Rational 统一开发过程

软件开发队伍的最佳实践

翻译：*Adams Wang*

Rational Unified Process

Best Practices for Software Development Teams

A Rational Software Corporation White Paper

目录

什么是 Rational Unified Process?

Rational Unified Process 是***软件工程化过程***。它提供了在开发机构中分派任务和责任的纪律化方法。它的目标是在可预见的日程和预算前提下，确保满足最终用户需求的高质量产品。[11，13]

Rational Unified Process 是 Rational 公司开发和维护的***过程产品***。*Rational Unified Process* 的开发队伍同顾客、合伙人、Rational 产品小组及顾问公司共同协作，确保开发过程持续地更新和提高以反映新的经验和不断演化的实践经历。

Rational Unified Process 提高了***团队生产力***。对于所有的关键开发活动，它为每个团队成员提供了能使用准则、模板、工具指导来进行访问的知识基础。而通过对相同知识基础的理解，无论你是进行需求分析、设计、测试、项目管理或配置管理，均能确保全体成员共享相同的知识、过程和开发软件的视图。

Rational Unified Process 的活动创建和维护***模型***。*Unified Process* 强调开发和维护*模型*——语义丰富的软件系统表达，而非强调大量的文本工作。[3，7，8]

Rational Unified Process 是有效使用 **Unified Modeling Language** (UML)的指南。UML 是良好沟通需求、体系结构和设计的工业标准语言。UML 由 Rational 软件公司创建，现在由标准化对象管理机构（OMG）维护。[4]

Rational Unified Process 能对大部分开发过程提供自动化的***工具***支持。它们被用来创建和维护软件开发过程（可视化建模、编程、测试等）的各种各样的产物——特别是模型。另外在每个迭代过程变更管理和配置管理相关的文档簿记工作支持方面也是非常有价值的。

Rational Unified Process 是***可配置的过程***。没有一个开发过程能适合所有的软件开发。*Unified Process* 既适用小的开发团队，也适合大型开发机构。*Unified Process* 建立简洁和清晰的过程结构为开发过程家族提供通用性。并且，它可以变更以容纳不同的情况。它还包含了开发工具包，为配置适应特定组织机构的开发过程提供了支持。

Rational Unified Process 以适合于大范围项目和机构的方式捕捉了许多现代软件开发过程的***最佳实践***。部署这些最佳实践经验——使用 *Rational Unified Process* 作为指南——给开发队伍提供了大量的关键优势。在下节中，我们对 *Rational Unified Process* 的 6 个基本最佳实践经验进行描述。

6 个最佳实践的有效部署

Rational Unified Process 描述了如何为软件开发队伍有效的部署经过商业化验证的软件开发方法。它们被称为“最佳实践”不仅仅因为你可以精确地量化它们的价值，而且它们被许多成功的机构普遍的运用。为使整个团队有效利用最佳实践，*Rational Unified Process* 为每个团队成员提供了必要准则、模板和工具指导:

1. 迭代的开发软件
2. 需求管理
3. 使用基于构件的体系结构
4. 可视化软件建模
5. 验证软件质量
6. 控制软件变更

迭代的开发产品——面对当今的复杂的软件系统，使用连续的开发方法: 如首先定义整个问题，设计完整的解决方案，编制软件并最终测试产品，是不可能的。需要一种能够通过一系列细化，若干个渐进的反复过程而生成有效解决方案的迭代方法。***Rational Unified Process*** 支持专注于处理生命周期中每个阶段中最高风险的迭代开发方法，极大地减少了项目的风险性。迭代方法通过可验证的方法来帮助减少风险——经常性的、可执行版本使最终用户不断的介入和反馈。因为每个迭代过程以可执行版本告终，开发队伍停留在产生结果上，频繁的状态检查帮助确保项目能按时进行。迭代化方法同样使得需求、特色、日程上战略性的变化更为容易。[1，2，10]

需求管理——*Rational Unified Process* 描述了如何提取、组织和文档化需要的功能和限制; 跟踪和文档化折衷方案和决策; 捕获和进行商业需求交流。过程中用例和场景的使用被证明是捕获功能性需求的卓越方法，并确保由它们来驱动设计、实现和软件的测试，使最终系统更能满足最终用户的需要。它们给开发和发布系统提供了连续的和可跟踪的线索。[7]

基于构件的体系结构——该过程在全力以赴开发之前，关注于早期的开发和健壮可执行体系结构的基线。它描述了如何设计灵活的，可容纳修改的，直观便于理解的，并且促进有效软件重用的弹性结构。*Rational Unified Process* 支持基于构件的软件开发。构件是实现清晰功能的模块、子系统。*Rational Unified Process* 提供了使用新的及现有构件定义体系结构的系统化方法。它们被组装为良好定义的结构，或是特殊的、底层结构如 Internet、CORBA 和 COM 等的工业级重用构件。[5]

可视化软件建模——开发过程显示了对软件如何可视化建模，捕获体系结构和构件的构架和行为。这允许你隐藏细节和使用“图形构件块”来书写代码。可视化抽象帮助你沟通软件的不同方面，观察各元素如何配合在一起，确保构件模块一致于代码，保持设计和实现的一致性，促进明确的沟通。Rational 软件公司创建的工业级标准 Unified Modeling Language（UML）是成功可视化软件建模的基础。[4，12]

