

XP 和 RUP 的比较

1. XP 简介

XP (Extreme Programming)是 Kent Beck 和 Ward Cunningham 于 1996 年提出的一套软件开发过程理论。它不同于以往的软件开发理论，没有对软件开发的整个过程进行强制而繁琐的规定，而是给出了一套在实际软件开发过程中需要遵守的活动原则。XP 没有强调复杂的过程和繁琐的文档，可以说 XP 是轻量级的软件开发过程理论。当然，与任何软件过程理论一样，XP 也是为了利用更少的成本而开发出品质卓越的软件，满足最终用户的需求，所以它非常强调客户满意度和客户在软件项目中所扮演的角色。在 XP 中，主要强调从四个基本方面来提升软件质量：

- 提高团队成员之间的沟通
- 以最简单的方法设计和编写程序
- 从最终用户处得到持续的反馈
- 在项目开发过程中，保持整个团队的士气和勇气

在现代软件开发过程中，能够唯一保持不变的就是不断的变化。客户的需求随时可以更改；竞争对手会在开发过程中推出新的版本，不得不使我们更改原有的软件开发计划和需求定义。XP 充分认识到了这个不可更改的现实，整个理论强调了作为软件开发团队要充分适应这种不断的变化，即使是在项目开发的后期。

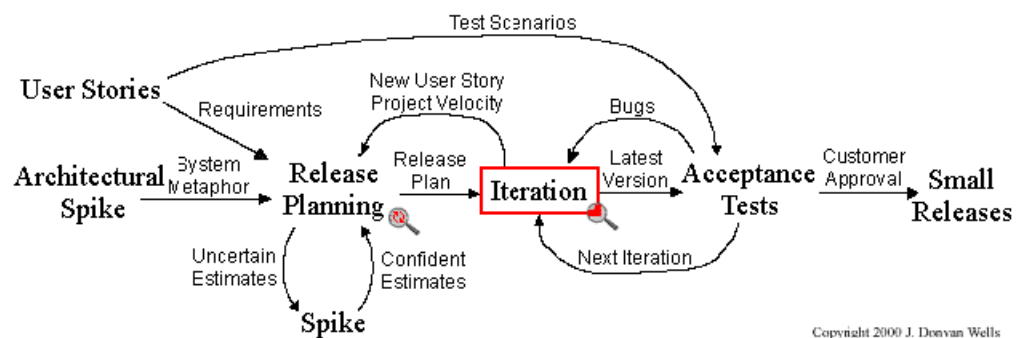


图 1: XP

2. RUP 简介

RUP 是 IBM Rational 的统一过程理论，是一套以文档为主的软件产品，是一套面向过程的软件开发理论。RUP 是一套以架构为中心，用例驱动的迭代开发过程，具有自己的概念，最佳实践和核心工作流程。下面就对这些内容作一简单介绍。

概念：

RUP 强调在软件开发的过程中，需要遵守一定的开发流程。目的是为了按照客户需求开发出高质量的软件产品。并且，为了可以使 RUP 能够可以被更广泛的应用到各种软件开发项目中，RUP 强调了本身的可定制性：即任何组织和项目都可以根据自身的需求开发出符合自身的流程。

- 角色：定义了软件项目中，对某项工作负责的个人或小组。
- 活动：在开发流程中，角色所从事的活动。
- 工件和工件集：工件是流程的工作产品。角色使用工件执行活动，并在执行活动的

过程中生成工件。工件可以是项目计划，需求文档，或是存贮在 Rational Rose 中的设计模型等。工件集是打算用来完成相似目的的一组相关的工件。

- 模板：模板是工件的模型。
- 工作流程：是一系列有顺序的活动。对于 RUP 的核心工作流程，使用活动图来表示。流程中的每一步，都详细规定了哪些角色使用何种模型进行什么活动，用以生产出何种工件。

