**（PPT根据助教说明已删减请注意，明天我会讲一下有关被点中讲PPT后上去讲的注意事**

**项**）

* （常见的系统开发周期模型：瀑布模型，V模型，原型化模型、螺旋模型、迭代模型）
* 迭代周期(一次完整地经过所有工作流程的过程)

迭代模型概念

为了进一步规避项目风险，通常根据需要在各里程碑阶段中划分一次或多次迭代开发过程，以滚动演进的方式分次实现里程碑目标。

一次迭代，根据其所处的阶段，将不同力度的业务分析、需求开发、分析设计、编码实现、测试与部署等开发活动按一种松散的顺序组合在一起。在项目先启、精化架构阶段的迭代活动集中于项目管理、需求开发、设计等方面；而在构建源码阶段，则集中于设计、编码实现与测试等方面；到了产品化过渡阶段，其焦点则成了测试与部署等活动。

迭代概念的提出，很大程度上是为了便于实施进度控制。每次迭代置于时间框方式的管理下，项目组必须优先保证迭代满足计划进度，并通过裁减迭代的内容范围来跟上进度期限。

注意迭代过程的最终目标仍是实现其所在里程碑阶段设定的目标。比如某项目的架构设计难度很大，存在风险，于是在其精化架构阶段划分验证原型开发与系统构架优化两次迭代，这两次迭代的目的都是为了建立正确的构架基线，而此阶段的验证原型不能用于对用户的交付（这是源码构建和产品化过渡迭代才能做的事）。为了赶工期，可以在源码构建阶段，根据产品功能优先级划分多次源码构建迭代，并将产品化过渡阶段提前与构建阶段的后续迭代重叠，分别向用户交付迭代演进版本。

需强调的是，产品化过渡阶段提前到与精化架构阶段重叠是有风险的。

软件工程领域，与RUP齐名的软件方法还有：

净室软件工程、CMMI；

极限编程（extreme programming，简称 XP）和其他敏捷软件开发（agile methodology）方法学。

迭代式开发

在软件开发的早期阶段就想完全、准确的捕获用户的需求几乎是不可能的。实际上，我们经常遇到的问题是需求在整个软件开发工程中经常会改变。迭代式开发允许在每次迭代过程中需求可能有变化，通过不断细化来加深对问题的理解。迭代式开发不仅可以降低项目的风险，而且每个迭代过程都以可执行版本结束，可以鼓舞开发人员。

管理需求

确定系统的需求是一个连续的过程，开发人员在开发系统之前不可能完全详细的说明一个系统的真正需求。RUP描述了如何提取、组织系统的功能和约束条件并将其文档化，用例和脚本的使用已被证明是捕获功能性需求的有效方法。

体系结构

组件使重用成为可能，系统可以由组件组成。基于独立的、可替换的、模块化组件的体系结构有助于降低管理复杂性，提高重用率。RUP描述了如何设计一个有弹性的、能适应变化的、易于理解的、有助于重用的软件体系结构。

可视化建模

RUP往往和UML联系在一起，对软件系统建立可视化模型帮助人们提供管理软件复杂性的能力。RUP告诉我们如何可视化的对软件系统建模，获取有关体系结构于组件的结构和行为信息。

验证软件质量

在RUP中软件质量评估不再是事后进行或单独小组进行的分离活动，而是内建于过程中的所有活动，这样可以及早发现软件中的缺陷。

控制软件变更

迭代式开发中如果没有严格的控制和协调，整个软件开发过程很快就陷入混乱之中，RUP描述了如何控制、跟踪、监控、修改以确保成功的迭代开发。RUP通过软件开发过程中的制品，隔离来自其他工作空间的变更，以此为每个开发人员建立安全的工作空间。

过程简介

二维结构

开发过程可以用二维结构或沿着两个坐标轴来表达：

横轴代表了制订开发过程时的时间，体现了过程的动态结构。它以术语周期（cycle）、阶段(phase)、迭代（iteration）和里程碑(milestone)来表达。

纵轴表现了过程的静态结构：如何用术语活动（activity）、产物(artifact)、 角色(worker)和工作流(workflow)来描述。

初始阶段（**为系统建立商业案例和确定项目的边界**）

意义：在这个阶段中，关注的是整个项目进行工程中的业务和需求方面的主要风险。对于建立在原有系统基础上的开发项目来说，初始阶段的时间可能很短。

本阶段的主要目标如下：

* 明确软件系统的范围和边界条件，括从功能角度的前景分析、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（Use-case） 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)
* 估计出潜在的风险(主要指各种不确定因素造成的潜在风险)
* 准备好项目的支持环境

初始阶段的产出是:

* 蓝图文档核心项目需求关键特色主要约束的总体蓝图
* 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)
* 原始商业案例,包括业务的上下文、验收规范（年度映射、市场认可等等），成本预计
* 原始的风险评估
* 一个或多个原型
* 里程碑：生命周期的目标

初始阶段结束时是第一个重要的里程碑：生命周期目标里程碑。初始阶段的评审标准：

* 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、风险和开发过程的可信度
* 被开发体系结构原型的深度和广度
* 实际开支与计划开支的比较