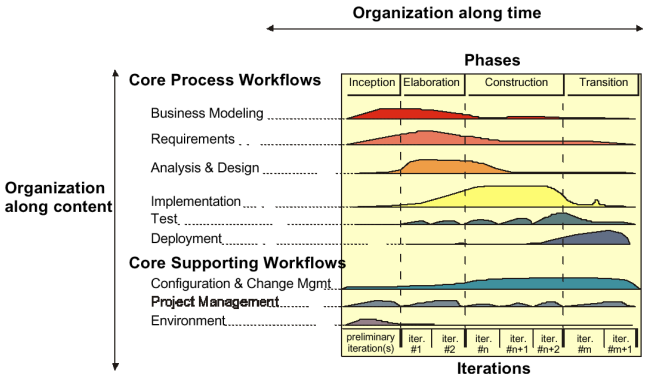
验证软件质量——拙劣的应用程序性能和可靠性是戏剧性展示当今软件可接受性的特点。从而，质量应该基于可靠性、功能性、应用和系统性能根据需求来进行验证。*Rational Unified Process* 帮助计划、设计、实现、执行和评估这些测试类型。质量评估被内建于过程、所有的活动，包括全体成员，使用客观的度量和标准，并且不是事后型的或单独小组进行的分离活动。

控制软件的变更——管理变更的能力——确定每个修改是可接受的，能被跟踪的——在变更不可避免环境中是必须的。开发过程描述了如何控制、跟踪和监控修改以确保成功的迭代开发。它同时指导如何通过隔离修改和控制整个软件产物（例如，模型、代码、文档等）的修改来为每个开发者建立安全的工作区。另外，它通过描述如何进行自动化集成和建立管理使小队如同单个单元来工作。

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。



The Iterative Model graph shows how the process is structured along two dimensions.

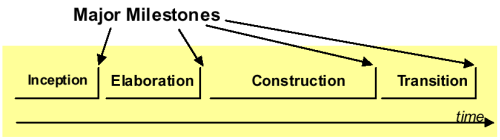
阶段和迭代——时间轴

这是开发过程沿时间的动态组织结构。

软件生命周期被分解为*周期*，每一个周期工作在产品新的一代上。*Rational Unified Process* 将周期又划分为四个连续的*阶段*[10]

- 初始阶段
- 细化阶段
- 构造阶段
- 交付阶段

每个阶段终结于良好定义的*里程碑*——某些关键决策必须做出的时间点，因此关键的目标必须被达到。



The phases and major milestones in the process.

每个阶段均有明确的目标。

初始阶段

Inception | Elaboration | Construction | Transition

初始阶段的目标是为系统建立商业案例和确定项目的边界。

为了达到该目的必须识别所有与系统交互的外部实体，在较高层次上定义交互的特性。它包括识别所有用例和描述一些重要的用例。商业案例包括验收规范、风险评估、所需资源估计、体现主要里程碑日期的阶段计划。[10，14]

本阶段具有非常重要的意义，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:

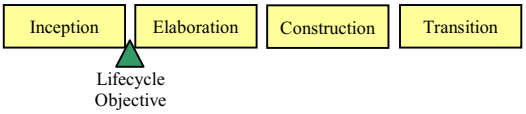
- 明确软件系统的范围和边界条件，包括从功能角度的前景分析、产品验收标准和哪些做与哪些不做的相关决定
- 明确区分系统的关键 Use-case 和主要的功能场景
- 展现或者演示至少一种符合主要场景要求的候选软件体系结构
- 对整个项目做最初的项目成本和日程估计（更详细的估计将在随后的细化阶段中做出）

- 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：生命周期的目标里程碑



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。

阶段评审的文档和需要达到的状态列表：

产品（按照重要性排序）	应达到的状态
Vision(SA)	核心需求，关键特色和主要约束都已形成文档
Business Case(PM)	建立并已确认
Risk List & Risk Management Plan (PM)	早期的项目风险已经标识出来
Business Rules(BPA)	核心业务规则已经归档
Stakeholder's Request & Target Organization Accessment(BPA)	完成
Business Architecture Document & Supplementary Business Specification（BPA）	完成
Software Development Plan(PM)	标识出早期的阶段、持续时间和相关目标；SDP 中的资源估计(主要是对时间、项目组成员、和开发环境等成本的估计)应该和 Business Case 保持一致。资源估计可以包括直到产品交付的整个项目，也可以只估计到细化阶段就可以了。对于完整的项目估计来说，可能是非常粗略的。这些估计将在随后的每个阶段和循环过程中不断细化、精确和修正。根据项目要求的不同，SDP 中可以包

	含一个或多个专门的计划文档，此外，SDP 中的工作指导类文档至少应该有草案的形式。
Iteration Plan(PM)	为细化阶段的第一个循环准备的循环计划已经完成并通过评审
Product Acceptance Plan(PM)	已经评审并建立基线，该计划在随后的循环过程中如果出现了附加的需求将加以细化
Development Case(PE)	基于 Rational Unified Process 建立，可以修改或扩展，已形成文档并通过评审
Project-Specific Templates(PE)	整个项目的相关模版已准备好
Use-Case Modeling Guidelines(SA)	已纳入基线
Tools(TS)	所有支持项目的工具已经选好，下一步循环工作必需的工具安装完毕
Glossary(SA)	主要的术语已经定义并经过评审
Use-Case Model (Actors, Use Cases)(SA,UID,UCS)	重要的 Actors 和 Use-case 已经标识出来，最为关键的 Use-case 的事件流框架形成。
User Interface Prototypes (UID)	主要界面原型
Business Use-case Model 和 Business Object Model (Organization Unit,Business Use-case Realization,Business Entity,Business Worker)(BPA & BD)	系统中用到的关键概念已经形成文档并经过评审。
Prototypes（可选）	针对 Vision 或 Business Case 或者某个特定风险准备好一个或多个概念原型
Requirement Management Plan & Requirement Attribute Repository(SA)	RMP 完成，准备好需求属性库

