将 Rational Unified Process 用于小型项目：

详述 eXtreme Programming

Gary Pollice

Rational Software 白皮书

TP 183, 3/01

Rational
the software development company

目录

摘要

Rational Unified Process® 或 RUP®产品是完整的软件开发流程框架，它附带了几个现成的实例。从 RUP 派生的流程包括从轻量级流程（满足生产周期短的小型项目的需要）到更复杂的流程（满足大型的，可能是分布式的项目团队的多种需要）。所有类型和规模的项目都已成功使用了 RUP。此本白皮书描述了如何以轻量级方式将 RUP 应用于小型项目。我们将描述如何在一个完整项目的更广泛环境中有效地应用 eXtreme Programming（XP）技术。

简介

一个简短的故事

一天早上，经理过来问我是否可以花几周的时间为公司将启动的新业务设置一个简单的信息系统。我正对手头的项目感到厌烦，渴望着新项目能够使我振奋，因此我欣然接受这个任务 — 我将能够不受所在庞大组织内繁文缛节的羁绊，放手开发很棒的新解决方案。

事情开始时很顺利。最初的六个月，主要是我独自工作 — 时间很长，但过得很愉快。我的生产力是不可思议的，我进行着我职业生涯最好的工作之一。开发周期非常快，我一般是每隔几周生成系统的一些主要新部件。与用户的交互是简单直接的（我们都是关系严密的团队的一份子），所以我们可以免除固定的格式和文档。设计中也几乎没有固定的格式，代码即是设计，设计即是代码。情况好极了！

情况好极了，暂时是这样的。随着系统的发展，有越来越多的事情要做。随着问题的变更，原有的代码必须改进，我们对我们需要做的事情的想法进行了优化。我雇用了几个人来帮助进行开发。我们作为一个单独的单位进行工作，通常成对地解决问题的某个部分。这增进了沟通，且我们可以免除一些固定的手续。

一年过去了。

我们继续增加人员。团队成员增加到三个人，然后是五个人，再到七个人。每次有新成员开始工作，就会经历漫长的学习弯路，由于没有经验优势，理解和解释整个系统（即使是概述）都变得更困难。我们开始使用白板图，它能够更加正式地显示系统的总体结构，主要概念和接口。

我们仍然使用测试作为验证系统是否实现了其应有功能的主要手段。因为许多新的人员在“用户”群中我们发现，在项目早期起作用的非正式需求和人际关系已经不够了。需要花费更多的时间来确定我们应当构建的系统。因此，我们坚持对讨论进行记录，这样就不必总是回顾我们已决定的问题。我们还发现，描述需求和使用场合有助于培训新的用户使用该系统。

随着系统规模和复杂度的不断增长，出现了意外情况 — 系统的体系结构需要特别关注。在早期，体系结构大体上存在于我的头脑中，稍后则随手记录在便笺或挂图上。然而，随着项目人员的增加，体系结构的控制难度也在加大。因为并非每个人都持有与我相同的历史观点，他们无法了解对体系结构所作特定变更的含义。我们必须以更精确的方式定义系统的体系结构约束。任何可能影响体系结构的变更都需要团队的一致同意，并由我最终核准。我们很艰难地发现了这一问题，在获取了一些沉重的教训之后，我们才真正意识到体系结构的重要性。

这是一个真实的故事。它仅仅描述了这个项目中一部分痛苦的经历。这些经历仅在一个方面不同寻常：我们之中有几个人从头至尾干了几年的时间。通常，人们参加并进行某个项目，但并不了解自己的行为产生的后续影响。

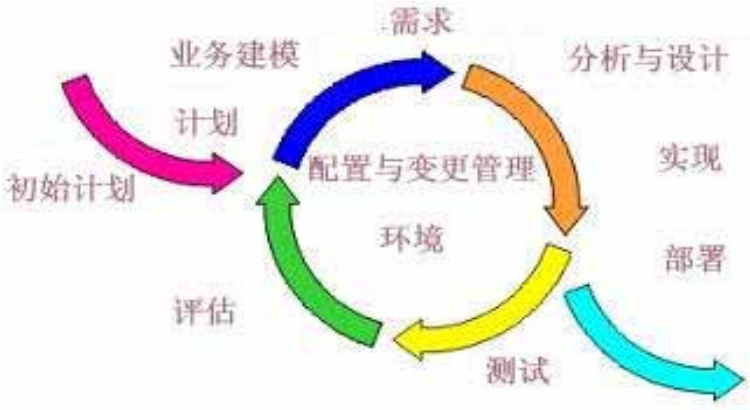
一点点地流程可能会对项目有帮助。太多的流程会阻碍进展，但是缺少流程则会带来新的风险。正如投资高风险股票的人几乎不使用流程，他们只看到高回报，却忽略了项目环境中的主要风险，正所谓“抱着最大的希望，却没有做好最坏的打算”。

概述

本文讨论如何像刚才描述的那样，将流程应用于项目。我们的重点在于获得“级别恰当”的的流程。通过了解开发团队所面临的挑战以及项目运作时所在的业务环境，得出恰当的流程正式级别。一旦我们了解了这些挑战，就能够提供恰好足够的流程以减轻风险。不存在“放之四海而皆准”的流程，轻量级的或许未必如此。在以下部分，我们探讨关于适当级别的流程是风险的函数这一思想。

