RUP® 与 XP 的比较

John Smith

Rational Software 白皮书

TP 167, 5/01

Rational
the software development company

目录

简介

本白皮书将 Rational Unified Process®（RUP）与 Extreme Programming（XP）进行了比较。RUP 是由 Rational® Software 多年来进行优化的一种流程框架，在从小型到大型的众多软件项目中得到广泛应用。XP 是一种软件开发方法，它作为在需求不断变化的环境中构建小系统的有效方法，正获得越来越多的认可。

比较将针对这两者的流程描述的性质和风格、底层结构、假定和表示。本白皮书还探讨这两者的潜在目标以及它们的使用结果。

大多数流程都具有一些公共的元素，这些元素使得可以进行系统化的比较。它们需要由活动组成的序列（或活动组），这些活动是由角色执行的（通常是由以个人或团队形式工作的人员担当角色），目的是生成工件或工作产品，会将其中的一些或全部交付给客户。大多数流程还认可：流程的实例具有时间维度，带有起点和终点，以及一些有趣的中间里程碑，里程碑代表重大活动（或活动组）的完成和相关工件的生成。相应地，本白皮书将探讨流程的下列维度，并使用它们来进行比较：

- **时间和工作分配** —讨论如何随时间推移来安排每个流程，以及如何比较人员工作分配。
- **工件** —比较工作产品，它们是在基于 XP 和 RUP 的项目过程中生成的。
- **活动** —讨论每个流程所声明的工件生成方式。
- **规程** —比较 XP 和 RUP 描述软件工程中重大关注方面的方式。
- **角色** —探讨 RUP 中（执行活动）的那些角色与 XP 中更接近于团队中的职位的那些角色之间的区别。

本白皮书的动机是阐明 RUP 和 XP 在流程领域的相对地位，以了解两者能够互相补充些什么，并驳斥下列观点：XP 是轻量级的，因此是重量级 RUP 的理想替代方案。

编写本白皮书时所用 XP 信息的主要来源是 Addison-Wesley XP 系列的三本书：

- Extreme Programming Explained [Beck00]
- Extreme Programming Installed [Jeffries01]
- Planning Extreme Programming [Beck01]

之所以选择这些书，是因为它们是最显见的 XP 信息库，许多对 XP 感兴趣的读者或 XP 的潜在用户将从阅读这些书着手。还计划使用其他书，但是它们在编写本白皮书时尚不可用。 RUP 信息的来源是 Rational Software Corporation 的 RUP 产品本身。

本白皮书并不打算成为 XP 或 RUP 的教程，因此对于已对这两种方法有一定程度了解或者已读过某些参考资料（例如：对于 XP 是 [Beck00]，对于 RUP 是 [Kruchten00]）的读者，本白皮书将会更加易读。

时间和工作分配

RUP 处理的是项目（用于开发软件），项目的生命周期分为若干阶段，名为：先启、精化、构造和移交。每个阶段可进一步分为若干迭代，每个迭代可能需要若干构建。

对于 RUP 中的大多数项目，迭代的持续时间将在两周到六个月之间[1]，并且项目生命周期中的迭代数量将在三个到九个之间。因此在这些缺省设置下，RUP 所涉项目的持续时间可以从 6 周到 54 个月。

XP 的迭代在持续时间上大约为 2 周。 XP 的发布则为两个月（或稍微长些），它们是由客户定义的，并且发布给客户。在持续时间上，XP 的发布类似于 RUP 的迭代 —至少在精化或构造[2]期间，对于适用于 XP 的项目，举例来说，对于最大团队规模为 10 人[3]且基于已建立的体系结构基线的那些项目。[4]

为了说明这一点，假定 XP 最大团队规模为 10 人，请为这样的团队找到合理的最大规模项目，然后查看 RUP 将如何处理不大于该规模的项目。如果我们使用 COCOMO II[5] 作为估算模型，则它将预测：10 人的团队通常与最长持续时间约 15 个月的项目关联。持续时间超过 15 个月的项目通常需要大于 10 人的团队。您可以只使用 10 人来构建大系统，前提是您已准备好给他们安排时限较长的进度，尽管通常进度的重要性足以使您计划增加更多人员（如果您可以有效地这样做）。该示例项目将从头开始提供大约 40,000 - 50,000 条源代码行（sloc）（在 Java 中是 800-1000 个功能点[6]）。

根据该模型，这是您对于 XP 规模的团队所能够尝试的最大项目（如果您希望它迅速完成）。并且，小团队无法有效构建大系统可能还有其他原因，例如：由于随着时间的流逝，不再熟悉已经构建的代码。（大团队能够并行进行的活动，小团队只能串行进行。因此，当到了集成测试和调试的时间，小团队只能重新访问它相当早的时候所写的代码。）

如果我们查看一组该规模的项目，将发现 XP 发布和 RUP 迭代之间存在合理的映射。下列示例说明 RUP 中如何规划它们。这些数字只是为了说明问题，但是对于这些规模的项目却是合理的。它们包含用于准备所有必需 RUP 工件所需的工作和时间，除了代码外，这些工件也是可交付成果，例如：用户指南、安装和操作信息等。

XP 适用项目示例

示例 1 说明了一个包含 5,000 条新的 Java 代码行的非常小的项目，需要大约 12 人月，耗用时间为 7 个月。[7]

[1] 电子商务开发中迭代的期望持续时间可能要短些 —更有可能在 2 到 6 周的范围内。

[2] 先启阶段的迭代通常更短。还请注意，RUP 的阶段模型包容了范围广泛的情况，因此在漫长的移交阶段，您可以观察一系列迭代，每个迭代都在将软件交付给客户时达到顶点，并且时间间隔同样为两个月；但是进入移交阶段则表示体系结构已完全稳定，并且尝试的更改在很大程度上已具有适应性、完备性和修正性。

[3] 请参阅 [Beck00]，该书声称：您很可能无法以 20 个程序员来运行 XP 项目，但是 10 个却“肯定行”。

[4] XP 特别地将重点放在向客户提供直接的业务价值，主要是作为功能提供的。 XP 几乎没有直接地考虑过体系结构，当添加功能时，通过让软件发生一系列重构而允许此情况发生。如果解决方案的体系结构尚未很好地建立，则存在风险，该风险将导致一系列严重的中断，这是因为：功能的添加将使先前的假定（以及特别的解决方案）无效，并且将带来一些问题，而这些问题并不适合进行局部重构。

[5] COCOMO II 是对原先由 Barry Boehm 博士开发的经典 COCOMO 软件成本估算模型的改进。它针对小到只有 2,000 sloc 的项目。请参阅 [Boehm00]。

[6] 功能点是与源代码无关的软件规模度量，通过量化向用户提供的功能而获得，并且仅基于逻辑设计和功能规范。该定义来自“国际功能点用户组”Web 站点（http://www.ifpug.org/）。

[7] 这可能看上去明显很漫长，但是注意到它涵盖了整个生命周期，从起点开始，那时根本没有任何人员和任何已定义的需求（只有一点点构想），直到项目验收和关闭。它还是 COCOMO II 模型的输出，实现了更大的进度压缩，但是以增加成本和风险为代价。然而，根据表中的进度，在项目开始后的三个半月（第一次构造迭代结束时）这么早的时间，就可以提供具有有用功能的成果。

示例 1

	先后	精化	构造	移交
人员	1	1.5	2	2
持续时间（单位：周）	3	6	18	3
迭代数（以及 每个迭代的持续时间，单位：周）	1（3）	1（6）	3（6）	1（3）