补充说明

1. 角色说明:
   - SA(System Analyst),UID(UI Designer),UCS(Use-case Specifier)
   - PM(Project Manager)
   - BPA(Business Process Analyst) & BD(Business Designer)
   - PE(Process Engineer)
   - TS(Tool Specialist)

2. 需求类主要文档
   - Vision
   - Use-case Model(包括和核心 use-case 实现相关的 Object Model)
   - Core Use-case,UI Prototype
   - Business Use-case Model(含 Business Object Model,包括 Worker,Organization Unit,Entity 等)
   - Stakeholder's Request & Target Organization Accessment
   - Business Architecture Document & Supplementary Business Specification
   - Business Rules

- Use-case Model Guideline 的 Draft
- Glossary
- Requirement Management Plan & Requirements Attribute Repository

3. PM 类主要文档

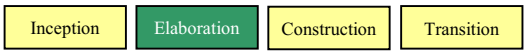
- Business Case
- Risk List & Risk Management Plan
- SDP
- Interation Plan & Interation Accessment
- Product Acceptance Plan
- Problem Resolution Plan, Measure Plan, State Assessment, Quality Assurance Plan
- Work Order
- Review Record(Project Reviewer)

4.PE 类主要文档

- Project-specific Templates
- Development Case

6.配置和变更管理：建立配置工作区和管理库，准备好变更请求库。

细化阶段



细化阶段的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。

为了达到该目的，必须对系统具有“英里宽和英寸深”的观察。体系结构的决策必须在理解整个系统的基础上作出：它的范围，主要功能和如性能等非功能性需求。

容易引起争论，细化阶段是四个阶段中最关键的阶段。该阶段结束时，硬“工程”可以认为已结束，项目则经历最后的审判日：决策是否项目提交给构建和交付阶段。对于大多数项目，这也相当于从移动的、轻松的、灵巧的、低风险的运作过渡到高成本、高风险并带有较大惯性的运作过程。而过程必须能容纳变化，细化阶段活动确保了结构、需求和计划是足够稳定的，风险被充分减轻，所以可以为开发结果预先决定成本和日程安排。概念上，其逼真程度一致于机构实行费用固定的构建阶段的必要程度。

在细化阶段，可执行的结构原形在一个或多个迭代过程中建立，依赖于项目的范围、规模、风险和先进程度。工作量必须至少处理初始阶段中识别的关键用例，关键用例典型揭示了项目主要技术的风险。通常我们的目标是一个由产品质量级别构件组成的可进化的原型，但这并不排除开发一个或多个探索性、可抛弃的原型来减少如：设计/需求折衷，构件可行性研究，或者给投资者、顾客即最终用户演示版本等特定的风险。

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

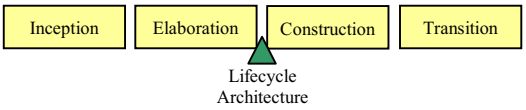
的成本和日程的程度。

- 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。

初始阶段的产出是:

- 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：生命周期的结构里程碑



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。

阶段评审的文档和需要达到的状态列表:

产品（按照重要性排序）	应达到的状态
Prototypes	针对系统的关键功能和主要结构场景建立的一个或多个可执行的结构化原型。
Risk List & Risk Management Plan（PM）	更新并经过评审。
Development Case(PE)	根据早期的项目管理经验加以细化。为构建小组提供支持的开发环境、过程、工具和各种自动化工具已经到位。

Project-Specific Templates(PE)	文档模版已经用于建立软件文档产品
Tools(TS)	用于细化阶段的工具已经安装
Software Architecture Document 和 Reference Architecture(AR)	已经创建并纳入基线，其中包括详细描述对软件结构构成重大影响的 use-case（通过 use-case 图）,对关键的实现机制和设计元素的标识，外加对流程和部署的定义和描述。
Analyse_Design Model (包括所有的子产品，包括对象模型、设计类、子系统和包、接口、事件，静态结构图，动态序列图、活动图，状态转移图等主要内容)（AR,D）	已经定义并纳入基线，对于和软件结构构成重大影响的 use-case 的实现已经定义，相关的行为已经分配到适当的设计元素中。 组件已经标识出来，创建/购买/复用决定做出，选好的结构化组件针对主要场景已经加以集成和评估。从这些活动中获得的教训可能会导致重新设计软件结构，重新考虑替代的设计方案和对需求的重新思考。
Data Model（DD）	定义并纳入基线，主要的数据库元素已经设计出来并通过评审
Implementation Model (and all constituent artifacts, including Components)(AR,IMP)	早期结构已经创建，主要组件已经确认并原型化
Vision(SA)	基于本阶段获得的新的信息加以细化。对形成软件结构和计划决策起主要作用的 use-cases 建立一个稳固的理解
Software Development Plan（PM）	更新并扩展到构建和移交阶段。
Guidelines, such as Design Guidelines and Programming Guidelines.	工作知道已经用于指导相关工作。
Iteration Plan & Iteration Assessment（PM）	针对构建阶段的循环计划已经完成并经过评审。
Use-Case Model (Actors, Use Cases)(SA,UCS)	use-case 模型基本完成- 所有的 use-case 已经标识出来，所有的 Actors 也已标识出来，大多数 use-case（从需求分析中捕获的）的说明都已提供。
Software Requirement Specification (UCS)	纳入基线
User Interface Prototype(UID)	完成
Supplementary Specifications（SA）	捕获非功能需求的辅助规范已经形成文档并经过评审。
Business Case(PM)	如果结构化工作中发现了新的导致变更基本项目假定的问题，本文档将加以更新。
User Manual and other Training Material（TW,CB）	基于 use-case 形成早期版本。

