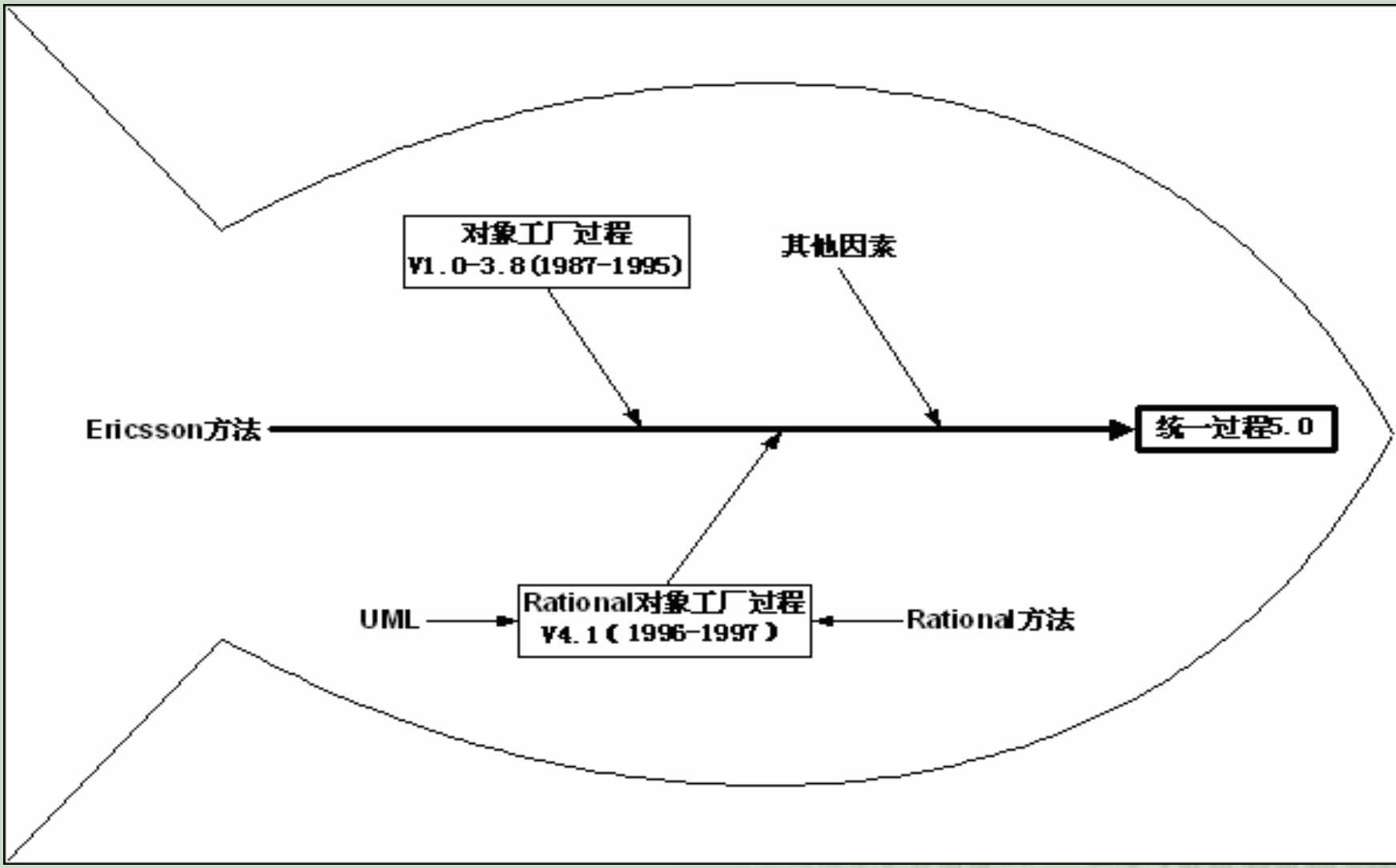
UML与软件过程

- UML只是一种建模语言，并不包含对软件开发过程的指导。
- UML是独立于过程的，可以用于不同的开发过程。
- UML的使用方式在很大程度上取决于过程。
- 如果不了解建模技术如何适应过程，那建模技术也就没有意义了。