示例 2 说明一个包含 10,000 条新的 Java 代码行的小项目，需要大约 27 人月，耗用时间为 8 个月。

示例 2

	先后	精化	构造	移交
人员	1.5	2.5	4	3
持续时间（单位：周）	4	7	20	4
迭代数（以及 每个迭代的持续时间，单位：周）	1（4）	1（7）	3（7）	1（4）

示例 3 说明一个包含 40,000 条新的 Java 代码行的中型项目，需要大约 115 人月，耗用时间为 15 个月。

示例 3

	先后	精化	构造	移交
人员	3	5	10	8
持续时间（单位：周）	6	16	36	6
迭代数（以及 每个迭代的持续时间，单位：周）	1（6）	2（8）	4（9）	1（6）

在这些约束下（所涉开发的范围相当广泛），在意向和持续时间上，RUP 迭代非常接近于 XP 的发布。

XP 不适用的大型系统开发

在非常大型的开发（RUP 可处理这类开发，而 XP 不能）中，RUP 迭代的持续时间要长得多。示例 4 显示了一个包含 1,500,000 条新的 Java 代码行的项目，需要 4,600 人月，耗用时间为 45 个月。[8]

示例 4

	先后	精化	构造	移交
人员	35	70	140	100
持续时间（单位：周）	20	50	100	20
迭代数（以及 每个迭代的持续时间，单位：周）	2（10）	2（25）	3（33）	2（10）

[8] 您可能会争辩说：您绝不会以这种方式处理项目，该规模的项目应当分为许多较小的项目（XP 可能适用于其中的每一个）。当然，该规模的项目构造为“系统套子系统”（或者，对于非常大型的项目，则为“系统套系统”），但是当这些子系统或系统需要集成到一个产品中时，您将需要考虑体系结构，特别要考虑软件的聚集与生成这些软件的团队之间的界面。现有形式的 XP 不能处理这些问题。

显然，在此示例中，人员并未组织成一个统一团队 — 项目包含若干个团队，每个团队负责处理子系统，这些子系统可能再包含子系统，因此在较低的级别，将存在规模上与 XP 适用规模的项目类似的规划。然而，该项目的目的是表示集成系统，因此在最高级别，必须对远远超出了 XP 规定的大型迭代（33 周）进行规划。这些迭代具有如此长的持续时间，是因为该规模的集成项目的规划惯性所致。 RUP 的迭代内规划是通过迭代构建计划（可能以系统级别和子系统级别存在）完成的，并且这些规划以接近于 XP 的发布的规模构建（对于这一非常大型的系统），在最低的级别则接近于 XP 的迭代（需要约两周）的规模，特别是在后期的精化和构造中。

因此，回到小系统的示例，RUP 的迭代或多或少等价于 XP 的发布，RUP 的构建或多或少等价于 XP 的迭代。

RUP 的各个阶段如何映射到 XP 中？

至少初看上去，RUP 的各个阶段映射到 XP 周期的各个方面，但是这些方面并未得到很好的描述。 XP 看似已经得以构造，或者至少已经得以描述，来与那些包含迭代的节奏稳定的发布（与业务周期相协调）相适应。已经认识到：存在某种必须引导才能存在的项目（请参阅 [Beck01] 中的“Scoping a Project”），此时构造的远大计划将随后分成多个发布，而发布接着又分成多个迭代。 [Beck01] 声称，远大计划所做的一件事就是帮助我们作出决策：对项目投资并不是在干傻事。

因此在 XP 中，在发布和迭代周期开始前有一段时间，这段时间中对项目进行的工作是作出下列决策：项目是否可行以及成本将是多少，为此，必须对必需功能构造粗线条的概念。 RUP 中将它称为先启阶段。在 XP 中给您的印象是：这必须迅速完成，要非常迅速。在 RUP 中，这将是作出审慎决策所必需的时间 — 取决于内在的风险。即使是在 XP 中，此时也必须（由少数几个人员）执行下列工作：

- 需求引发（编写“远大构想”）
- 规划
- 与客户协商（设置期望值）
- 分析所有的业务约束

显而易见，从冷启动进行到达成一致的“远景”（这是“远大构想”在 RUP 中的提法）和业务案例（能够证明项目合理性的预算和投资收益率（ROI）），并且着手制订软件开发计划，可能需要几天时间。

在 RUP 中，象在 XP 中一样，您需要在项目开始时进行充分的探索和规划，以追寻您所建立的目标，这样您成功的机会就会很大。如先前在示例 2 中所示，RUP 规划建议在先启阶段（在工作方面）投资大约 $12,000，来证明总价约 $250,000 的项目费用的合理性。这不是一成不变的：在一些已彻底处理过的问题区域，您不需要如此费事，例如：您可能不是冷启动的。在其他场合，如果某个领域需要某种程度的研发，而您要在这个不熟悉的领域中从头开始，您会更加费事。

XP 经过了先启的等价阶段之后，将直接转为规划第一次发布。 RUP 声称先启之后是精化阶段，在精化迭代中，解决方案的体系结构将得到稳定。作为很明显的对比，XP 则建议发布规划应当面向功能，因此所有发布都要向客户提供业务价值。 XP 很明确地表示，它很看轻它称为基础结构规划的那种活动：为了支持针对该发布所选择的功能，而去做一些事情，这已经足够。以后将按需要添加更多的基础结构，并且，如果后面添加的功能已使得先前的基础结构决策失效，而使系统发生崩溃，那么将对代码进行重构以使它工作。[9] 设想是：不要花费很多的时间构建不能对客户提供价值的东西。

[9] 然而请注意，重构只带来局部的改善：重构并不针对整体框架。如果整体框架解决错了问题，
那么重构不会生成正确的解决方案。唯一的解决方案是作出激进的更改，或者极端的做法是：从头开始。这显然具有预算和进度方面的风险。

比较下来，需要对 RUP 作出以下几点说明：

- ***RUP 的体系结构概念不仅仅是基础结构***

  引自 RUP：“…软件系统的体系结构（在给定时点）是系统中重要组件的组织或结构，这些组件通过接口与由更小的组件和接口组成的组件交互。”

  因此，从一组特定的视角并以相应的抽象级别来看，体系结构针对的是整个解决方案（而不仅是基础结构）。

- ***RUP 的可执行体系结构的观点向客户提供了业务价值***

  它使得能够在早期证明解决方案满足客户的非功能需求（并且可能还满足流程中一些关键的功能需求）。

  通过这种方式降低风险（非功能需求通常是具有风险的需求），这本身就对客户具有相当的业务价值。

我们为什么认为有必要以这种方式描述流程？因为 RUP 致力于全面排除风险，并且我们认为有些类型的问题和系统就是不适用 XP 方法，XP 方法允许从代码重构中形成体系结构，通常会带来更大、更复杂的风险。其中一个风险是：通过重构作出局部更改（有必要限制范围）的情况下，将生成局部优化，这些局部优化加在一起，将导致解决方案的优化程度变差。这并非只是我们的一家之言：在 [Beck00] 中名为 *Four Values* 一章的标题 *Courage* 下，Kent Beck 自己也已说得很多。难处在于 XP 对体系结构上具有重要意义的那些更改的灵感起源说得过少 － 它只是说应用这些更改可能需要一些勇气[10]！

然后就会有一些更改，而这些更改恰恰使重构变得艰难。例如，要将单用户、单机的系统转为多用户、多处理器的系统，可能必须进行很多的重新设计，然而最后仍然得到存在很多约束的系统。