RUP 的最佳实践包括如下内容（部分）：

- 迭代开发

在 RUP 中，每一代软件产品都要经历四个阶段：先启，精化，构建，产品化。为了减少开发过程中的风险，提高软件质量，适应变更，RUP 要求在每一个阶段中采用迭代开发的方式。

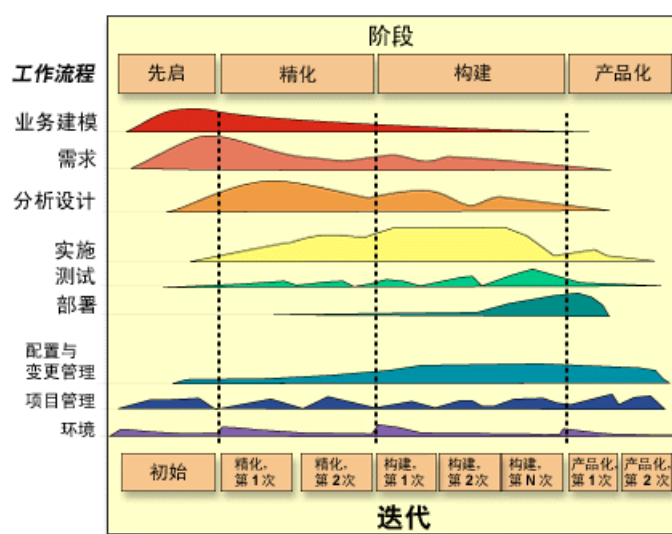


图 2：RUP

- 需求管理

有效记录系统需求及其属性，其间的关联性。用于在客户与开发团队之间就系统的功能达成一致。有利于最大限度的提高客户满意度，促进产品开发团队内的交流。

- 构件构架

RUP 强调系统架构的设计，在迭代的初期需要产生并验证一个构架，并在以后的逐次迭代中逐步完善。好的架构可以明确系统各个组成部分之间的接口，并且可以方便项目管理。

构件是具有明确功能和边界的一个软件模块。架构决定了构件的集成方式。测试是由构件开始，慢慢过渡到整个系统，是自底向上的测试方法。

- UML

使用 UML 对整个系统建模。

- 质量管理

确定质量目标及其评估方法。贯穿于 RUP 整个过程。

- 配置和变更管理

严格的执行变更和配置管理。

➤ Rational 工具

RUP 是一个商业产品，在实施 RUP 的过程中，它推荐使用 Rational 公司的其他产品进行建模和对需求、配置、变更进行管理。

➤ 可配置流程

RUP 是一个可配置的流程。这个软件工程流程可以被用户调整、定制、更改和扩展。

➤ Use Case 驱动

核心工作流程：

在 RUP 中，详细规定了以下流程的工作内容：*业务建模、需求、分析设计、实施、测试、部署、配置和变更管理、项目管理和环境*。对于每一流程，都给出了详细的工作流程明细图，说明了每一个活动所要求参与的角色，所需要的模型和产生的工件。对于流程中的每一步，还给出了一些工作指南，对于用户进行指导。

3. XP 和 RUP 之比较

下面我们就根据 XP 的 12 条基本开发规则来与 RUP 进行一个简单的比较：

3.1 做好计划（The Planning Game）

为了减少软件开发过程中的风险，确保软件质量，XP 和 RUP 均要求采取迭代式的开发方式。在 XP 项目中，项目组要做好软件开发计划，开发人员要估计出完成每一个用户需求（User Story）所需要的时间和投入；客户根据开发人员的估计和自身的需求来决定这些 User Story 的优先级。同时要做好迭代开发计划，明确每一次迭代的目标。XP 中要求每次迭代为 1 至 3 个星期，在项目整个开发的过程中应该尽量保证迭代周期的一致性。在每一次迭代开始时，由用户和开发人员共同商讨决定在本次迭代中需要实现的 User Story。