我们讨论的重点是如何使用 Rational Unified Process（RUP）和 eXtreme Programming（XP）这两种常用的方法论来获取恰当的流程级别。我们将展示如何为小型项目定制 RUP 以及如何将其用于 XP 未顾及的许多领域。这种组合为项目团队减轻风险和实现软件产品交付目标提供了所需的指导信息。

RUP 是由 Rational Software 开发的一个流程框架。它是一种迭代开发方法，基于六种经行业验证的最佳实践（请参阅 RUP 附录）。按照时间顺序，基于 RUP 的项目经历四个阶段：先启阶段、精化阶段、构造阶段和移交阶段。每个阶段包含一次或多次迭代。在每次迭代中，将在每个规程（或工作流，例如需求、分析与设计，以及测试等）上花费不同的工时量。RUP 的主要驱动因素是减轻风险。RUP 已经在数以千计的 Rational 客户与合作伙伴的数千个项目中使用，从而得到了优化。下图说明通过一个典型迭代中的流：



典型迭代流

作为风险塑造流程的示例，我们可能会问是否应该对业务建模。如果未能了解某个重大的风险，业务可能使我们构建错误的系统，因此我们可能应该执行一定量的业务建模。对于建模工作，我们是否需要以正式方式进行。这取决于我们的受众 — 如果是小团队非正式地使用结果，则我们可以只做一些非正式的注释。如果组织中的其他人想要使用或复审结果，我们可能必须投入额外的工时，并且在演示资料的正确性和可理解性方面投入更多的精力。

您可以定制 RUP 以适合几乎任何项目的需要。如果没有现成的流程或路线图适合您的特定需求，您可以轻松地生成自己的路线图。路线图描述了项目如何规划使用流程，因而代表该项目的一个特定流程实例。它所要表达的意思是：RUP 根据需要可以是轻量级或重量级，这也是本书所要阐明的。

XP 是用于小型项目的轻量级、以代码为中心的流程（请参阅 XP 附录）。它由 Kent Beck 构想而出，并在大约 1997 年由于 Chrysler Corporation 的 C3 薪水册项目而引起软件行业的关注。与 RUP 一样，它也是基于迭代的，迭代中包含若干个实践，比如小规模发行版 (Small Releases)、简单设计 (Simple Design)、测试 (Testing) 和持续集成 (Continuous Integration)。XP 促进了对相应的项目或环境有效的几项技术；然而还有隐藏的假设、活动和角色。

RUP 和 XP 来自不同的理论。 RUP 是由流程组件、方法和技术构成的框架，可以将其应用与任何特定的软件项目；我们希望用户专门研究 RUP。另一方面，XP 是约束较多的流程，

需要附加内容才能适合整个开发项目。这些不同点解释了整个开发社区的观念：从事大型系统开发的人员把 RUP 视为其问题的解决之道；而小型系统社区则把 XP 视为其问题的解决方案。我们的经验表明：大多数的软件项目介于这两者之间 － 针对他们的具体情况尝试实现级别恰当的流程。对于项目而言，仅凭两者中的任何一个都是不够的。

当您将 RUP 的广度与 XP 的某些技术结合在一起时，就能够实现适合于所有项目成员的恰当流程量，并确定了所有主要的项目风险。对于在相对信任度高的环境中工作的小型项目团队，对于在信任度相对较高的环境中工作的小型项目团队，其中的用户是团队不可缺少的一部分，XP 可以很好地发挥作用。随着系统变得越来越分散，且代码库不断增长，或者体系结构没有很好地定义，您就需要其他的手段。对于具有合同风格的用户交互的项目，仅 XP 是不够的。 RUP 是一个框架，通过它可以使用一组更强大的技术（根据需要）对 XP 进行扩展。

本书的剩余部分将描述一个基于 RUP 四个阶段的小型项目。在每个阶段中，我们将确定活动和生成的工件。[1] 虽然 RUP 和 XP 确定不同的角色和职责，但是我们在这里并不关注这些不同点。对于任何组织或项目，实际的项目成员必须与流程中的适当角色相关联。

项目开始 － 先启

先启阶段对于新的开发工作意义重大，在这一阶段，您必须先提出重要的业务和需求风险，然后项目才能继续进行。对于专注于改进现有系统的项目，先启阶段较简短一些，但确保项目值得进行并可能进行仍然是重点。

在先启阶段，您将建立用于构建软件的业务案例。 远景是先启阶段中生成的重要工件。它是对系统的高级别描述。它告诉所有人该系统是什么，也可能讲述谁将使用该系统、为什么要使用它、必须提供什么功能，以及存在哪些约束。远景可能很简短，或许只有一两个段落。通常远景中包含软件必须提供给客户的关键功能。

以下示例展示了为重新设计 Rational 外部 Web 站点这一项目编写的简短远景。

> *通过一个动态的、个性化的 Web 站点向访问者提供自助服务、支持以及有针对性的内容，以此增强客户关系，从而使 Rational 成为电子开发（工具、服务和最佳实践）领域的全球领导者。 新流程和实现技术将使提供商能够通过简化、自动化的解决方案提高内容的发布速度和质量。*