如果无法通过这些里程碑，则项目可能被取消或仔细地重新考虑。

**细化阶段（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。）**

本阶段的主要目标如下：

* 确保软件结构、需求、计划足够稳定；确保项目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。

细化阶段的产出是：

* 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非关联于特定用例要求的需求
* 软件体系结构描述\_可执行的软件原型
* 经修订过的风险清单和商业案例
* 总体项目的开发计划，包括纹理较粗糙的项目计划，显示迭代过程和对应的审核标准
* 指明被使用过程的更新过的开发用例
* 用户手册的初始版本（可选）

里程碑：生命周期的结构

细化阶段结束是第二个重要的里程碑：生命周期的结构里程碑。此刻，检验详细的系统目标

和范围、结构的选择以及主要风险的解决方案。主要的审核标准包括回答以下的问题：

* 产品的蓝图是否稳定？
* 体系结构是否稳定？
* 可执行的演示版是否显示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蓝图是可实现的？
* 实际的费用开支与计划开支是否可以接受？

如果无法通过这些里程碑，则项目可能被取消或仔细地重新考虑。

**构建阶段（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）**

本阶段的主要目标如下：

* 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
* 据实际需要形成各个版本（Alpha,Beta,and other test 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

构建阶段的产出是可以交付给最终用户的产品。它最小包括：

* 特定平台上的集成产品
* 用户手册
* 当前版本的描述

里程碑：初始运作能力

创建阶段结束是第三个重要的项目里程碑（初始功能里程碑）。此刻，决定是否软件、环境、用户可以运作而不会将项目暴露在高度风险下。该版本也常被称为"beta"版。

构建阶段主要的审核标准包括回答以下的问题:

* 产品是否足够稳定和成熟得发布给用户?
* 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?

如果无法通过这些里程碑,则移交不得不被延迟。

交付阶段（目的是将软件产品交付给用户群体）

本阶段的目标是确保软件产品可以提交给最终用户。本阶段根据实际需要可以分为几个循环。本阶段的具体目标如下：

* 进行 Beta 测试以期达到最终用户的需要
* 进行 Beta 测试和旧系统的并轨
* 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 Vision 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 Vision 中标准的共识
* 该阶段依照产品的类型,可能从非常简单到极端复杂的范围内变化。例如，现有的桌面产品的新版本可能非常简单，而替代国际机场的流量系统会非常复杂。
* 里程碑：产品发布

在交付阶段的终点是第四个重要的项目里程碑，产品发布里程碑。此时，决定是否目标已达到或开始另一个周期。在许多情况下，里程碑会与下一个周期的初始阶段相重叠。

发布阶段的审核标准主要包括以下两个问题：

* 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？

**开发过程中的静态结构(Static Structure of the Process)**

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 Rational Unified Process：

* 角色(Workers),"谁"
* 活动(Activities),"如何"
* 产物(Artifacts),"某事"
* 工作流（Workflows），"何时"

活动、产物、角色

角色

角色定义了个人或由若干人所组成小组的行为和责任。可以认为角色是项目组中个人戴的"帽子"。单个人可以佩戴多个不同的帽子。这是一个非常重要的区别。因为通常容易将角色认为是个人或小组本身，在 Unified Process 中，角色还定义了如何完成工作。所分派给角色的责任既包括某系列的活动，还包括成为产物的拥有者。

活动

某个角色的活动是可能要求该角色中的个体执行的工作单元。活动具有明确的目的，通常表现为一些产物，如模型、类、计划等。每个活动分派给特定的角色。活动通常占用几个小时至几天，常常牵涉一个角色，影响到一个或少量的产物。活动应可以用来作为计划和进展的组成元素；如果活动太小，它将被忽略，而如果太大，则进展不得不表现为活动的组成部分。

活动的例子：

* 计划一个迭代过程，对应角色：项目经理
* 寻找 use cases 和 actors, 对应角色:系统分析员
* 审核设计,对应角色:设计审核人员
* 执行性能测试,对应角色:性能测试人员

产物

产物是被产生的、修改，或为过程所使用的一段信息。产物是项目的实际产品、项目产生的事物，或者供向最终产品迈进时使用。产物用作角色执行某个活动的输入，同时也是该活动的输出。在面向对象的设计术语中，如活动是活动对象（角色）上的操作一样，产物是这些活动的参数。

产物可以具有不同的形式：

* + 模型，如 Use-Case 模型或设计模型
  + 模型组成元素,即模型中的元素，比如类,用例（use case） 或子系统般的元素
  + 文档,如商业案例或软件结构文档
  + 源代码