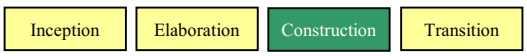
补充说明

1. 角色说明:
   - AR(Architect),D(Designer),DD(Database Designer),IMP(Implementor)
   - TW(Technical Writer), CB(Cource Builder)
2. 需求类文档

- Vision
- Use-case Model(包括 Actors,Use-cases,Use-case Realization)
- SRS & Supplementary Specification
- User Interface Prototype

3. PM 类文档
   - Risk List
   - SDP
   - Iteration Plan
   - Business Case
4. 分析和设计类文档
   - Software Architecture Document & Reference Architecture
   - Analysis_Design Model(Include Object Model, Design Class, Package, SubSystem, Interface, Event and Diagrams)
   - Implementation Model(Include SubSystem,Component)
   - Data Model
   - Prototypes
5. 细化阶段需求建模结束、概要和详细设计工作也基本结束，一部分和结构相关的组件已经开始编码。
6. 本阶段可以分为两个子阶段，第一个子阶段是以概要设计结束（完成软件体系结构设计）和需求建模基本结束为标志，此阶段主要参与人员是SA(& Use-case Specifier)和 Architect. 可以称为概要设计阶段。此后是详细设计阶段，在详细设计阶段，需求建模只剩下 Refine System Defination 和 Manage Change Requirements 两项主要工作。对于分析和设计工作来说，可能会有多个设计人员（包括 Designer,Database Designer,甚至部分 Implementor）参与进来，这时,Architect 主要对软件的整体体系结构负责。
7. 需求评审可以在概要设计结束和详细设计开始时进行。

构建阶段



在构建阶段，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。

构建阶段，从某种意义上说，是重点在管理资源和控制运作以优化成本、日程、质量的生产过程。就这一点而言，管理的理念经历了初始阶段和细化阶段的智力资产开发到构建阶段和交付阶段可发布产品的过渡。

许多项目规模大的足够产生许多平行的增量构建过程，这些平行的活动可以极大地加速版本发布的有效性；同时也增加了资源管理和工作流同步的复杂性。健壮的体系结构和易于理解的计划是高度关联的。换言之，体系结构上关键的质量是构建的容易程度。这也是在细化阶段平衡的体系结构和计划被强调的原因。

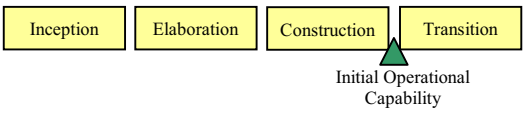
本阶段的主要目标如下:

- 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：初始功能里程碑



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。

本阶段里程碑处的产品和相关状态如下:

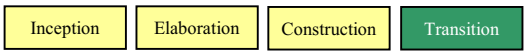
产品 (按重要性排序)	达到里程碑时的状态
"The System"(IN)	准备好进入 Beta 测试
Deployment Plan(DM)	开发、评估并已纳入基线
Implementation Model (and all constituent artifacts, including Components)(AR,IMP)	从细化阶段的模型扩展到所有组件，完成
Test Plan, Test Procedure, Test Use-case, Test Results，Workload Analysis Document(TD,T)	完成
Training Materials(TW,CB)（可选）	手册和培训资料基本完成
Iteration Plan（PM）	为移交阶段制订的循环计划已经完成并经过评审
Design Model (and all constituent artifacts)(D)	更新并完成
Development Case(PE)	基于早期项目经验完成，为支持产品移交小组的环境、工具和自动化工具已经到位。
Project-Specific Templates(PE)	已经用于建立软件产品文档

Tools(TS)	支持产品构建的工具已经安装
Data Model(DD)	更新，包括所有支持持久存储的元素 (e.g. tables, indexes, object-to-relational mappings, etc.)
Supplementary Specifications(SA)	如果有新的需求出现将加以更新。
Use-Case Model (Actors, Use Cases)(SA,UCS)	如果实现阶段发现新的需求将加以更新

补充说明

1. 角色说明
   - IN(Integrator)
   - DM(Deploy Manager)
   - TD(Test Designer), T(Tester)
2. 测试类文档
   - Test Plan(包括测试计划，集成测试计划)
   - Test Procedure
   - Test Use-case
   - Test Results
   - Test Review Report