在 RUP 中指定的四个基本先启活动为：

- **确定项目范围** － 如果我们要生产一个系统，就需要了解该系统是什么样子的以及它将如何满足项目干系人。在此活动中，我们以足够的详细程度捕获环境和最重要的需求，以得出产品的验收条件。
- **规划并准备业务案例** － 使用远景作为指导信息，我们定义风险减轻策略，开发初始的项目计划，并确定已知成本、进度和利润权衡。
- **合成候选体系结构** － 如果考虑中的系统几乎没有新颖度，并且具有很好理解的体系结构，则您可以跳过这一步。一旦我们知道客户需求，就分配时间研究潜在的候选体系结构。新的技术给软件问题带来
新的或改进的解决方案的可能性。在流程的早期花时间评估购买与构建的利弊，以及选择技术和开发初始原型（可能），这些举措可以减少项目的一些主要风险。
- **准备项目环境** —任何项目都需要一个项目环境。您是使用一些 XP 技术（比如结对编程），还是使用更加传统的技术，需要确定物理资源、软件工具和项目将遵循的过程。

[1] XP 定义了三个阶段：探索、落实和操纵。它们与 RUP 阶段不能很好地映射，因此我们选择使用四个 RUP 阶段来描述流程。

在小型系统中，不会花费许多“流程时间” 来执行先启阶段。通常，您可以在两三天或更短的时间内完成先启阶段。以下部分描述了这一阶段中除远景之外的其他预期工件。*Vision*

一个经核准的业务案例

项目干系人有机会从业务的角度同意项目是值得去做的。 RUP 和 XP 都认为在早期就发现项目不值得进行要比在一个注定失败的项目上浪费宝贵资源要好。正如在“计划 *Extreme Programming*[1]”中描述的那样，XP 在项目如何形成以及涉及哪些角色方面是很含糊的（似乎在现有的业务或系统环境中最清晰），但是在 XP 的探索阶段，它涉及与 RUP 先启阶段同等的工件。

您需要考虑是像使用 XP 那样非正式地考虑业务问题，还是像使用 RUP 那样使业务案例成为一级项目工件。

风险列表

您需要在整个项目中维护风险列表。这可以是具有已规划的风险减轻策略的简单风险列表。对风险划分了优先级。与项目相关的任何人都可以了解到有哪些风险以及怎样随时及时处理风险。Kent Beck 描述了 XP 处理的一组风险以及 XP 是如何处理它们的，但并没有提供处理风险的通用方法。

初步项目计划

在该计划中包含资源估计、范围和阶段计划。在任何项目中，这些估计不断变更，您必须对它们进行监视。

项目验收计划

您是否有一个正式的计划取决于项目的类型。您必须决定客户如何评估项目是否成功。在 XP 项目中，采用验收测试的形式（由客户创建）来对此进行确定。在更普遍的流程中，客户实际上可能不构造测试，但是验收条件必须直接由客户或其他角色（比如直接与客户交互的系统分析员）来驱动。可能还有其它的验收条件，比如 XP 没有涵盖的最终用户文档和帮助。

初始精化迭代计划

在基于 RUP 的项目中，您将在上次迭代的末尾详细地计划每次迭代。在迭代末尾，您将根据该迭代开始处指定的条件评估进度。XP 提供了一些监视和评估迭代是否成功的好方法。这些度量值很简单，您可以很容易将他们并入迭代计划和评估条件中。

精化

精化阶段的目标是建立系统体系结构的基线，以便为构建阶段中的大量设计和实施工作提供稳固的基础。体系结构可引发对最重要需求（那些对系统体系结构有巨大影响的需求）和风险评估的考虑。体系结构的稳定性是通过一个或多个体系结构原型来评估的。

在 RUP 中，设计活动围绕系统体系结构的概念（对于软件密集型系统，是软件体系结构）。使用组件体系结构是 RUP 中包含的软件开发六个最佳实践之一，它建议

[1] 这是最近发布的关于 eXtreme Programming 的三本书之一。

[2] 在 *eXtreme Programming eXplained*中， Kent Beck 描述了八个风险以及 XP 如何解决这些风险（pp 3-5）。我们欢迎读者查看它们并确定是否足够
。我们相信尚有许多其他的风险，并且一个通用的风险处理策略是任何流程的必要组成部分。

花时间开发和维护体系结构。花费在这项工作上的时间减轻了与脆弱和不灵活的系统相关的风险。

XP 用“隐喻”替代了体系结构的概念。隐喻捕获了部分体系结构，而体系结构的剩余部分则随着代码开发的自然结果而演进。XP 假定体系结构由生成最简单的设计和不断地重构代码而产生。

在 RUP 中，体系结构不仅仅是隐喻。在精化阶段，您将构造可执行的体系结构，通过该体系结构可以减轻与满足非功能性需求（如性能、可靠性和强壮性）相关的许多风险。通过阅读 XP 文献，可以推断 RUP 在精化阶段规定的某些内容，特别是在 XP 中称为基础结构的不当关注，这是在浪费精力。XP 认为在需要之前就在构建基础结构上花费工时，将导致过度复杂的解决方案和产生对客户没有任何价值的内容。在 RUP 中，体系结构和基础结构是不同的。