统一过程

- 统一过程(Unified Process)是一个将用户需求转化为软件系统所需要完成的一系列任务的框架，规定了完成各项任务的步骤。
- 统一过程使用UML来制定软件系统的蓝图。

统一过程的历史


对象工厂过程
V1.0-3.8(1987-1995)
其他因素
Ericsson方法
统一过程5.0
UML
Rational对象工厂过程
V4.1(1996-1997)
Rational方法

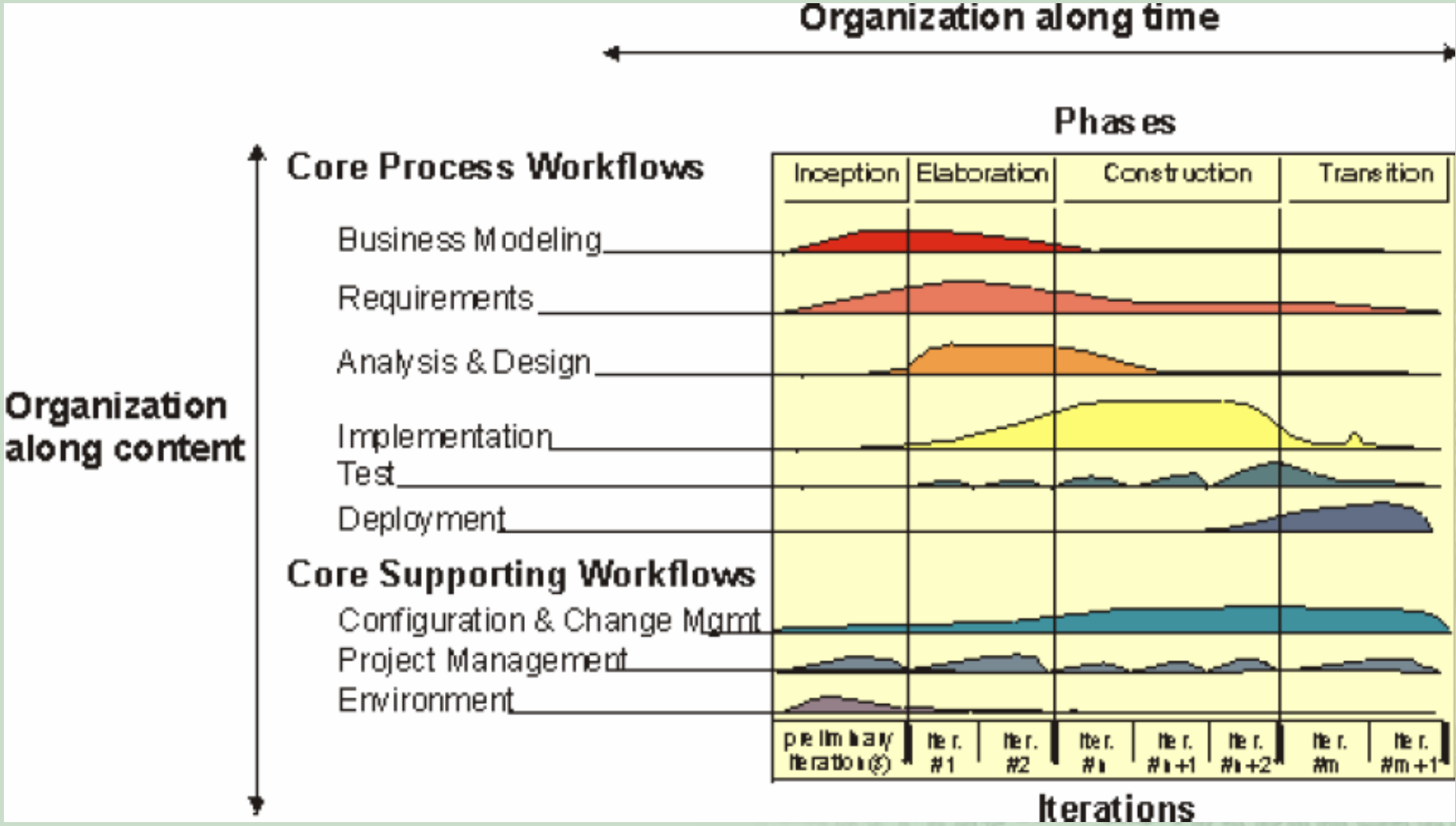
内容

- 概述
- 过程模型
- 特点
- 最佳实践
- 剪裁

统一过程模型

统一过程是一个二维生命周期模型，纵轴代表核心工作流，横轴代表时间。一个工作流可以分配在若干恰当的时间阶段完成；一个阶段可以涉及多项工作流。

统一过程模型


Organization along time
Phases
Inception
Elaboration
Construction
Transition
Core Process Workflows
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Core Supporting Workflows
Configuration & Change Mgmt
Project Management
Environment
Organization along content
Iterations

核心工作流

UP为每次软件开发过程定义了九个核心工作流，以保证开发过程的有效执行，其中包括六个技术性工作流和三个管理性工作流。

- 技术性工作流包括：业务建模、需求、分析和设计、实现、测试、部署。
- 管理性工作流包括：项目管理、配置和变更管理、环境。

阶段

- UP将大型项目开发分解为较小的几部分，先分析一部分需求和风险，并加以实现和确认，再做更多的需求分析、实现和确认，通过迭代重复一系列过程，降低每次迭代的风险，以保证最终向用户提交一个完整、稳定的高质量产品。
- 整个迭代的渐进式软件开发过程包括4个阶段：初启(Inception)、细化(Elaboration)、构造(Construction)和移交(Transition)。每个阶段可以根据需要细分为多次迭代。