交付阶段



交付阶段的目的是将软件产品交付给用户群体。

只要产品发布给最终用户，问题常常就会出现：要求开发新版本，纠正问题或完成被延迟的问题。

当基线成熟得足够发布到最终用户时，就进入了交付阶段。其典型要求一些可用的系统子集被开发到可接收的质量级别及用户文档可供使用，从而交付给用户的所有部分均可以有正面的效果。这包括：

- 对照用户期望值，验证新系统的“beta 测试”
- 与被替代的已有系统并轨
- 功能性数据库的转换
- 向市场、部署、销售队伍移交产品

构建阶段关注于向用户提交产品的活动。典型的，该阶段包括若干重复过程，包括 beta 版本、通用版本、bug 修补版和增强版。相当大的工作量消耗在开发面向用户的文档，培训用户。在初始产品使用时，支持用户并处理用户的反馈。开发生命周期的该点，用户反馈主要限定在产品性能调整、配置、安装和使用问题。

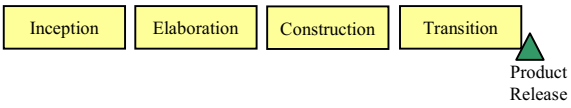
本阶段的目标是确保软件产品可以提交给最终用户。本阶段根据实际需要可以分为几个循

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?

达成本阶段里程碑的主要产品和相关状态如下:

产品 (按重要性排序)	达到里程碑时的状态
The Product Build	完成并满足需求
Release Notes	完成
Installation Artifacts	完成
Training Material	完成，并确保用户对产品的使用和自身维护工作已经满意。
End-User Support Material	完成，并确保用户对产品的使用和自身维护工作已经满意。

迭代过程

Rational Unified Process 的每个阶段可以进一步被分解为迭代过程。***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。[10]

迭代方法的益处

与传统的瀑布式方法相比，迭代过程具有以下的优点。

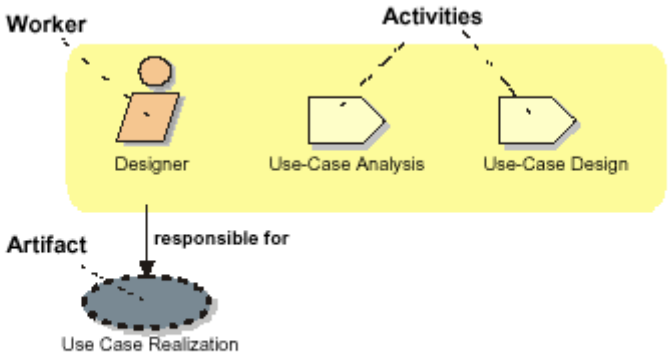
- 减小了风险
- 更容易对变更进行控制
- 高度的重用性
- 项目小组可以在开发中学习
- 较佳的总体质量

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”

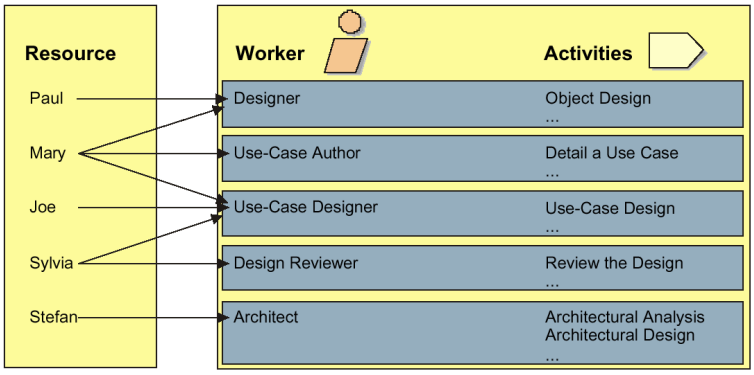
活动、产物、角色



Workers, activities and artifacts.

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。



People and Workers.

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:

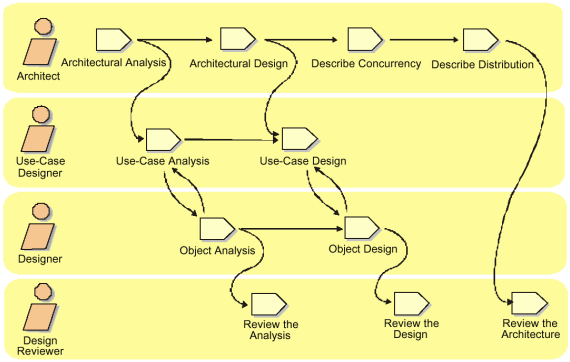
- 模型，如 Use-Case 模型或设计模型
- 模型组成元素，即模型中的如类，use case 或子系统般的元素
- 文档，如商业案例或软件结构文档
- 源代码
- 可执行文件

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。



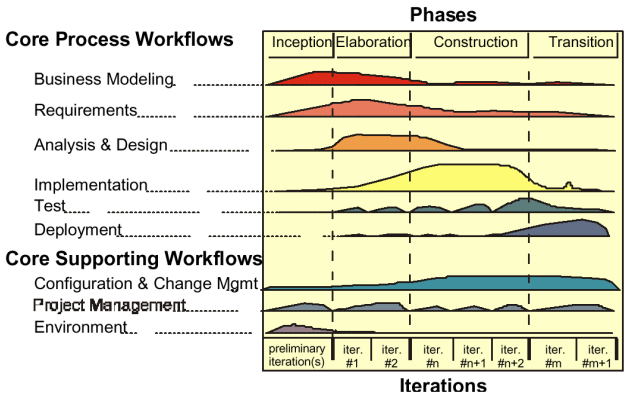
Example of workflow.