RUP 和 XP 中的体系结构方法截然不同。 RUP 建议您关注体系结构以避免随时间推移而增加的风险、增加项目规模和新技术。XP 假设已存在体系结构或者体系结构很简单或能被很好地理解，从而足以随着您的编码而演进。XP 建议不要为明天设计，而应当为今天实施。如果您使设计尽可能的简单，相信明天可以做好明天的事。RUP 建议您评估此类提议的风险。如果系统或其组成部分将来必须重新编写，XP 表示这仍然比为可能性做规划要好且通常花费更少。对于某些系统，这将是正确的，而通过使用 RUP，您在精化阶段对风险的考虑将使您得到这一结论。RUP 并不断言这对于任何系统都是正确的，经验表明对于较大、较复杂且没有先例的系统，这将会很糟糕。

而以下观点则是正确的：过于关注可能从来都不会发生的未来可能性是一种浪费，适当的关注则是一种明智的做法。有多少公司能够负担不断地重写甚至重构代码？

对于任何项目，在精化阶段应该至少完成三项活动：

- **定义、验证体系结构，并为体系结构设立基线** － 从先启阶段使用风险列表开发候选体系结构。我们感兴趣的是确保预想的软件是可行的。如果选择的技术没有任何新颖度或者系统不复杂，则该任务将不会花费很多时间。如果您是对现有的系统进行改进，如果不需要对现有的体系结构进行变更，则该任务是不必要的。当真正的体系结构风险出现时，您不希望让体系结构自生自灭。

  作为该活动的一部分，您可以执行某些组件选择和做出购买／构建／重用的决定。如果这需要大量的工时，您可以将它分化出来作为单独的活动。

- **优化远景** － 在先启阶段曾开发了一个远景。当您确定了项目的可行性，并且项目干系人有时间对系统进行复审并提出意见时，可能对远景文档和需求进行变更。通常在精化阶段修改它（远景文档）和需求。在精化阶段的末尾，您已经建立了对驱动体系结构和计划决策的最关键用例的可靠理解。项目干系人需要同意：如果在当前体系结构环境中执行当前计划来开发整个系统，当前远景可以发生。在随后的迭代中，变更的量应该减少，但您可能希望在每次迭代中分配一些时间用于需求管理
  。

- **创建构造阶段的迭代计划并建立基线** － 现在，填写您计划的详细信息。在每次构造迭代的最后，您需要复查计划并进行必要的调整。进行调整通常是由于错误地估计了工时、业务环境改变或者需求改变。划分用例、场景和技术性工作的优先级，然后将它们分配到迭代中。在每次迭代的最后，您都计划生成一个可以为您的项目干系人提供价值的工作产品。

在精化期间，您可以执行其他活动。我们建议您建立测试环境并开始开发测试。虽然可能不存在详细的代码，您仍然可以设计并可能实施集成测试。程序员应该准备开发单元测试并知道如何使用为项目选择的测试工具。XP 建议在编写代码之前先编写测试。这是一个好主意，尤其当您是对现有的代码体进行添加时。但是，如果您选择进行测试，则建立常规测试制度的时间是在精化阶段。

RUP 描述的精化阶段包含 XP 中探索和落实阶段的元素。 XP 处理技术风险（例如，新颖度和复杂度）的方法是“最先进”的解决方案，即花一些时间进行实验以估计工时。这种技术在许多情况下是有效的，当较大的风险未包含在单个用例或故事中时，您需要花费更多一些的时间以确保系统成功和准确估计工时。

您通常在精化阶段更新来自先启阶段的工件，例如需求和风险列表。在精化阶段可能出现的工件有：

- **软件体系结构文档（SAD）** — SAD 是一个综合工件，它在整个项目中提供技术性信息的单一来源单一来源。在精化阶段的最后，它可能包含关于对体系结构具有重大意义的用例以及关键机制和设计元素确定的详细描述。当项目是对 现有的系统进行改进，您可能使用先前的 SAD 或者您确定没有该文档不会有风险。在任何情况下，thought 您都应该执行生成该文档的全部流程。
- **构造阶段的迭代计划** —在精化阶段，您计划了构造迭代的次数。每一次迭代都有分配给它的特定用例、场景和其他工作项。可在迭代计划中捕获该信息并对其设立基线。复审并核准作为精化阶段退出条件一部分的计划。

在很小且简单的项目中，您可以将精化阶段与先启和构造阶段合并。虽然仍然要执行基本的活动，但是用于迭代计划和复审的资源将减少。

初始用例模型

虽然这听起来是正式且吓人的，事实上它相当简单易懂。用例对应于 XP 中客户所编写的“故事”。不同之处在于，用例是由参与者、提供可见价值的系统外部人员或事物启动的一组完整操作。用例可能包含多个 XP 故事。为了定义项目的范围，RUP 建议在先启阶段确定用例和参与者。从用户的观点关注整组操作有助于将系统分割为多个提供价值的部分。这有助于确定合适的实施功能，因此我们在每次迭代最后都有一些东西可交付给客户（可能早期的先启阶段和精化阶段迭代除外）。