在 RUP 中，每一个阶段（先启，精化，构建，产品化）都由若干次迭代组成，只不过每一次迭代开发的侧重点不一样。举例来说，在先启阶段进行一次迭代，用以确定项目的规模和前景；在精化阶段的一次迭代用以确定软件需求和架构；在构建阶段进行若干次迭代，实现用例并充实架构；再在产品化阶段经历几次迭代，把产品最终转移到用户群。每一个阶段的结束都是一个主要里程碑。项目经理要在项目初期制定软件开发计划和阶段与迭代计划。

从这里我们可以看到，XP 和 RUP 都要求做好开发计划和迭代计划，明确每一次迭代的目标。这是现代商业软件开发普遍采用的方式。

XP 使用 User Story 的形式记录系统用户的需求。“User Story”指的是软件所需要满足的用户需求（写在 4×6 的卡片上），在 XP 中用于估计软件开发时间，并且作为验收测试的依据。RUP 则是一套以 Use Case 驱动的开发过程（Use case 用来叙述一个用户在外部使用系统的需求情况）：用户可以按照 Use Case 来组织需求；在设计中，要表现各个对象如何通过交互来实现 Use Case；在测试阶段，要通过 Use Case 生成测试用例，用于测试、验证系统的功能。User Case 和 User Story 都是从系统外部描述用户的需求，并且作为以后的开发和测试工作的依据，从这个意义上讲，二者非常的相似。

XP 中并不要求 User Story 的叙述一定要详细到以往软件工程中要求的详细地步（4×6 的卡片上能包含多少内容呢？）。这主要是因为充分认识到了现代商业软件开发的众多不确定因素，为了能够使过程本身可以很好的适应外界因素不断的变化。而在 RUP 中对需求的生成和管理都提出了较高的要求。RUP 的核心工作流程中包含“需求”流程，其中详细定义了需求产生的过程及产出的工件，主要包括：词汇表、需求管理计划、前

景、用例、需求规约（SRS）、需求属性等等。在 RUP 的最佳实践“需求管理”中要求严格控制需求的变更，并根据可跟踪链接（Traceability）确定受到影响的其它工件。从对需求的详细程度、过程要求和产出工件等方面，我们可以看出，RUP 相对 XP 来说是一种“重量级”的软件开发过程理论。

XP 中对需求的要求相对来说要比 RUP 简单的多，所以它要求在开发团队中随时可以找到客户代表，可以一起商讨软件需求。

3.2 小版本发布（Small Releases）

XP 要求每一次迭代为 1 至 3 个星期，项目组应该尽早使客户得到一个可以使用的软件版本，之后在此基础上进行不断的演进。在项目的开发过程中，应持续的从 on-site customer 处得到反馈，用以不断的完善软件。持续不断的与客户进行交流，可以弥补 XP 中需求不够详细的弱点。XP 更强调项目组成员（项目经理、程序员、测试人员、客户等）之间的交流，面对面的交流可以避免由于书写需求和阅读需求产生歧义而带来的损失。

在 RUP 中，由于先期的迭代主要是集中在系统的需求和设计上，一般要到项目后期才能够得到可以使用户真正使用的版本（不是在需求阶段产生的 UI Prototype）。要做到小版本发布，一定要注意软件产品 build 要做到自动化（可以使用 ant, gnu make 等工具）。否则，如果每次产品发布时都会发生 break build，会为项目组带来许多不必要的麻烦。

3.3 设计简单（Simple Design）

KISS (Keep It Simple and Stupid)! XP 要求使用尽可能简单的设计来完成目前开发阶段的用户需求（User Story）。不要担心以后的用户需求会对系统造成何种影响。用户的需求很有可能会出现变化，所以不要进行过于复杂的设计。同时，简单的设计也有利于节省时间，保证系统质量，减少出错的机会。XP 对设计的生成过程和文档没有提出更多的要求。