初启

在初启阶段，软件项目的发起人确立项目的主要目标和范围，并进行初步的可行性分析和经济效益分析。

细化

细化阶段的开始标志着项目的正式确立。

软件项目组在此阶段需要完成以下工作：

- 初步的需求分析
- 初步的高层设计
- 部分的详细设计
- 部分的原型构造

细化

初步的需求分析：

采用UML的用例描述目标软件系统所有比较重要、比较有风险的用例，利用用例图表示参与者与用例以及用例与用例之间的关系。

采用UML的类图表示目标软件系统所基于的应用领域中的概念与概念之间的关系。这些相互关联的概念构成领域模型。如果领域中含有明显的流程处理成分，可以考虑利用UML的活动图来刻画领域中的工作流，并标识业务流程中的并发、同步等特征。

细化

初步的高层设计：考虑根据用例、类在业务领域中的关系，或者根据业务领域中某种有意义的分类方法将整个软件系统划分为若干个包，利用UML的包图刻画这些包及其间的关系。这样，用例、用例图、类、类图将依据包的划分方法分属于不同的包，从而得到整个目标软件系统的高层结构。

细化

部分的详细设计：

对于系统中某些重要的或者风险比较高的用例，可以采用交互图进一步探讨其内部实现过程。

同样对于系统中的关键类，也可以详细研究其属性和操作，并在UML类图中加以表现。

细化

因此这里倡导的软件开发过程并不在时间轴上严格划分分析与设计、总体设计与详细设计，而是根据软件元素（用例、类等）的重要性和风险程度确立优先细化原则，建议软件项目组优先考虑重要的、比较有风险的用例和类，不能将风险的识别和解决延迟到细化阶段之后。

细化

部分的原型构造：在许多情形下，针对某些复杂的用例构造可实际运行的原型是降低技术风险、让用户帮助软件项目组确认用户需求的最有效的方法。为了构造原型，需要针对用例生成详细的交互图，对所有的相关类给出明确的属性和操作定义。

细化

在细化阶段可能需要使用的UML语言机制包括：描述用户需求的用例及用例图、表示领域概念模型的类图、表示业务流程处理的活动图、表示系统高层结构的包图和表示用例内部实现过程的交互图等。