可执行文件

**工作流**

仅依靠角色、活动和产物的列举并不能组成一个过程。需要一种方法来描述能产生若干有价值的有意义结果的活动序列，显示角色之间的交互作用。

工作流是产生具有可观察结果的活动序列。

UML 术语中,工作流可以表达为序列图、协同图，或活动图。在本文中，使用活动图的形式来描述。

9个核心的过程工作流

核心工作流分为6个核心"工程"工作流

**商业建模工作流** ：许多项目可能不进行商业建模。

**需求工作流** ：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。为了达到该目标，进行提取、组织、文档化需要的功能和约束；跟踪、为折衷方案及决定形成文档。

**分析和设计工作流** ：每一个用例被仔细地描述。用例描述显示了系统如何与 actor 交互及系统的行为.非功能性的需求在补充说明中体现。

**实现工作流** ：

定义代码的组织结构--以层次化的实施子系统的形式

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

对由单个实现者（或小组）产生的结构集成为可执行的系统

**测试工作流** ：

测试的目的是：

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

识别并确保载软件发布之前缺陷被处理

**分发工作流** ：

发布工作流的目标是成功地生成版本，将软件分发给最终用户。它包括了范围广泛的活动。

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 Beta 测试版

移植现有的软件或数据

正式验收

3个核心"支持"工作流

**项目管理工作流** ：

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

它并不是成功的灵丹妙药，但提供了管理项目能显著提高软件成功发布的方法。

**配置和变更控制工作流** ：

本工作流中，描绘了如何在多个成员组成的项目中控制大量的产出物。控制有助于避免混乱，确保不会由以下的问题而造成产品的冲突。

同步更新--当两个或两个以上的角色各自工作在同一产物上时，最后一个修改者会破坏前者的工作。

通知不达--当被若干开发者共享的产品中的问题被解决时，修改未被通知到一些开发者

多个版本--许多大型项目以演化的方式开发。一个版本可能供顾客使用，另一个版本用于测试，而第三个版本处于开发阶段。如果问题在其中任何一个版本中被发现，则修改需要在所有版本中繁衍，从而可能产生混乱导致昂贵的修改和重复劳动，除非变更被很好地控制和监控。

**环境工作流：**

提供了定制流程所必须的准则、模板、工具的开发工具箱。开发工具箱在本文后续的"定制流程的开发工具箱"中更详尽的讨论。

环境工作流没有被牵涉到如选择、获取、使其运行和维护开发环境等的具体方面。

**Rational Unified Process －具体产品  
Rational Unified Process 产品包括：**

* 能使用 web 搜索的知识基础--为全部团队成员就所有关键的开发活动提供准则、模板、工具指导。知识基础进一步划分为：
  + 扩展的准则供全部成员对软件生命期所有组成部分进行参考。提供了既为高层次的思考过程，也为日常沉闷的活动进行指导的指南。该指南发布为 HTML 格式以利于独立于平台的桌面访问。
  + 工具指导涵盖整个生命周期工具的指引。同样，它以 HTML 格式发布.详细情况参见"工具集成"。
  + Rational Rose 的例子和模板为在遵循 Rational Unified Process 时如何组织**Rational Rose 的信息提供参考。**
  + SoDA 模板提供10个以上 SoDA 模板以协助软件文档自动化。
  + Microsoft Word 模板提供了超过30个模板以帮助工作流和生命期所有部分文档化。
  + Microsoft Project Plans 许多项目经理发现很难做出反映迭代开发方法的项目计划。该模板根据 Rational Unified Process 为该方法提供一个好的开端。
  + Development kit 介绍了如何定制和扩展 Rational Unified Process 至采用该方法机构或项目的特定需求,同时提供了工具和模板来辅助该工作.开发工具包在本节的后续部分阐述。

**支持 Rational Unified Process 的工具清单。**

Rational Unified Process 对于大多数产品均提供了工具指引（Tool Mentors）。工具指引是详细介绍如何操作工具以实现开发过程的指南（即弹出什么样的窗口，对话框中的信息及如何漫游的工具）。工具指引允许将独立于工具的过程链接至日常工作的实际操作工具。

* Rational Requisite Pro --通过使需求更易于书写，交流和修改使在整个应用开发中全体开发小组能实时更新和跟踪。
* Rational ClearQuest -- 基于窗口的和 Web 的需求变更管理产品,时项目小组能跟踪和管理开发生命期中的所有变更活动。
* Rational Rose98 -- 世界领先的用于商业过程建模，需求分析，构建结构设计的可视化建模工具。
* Rational SoDA -- 为整个软件开发过程提供产品文档自动化的工具，极大减少了文档工作的时间和成本。
* Rational Purify -- C/C++构件和应用程序开发者使用的运行错误检查工具,帮助检查内存错误。
* Rational Visual Quantify -- C/C++、VB、Java构件和应用程序开发者使用的高级性能评测工具,帮助评估性能瓶颈。
* Rational Visual PureCoverage -- 自动的软件测试覆盖率工具，使开发者能全面地有效地测试他们的应用程序。
* Rational TeamTest -- 创建、维护和执行自动化的功能测试，允许全面地测试代码和决定软件是否满足期望的需求和性能。
* Rational PerformanceStudio -- 评测和预计 Client/Server 和 Web 系统性能的易于使用、准确和可升级的工具。
* Rational ClearCase?--主导市场的软件配置工具，为项目经理提供跟踪每个软件开发项目进化的能力。