在 RUP 中，强调了架构的重要性，在开发过程中的“精化”阶段，要生成并验证一个软件架构。该架构是系统的部分实现，是一个可执行的架构原型。使用该架构可以验证系统某方面的功能，并且可以降低以下几方面的风险：性能、吞吐量、可靠性等。并且，在 RUP 中还使用基于构件的开发（CBD）。基于构件的开发，可以带来如下好处：分清系统每一部分的责任，使每一部分具有清晰的边界；可以执行有效的复用；可以作为项目管理的基础；可以方便地进行自底向上的集成测试。

值得注意的是，在 RUP 中也并不否定“设计简单”的原则，简单的设计可以保证系统的健壮性和易于维护（能使用最简单的设计来满足用户的需求才是真正的挑战）。RUP 和 XP 的不同在于架构在整个软件开发过程中所处的地位。RUP 提倡以架构作为软件开发的中心，可见架构在 RUP 中的重要性。而使用 XP，有可能在开始编程的时候，系统根本不具备一个体系架构，只使用最简单的方式去满足 User Story 要求的功能。系统的整体架构是在不断的迭代开发过程中逐渐形成的，并且程序的设计是蕴藏在一行一行的代码中。XP 的侧重点是在于如何灵活地处理不断的变化。在使用 XP 的项目中，不会等到需求和设计都写的非常详细了才会开始编写程序，而是把编写需求和设计的工作减少到最少，尽快的开始编程，通过短周期的迭代不断地产出可以让用户使用的版本，并从客户处得到持续的反馈，在不断的发展过程中逐渐完成系统的每一部分。

由于没有详细的文档来说明需求和设计，在 XP 团队中沟通就显得尤为重要，项目组成员需要面对面的讨论软件的具体问题。XP 的这种做法可以避免由于写作和阅读文

档的歧义而引起的错误，但也因此限制了团队规模。一般情况下，XP 可以应用于 12 个程序员以下的团队。相对而言，RUP 可以应用于较大规模的团队，不同的团队之间通过书写完善的文档来进行交流。

3.4 重构 (Refactoring)

随着开发进程的进行，很有可能发现在项目早期进行的系统设计和程序代码不能适应进一步的开发。在程序中可能会出现重复的代码，从来没有使用过的类或方法。慢慢的这些代码很有可能变成整个系统的累赘，没有人敢于去修改这些莫名其妙的代码。XP 强调要在整个软件开发周期内不断的进行重构，并且对于这些不合适的代码进行坚决的修改，来保持系统程序的简洁和准确，从而达到保证软件质量的目的。

RUP 中没有明确的提到重构。重构是确保源代码质量的一种重要的手段。通过重构，可以最大限度的减少代码的重复，保持简单清晰的设计风格。XP 中建议程序员经常对程序进行重构。

(注：在一般的 IDE 环境 (例如 JBuilder, Eclipse, IntelliJ) 中都集成有重构功能。)

3.5 双人编程 (Pair Programming)

使用 XP 的项目中，代码是由两个程序员共同编写完成的。XP 中是这样形容的：两个程序员共同坐在同一台计算机前，其中一个执行编程任务，另外一人在旁边审查代码；两个程序员会在适当的时候交换他们的工作。乍看起来，可能会觉得这种方法浪费生产力，但 XP 理论认为这种生产方式可以显著提高软件质量，并且不会因为某一程序员的离开而造成软件开发的停顿。通过和不同的程序员进行双人编程，可以使软件的知识传播到每一个团队成员。在实际项目中，真正要实施双人编程可能会有一定的困难。首先，公司的领导层不一定同意这种做法 (既然在同一时刻只有五个人在编程，为什么要十个人呢？)；其次，程序员本身不一定习惯这种编程方式。应为编写程序是一种精神高度集中的脑力劳动过程，需要一个安静和不受其他人打扰的环境；第三，两个个性完全不同的人也很难能够坐在一起共同工作。

如果不使用双人编程的开发方式，可以通过 code review (代码复查) 的方式来复查代码。通过 code review，不仅可以促使程序员开发出优秀、易懂的程序，还可以发现许多潜在的 bug，使团队的每一个成员都对程序有一个整体的了解。