细化

细化阶段的结束条件是，所有主要的用户需求已通过用例和用例图得以描述；所有重要的风险已被标识，并对风险应对措施了如指掌；系统的构架已被确认；能够比较精确地估算实现每一用例的时间。

构造

在构造阶段，开发人员通过一系列的迭代完成对所有用例的软件实现工作，在每次迭代中实现一部分用例。

构造

在实际开始构造软件系统之前，有必要预先制定迭代计划。

计划的制定需要遵循两项原则：1.用户认为业务价值较大的用例应优先安排；2.开发人员评估后认为开发风险较高的用例应优先安排。

在迭代计划中，要确立迭代次数、每次迭代所需时间以及每次迭代中应完成（或部分完成）的用例。

构造

每次迭代过程由针对用例的分析、设计、编码、测试和集成5个子阶段构成。

在集成之后，用户可以对用例的实现效果进行评价，并提出修改意见。这些修改意见可以在本次迭代过程中立即实现，也可以在下次迭代中再予以考虑。

构造

构造过程中需要使用UML的交互图来设计用例的实现方法。

为了与设计得出的交互图协调一致，需要修改或精化在细化阶段绘制的作为领域模型的类图，增加一些为软件实现所必需的类、类的属性或方法。

如果一个类有复杂的生命周期行为，或者类的对象在生命周期内需要对各种外部事件的刺激作出反应，应考虑用UML状态图来表示类的对象的行为。

构造

UML的活动图可以在构造阶段用来表示复杂的算法过程和有多个对象参与的业务处理过程。活动图尤其适用于表示过程中的并发和同步。

在构造阶段的每次迭代过程中，可以对细化阶段绘出的包图进行修改或精化，以便包图切实反映目标软件系统最顶层的结构划分状况。

构造

综上，在构造阶段可能需要使用的UML语言机制包括：

- 用例及用例图：它们是开发人员在构造阶段进行分析和设计的基础。
- 类图：在领域概念模型的基础上引进软件实现所必需的类、属性和方法。
- 交互图：表示针对用例设计的软件实现方法。
- 状态图：表示类的对象的状态-事件-响应行为。
- 活动图：表示复杂的算法过程，尤其是过程中的并发和同步。
- 包图：表示目标软件系统的顶层结构。
- 构件图
- 部署图

移交

在移交阶段，开发人员将构造阶段获得的软件系统在用户实际工作环境（或接近世界的模拟环境）中运行，根据用户的修改意见进行少量调整。

UP工件和时限样例

Discipline	Artifact Iteration→	Incep. I1	Elab. E1..En	Const. C1..Cn	Trans. T1..T2
Business Modeling	*Domain Model*		s		
Requirements	Use-Case Model (SSDs)	s	r		
	Vision	s	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	
Implementation	Implementation Model		s	r	r
Project Management	SW Development Plan	s	r	r	r
Testing	Test Model		s	r	
Environment	Development Case	s	r		

内容

- 概述
- 过程模型
- 特点
- 最佳实践
- 剪裁

统一过程的特点

统一过程的特点是：用例驱动、以构架为中心、迭代和增量的。

用例驱动

统一过程是用例驱动的。

软件系统是为了服务于它的用户而出现的。因此为了构造一个成功的软件系统，必须了解其预期的用户所希望和需要的是什么。

用户代表了与所开发的系统进行交互的某个人或其它系统。用例是能够向用户提供有价值结果的系统中的一种功能。

用例驱动

用例获取的是功能需求。所有的用例合在一起构成用例模型，描述了系统的全部功能：系统应该为每个用户做什么。

用例促使我们从系统对用户的价值方面来考虑问题。

用例驱动

用例不只是一种确定系统需求的工具，还能驱动系统设计、实现和测试的进行，即用例可以驱动开发过程。

用例不仅启动了开发过程而且使其结合为一体。用例驱动表明开发过程是沿着一个流——一系列从用例得到的工作流前进的：用例被确定，用例被设计，用例被实现，最后用例又成为测试人员构造测试用例的基础。