注意要表达活动之间的所有依赖关系并不是总可能或切合实际的。常常两个活动较表现的交织得更紧密，特别是在涉及到同一个角色或个人时。人不是机器，对于人而言，工作流不能按字面地翻译成程序，被精确地、机械地执行。

下节中，我们将讨论开发过程中最基本的工作流，称之为***核心工作流***。

核心工作流（Core workflows）

Rational Unified Process 中有 9 个***核心工作流***，代表了所有角色和活动的逻辑分组情况。



The nine core process workflows.

核心工作流分为 6 个核心“工程”工作流

1. 商业建模工作流
2. 需求工作流
3. 分析和设计工作流
4. 实现工作流
5. 测试工作流
6. 分发工作流

和 3 个核心“支持”工作流

1. 项目管理工作流
2. 配置和变更控制工作流
3. 环境工作流

尽管 6 个核心工程工作流能使人想起传统瀑布流程中的几个阶段，但应注意迭代过程中的阶段是不同的，这些工作流在整个生命期中一次又一次被访问。9 个核心工作流在项目中的实际完整的工作流中轮流被使用，在每一次迭代中以不同的重点和强度重复。

商业建模

决大多数商业工程化的主要问题，是软件工程人员和商业工程人员之间不能正确地交流。这导致了商业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。*Rational Unified Process* 针对该情况为两个群体提供了相同的语言和过程，同时显示了如何在商业和软件模型中创建和保持直接的可跟踪性。

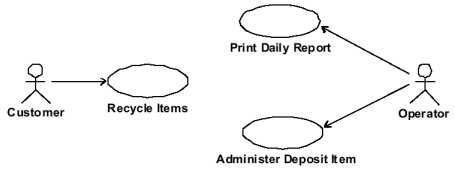
在商业建模中，使用商业用例来文档化商业过程，从而确保了组织中所有支持商业过程人员达到共识。商业用例被分析以理解商业过程如何被业务支持，而这些在商业对象模型中被核实。

许多项目可能不进行商业建模。

需求

需求工作流的目标是描述系统应做“*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。

蓝图被创建，需求被提取。代表用户和其他可能与开发系统交互的其它系统的***Actor***被指明。***Use case***被识别，表示系统的行为。因为 *use case* 根据 *actor* 的要求开发，系统与用户之间的联系更紧密。系统展示了用于再生系统的用例模型。



An example of a use-case model with actors and use cases.

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。

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。

分析和设计

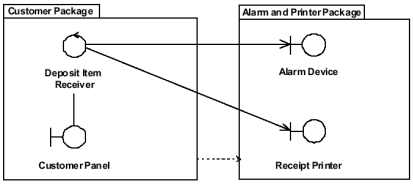
分析设计工作流的目标是显示系统*“如何”*在实现阶段被*“实现”*的。你需要一个如下系统：

- 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
- 健壮的被建造（如果功能需求发生变化，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*。设计模型是源代码的抽象；即设计模型充当源代码如何被组建和编制的“蓝图”。

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，而描述则体现了类的对象如何协同工作实现用例的功能。

下图是上例 use-case 模型的设计模型示例的一个部分。



Part of a design model with communicating design classes, and package group design classes.

设计活动以体系结构设计为中心。结构的产物和验证是早期迭代的主要焦点。结构由若干结构视图来表达，这些视图捕获了主要的构架设计的决定。本质上，结构视图是整个设计的抽象和简化，该视图中细节被放在了一旁，使重要的特点体现得更加清晰。结构不仅仅是良好设计模型的承载媒介，而且在系统的开发中能提高任何被创建模型的质量。

实现

实现阶段的目的:

- 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

系统通过完成构件而实现。*Rational Unified Process* 描绘了如何重用现有的组件，或实现经过良好责任定义的新构件，使系统更易于使用，提高了系统的可重用性。

构件被构造成实施子系统。子系统被表现为带有附加结构或管理信息的目录形式。例如，子系统可以 被创建为文件系统中的文件夹或目录，或 *Rational Apex for C++ or Ada*，或 *Java* 中的包。

测试

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

Rational Unified Process 提出了迭代的方法，意味着在整个项目中进行测试，从而允许尽可能早的发现缺陷，从根本上降低了修改缺陷的成本。测试类似于三维模型，分别从可靠性、功能性、应用和系统性能来进行。流程从每个维度描述了如何经历测试生命周期的几个阶段，计划、设计、实现、执行和审核。

另外，描述了何时及如何引入测试自动化的策略。使用迭代的方法，测试自动化是非常重要的，它允许在每次迭代结束及为每个新产品进行回归测试。

发布

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

尽管发布工作流主要被集中在交付阶段，但早期阶段需要加入为创建阶段后期的发布做准备的许多活动。

Rational Unified Process 中的发布和环境工作较其它工作流包含了较少的内容。

项目管理

软件项目管理是一门艺术，它平衡了互相冲突的目标，管理风险，克服各种限制来成功地发布满足投资用户和使用者需要地软件。如此少的无争议的成功项目无疑是该项任务难度的证明。

工作流主要集中在迭代开发过程的特殊方面。本节我们的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。[14]