RUP 和 XP 都有助于确保我们不会处于以下情况：系统的 80% 已经完成，但是未以可交付的形式完成任何内容。我们总是希望能够通过发放系统来向客户提供一些价值。

这里的用例模型确定了用例和参与者，但没有或几乎没有支持的详细信息。它可以是简单的文本，或者是通过手工或画图工具绘制的 UML（Unified Modeling Language，统一建模语言）图。该模型有助于确保我们已经包含了项目干系人的正确功能并且未遗忘任何内容，还使我们能轻松地了解整个系统。用例是根据多种因素（例如风险、对客户的重要性和技术困难）划分优先级的。

先启阶段的所有工件都不需要过度正式或大型。使它们保持简单，或根据需要设定正式级别。 XP 包含关于计划和系统验收的指导信息，而 RUP 则在项目早期添加更多一些的内容。通过提出更完整的风险集，这些少量的添加可能会带来很大的收益。

构造

构造阶段的目标是完成项目的开发。从某种意义上说，构造阶段是一个制造流程，在该流程中强调对资源进行管理和对操作进行控制，以优化成本、进度和质量。从这个意义上说，该管理理念进行了一个转化，它将先启和精化阶段的智力资产开发转化为构造和转换阶段的可部署产品开发。

XP 专注于构造阶段。构造阶段是您生成代码的阶段。 XP 阶段是为了计划目的，但是 XP 的重点是构建代码。

每一次构造阶段迭代有三项基本活动：

- **管理资源以及控制流程** —每个人都需要了解谁将做、做什么以及什么时候做。您必须确保工作负载未超出您的能力并且工作是根据时间表进行的。
- **开发并测试组件** —构建满足迭代的用例、场景和其他的功能所需的组件。您通过单元和集成测试对它们进行测试。

- **评估迭代** － 一旦迭代完成，您就需要确定是否已达成了迭代目标。如果没有，您需要重新划分优先级并管理范围以满足您的交付日期。

不同类型个系统需要不同的技术。 RUP 为软件工程提供了不同的指南并帮助构建合适的组件。用例形式的需求和补充（非功能性）需求对于工程师完成工作而言已足够详细。RUP 中的多个活动提供了关于设计、实施和测试不同种类组件的指南。一个有经验的软件工程师不需要详细地查看这些活动。 而一个不太有经验的工程师将在最佳实践的帮助中获益非浅。每个团队成员都可以根据需要简单或深入了解流程。但是，他们都查看一个流程知识库。

在 XP 中，故事驱动实施。在“*Extreme Programming Installed*”一书中，Jeffries, et al 表示故事是程序员的“对话承诺” 。[1]持续有效的沟通是有益的。虽然始终有一些需要澄清的细节，但是如果故事对于要做大部分工作的程序员来说不够详细，则他们就无法就绪。对于实施用例的程序员来说，用例必须足够详细。在许多情况下，程序员帮助编写用例的技术性详细信息。Jeffries, et al 同时表示这些对话将被记录下来并附加到故事中。 除了可根据需要设定正式级别的用例规范形式，RUP 也建议这样做。保留和管理对话的成果是您必须管理的任务。

XP 专注于构造阶段。有一些适用于大多数团队的至理名言和指导信息。最值得一提的 XP 实践为：

- **测试** － 程序员编写代码的同时不断编写测试。测试反映了故事。XP 力劝您首先编写测试，这是极好的实践，因为它使您深入理解故事并在必要的时候提出更多问题。不管在编写代码之前还是之后编写测试，都要编写它们。将它们添加到您的测试套件中并确保在每次代码变更时运行它们。
- **重构** － 在不改变系统行为的情况下，不断地重构系统以使其更加简单或增加灵活性。您需要确定这对于您的团队而言是否是一个好的实践。对某人简单的东西对另一个人而言可能很复杂。有一个例子，项目中的两个非常聪明的工程师每天晚上都要重新编写另一个人的代码，因为他们认为它过于复杂。作为附带的影响，第二天他们不断地打断团队其他成员的构件。测试是有帮助的，但是如果他们没有进入这种代码战，团队可能会好过一些。
- **结对编程** — XP 声称结对编程能够在更短的时间内生成更好的代码。证据是以下例子。[2]您在实施该实践时需要考虑许多人员和环境因素。程序员是否愿意这样做？您的物理环境是否宽敞足够两个程序员在一个工作台上高效地工作？您将如何利用远程办公或位于其他地点的程序员？
- **持续集成** —经常性地集成并构建系统，可能每天多次。这是确保代码结构完整性的好方法，并且它还允许在整个集成测试期间持续进行质量监控。
- **集体所有权** － 任何人都有权随时更改代码。XP 依赖于以下事实：好的单元测试集将减少实践的风险。让每个人都熟悉每件事的好处不会超过某个比例 － 一万行代码；两万行，当然少于五万行？
- **简单设计** － 与重构一样，系统的设计必须不断变更以减少复杂性。再次强调，您需要确定可将此变更进行到何种程度才会无法继续减少复杂性。如果您在精化阶段花一些时间设计体系结构，我们相信将会出现简单的设计并很快会变得稳定。
- **编码标准** － 这始终是一个好的实践。只要您有一个标准且每个人都同意使用它就可以，至于标准是什么没有关系。

RUP 和 XP 都认为您必须管理（或掌控）迭代。度量值可以提供好的计划信息，因为它们有助于您为团队选择最好的计划。有三个因素需要评估：时间、规模和缺陷。通过此评估，您可以

[1] 该描述出自 Allistair Cockburn。

[2] *Strengthening the Case for Pair Programming*, IEEE Software, July/August, 2000.