RUP 强调体系结构，目的是为了针对下列情况：初始方法可能是错误的，需要完全重新考虑。对于小系统，这样风险更小 － 重新分析可以将系统更全面地检查一遍，并且初始体系结构出错的可能性更小。

还请注意：如果几乎不存在任何能够感知到的风险（不能感知的原因包括：规模、新技术、低熟悉度、复杂度，以及关于性能、可靠性和安全等方面的其他苛求），并且能够通过很好理解的现有框架交付解决方案，那么 RUP 精化阶段（处理体系结构事宜的阶段）可能非常简短（并且可能更加关注重要功能需求的引发、优化和证明）。即，RUP 生命周期退化为接近 XP 生命周期。当问题能够通过 XP 得到成功解决时，RUP 开发案例也将产生类似“轻量级”的结果（尽管并不完全一样）。

RUP 的构造阶段等价于 XP 的一系列发布，原因如下：如果您遵循 RUP，那么您必须向客户提供每个构造迭代的结果，以便具有与 XP 相同的效果。XP 实际上不具有与 RUP 的移交阶段等价的阶段，因为每个 XP 发布已交到客户手中了。当 XP 项目结束（[Beck00] 称为 project death）时（请参阅稍后标题为 *XP* 生命周期的内容），将发生一些相同的项目收尾活动。

XP 的各个阶段如何映射到 RUP 中？

XP 的各个阶段（同时适用于 XP 的发布和迭代）如下：探索、承诺和开动。在发布级别，它们与 RUP 项目管理规程中的各组特定活动（在 RUP 中称为工作流程明细）相一致。它们是：规划下一迭代（映射到“探索”和“承诺”），监视并控制项目（映射到“开动”）。这将在以下各部分中说明。

[10] 谁也不会否认“勇气是美德”－ 并且还暗示它是“好人”拥有的一种品质。然而，即使是最有勇气的人，有时也需要有条理的工作方式，并且需要一个能够用来解决棘手问题的框架。

XP 阶段

在 [Beck00] 中，有一章是关于理想 XP 项目的生命周期的，一级标题为 **exploration**、**planning**、**iterations to first release**、**'productionizing'**、**maintenance** 和 **death**。它们看似顶级阶段，但其实这样说并不十分准确。 XP 项目由初始探索阶段和项目收尾时的"终结"进行定界。然而，贯穿其中的主旋律是"发布"，并且每个发布都具有探索阶段。因此，作为项目的开始（并将导致第一次发布）的那个探索阶段是一个特例，这是因为：需要通过一个初始关口（请参阅 [Beck01] 中的 *Scoping a Project*），在该关口要运用智慧对"是否继续"的问题作出决策。 XP 发布和迭代都具有三个阶段：探索、承诺和发动。"维护" 确实使 XP 项目的性质在第一次发布之后得到显现。

XP 生命周期

总体上说，对于具有 7 个发布且耗用 15 个月的项目，XP 生命周期可能类似于图 1。

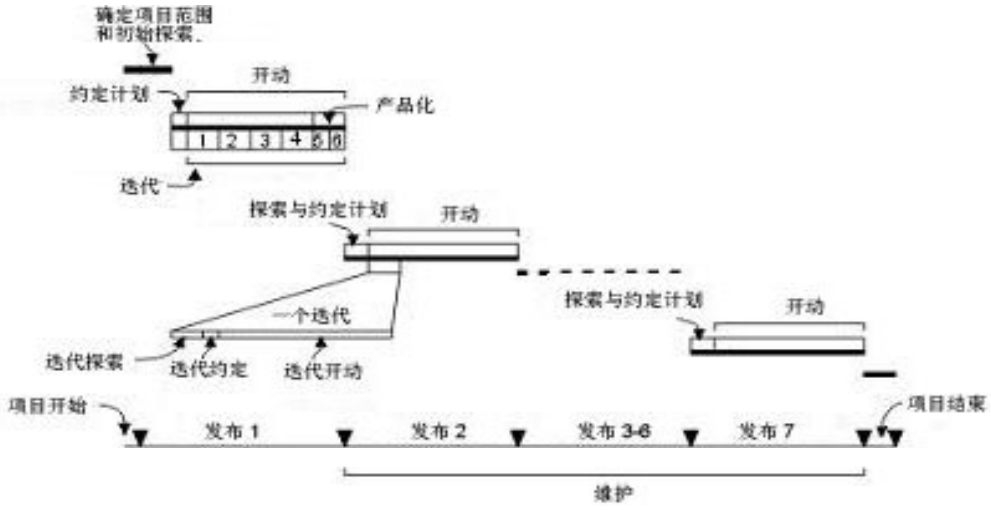

图 1

第一次发布的持续时间为 3 个月，其后的每次发布在持续时间上都约为 2 个月，并且每个迭代都约为 2 周；但是发布的后期阶段（[Beck00] 中称为"productionizing"）除外，在那些阶段，迭代的节奏将加快。

从规划度量的角度看，这有意义吗？在这个设计的示例中，我们尝试遵循 XP 所规定的指南（发布的持续时间应为大约 2 个月，第一个发布则为 2 到 6 个月），并且已使团队规模在 10 人以下。如果该项目象先前所示的示例 3 一样，那么它总共将提供大约 40,000 条 Java 代码行或 800 个功能点（如果我们接受 COCOMO II 模型中的转换因子）。如果这在发布之间是均匀分布的，那么每个发布提供大约 115 个功能点。

有疑问的地方将是第一次发布：从头开始（比如：分配了一个人员、未记录任何需求、尚未确定任何范围），直到在 3 个月内向客户交付具备生产质量的成果，将是一个难题。即使我们实现了高生产力，比如 12 个功能点／人月（根据 David Consulting Group 的业界数据，请参阅 http://davidconsultinggroup.com/indata.htm），仍然不可能以任意高的速度扩展人员队伍 — 此处我们处理的问题空间（115 个功能点）很小，无法分为任意小的部分以便让大团队攻克它。

然而，如果我们假定可以做到这一点，那么所需的平均团队规模将大约是 4 人，并且项目将在 7 人团队的情况下退出第一个发布。如果在项目的持续时间中向团队添加了一个人员，那么对于项目的剩余部分，我们将因 8 人团队而处于 XP 的常规方式（维护）— 8 人团队在 XP 中处于合适规模区域。随着所提供规模的扩大，生产力几乎总会有一点下降，降低的发布时间间隔（2 个月）将抵消由于添加人员而带来的好处，因此每个发布将提供与第一个发布大致相同的附加功能权重。总体项目工时则为 108 人月 — 略低于先前所示的示例 3 中的数据。

缺省 RUP 生命周期

作为对比，类似规模的 RUP 项目的缺省生命周期如下：

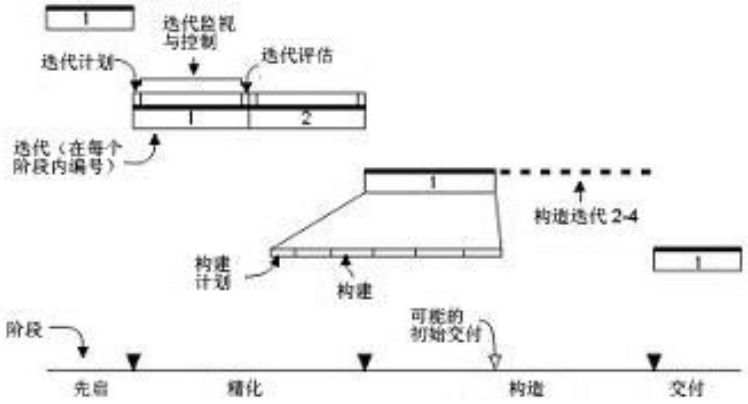

图 2

同样，该规划也基于示例 3。对每个阶段中的 RUP 迭代进行了编号。使用缺省 RUP 生命周期的情况下，向客户交付接近生产质量的成果的最早机会最有可能是在第一个构造迭代结束时，这发生在项目的第 7 个月。然而，此时 RUP 可以交付总能力的大约 1/4，或 200 个功能点（与 XP 三个月后的 115 个功能点形成对照）。通常，对客户作出的第一次交付发生在构造阶段结束时，对于此例是在项目的第 13 个月，那时实质上所有能力均已存在。这并不表示这是客户第一次看到产品；对于 RUP，客户将被邀请参加迭代评估，并且能看到接受测试的产品不断发生演进。