配置和变更管理

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。

工作流提供了准则管理演化系统中地多个变体，跟踪给定软件创建过程中的版本。根据用户定义地版本规则建造程序或整个 版本，实施特定于现场的开发策略。

工作流描述了了如何管理并行开发、分布式开发，如何自动化创建工程。这在如每天均需要频繁编译链接的重复过程中尤为重要。如果没有有力的自动化是不可能的，同时也阐述了对产品修改原因、时间、人员保持审计记录。

工作流也涵盖了变更需求管理，即如何报告缺陷，在它们的是生命周期中如何管理，及如何使用缺陷来跟踪进展和发展的倾向。

环境

环境工作流的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。

工作流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。

环境工作流还包含了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 产品包括:

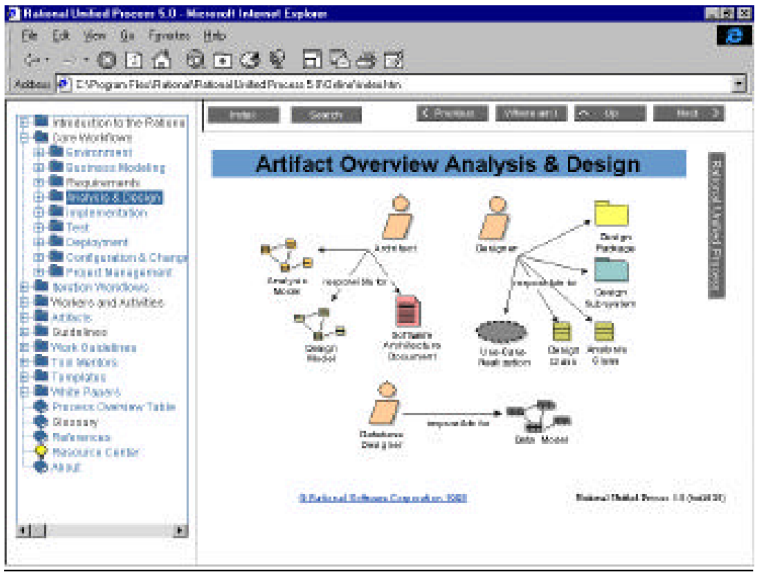
- **能使用 web 搜索的知识基础——**为全部团队成员就所有关键的开发活动提供准则、模板、工具指导。知识基础进一步划分为:
  - **扩展的准则**供全部成员对软件生命期所有组成部分进行参考。提供了既为高层次的思考过程，也为日常沉闷的活动进行指导的指南。该指南发布为 HTML 格式以利于独立于平台的桌面访问。
  - **工具指导**涵盖整个生命周期工具的指引。同样，它以 HTML 格式发布。详细情况参见“工具集成”。
  - **Rational Rose 的例子和模板**为在遵循 *Rational Unified Process* 时如何组织 Rational Rose 的信息提供参考。
  - **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。
- Addison-Wesley 出版，Philippe Kruchten 著的《**Rational Unified Process A Introduction**》。本书共 277 页，对开发过程和知识基础提供了很好的介绍和概括。

知识基础的浏览

Rational Unified Process 允许使用任何流行的浏览器如：IE、Netscape Navigator 进行浏览。

使用 *Rational Unified Process*，你仅需少许的鼠标点击即可获得所需的信息。该知识基础包含了许多超文本链接，各种过程元素用交互的图案来表达，很容易直观的寻找相关信息。功能强大的搜索引擎、索引、“资源管理器式外观”的树状浏览使得使用该过程很容易。浏览按钮允许如同读书一般前后翻页。

信息在若干不同的视图中表现，允许你相关于角色、特定活动或工作流来查看信息。对于关键的项目角色，提供易于学习的指导过程。



Interactive images and navigational buttons make it easy to find the specific information you are looking for

定制过程的开发工具包

Rational Unified Process 足够通用及完备，如同它名字所描述的一般。然而，在某些情况下，该软件开发过程需要被修改、调整和剪裁以容纳具体的特性、约束和机构的历史情况。特别的，该过程不应该盲目的被遵循，形成许多无用的工作，产生无用产物。它应尽可能的精炼并能够快速地、可预见地产生高质量的软件。

开发过程包含了**开发工具包**，它包括了指导如何定制过程以满足机构或项目特定需要的指南。工具包还包括了创建过程，以及搜索引擎、索引、场所地图、树型浏览器等的衍生和操作的模板。开发工具包使得能定制组织结构维持 *Rational Unified Process* 的感观。

开发过程定制程度越高，则定制化向未来过程版本的移植越困难。开发工具包描述了减少定制化移植时工作的策略、工具和技术。

工具集成

软件工程化过程需要工具支持系统生命期的所有活动，尤其是支持开发、维护和各种各样产物的簿记——特别是模型。迭代的开发过程对使用的工具集提出了特殊的要求，如工具的集成和模型代码之间的双向工程。同样，还需要支持跟踪需求，文档自动化以及测试自动化如促进回归测试等的工具。*Rational Unified Process* 可以与各种各样的工具一同使用，包括 Rational 公司和其它供应商的产品。而 Rational 提供许多有效支持 *Rational Unified Process* 良好集成的工具。