用例驱动

用例与系统构架是协调发展的：

用例驱动系统构架，系统构架反过来影响用例的选择。

系统构架和用例会随着生命周期的延续而逐渐发展。

以构架为中心

统一过程是以构架为中心的。

软件系统的构架从不同角度描述了即将构造的系统。

软件构架概念包含了系统中最重要的静态和动态特征。

构架刻画了系统的整体设计，它去掉了细节部分，突出了系统的重要特征。

以构架为中心

软件开发并不仅是依赖用例驱动工作流就能够完成的。

大型软件系统如果没有构架设计，会造成开发人员无法准确理解要开发的系统，无法向同一个目标共同努力。

软件构架的好处包括：便于理解系统、利于组织开发、适用于重用要求、有利于系统的改进和发展。

以构架为中心

构架是根据企业的需要逐渐发展起来的，受到用户和其他项目相关人员需求的影响，并在用例中得到反映，同时受到其他许多因素的影响，如：软件应用平台、是否有可重用的构造块、如何考虑实施问题、如何与遗留系统整合以及非功能需求。

以构架为中心

用例和构架是如何相关的？

每种产品都具有功能和表现形式两个方面，必须恰当权衡才能得到成功的产品。功能与用例对应，表现形式与构架对应。

用例和构架是相互影响的，一方面用例在实现时必须适合于构架；另一方面，构架必须预留空间以实现现在或将来所有需要的用例。

二者必须并行进化。

以构架为中心

构架必须设计得使系统能够进化，不仅要考虑系统的初始开发，而且要考虑将来的发展，应该是开放的。

迭代和增量的过程

统一过程是迭代和增量的过程。

开发一个复杂软件产品是一项艰巨的任务，将它划分为较小的部分或袖珍项目是切实可行的。每个袖珍项目都是一次能够产生一个增量的迭代过程。

迭代是指工作流中的步骤，而增量是指产品中增加的部分。

为了获得最佳的效果迭代过程必须是受控的。

迭代和增量的过程

开发人员基于两个因素来确定在一次迭代过程中要实现的目标：