在 RUP 中的核心工作流程——“实施”中也要求在构件在经过了编码和单元测试之后，要进行“复审代码”的活动。该项活动的输入工件为经过编译的源代码和编程指南，输出的工件是被接受或拒绝的源代码和复审记录。复审代码可以促使所有程序员使用相同的编程规范，发现自动测试所忽略的 bug，促进项目成员之间的沟通。可以说，代码审查是人们在长期的软件开发过程中总结出来的宝贵经验，对于提高软件质量有着不可替代的作用。

3.6 集体主义 (Collective Ownership)

XP 强调开发中的软件作为一个整体是属于整个开发团队的。每一个程序员都可以修改任何错误，重构任何一段程序，不只局限于自己开发的代码 (建议在更改别人的代码之前，一定要在源程序中加入注释，用来解释为什么更改，如何更改等问题)，但前提是必须做好单元测试。并且，系统的架构由整个团队负责开发的，每一个程序员都参与，不需要一个特别的架构设计师。

在 RUP 的核心工作流程“分析设计”中，要由角色“架构设计师”来进行架构分析的工作；在编码阶段，要由开发人员进行编码。由于在 RUP 中把架构设计和编码实现分成两个不同的工作流程，很自然的就把不同阶段的工作对应到不同的角色上去。

从这里我们可以引申出一个问题，就是 XP 团队中对成员的要求实际上是很高的。在 XP 中，程序员首先要负责和客户代表讨论具体的需求，决定使用何种技术去满足客户的商业需求（User Story），并且和客户一起制定出迭代开发计划；其次，XP 中无特定的架构设计师，而是强调每一个程序员都是系统的设计者，程序的设计是包含在代码中的；要能够很好的理解其它程序员编写的代码；并且，程序员还要写好单元测试用例，执行单元测试。所以我们看到，在 XP 中，每一个程序员都会身兼多个角色，他必须要具有广泛的知识基础，要有很强的沟通能力和应变能力，还要能够合理的安排个人的工作时间和具有很强的责任心。由此可见，XP 的理论基础是非常强调个体在团队中所起的作用。

如果团队成员的水平参差不齐、有强有弱，最好不要使用 XP 进行开发，可以考虑类似 RUP 的做法：由少数比较有经验的程序员（senior engineer）负责软件整体的架构，而由其它的成员负责编码。

3.7 隐喻命名（System Metaphor）

XP 要求使用一套简单易懂、具有象征意义的词语来命名程序中的类和方法。使任何人都可以通过名称容易地理解整个系统。使用隐喻命名的方式，可以使每一个开发人员能够更好的了解系统每一部分的功能及其相互之间的关系。

相对于 XP 来说，RUP 是使用系统架构来说明每一部分的关系、接口、以及功能的。RUP 的最佳实践包括使用 UML 来进行系统的分析设计，UML 可以在所有的团队成员之间提供可供交流的统一语言。并且，因为 UML 已经非常的普及，项目成员可以使用 UML 来与非项目成员进行有效的交流。

相比之下，使用 UML 的做法比较严谨，不容易产生歧异，可以更好地起到交流的作用。

3.8 持续集成（Continuous Integration）

对于 XP 项目中的程序员来说，至少每天要和源代码管理系统进行一次同步，并且至少要 check in 自己的代码一次。持续集成的好处的显而易见的：可以使程序员工作在最新的代码之上，在早期避免一些由于兼容性而引起的问题。如果程序员在开发了 2—4 周之后才与系统进行集成，所付出的代价将是巨大的：他很可能还需要 1—2 周的时间调整自己的代码来与其它程序员的改动进行集成，并且非常容易产生错误。

在 RUP 中，建议用户 Rational 工具进行配置管理。持续集成可以说是软件配置管理（SCM）的最佳实践之一，无论使用任何过程，任何工具都应该在开发过程中做到这一点。很多新一代的软件配置管理工具可以完全支持持续集成，甚至在产品中彻底的贯彻了持续集成的思想。