获取所有感兴趣的统计信息。 XP 提供了用于确定进度和预测完成情况的简单度量值。 这些度量值围绕已完成的故事数量、已通过的测试的数量以及这些统计信息中的趋势。 XP 是使用最小数量的度量值的强有力例子，因为看到更多的度量值并不一定能够提高项目成功的机会。RUP 提供关于度量什么以及如何度量的指导信息，并且给出一些度量值的例子。在任何情况下，度量值都必须简单、客观、易于收集、易于解释并不易曲解。

在构造迭代中出现哪些工件？您可以创建以下任何工件，这取决于迭代处在构造阶段的早期还是后期：

- **组件** － 一个组件代表一段软件代码（源代码、二进制代码或可执行代码），或者一个包含信息的文件；例如，启动文件或自述文件。组件还可以是其它组件的聚集，例如，由多个可执行文件组成的应用程序。
- **培训材料** － 如果系统具有大量用户界面方面的内容，应根据用例尽早生成初步的用户手册草稿和其他的培训材料。
- **部署计划** － 用户需要一个系统。部署计划描述了安装、测试和有效地将产品移交给用户社区所需要的一组任务。对于以 Web 为中心的系统，我们已经发现部署计划具有更大的重要性。
- **移交阶段迭代计划** － 当您达到将软件部署给用户的时候，您就完成并复审了移交阶段迭代计划。

真的全部关于代码吗？

RUP 和 XP 之间除了在体系结构的方法上有所不同之外，还有其他的不同点。不同点之一是您交流设计的方法。XP 表示“代码即是设计，设计即是代码”。这种说法是正确的：代码与其本身始终保持一致。我们相信除了代码，值得花一些精力来捕获和交流设计。以下简短的故事可以说明这一点。

某个工程师软件有两次软件项目的开发经历；在这两个项目中，设计存在于代码之中，并且代码是唯一可以找到设计信息的地方。这两个项目都与编译器相关：一个是为 Ada 编译器改进和维护优化器，另一个是将编译器的前端移植到新的平台上并将第三方代码生成器与其相链接。。

编译器技术很复杂，但是为人所熟知。在两个项目中，该工程师需要关于编译器（或优化器）的设计和实施概述。在每种情况下，他都收到一堆源代码列表，有几英寸厚，并且被告知“在其中查找”。他迫切渴望得到一些具有支持文本、且构造良好的图。优化器项目从未完成。由于与代码一起开发的大量测试，该编译器项目得以成功完成，且具有很高的代码质量。该工程师花费了几天的时间用调试器遍历所有代码，试图了解它的功能。个人花在较小故障上的成本与团队的成本并不值得。我们不能够像 XP 建议的那样选择 40 小时后停止，并且我们花费了大量的工时完成该工作。

仅仅含有代码的一个原则性问题是：代码（不管记录得多么好）不会告诉您它要解决的问题，它只传达了问题的解决方案。有些对需求的记录颇费周折地说明原始目标，但在这之前，最初的用户和开发人员早已转移目标。为了维护系统，您通常需要知道原始项目团队脑海中的项目。一些高级别的设计文档是类似的 － 通常代码的抽象级别过低，以至于实际上无法表明系统总体上所试图实现的功能。在面向对象的系统中尤其如何；在此类系统中，仅通过查看类很难或不可能理解线程是如何执行的。当以后出现问题时（迟早会出现），设计文档向您指示应到何处进行查看。

该故事要告诉您的是：在捕获和维护设计文档上花费一些时间确实很有帮助。它减少了曲解的风险并能够加快开发的速度。XP 方法是花几分钟时间为设计绘制草图或使用 CRC 卡片。[1] 团队无需维护这些内容，而只需对代码进行处理。这里有一个隐含的假设，即任务足够简单，我们已经了解如何进行。即使我们了解，但随后加入的人员可能未必如此幸运。RUP 建议花费更多一些时间捕获并维护这些设计工件。

[1] 由面向对象设计的培训工作者和元老 Kent Beck 和 Ward Cunningham 开发的 CRC（类、职责和协作）卡片。

移交

移交阶段的重点是确保最终用户可以使用软件。移交阶段包含为发行版准备的产品测试，以及根据用户反馈作出较小的调整。在生命周期中的此阶段，用户反馈应主要集中在产品微调、配置、安装和可用性问题上。

尽早并经常发行是个好主意。但是，发行版的含义是什么呢？ XP 关于这个概念的解释很模糊，并且未提出发行商业软件必须考虑的制造问题。您或许能够在内部项目上省略一部分问题，但即便如何，您仍然需要进行记录、培训等。支持并变更管理是关于什么？期望现场客户控制这些内容是否实际？Bruce Conrad 在他对 XP[1] 的 InfoWorld 回顾中指出：客户可能不希望得到经常变更的软件。您必须在对客户的迅速改变的优点和变更以及可能降低稳定性的缺点之间进行权衡。