RUP 和 XP 之间为什么存在这一差别？发生这种情况是因为，在这一缺省示例中，已假定解决方案的体系结构中存在足够的风险（原因包括：体系结构是先前未经历过的、技术是新的，或者非功能需求特别难以负担），有必要在精化阶段用两个迭代来使它稳定，然后再尝试构建客户所需的功能。这意味着在精化结束时，已探索（并且可能已实施）的将只有那些在体系结构上具有重要意义的需求。

XP 并不这样做。对于每次发布，它都提供一系列在功能意义上完整的解决方案，它的假设条件是：体系结构不会在发布之间发生崩溃，或者重构可以修复发生的任何崩溃。我们认为这会将 XP 适用的对象限制为能够构建在能力已知的现有体系结构上的那类系统。

实际 RUP 生命周期

对于这样的系统，实际的 RUP 生命周期略有不同：精化阶段将短得多，先启阶段也很有可能更短。如果我们将一个精化迭代替换为一个构造迭代，那么我们可使第一个构造迭代的结束时间提前几个月，将从 7 个月缩短为 5 个月。

在缺省示例中将可能提供略少于 200 个的功能点（原因是处理先前构造迭代的人数将有所减少），但这已接近 XP 进度的灵敏度 － 通过足够的松弛来大大降低风险。作为极端情况，我们可以想象一个基于 RUP 的方法，该方法是围绕小系统（比如，象 XP 示例中的 115 个功能点）的初始交付建立的，后面跟随许多完整的 RUP 演进周期（完整的“先启、精化、构造和移交”序列），以便通过类似于 XP 的维护方式来交付系统的剩余部分。

同样地，在大约 6 个月时，初始交付可能处于 XP 所建议的时间范围的外边界，但是随后 RUP 将认可并要求您规划（并为之分配时间和工作）那些 XP 似乎要掩盖过去的活动，例如发布的部署。

我们可从这些观测中推出关于 XP 适用项目的性质的其他限制。 XP 明确要求在客户与开发团队之间有非常密切的关系，要求有“全职”客户在现场，并且要求客户编写构想并定义发布。在 XP 中，客户实际上是团队中的一个角色：担当该角色的人员或小组可能属于也可能不属于独立的采购方公司，但是他们有权代表采购软件的人。从本质上说，他们具有需求。

相比于为外部客户进行的正式签订合同的开发，这种关系更常见于内部开发。 2 个月的发布周期在这种环境中也将更加可以接受。跨越公司内部边界的每 2 个月一次的新发布（带有新的特性，而不只是缺陷修订）部署，可能在运营方面具有不可接受的开销。

有了这些种类的松弛（小团队、已建立的体系结构、内部样式的开发和正式程度较低的部署），适当定制的 RUP 周期将具有与 XP 开发类似的效果。初始发布的持续时间可能略长于 XP 的最优值，但是它将使风险更低。

对于相反的情况（大团队、先前未经历过的体系结构、对交付和部署都有约束的正式签订合同的开发），则很难看到 XP 如何伸缩 — 公平地说，本白皮书的来源并未声称它能够。而在另一方面，RUP 的设计目的是对付伴随此类项目的正式性和正规性。也许，类似 XP 的方法（即：使用结对编程和重构之类的 XP 方法）可以嵌入到较大的 RUP 项目中，但仅仅是在体系结构稳定的情况下。

工件

RUP 描述了 100 多个工件，这招来了批评，指责它对小项目强加了无法忍受的“繁文缛节”般的开销。这种观点是不正确的，理由有以下四点：

- 项目无需生成所有的工件：选择和定制工件是流程的必要组成部分。RUP 针对“什么是可省略的内容”以及“什么是可定制的内容”提供了指南。
- 工件（在 RUP 中，是工件描述）是规范实体，用于指定要由活动生成的信息，并且为了以系统化的方式实现这一点，它可能描述抽象形式的工件。对于以模型、模型元素或文档（约有半数的 RUP 工件分类为文档）形式存在的工件加以 RUP 描述，其目的并不是暗示特定的实现。

  可使用专门用途的工具（例如 Rational Rose）来捕获和展示 UML 模型，或者，作为另一极端，也可以使用通过简单图形工具所作的图，甚至可以是白板上的草图 — 在 RUP 术语中，这将仍然算作工件（尽管存在白板遗失的风险！）。

  术语“文档” 一直以来都带有某些隐含意义，这使得人们对它们的认识仍然近乎偏见，认为它们是离散的纸质工作产品，带有一组特定（且必需）的段落标题 — 所有的文档必须填充文本（可能还有图），这样文档才算完整。但是这仅是 RUP 文档的一种实现形式。

  RUP 文档是由文本和图构成的信息集。为了提供帮助，RUP 指定了这些信息应该是什么，对此，方便而系统化的实现方式是指向一个模板，该模板枚举并描述要考虑的事项和问题。当然可以按照直接而正式的方式来使用这些模板，以实现（宽泛意义上的）文档形式的工件，即：电子文档或纸质文档。认识到许多项目需要这样做以后，我们选择了提供一组模板，这些模板或多或少可以通过这种方式直接使用。但是这样做不是最根本的（并非某个模板中提供的一切内容都一定与一个特定的项目相关），RUP 使得可以针对项目来为工件自由地选择合适的实现方式（以及交付机制）。重要的是它所包含的信息。

- 我们认为，清晰地确定流程的所有可能工作产品是最根本的，这样可以供流程工程师和项目经理考虑，以便他们在清醒且完全了解省略的后果的前提下，作出是省略它们还是定制它们的决策。我们认为，通过这一方式使工件列表保持完整和明确，将使流程的配置具有客观性。它还能够对不太有经验的项目经理提供支持，支持途径为：使他们注意到一些内容，更有经验的经理会对这些内容“视而不见”，就好象没有提到一样，而生手可能恰恰忽视了这些内容。具备定义良好的文档集，还有助于客户和项目经理及早就交付内容和交付形式达成一致，以避免在项目收尾时频繁发生的不愉快的争吵。
- RUP 描述了一些工件组合体，并且还描述了它们的组成工件。这使得能够在适用的场合对活动的输入和输出作出细线条的描述。在描述中，RUP 可能在组合体（例如：设计模型）处打住，并且只是将“类”称为“设计模型的组成部分”，而不是具有各自意义的工件。将它们确定为工件使得可以精确地描述它们的使用，同时又是以增加总工件数为代价来遵循流程元模型。

XP 看上去没有遭受这种“工件过多”的批评，但这其实是过度简化，原因有以下两点：

- 对 XP 已进行了定制以适合某类开发，以特定规模处理 RUP 的一部分规程，因此希望 XP 要求的工件少些。
- XP 强调用户构想和代码的重要性，而在描述流程时，其他工作产品则轻轻带过，因此工件数并不象第一眼看到的那么少。