3.9 编程规范（Coding Standards）

XP 要求开发团队应使用统一的软件编程规范，使得代码更加清晰和易于其它程序员理解。

RUP 中缺省提供了三种编程语言的规范：Ada，C++，和 Java。可以直接应用到软件开发项目中。

使用统一的编程规范是一个成熟的开发团队的标志之一。使用一套好的编程规范，不仅可以使自己编写的程序易于被其它开发人员理解，还可以使每一个程序员都可以通过源代码本身直接地进行交流。

3.10 不加班 (No Overtime)

为了保证开发团队的士气和精力，XP 不鼓励进行加班，应该严格保证每周 40 小时的工作时间。如果能够实现 XP 的要求，恐怕所有程序员都会认为来到了理想国了。现实世界中，加班是不可避免的（不只是软件开发，各种行业的从业人员都会加班），但是要采取有效的手段激励团队的士气，使得每个成员都具有必胜的信心和旺盛的斗志。

3.11 测试 (Testing)

XP 中主要从两方面强调了测试的重要性：单元测试和验收测试。

- 单元测试：在程序开发之前，需要为所开发的代码编写测试用例。程序员所编写的每一行代码都需要经过单元测试。并且单元测试要做到自动化（使用自动化测试工具：JUnit 等）。所有为单元测试所编写的代码也需要和源代码一起 check in 到版本控制系统中。
- 验收测试：客户为了验证整个系统是否满足需求 (User Story) 而进行的测试。一个 User Story 直到通过了所有和它相关的验收测试才可以认为是真正完成。验收测试需做到自动化，可以用于在产品发布之前的回归测试。

测试的最终目的的保证系统的质量。质量是无论任何过程理论都必须强调的地方。RUP 中的最佳实践中包括了“检验质量”和“控制变更”；质量管理贯穿 RUP 的所有工作流程、阶段和迭代过程。在 RUP 的核心工作流程“测试”中，详细定义了以下和测试相关的活动：制定测试计划、设计测试、实施测试、集成测试、系统测试、以及评估测试；并且在“实施”流程中要求进行单元测试；在部署软件之前要执行验收测试。

除了验收测试和单元测试，RUP 同时强调了集成测试、可靠性测试、压力测试等其它类型的测试。并且给出了一些实践指导原则。在实际软件开发项目中（例如一个 B/S 结构的应用），可靠性测试和压力测试等都具有很重要的意义。即使使用 XP，也不能忽略这些测试。

3.12 客户代表 (On-site Customer)

应用 XP 的开发团队中应该随时可以找到一名客户代表。程序员可以与该代表商量系统的需求。该客户代表还需要参与产品发布计划和迭代开发计划的制定，决定 User Story 的优先级和产品发布时间；以及进行相应的验收测试。

在 RUP 中的“业务建模”和“需求”两个核心工作流程中几乎每一个活动（这些活动主要在先启和精化阶段进行）都涉及到了客户和最终用户。客户和最终用户主要的作用是提出系统需求，协助系统分析员决定需求的优先级、工作量、成本、风险等属性，并且需要协助业务流程分析员完成对目标组织的业务流程分析和建模工作。

客户在现代软件开发中所扮演的角色是至关重要的。开发团队的最终目的就是满足客户提出的需求。能够与客户进行有效的沟通是软件项目成功的必要条件。由于 XP 并没有要求在项目开始阶段写出完整的客户需求，所以它为了把握软件开发的方向，需要不断的与客户进行交流。客户水平的高低，也有可能直接影响到项目开发的成败。所以在 XP 的团队中，不止是每一个程序员要有较高的水平和责任心，团队中的客户也需要