当您决定发行时，需要向最终用户提供的不仅仅是代码。移交阶段中的活动和工件将引导您完成软件开发流程中的此阶段。这些活动的重点在于获得客户可使用的产品。移交阶段的基本活动如下：

- **最后确定最终用户支持材料** — 此活动可能如同从清单中选择商品那样简单，但您必须确保您的组织已经准备好支持客户。
- **在客户环境中测试可交付产品** — 如果您可以在您的位置上模拟用户环境，则直接这样做。否则，请到客户位置安装您的软件并确保它能够工作。您不会希望尴尬地对客户说“但是它在我们的系统上是工作的”。
- **根据客户反馈对产品进行微调** — 如果可能，计划一个或两个 beta 测试期，在此期间，您可将软件交付给数量有限的客户。如果您进行此 beta 测试，则需要管理此测试期并在测试的最后阶段考虑客户的反馈。
- **向最终用户交付最终产品** — 有许多关于封装、制造和其他生产问题的细节需要解决，这取决于软件产品和发行版的类型。很少会将软件放在一个目录下，然后发出邮件通知您的客户社区已经为他们准备好了软件。

与大多数其他阶段一样，流程的正式度和复杂度不尽相同。但是，如果您不注意部署细节，则可能抵消几周乃至几个月的有效开发工作，最终得到一个将在目标市场上失败的产品。。

在移交阶段你可以生成多个工件。如果您的产品是有将来发行版的产品（将有多少？），您应该开始为下一个发行版确定功能和缺陷修订。任何项目的基本的工件为：

- **部署计划** — 最终确定您在构造阶段启动的部署计划并将它用作客户交付的路线图。
- **发行说明** — 它是特殊的软件产品，不含针对最终用户的最终指示信息。请为您的说明计划一种可使用的统一格式。
- **培训资料和文档** — 此类资料可以有各种各样的形式。您要在线提供任何资料吗？您是否有教程？您的产品帮助是否完整且可使用？不要想当然地认为您的客户知道您所知道的内容。您的成功决定于帮助客户取得成功。

摘要

构建软件不仅仅是编写代码。软件开发流程必须以向您的客户交付质量而必须采取的活动为重点。一个完整的流程不一定要很庞大。我们已经展示了如何通过关注项目的基本活动和工件来创建一个小巧而完整的流程。执行活动或生成工件（如果它有助于减轻风险）。
为您的项目团队或组织使用尽可能多或尽可能少（根据需要）的流程或形式。

[1] http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml

RUP 和 XP 不一定是互斥的。通过合并两种方法的技术，您可以实现有助于您比现在更快地交付更高质量软件的流程。Robert Martin 描述一种称为“*dX* 流程”的流程，他声称这种流程与 RUP 兼容。[1]它是从 RUP 框架构建的流程实例。

好的软件流程合并了经业界验证的最佳实践。最佳实践就是经过实际的软件开发组织反复测试的实践。XP 是一种目前备受关注的方法。它以代码为中心，承诺提供最小的流程开销和最大的生产力。XP 中的许多技术保证在合适的情况下的注意事项和采用。

XP 的重点是故事、测试和代码 — 它以相当的篇幅讨论计划，却轻描淡写得对待计划的捕获。 XP 表示您可以生成其他的内容，例如“使用一些卡片进行 CRC 设计，或对 UML 绘制草图...” 或者“不生产文档或其他不使用的工件”，但是顺便考虑它们。 RUP 建议您考虑仅生成在制定或更新开发计划时有用和需要的工件，而该开发计划则指定了这些工件是什么。

RUP 是一个针对整个软件开发生命周期的流程。它的重点是从数千个项目中演进而来的最佳实践。我们鼓励研究并发明能够产生最佳实践的新技术。 当新的最佳实践出现时，我们希望将它归入 RUP 中。

附录 A：Rational Unified Process

Rational Unified Process 或 RUP 提供了一种规范的软件开发方法。它是一个由 Rational Software 开发和维护的流程产品。它附带了多个针对不同类型的软件项目的现成路线图。RUP 同时提供了有助于您使用其他 Rational 软件开发工具的信息，但是它不要求向组织实际应用这些 Rational 工具；RUP 可以与其他供应商的产品相集成。

RUP 为软件项目的所有方面提供指导信息。它不需要您执行特定的活动或生成任何特定的工件。它提供了信息和指南供您决定适用于您组织的内容。它还提供了有助于您定制流程的指南（如果没有现成的路线图适合您的项目或组织）。