如果考虑了 XP 方法，那么在我们用作这一比较（因为它们都是流程的旗舰方法）的来源材料的三本书中，术语“工件”和“工作产品”没有出现在索引中，也许就不令人惊奇了。然而，通读文本并提取出工件的参考信息，并不是件难事。下面是一些示例：

- 构想
- 约束
- 任务
- 技术任务
- 验收测试
- 软件代码
- 发布
- 隐喻
- 设计 — CRC、UML 框架
- 设计文档 — 在项目结束时生成
- 编码标准
- 单元测试
- 工作空间（开发和其他工具）
- 发布计划
- 迭代计划
- 会议报告和记录
- 总体规划 — 预算
- 进度报告
- 构想评估
- 任务评估
- 缺陷（和关联数据）
- 来自对话的其他文档
- 支持文档
- 测试数据
- 测试框架工具
- 代码管理工具
- 测试结果
- 峰值（解决方案）
- 任务工作时间记录
- 度量数据
  - 资源
  - 范围
  - 质量
  - 时间
- 其他度量
- 跟踪结果

这里我们有大约 30 个工件，其中一些还是工件组合体。该列表只是为了说明问题，而不是穷尽列举。这些 XP 书籍并未花费许多篇幅描述这些工件中的大多数，然而以这些书籍的编写水平，我们也不期望他们会这样做。然而，一旦项目必须实现这些工件时，就需要关于它们的内容和形式的更详细信息。

项目当然可以到时候再这样做，但是这占用了实际工作的时间；与此不同的是，RUP 预先提供了指导信息，因而节省了项目时间。

为什么 XP 不需要 RUP 的所有工件？

一个原因是 XP 不具备 RUP 的范围。这样做是有意的：XP 是关于编程的，是通过编程来满足业务需求。该业务需求是如何发生的（以及如何对它建模、捕获它，并分析原因），则不是 XP 要关心的主要问题。身为客户的那个 XP 团队成员将提炼的需求作为“构想”提供给 XP 开发团队，他们同时又是业务价值的仲裁者。构想是如何以那种形式表达的（或者可以通过那种形式表达），这当中的奇妙之处不是 XP 所要关心的。因此，举例来说，RUP 在其“业务建模”规程中描述的内容超出了 XP 的范围（RUP 中的“业务建模”有 14 个工件）。 XP 描述了一个流程，该流程向客户定期提供发布。部署这些发布的运营工作不是开发所要关心的内容，因此 RUP“部署”规程在很大程度上超出了 XP 的范围（RUP 中的“部署”有 9 个工件）。因此，对于 XP 适用的小项目，您会希望省略定制 RUP 时的 23 个工件。

另一个原因是 XP 声称可以非常简单地完成需求和设计的捕获：需求是作为用户构想（有时可能必须分割这些构想）捕获的，然后将需求分解为任务（而任务本质上就是设计），需要记住关于系统的隐喻。这对所有系统都可行吗？肯定不是。这对某些系统可行吗？当然。公平地说，XP 并没有声称针对所有系统。并且，可能是 XP 作者在作出这些声明时特别谨慎。

如果没有系统化的方法（例如用例），要清晰地表达更大、更复杂系统的期望行为，会非常困难。要依赖于客户和开发人员之间的对话，从而一致地详述复杂的用户构想，也是不可能的，人类的记忆是容易出错的。并且，开发用于实现大系统中的复杂行为的那些结构，需要辅以一定的能力，即：能够形成该系统体系结构的各种抽象视图，并对之作出评判。 RUP 在“需求”（14 个工件）和“分析与设计”（17 个工件）规程中描述的工件使 RUP 能够应对这些系统的变化、规模和复杂性。

在 RUP 的小项目指示信息中，工件数（对于“需求”和“分析与设计”）减到了 7 个，其中的一些削减是通过简单地引用工件组合体而实现的。这并不是玩什么花样，它处理工件的方式恰与 XP 一样。例如，由于 RUP 在小项目中的可能实现方式，RUP 作出关于设计模型的下列声明：

> “设计模型应随着一系列头脑风暴会议而发生演进，在这些会议上，开发人员将使用 CRC 卡片和手绘的图片来探索和捕获设计。只有当开发人员觉得有用时，才会维护设计模型。它并不与实施保持一致，但是会将它归档以供参考。”

最后，RUP 在项目管理的必要场合实现了更高的正式性（和“正规性”），在一些合同安排上将是有必要的。同样，许多 RUP 项目管理工件（“项目管理”规程中有 15 个）属于工件组合体，仅当项目的正式性要求时，才需作为独立文档实现。例如，在 RUP 中，软件开发计划“包含” “风险管理计划”和“产品验收计划”之类的工件。在较小、不太正式的项目中，也许只需使用“软件开发计划”中的一两段内容，就可以处理这些计划所针对的问题。在 RUP 的小项目指示信息中，项目管理工件的数目减到了 6 个。

比较小项目的工件

因此，当您为小项目定制 RUP 并定制工件需求时，在总体上将发生什么情况？当您查看 RUP 的小项目指示信息，并数出工件数时，将发现数字是 30（如果您略去部署，则为 26）－ 工件其实并不算太多！ RUP 只是清晰地描述了 XP 含糊不清的地方，使您能够确定什么是必要的，并且提供了关于如何作出选择的指导信息。现在，详细程度对于两方面是不同的，但关键是：对于 XP 能够轻易解决的那类小项目，RUP 中工件数的级数与 XP 的相同。

XP 工件到 RUP 工件的大致映射

XP	RUP 小项目指示信息
构想 来自对话的其他文档	远景 词汇表 用例模型
约束	补充规范
验收测试 单元测试 测试数据 测试结果	测试模型
软件代码	实施模型
发布	产品 发行说明
隐喻	软件体系结构文档
设计 － CRC、UML 框架 任务 技术任务 设计文档 － 在项目结束时生成 支持文档	设计模型
编码标准	设计指南 编程指南
工作空间（开发和其他工具） 测试框架工具	工具
发布计划 构想评估 任务评估 迭代计划	软件开发计划 迭代计划
总体规划 － 预算	业务案例 风险列表 产品验收计划
进度报告 任务工作时间记录 度量数据：资源、范围、质量、时间 其他度量 跟踪结果 会议报告和记录	状态评估
缺陷和关联数据	变更请求
代码管理工具	配置管理计划 项目存储库 工作空间
峰值	原型
	开发案例 特定于项目的模板

指南

RUP 与工件相关联，并且提供了指南，这些指南在本质上是关于该工件的更多信息：它的含义、表示方法、与其他工件的关系、内容和使用。这些指南如果打印出来，会有几百页，这可能看上去使人眩晕，然而，您只需针对所需的那些工件来阅读所需的内容。

这些 XP 书籍还包含关于做法和工件的许多指导信息，但并不尝试严格地对这些关系建模（或者说没有这方面的需要）。文献中关于 XP 的内容加在一起比较庞大。写作时可以读到的这三本书加起来几乎 600 页，另外还有两本也很快可以读到了，后两本加起来又有 700 页左右。还有关于重构的书 [Fowler99]，有 400 多页。

不仅如此，还有好几个 Web 站点都涉及了 XP。然而涉及面如此之大，难免会互相重叠 — 它们从不同的角度查看 XP，并增添了经验库的内容。实际上，这说明了 RUP 和 XP 之间的另一差别：RUP 是产品，而 XP 不是。对于 XP 而言没有一个来源，尽管这些书籍将会是好的开始。“获取”XP 的最快方式是通过培训，支持 XP 的那些思想领袖已在出售这种培训。[11]