有高水平和尽职尽责。

4. 小结

综上所述，我们可以看到相对于 RUP 来说，XP 是一种轻量级（light weight）的软件开发过程理论，主要体现在三个方面：

- 对过程的要求。
- 对过程所产出工件（文档等）数量的要求。
- 对文档内容的详细程度要求。

当然，RUP 强调了本身是可定制的，用户可以根据自身的需求对流程和工件等进行适当的剪裁（但是读懂 RUP 的所有内容，然后再作出合理的定制，绝对不是一件简单的事情，需要花费大量的精力），从而得到基于 RUP 的敏捷方法。

XP 是一种 people-oriented 的轻量化过程理论；RUP 更像是一种面向过程（process-oriented）的开发理论。可以说 XP 是由程序员来驱动的开发过程，而 RUP 则是过程驱动的开发。相对 XP 来说，RUP 显得非常的庞大，实践指导意义并不十分突出，如果一个项目完全遵循 RUP 去做，很有可能忽略了程序本身，而过于的注重过程控制，走向软件开发中的形式主义。过于的注重形式，势必使团队花费很多的时间去满足过程的要求，使得程序员不能专心在开发程序上。XP 则是强调尽量把时间放在实实在在的编写程序（coding）上。但是，在 XP 中，有一些必要的内容又被过分的省略了。举例来说，任何一个软件产品的发布，都不仅仅是通过验收测试，还应包括相应的文档、培训教材、在线帮助等支持性文档，这些在 XP 中都没有得到很好的体现。

XP 只给出了一般的开发活动原则（例如：计划、双人编程、重构等），没有硬性规定开发过程，所以我们可以说 XP 非常地适合软件需求变化的发生（这正是现代软件开发的一大特点），无论需求如何变化，使用 XP 的团队都可以及时的调整自己的开发计划，重构程序代码，在最短的时间内使软件满足客户的实际需求。开发团队也可以根据项目不同阶段的不同特点以及不同的人员状态，及时的进行开发流程的调整。所以我们说 XP 本身具有高度的适应性。RUP 强调了对于变更的控制，所以说它是力求控制变化，控制变化的发生，控制变化所产生的影响。在 RUP 中，也指明了在产品后期可以对需求和软件架构进行修改，但是变更要有严格的控制。

软件开发是所有团队成员脑力劳动的结果，其特点就是涉及到的人的因素非常多。为了使不同的人员能够有效的合作，生产出零缺陷软件，尽量减少软件项目中的风险，人们逐渐认识到了软件开发要遵循一定的过程规范，才能够避免开发过程中的混乱。但是，即使有了固定的过程控制，也绝不可能淡化人的因素对软件产品所产生的影响。最终编写代码和执行测试都是要由人来完成的。优秀的程序员不使用任何过程也可以开发出很好的软件，他们可以从实践中逐步总结出自己的最佳过程；而平庸的程序员无论使用何种优秀的过程实践，也只能编写出一般的软件产品。

XP 对团队成员的水平要求是很高的，需要通过人与人之间面对面的交流来替代书面文档。之所以这样，是因为长期的实践证明：要书写出没有歧异的软件需求文档需要花费太多的精力，与其这样，还不如在这段时间里开发出一个客户可用的版本，让客户使用并从客户处得到反馈。所以，XP 一般不适用于大规模的开发团队；一般来说，XP 可以应用在 12 个程序员以下的软件开发团队。如果把一个大的软件项目分解到若干个小团队，在每一个小的团队中使用 XP，一定要注意如何进行团队之间的沟通和交流，这也是 XP 理论没有强调的一部分。从理论上来说，RUP 可以适用于更大规模的软件开发团队，团队之间通过书写完备的文档来进行交流。

总之，软件项目由于涉及的人的因素非常之多，直接导致了项目本身的千变万化。在实

际的项目中，必然都存在着一个不断摸索的过程。实际项目本身的过程管理，很有可能包含一部分 XP 的内容，同时又具有 RUP 的因素，很有可能还有项目成员根据以往经验获得的最佳实践。在真正的软件开发过程中，我们不能盲目的去追求单纯的过程管理，而是要注意过程管理的最终目的：开发出优秀的软件产品。