RUP 强调采用现代软件开发中的某些最佳实践，以此减少开发新软件的固有风险。这些最佳实践为：

1. 迭代式开发
2. 管理需求
3. 使用基于组件的体系结构
4. 可视化建模
5. 持续验证质量
6. 控制变更

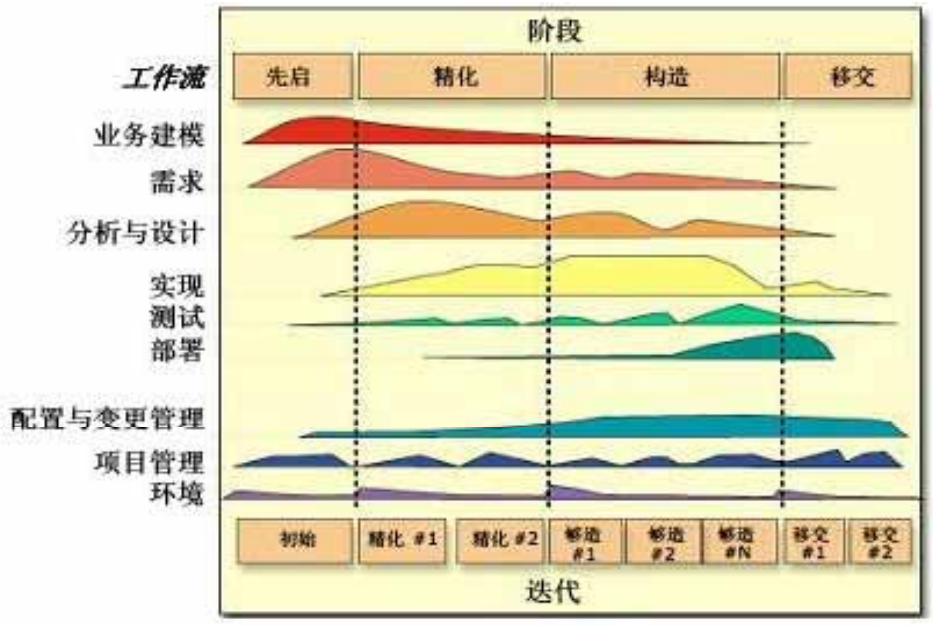
这些最佳实践融入 Rational Unified Process 定义的以下方面：

- 角色 — 执行的活动或拥有的工件的集合。
- 规程 — 软件工程工作的重点领域，例如需求、分析与设计、实施，以及测试。
- 活动 — 工件生成和评估方法的定义。
- 工件 — 在执行活动过程中使用、生成或修改的工作产品

RUP 是一个迭代的流程，将任何软件开发项目分为四个阶段。按照时间顺序，项目依次经历先启、精化、构造和移交这四个阶段。每个阶段包含一次或多次迭代，您可在这些迭代中

[1] http://www.objectmentor.com/publications/RUPvsXP.pdf. 这一章来自 Martin、Booch 和 Newkirk 的一本未出版的著作。

生成可执行文件，但是可能是不完整的系统（可能先启阶段中除外）。每次迭代期间，您将以不同的详细级别执行多个规程中的活动。以下是 RUP 的概述图。



RUP **概述图**

*"The Rational Unified Process, An Introduction, Second Edition"*对 *RUP* 进行了很好的概述。您可以在以下 Rational Software Web 站点找到更多的信息以及 RUP 的评估版：www.rational.com.

附录 B：eXtreme Programming

eXtreme Programming（XP）是 Kent Beck 在 1996 年开发的一个软件开发规程。它基于四个价值：交流、简化、反馈和勇气。它强调客户与开发团队成员之间不断交流，具体方法是邀请客户来到开发现场。由现场客户决定要构建的内容以及按何种顺序构建。您通过不断地重构代码和生成非代码工件的最小集合来实现简化。许多简短的发行版和持续的单元测试是反馈机制。勇气意味着做正确的事，而不是随波逐流。这也意味着应当坦诚说明您能够做到的事和不能做到的事。

有十二个 XP 实践支持以上四个价值。它们是：

- **计划阶段** – 通过结合已划分优先级的故事和技术估计来确定下一个发行版中的功能。
- **小型发行版** – 经常性地向客户发行版本递增很小的软件。 隐喻 - 隐喻是一种简单的共享故事或描述，内容是关于系统如何工作。
- **简单设计** – 通过保持代码简单保持设计简单。不断在代码中查找复杂的部分并立即除去它们。
- **测试** – 客户编写测试以测试故事。程序员编写测试以测试任何可以分解成代码的内容。应当在编写代码之前编写测试。
- **重构** – 这是除去重复和复杂代码的简化技术。

- **结对编程** – 两名程序员为一组，在一台计算机上开发所有的代码。其中一位程序员编写代码或驱动，与此同时，另一位程序员对代码进行复审以确保正确性和可理解性。
- **集合所有权** – 每个人都拥有所有的代码。这意味着每个人都可以随时变更代码。
- **持续集成** – 一旦有实施任务完成，就构建并集成系统，甚至一天多次。
- **每周 40 小时** – 如果程序员疲劳了，就无法以最高的效率工作。决不允许连续两周加班。
- **现场客户** – 真正的客户在开发环境中全日制地工作以帮助定义系统、编写测试和回答问题。
- **编码标准** – 程序员应采用一致的编码标准。

目前有三本描述 XP 的书：

1. eXtreme Programming Explained
2. Extreme Programming Installed
3. Planning Extreme Programming

有多个 Web 站点提供了关于 XP 的更多信息。

Rational
the software development company

两家总部：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
电话：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
电话：(781) 676-2400

免费电话：(800) 728-1212
电子邮件：info@rational.com
Web：www.rational.com
全球网址：www.rational.com/worldwide