[11] RUP 有时被说成是“专有财产”，但是任何人都可以购买 RUP 并利用其中的构想，对它进行添加，删除与其组织或项目无关的部分，等等，只要他们遵守版权和许可证协议。以书籍形式提供的 XP 同样是由作者和出版商所拥有的知识产权；唯一的差别是：从 RUP 和 XP 的来源中能够完全理解它们的程度是不一样的。 RUP 作为一个产品，努力做到完整；而当前的 XP 书籍则需要进行补充（通过“浸入式”培训，或者通过咨询）。

活动

RUP 正式将术语“活动”定义为由角色执行的工作，执行方式是使用和转换输入工件，并生成新的和更改过的输出工件。 RUP 接着枚举这些活动，并根据项目中的“规程” 或主要“关注面” （如 RUP 中所定义）将它们分类。这些规程如下：

- 业务建模
- 需求
- 分析与设计
- 实施
- 测试
- 部署
- 配置与变更管理
- 项目管理
- 环境

活动通过它们生成和使用的工件而与时间相关：当输入可用（且处于适当成熟的状态）时，活动从逻辑上就可以开始。这意味着，“生成者 － 使用者”活动对可以在时间上有重叠（如果工件状态允许的话）；它们不必严格依次进行。

RUP 中的活动旨在使生成工件的智能流程变得更透明 － 将针对生成方式提供有用的指导信息。还可以使用活动来帮助项目经理进行规划。贯穿于 RUP 中（通过“生命周期”、“工件”和“活动”来描述）的是“最佳做法”：最佳做法是软件工程原则，它们被证实可产出按照可预测的进度和预算构建的高质量软件。

RUP 通过活动及其关联的工件，支持和实现这些最佳做法 － 它们是贯穿 RUP 的主题。请注意，XP 也使用“做法”的概念，但是应该看到，它与 RUP 的“最佳做法”概念并不完全一致。

XP（请参阅 [Beck00] 第 9 章）以一种不可思议的简单方式来描述软件开发，将它视为包含四个要按照某些支持做法启用和构造的基本活动：

- 编码
- 测试
- 侦听
- 设计

实际上，对于范围而言，XP 的活动更接近 RUP 规程而不是 RUP 活动，并且针对 XP 项目所发生的许多活动（不仅是编码、测试、侦听和设计）将源自 XP 做法的精化和应用。

是否存在 RUP 活动的 XP 等价活动？

是的，有。但是 XP 的“活动” 没有得到正式确定或描述：例如，请查看 [Jeffries01] 中的第 4 章 *User Stories*，您将发现标题是“*Define requirements with stories, written on cards*”，然后在整整一章中综合了流程描述和关于用户构想是什么以及应如何（和由谁）生成的一些指导信息。该书以此方式继续描述，就象这些书都通过大标题（在 [Jeffries01] 是综合的：一些是针对工件的，一些是针对活动的）描述 XP 一样；“完成的任务” 和“生成的内容” 是以不同的规定性级别和详细程度来描述的。

RUP 的显而易见的规定性来源于它对活动及其输入输出进行系统化处理中的完整性和更高的正式性。 XP 不缺乏规定性，但可能试图保持“轻量级”，就略去了正式性和详细性。当对项目实施 XP 时，必须添加 RUP 中提供的详细信息。明确性的缺乏既不是优点也不是缺点，但在 XP 中详细信息的缺乏却不应混淆为简明扼要。有时，项目人员需要了解做什么，那个时候就将需要详细信息。

规程和工作流程

RUP 最近用术语“规程” 替代了“核心工作流程”。 RUP 中的规程是用来生成一组特定工件的一系列活动（及其关联概念），这组工件代表软件开发中的某个重要方面或关注点。这一用法与规程在字典中的定义（作为知识或教学的一个分支）很好地保持了一致。

如前所述，RUP 的规程是：业务建模、需求、分析与设计、实施、测试、部署、配置与变更管理、项目管理，以及环境。这并未涵盖组织或企业在雇佣人员开发、部署、操作、支持、销售、营销或以其他方式处理系统（很大程度上是软件）时可能做的事情的每个方面。例如，RUP 当前未包含“系统工程”。它也没有包含诸如 ISO 15504 的某些国际软件流程标准的所有需求（例如，ISO 15504 涉及了与软件获得和人力资源管理相关的方面）。这是经过选择的：其他那些方面，虽然很重要，但是却在 RUP 设计重点的范围之外。

XP 明确地对自身作进一步限制：它包含四个基本活动：编码、测试、侦听和设计（先前被认为与 RUP 规程紧密地保持一致），这些活动是使用一组做法执行的，这组做法要求执行其他活动，而这些活动映射到 RUP 中的其他一些规程。 XP 做法（逐字引自 [Beck00]）如下：

- “ *规划活动* — 通过结合业务优先级和技术评估来快速确定下一次发布的范围。当现实情况超出计划时，就更新计划。
- *小型发布* — 将简单系统快速投入生产，然后在很短的周期内发布新的版本。
- *隐喻* — 使用讲述整个系统如何运行的简单共享构想来指导所有开发。 简单设计 — 在任何给定时刻，应将系统设计得尽可能的简单。额外的复杂性一经发现即除去。
- *测试* — 程序员持续不断地编写单元测试，为了开发得以继续，测试必须无错误地运行通过。客户编写测试，证明功能已完成。
- *重构* — 程序员重构系统，而不更改它的行为，以消除重复、改进通信、简化系统或增加灵活性。
- *结对编程* — 所有生产代码都由两个程序员在一台机器上编写。
- *集体所有权* — 任何人可以在任何时间更改系统任何位置的任何代码。
- *持续集成* — 一天内多次集成和构建系统，每完成一项任务就执行一次。
- *每周 40 小时工作制* — 将每周工作不超过 40 个小时作为一条规则。不能连续两周加班。
- *现场客户* — 团队中包含真实存在的“全职”用户，负责回答问题。
- *编码标准* — 程序员遵循那些强调“通过代码进行通信”的规则来编写所有代码。”

关于主题简单设计，需要注意的是：“额外的复杂度” 在某种意义上是很主观的术语，因而是难以定义的，很大程度上是有经验的观察者的判断。

例如，作为规划活动这一做法的结果而执行的活动将主要映射到 RUP 的“项目管理”规程。然而，有些主题却不在 XP 的范围内，例如：业务建模。需求引发大部分在 XP 的范围之外 — 客户（与团队一起在现场）定义并提供需求（通过“构想”的形式）。已发布软件的部署则在 XP 的范围之外。由于 XP 所针对的开发的规模和类型，XP 能够以非常“轻量级”的方式处理“环境”和“配置与变更管理”规程中的问题，而 RUP 对这两个规程作了详述。

XP 做法在 RUP 中的使用

在 XP 和 RUP 都包含的那些规程中，XP 中描述的一些做法可以用在 RUP 中（并且一些已经用在 RUP 中），例如：

- 结对编程：XP 要求生产代码由在一个工作站上结对工作的程序员生成，并声称这对代码质量带来有益的影响，并且一旦掌握了该技能，将能够从中得到更大享受。 RUP 没有描述这一细线条级别的代码生成机制，在基于 RUP 的流程中当然也可以使用结对编程。关于该做法的一些信息以及关于测试优先设计和重构（请参阅下文）的一些信息，现在是通过白皮书的形式随 RUP 一起提供的。显然，并不要求在 RUP 中使用该做法，我们也不认为有必要强制这样做。