首先，迭代过程要处理一组用例，这些用例合起来能够扩展所开发产品的可用性。

其次，迭代过程要解决最突出的风险问题。

迭代和增量的过程

在每次迭代过程中，开发人员标识并详细描述有关的用例，以选定的构架为向导来创建设计，用构件来实现设计，并验证这些构件是否满足用例。

如果一次迭代达到了目标开发工作可进入下一次迭代。如果一次迭代未能达到预期目标开发人员必须重新审查前面的方案，并试用一种新的方法。

迭代和增量的过程

为了在开发过程中达到最佳的经济效益，项目组应设法选择实现项目目标所需要的迭代过程，并按照一定的逻辑顺序来排列这些迭代过程。

迭代和增量的过程

受控迭代过程具有很多好处：

将成本风险降低为获得一次增量所需要的费用。

可以降低产品不能按计划投放市场的风险。

可以控制甚至加快整个项目的进展速度。

有利于精确获取用户需求和变化。

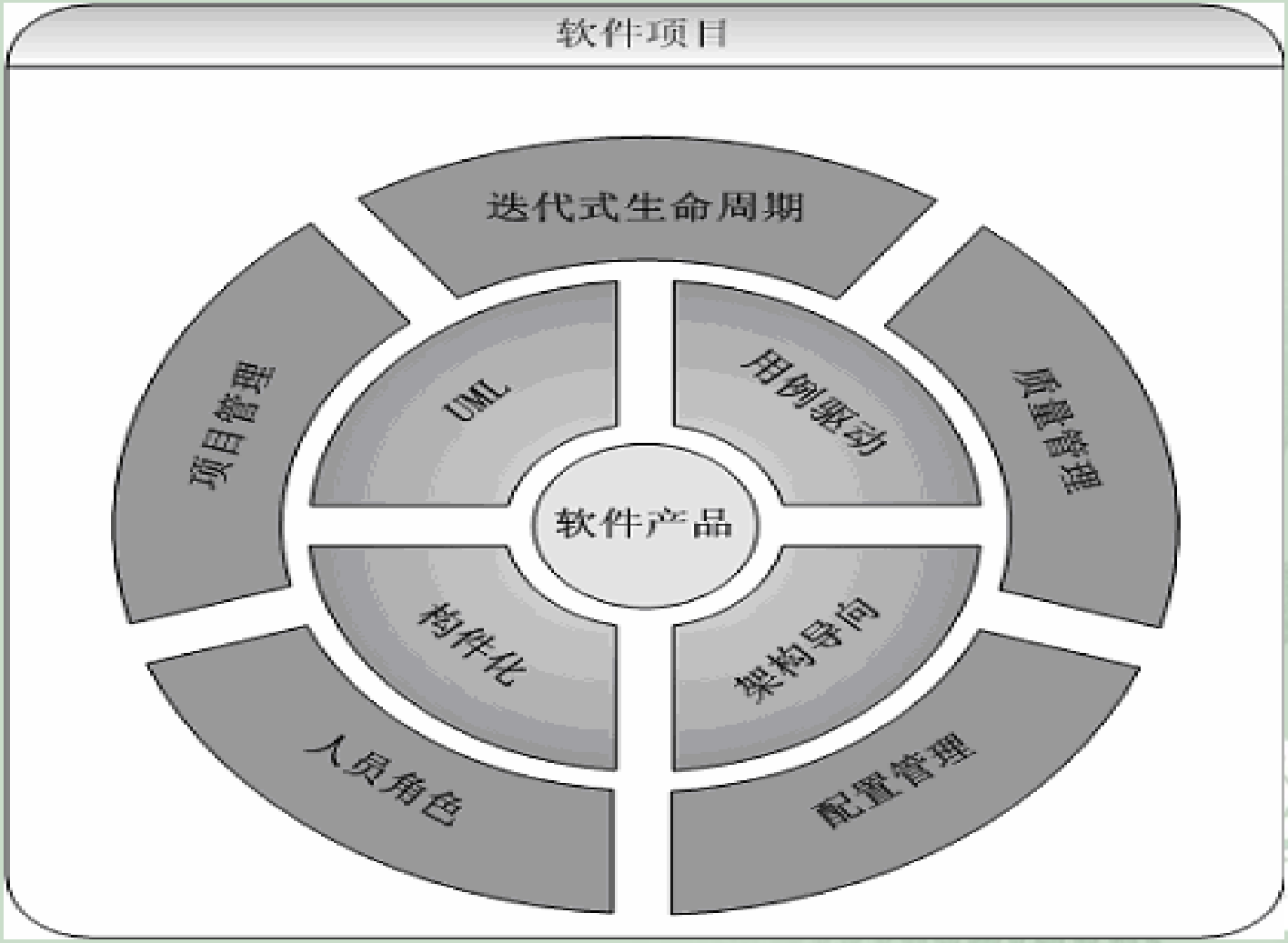
内容

- 概述
- 过程模型
- 特点
- 最佳实践
- 剪裁

最佳实践

- 迭代式开发
- 需求管理
- 使用基于构件的体系结构
- 可视化软件建模
- 验证软件质量
- 控制软件变更

最佳实践



UP最佳实践

内容

- 概述
- 过程模型
- 特点
- 最佳实践
- **剪裁**

UP的裁剪

没有一个适用于所有软件项目的任务集合，科学有效的软件过程应该定义一组适合当前项目特点的任务集合。

UP必须通过剪裁为特定项目进行实例化。

UP的裁剪

UP裁剪可以分为以下几步：

- 确定本项目需要哪些工作流
- 确定每个工作流需要哪些输入制品，进行加工后，需要输出那些制品
- 确定4个阶段之间如何演进
- 确定每个阶段内的迭代计划
- 规划工作流内部结构

内容

- 概述
- 过程模型
- 特点
- 最佳实践
- 剪裁