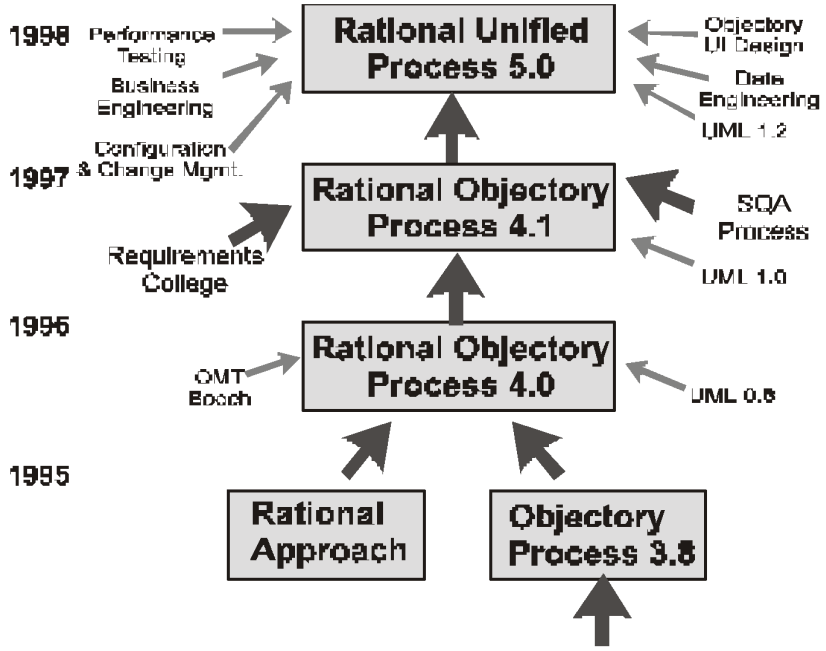
下面提供了支持 *Rational Unified Process* 的工具清单。

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™**——Windonw 和基于 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 Rose® 98**——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。

Rational Unified Process 的历史

Rational Unified Process 经过了许多年的成熟期并反映了许多人和公司的集体经验。让我们简要地看一下如下图所示它的历史:



Genealogy of the Rational Unified Process

从时间上回顾，*Rational Unified Process* 是 *Rational Objectory Process (version 4)*的直接继承者。*Rational Unified Process* 合并了数据工程、商业建模、项目管理和配置管理领域更多的东西，后者作为与 Prue Atria 的归并结果。它更紧密地集成至 Rational 软件工具集

Rational Objectory Process 是 *"Rational Approach"* 与 *Objectory process (version 3)*的综合，在１９９５年 *Rational* 软件公司与 *Objectory AB* 合并之后。从它的 *Objectory* 前任，继承了*过程结构*和 *use case* 的中心概念。从它的 *Rational* 背景，得到了*迭代开发*和*体系结构*的系统阐述。该版本同样包括了 *Requisite, Inc* 配置管理部分和从 *SQA,® Inc* 继承的详细测试过程。最后，该开发过程是第一个使用了统一建模语言(UML 0.8)。

Objectory process 作为 Ivar Jacobson 的 Ericsson 经验于１９８７在瑞典被创建。该过程称为他公司 *Objectory AB* 的产品。由于以 use case 概念和面向对象的方法为中心，它很快得到了软件工业的认可并被许多世界级的公司集成。*Objectory process* 的简单版本作为课本在１９９２年被出版。

Rational Unified Process 是更为通用方法的一个特定的、详细的实例，该通用方法在 Ivar Jacobson, Grady Booch, 和 James Rumbaugh 所著的课本《The Unified Software Development Process》有介绍。

参考资料

1. Barry W. Boehm, *A Spiral Model of Software Development and Enhancemen*t, *Computer,* May 1988, IEEE, pp.61-72
2. Barry W. Boehm, *Anchoring the Software Proces*s, *IEEE Software,* 13, 4, July 1996, pp. 73-82.
3. Grady Booch, *Object Solutions,* Addison-Wesley, 1995.
4. Grady Booch, Ivar Jacobson, and James Rumbaugh, *Unified Modeling Language 1.*3, White paper, Rational Software Corp., 1998.
5. Alan W. Brown (ed.), *Component-Based Software Engineerin*g, IEEE Computer Society, Los Alamitos, CA, 1996, pp.140.
6. Michael T. Devlin, and Walker E. Royce, *Improving Software Economics in the Aerospace and Defense Industry,* Technical paper TP-46, Santa Clara, CA, Rational Software Corp., 1995
7. Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaar*d, Object-Oriented Software Engineering—A Use Case Driven Approach,* Wokingham, England, Addison-Wesley, 1992, 582p.
8. Ivar Jacobson, M. Griss, and P. Jonsson, *Software Reuse—Architecture, Process and Organization for Business Success,* Harlow, England, AWL, 1997.
9. Philippe Kruchten, *The 4+1 View Model of Architectur*e, IEEE Softwar*e,* 12 (6), November 1995, IEEE, pp.42-50.
10. Philippe Kruchten, *A Rational Development* Process, CrossTalk, 9 (7), STSC, Hill AFB, UT, pp.11-16.
11. Ivar Jacobson, Grady Booch, and Jim Rumbaugh, *Unified Software Development Proces*s, Addison-Wesley, 1999.
12. Grady Booch, Jim Rumbaugh, and Ivar Jacobson, *Unified Modeling Language—User's Guid*e, Addison-Wesley, 1999.
13. Philippe Kruchten, *Rational Unified Process—An Introduction,* Addison-Wesley, 1999.
14. Walker Royce, *Software Project Management—A Unified Framewor*k, Addison-Wesley, 1998.