  若要在行业的规模水平谈论该做法的好处，则显得证据不足，就好象谈论趣闻轶事一般；[Nosek98] 和 [Williams00] 的研究将结对编程与个别编程（孤立地工作）进行了比较，而好的团队并不以那种方式工作。在团队环境中，在开放式通信的文化下，个人可以自由地向他们的同事（以及团队负责人和管理人员）请教问题，并且按照已定义的流程展开工作，我们斗胆猜测，结对编程的好处（在对总体生命周期成本的影响方面）是难以分辨的。在合作良好的团队中，人们将很自然地聚在一起讨论和解决问题，而不会被强迫这样做。

  [Beck00] 则对人员和流程采取几分悲观的态度，它写道：“结对编程的另一强大特性是：没有它，一些做法将无法执行。在压力下，人们将退避。他们会将测试编写跳过。他们会将重构推迟 …”。这些都被认为是很自然的做法，将导致更好的结果 － 为什么人们都不这么做？

  好流程的例子是非干扰的，建议必须在“微观”级别强制实施它将是索然无味的。但是，在某些情况下，结对工作显然是有优势的，因为每个人都可以帮助另一个人，例如：

  - 在团队形成的初期，当人员渐渐彼此认识时
  - 在对某种新技术没经验的团队中
  - 在混合了有经验者和生手的团队中

- 测试优先设计和重构：它们是可应用于 RUP 中的“实施”规程的好方法。特别是测试优先设计，它是用于详细阐明需求的出色方法。
- 现场客户：RUP 的许多活动将在效率提高方面大大获益，这种效率提高是使客户到现场成为团队成员所获得的。客户以此方式在现场工作，可以减少对许多中间可交付成果（特别是文档）的需求。

  XP 不愿意承认除了代码还应该捕获其他内容，而是强调对话是首选的通信途径。这要依赖于连续性和熟悉度才能成功。当必须转换系统时，即使是小系统，也需要生成其他内容。

  XP 最终确实承认了这一点，但却是事后才考虑到的想法；例如：项目结束时的设计文档。事实上，XP 并未禁止生成文档或其他工件（它只是闭口不谈而已），而是声称您应当只生成实际需要和用到的内容。 RUP 对此没有异议，但是当连续性和熟悉度不够理想时，RUP 会继续列出和描述您可能需要的内容。

- 编码标准：RUP 有一个工件（编程指南），它几乎总是被视为必需（大多数项目风险情况，作为定制的主要驱使因素，促成了这个事实）。
- 持续集成：RUP 通过（一个迭代内）子系统级别和系统级别的工作版本，支持此做法；单元测试组件是在新出现的系统环境中进行集成和测试的。

没有伸缩性的 XP 做法

然而，一些做法没有伸缩性（并且 XP 也没有声称它们具备伸缩性），因此对它们的使用要遵守 RUP 中的这种限制，例如：

- *集体所有权*：让小团队中负责小系统或者大系统中的子系统的成员熟悉其所有代码，这样做是很有用的。您是否愿意向团队的所有成员赋予在任何地方作出更改的均等权利，则取决于系统或子系统的性质和复杂度。由当前处理代码段的个人（或结对人员）做出修正通常将更快一些（并更安全）。

  即使是编写得最好的代码，对代码的熟悉程度也会随着时间的过去而急剧下降，当代码的算法很复杂时尤其如此。

- *重构*：在大型系统中，经常性的重构不是所缺乏的体系结构的替代品。 [Beck00] 声称，“XP 的设计策略类似于“爬山”算法（hill-climbing algorithm）。您获得一个简单的设计，然后使其变得稍稍复杂一些，接着简单一些，然后再更复杂一些。爬山算法的问题
  将发展成“局部最优化”，在这种情况下，任何小变更都无法改进状况，但大变更可以做到。”

  在 RUP 中，体系结构使得可以查看和访问“大山”（即：大目标），以使大型、复杂的系统易于处理。

- *隐喻*：对于复杂的大系统，像隐喻一样的体系结构是明显不够的。 RUP 为体系结构提供了丰富得多的描述性框架，并不只是 [Beck00] 轻视地将它描述成的“大盒子和连接”。

- *快速发布*：客户可以接受和部署新发布的速率将取决于许多因素，其中之一通常是系统的大小，该因素通常与业务影响相关。两个月的周期对于某些类型的系统来说可能太短了；部署的运营工作可能不允许。

一些初看显然很可靠并且可能可用在 RUP 中的做法，在普遍应用时需要再多用些心思，再谨慎一点：

- *简单设计*：XP 在很大程度上是功能驱动的：选择用户构想，将它们分解成任务，然后实现。根据 [Beck00]，“在任何给定时间，软件的正确设计是实现了下列能力的设计：

  1. 运行所有测试
  2. 没有重复逻辑…
  3. 声明每个对程序员很重要的意向
  4. 具有最少的可能存在的类和方法。”

  XP 不认同去添加当前还不需要的任何东西。此处存在一个问题，在某种程度上与“局部最优化”问题很相似：当处理 RUP 中称为非功能需求的一类需求时。这些需求将业务价值带给客户，但是将它们表达为构想会更困难 — XP 所谓的约束中，有一些归于此类。

  RUP 不提倡以任何推测的方式设计多于必需的内容，但提倡在头脑中构思体系结构模型，此体系结构模型是符合非功能需求的关键因素之一。

  因此 RUP 同意 XP 的以下观点：“简单设计” 应该包括运行所有测试，但附加条件是这包含那些用于证明软件将满足非功能需求的测试。同样，这只在系统规模和复杂性增加时，或者体系结构是先前未经历过的或非功能需求难以负担时，才成为主要问题。例如，对数据进行编组（在异构的分布式环境中操作）的需要似乎使代码过于复杂，但它对于全局仍然具有必要性。

- *每周 40 小时工作制*：与 XP 一样，RUP 建议加班不得作为长期状况而存在。 XP 并不建议 40 小时的硬性限制，它承认对工作时间的容忍度是不同的。软件工程师是以长时间工作而没有额外报酬而闻名的 — 只是为了看到事情完成而感到满意，而管理人员就不必去横加阻止了。

  管理人员不应该做的是利用这一点或者强制这一点 — 并且管理人员应该始终收集实际工作的小时数统计（即使是无报酬的）。如果很长一段时间以来，任何人的工作小时数记录都看上去很高，那么当然应该调查这种情况。但是，应当在发生问题的特定环境中、在管理人员和该个人之间解决这些问题，还要意识到团队其他成员可能关注的任何方面。 40 小时只是指导性的数字（但具有很好的指导性）。

角色

在 RUP 中，活动是由角色执行的。[12] 角色也负责特定的工件；负责的角色通常将创建该工件，并确保由其他角色作出的任何更改（如果允许此类更改）不会破坏该工件。 RUP 中的角色可由个人或一组人员执行。同样地，个人或小组都可以执行几个角色。角色并非必须映射到组织中的一个职位或“位置”；角色到组织单元的映射也可以是“多对多”。

RUP 角色

RUP 总共定义了 30 个角色：不存在对规程的严格映射，因为角色（例如：软件设计师）不一定限于一个规程，但是大致上会将角色放在它所属的主要位置：

- “业务建模”有 3 个角色
- “需求”有 5 个角色
- “分析与设计”有 6 个角色
- “实施”有 3 个角色
- “测试”有 2 个角色
- “部署”有 4 个角色
- “配置与变更管理”有 2 个角色
- “项目管理”有 2 个角色
- “环境”有 3 个角色

RUP 中为什么有这么多角色？

RUP 中的角色用于对活动进行划分，用于细致地区别执行角色所需的技能和能力，这将有助于指导如何选择执行角色的人员。当角色的重要性发生变化时，该划分级别还能帮助确定在组织中的新职位。例如，对于非正式的小项目，项目经理和配置经理的角色（RUP 角色）可由同一个人执行；而对于大型、正式、航空母舰型的项目，配置经理的工作可能就非常专业化和繁重，足以证明由一支小团队来担当该角色的必要性。通过以这种方式来描述角色，RUP 促成了这一映射。

XP 角色

[Beck00] 确定了适用于 XP 的 7 个角色，然后继续描述这些角色的职责以及执行角色的人员所必需的技能和特征。这些角色如下：

- 程序员
- 客户
- 测试员
- 跟踪人员
- 辅导人员
- 顾问
- 重要主管

在其他一些 XP 书籍中也引用了这些角色，并且还详述了角色执行的活动。

[12] 更精确（准确）地说，活动是由担当角色的个人或团队执行的。

XP 和 RUP 角色数量的差别可以很容易地解释如下：

- XP 不涵盖所有的 RUP 规程。
- XP 角色实际上比 RUP 角色更接近于“职位”，例如：XP 的程序员实际上执行多个 RUP 角色（实施人员、集成人员和代码复审人员），这些角色需要的能力略有不同。

当 RUP 角色映射到小项目时（例如在 RUP 的小项目指示信息中），所映射的类似 XP 的角色数（即：职位）原先为 30 个，现在显著减少。在小项目指示信息中，职位数为 5，如下表中所示。

映射到 ABC 公司小项目的 RUP 角色

ABC 公司工作头衔	RUP 角色
项目经理	项目经理 流程工程师 部署经理 需求复审人员 体系结构复审人员
ABC 公司执行官	项目复审人员 项目干系人 需求复审人员
首席程序员	系统分析员 需求指定者 用户界面设计人员 软件设计人员 设计复审人员 流程工程师 工具专家 配置经理 变更控制管理员 *在不太重要的程度上，与下列各角色等同*： 程序员
程序员	设计人员 实施人员 代码复审人员 集成人员 测试设计人员 测试员
管理助理	*负责*： - *维护“小项目” Web 站点* - *协助*项目经理*角色执行规划或调度活动* - *协助*变更控制管理员*角色对工件变更进行控制* - *还可能按需要为其他角色提供帮助*

总结

XP 实际上与 RUP 不是同一类东西。 RUP 是流程框架，可以从中配置并实例化特定流程。 RUP 必须得到配置，这实际上是 RUP 本身定义的一个必需步骤。严格地说，我们应该将定制版本的 RUP 与 XP 相比较，即：根据 XP 明确设定的项目特征（以及可推断出的项目特征）对 RUP 进行了定制。这样一个定制的 RUP 流程可容纳一些 XP 做法（例如结对编程、测试优先设计和重构），但是它仍然与 XP 不同，因为它承认体系结构、抽象（在建模中）和风险的重要性，并且它的时间结构不同（阶段、迭代）。

RUP 将允许在流程的构造中容纳那些由于规模或种类而不在 XP 范围内的项目。 RUP 之所以是重量级的，仅在于它是对一系列流程的完整描述，根据需要，这些流程在实施上可以是轻量级的，也可以是重量级的（工件、可交付成果、正式性、规定性、正规性，或任何其他关于“量级”的度量）。 XP 当然是轻量级的，因为它有意地针对（轻量级）流程的实施，而 XP 的描述（至少在这些书中）也不详细。因此在 XP 实施中，随时会有需要发现、创造或定义的东西。因此，与 RUP 比较，XP 在描述材料方面也是轻量级的。

事实上，这可能会发生变化，随着另外两本书的出版，可能已在发生变化，其中一本是 [Succi01]，有 512 页。然而，基于这些情况，对于这两种方法，为采纳它们所需进行的工作的概况将是不同的。 RUP 将培训需求和流程定制两方面的大量工作转为预先完成。一个组织还很有可能为了组织范围的应用而针对特定类型和规模的项目定制 RUP，并且将在若干项目中使用这些结果。对于 XP，将需要一些预先的培训，但是剩余的采纳工作将跨整个项目，因为它要扩展并捕获所有这些辅助性的内容，而这些内容是使 XP 能够工作所需的。 XP 没有明确地鼓励记录“公司经验”，使得实施采纳工作的组织（如果它未保存它的流程经验）很容易受到人员更替的影响。

如果不加进一步限定，将 RUP 标注为重量级、将 XP 标注为轻量级，对这两者都会造成损害，因为它没弄清它们各自是什么，打算做什么。并且，如果以错误的方式执行，则只是毫无意义的装腔作势。它们作为流程的实施才需要区分“重量级”或“轻量级”，并且应该根据环境的要求确定是重量级还是轻量级，并且应该根据环境的要求确定是重量级还是轻量级。

XP 并非形式自由，怎么做都可以 — 它专门侧重于软件开发的特定方面以及提供价值的方式，并对实现该目标的方式有非常具体的规定。

RUP 的覆盖面要广得多，并且涉及内容具有深度，这就解释了为什么它的“个头”明显很大。然而在流程的微观级别，RUP 偶尔允许并提供同样有效的替代方法（例如：结对编程的做法），而 XP 则不会。这并不是批评 XP，只是说明 XP 是如何（如其名称暗示的那样）缩小其侧重范围的。

参考资料

[Beck00] *Extreme Programming Explained*，Kent Beck，Addison-Wesley，2000 年

[Beck01] *Planning Extreme Programming*，Kent Beck，Martin Fowler，Addison-Wesley，2001 年

[Boehm00] *Software Cost Estimation with COCOMO II*，Barry W. Boehm 等，Prentice Hall PTR，2000 年

[Fowler99] *Refactoring: Improving the Design of Existing Code*，Martin Fowler 等，Addison-Wesley，1999 年

[Jeffries01] *Extreme Programming Installed*，Ron Jeffries，Ann Anderson，Chet Hendrickson，Addison-Wesley，2001 年

[Kruchten00] *The Rational Unified Process, An Introduction, Second Edition*，Philippe Kruchten，Addison-Wesley，2000 年

[Martin01] *Extreme Programming in Practice*，Robert C. Martin，James W. Newkirk，Addison-Wesley，2001 年（尚未出版）

[Nosek98] *The Case for Collaborative Programming*，John T. Nosek，选自 *Comm. ACM*，第 3 部分，第 41 卷，第 105-108 页，1998 年

[Succi01] *Extreme Programming Examined*，Giancarlo Succi，Michele Marchesi，Addison-Wesley，2001 年（尚未出版）

[Williams00] *Strengthening the Case for Pair Programming*，Laurie Williams，Robert R. Kessler，Ward Cunningham，Ron Jeffries，选自 IEEE Software，第 4 部分，第 17 卷，第 19-25 页

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

Rational、Rational 徽标和 Rational Unified Process 是 Rational Software Corporation 在美国和／或其他国家或地区的注册商标。Microsoft、Microsoft Windows、Microsoft Visual Studio、Microsoft Word、Microsoft Project、Visual C++ 和 Visual Basic 是 Microsoft Corporation 的商标或注册商标。其他所有名称均仅用于标识目的，它们是其相应公司的商标或注册商标。ALL RIGHTS RESERVED.

Copyright 2006 Rational Software Corporation.
如有更改，恕不另行通知。