RUP and XP

作者 Gary Pollice

审校：pany[AKA]
译者 shinelee FatCat_Hu guizhiyang beta 周鸿飞[AKA]

?

Part I: Finding Common Ground
第一部分：寻找共同点

eXtreme Programming (XP) is hot! Attend any software development conference today and XP presentations are standing room only. Why? I have observed that XP speaks to the programmer in almost every technical manager and practicing software developer. We all remember simpler times when we sat at the keyboard and coded. Some of us were even happy testing and fixing the code because we wanted it to be perfect. Our code was our legacy to future generations of programmers. And we all had heroes, legends who cranked out more code with fewer errors than we thought humanly possible.
　　极端编程（XP）现在很热门！参加现在的任何软件开发会议会发现听XP演讲只剩下站的地方了。为什么呢？我观察到XP对从技术经理到实习软件开发人员的几乎所有程序员都有吸引力。我们都还记得我们坐在键盘前面编写代码的纯朴年代。我们中的一些人甚至愉快地测试和修补代码因为我们希望它变得完美。我们的代码是我们给未来一代代程序员的遗产。而且我们心中都有英雄，传说他们弄出了超出我们所以为的人力极限更多的代码而只有更少的错误。

XP hints that we may yet return to the "good old days."
　　XP暗示我们也许还可以重回"美好昔日"。

Why don't tears of nostalgia well up in our eyes when we think of the Rational Unified Process (RUP)? Because RUP contains the dreaded "process" word, and most of us have had bad experiences with process. We recall that it was too heavy and too restrictive. Something that wasted our valuable time; something that kept us from coding. But the process itself was not the real culprit: our bad experiences stemmed from the way the process was implemented and used.
　　为什么当我们想到Rational Unified Process (RUP)的时候恋旧的眼泪并没有涌出我们的眼睛呢？因为RUP包含一个可怕的词"过程"，而我们大都在过程上有过很糟糕的经历。我们想起它有太多的负担,，太多的限制。这些是浪费我们宝贵时间的东西，是迫使我们不能编码的东西。但是过程本身并不是真正的罪魁，我们的糟糕经历是源于过程被实现和使用的方式。

In this two-part series, we will look at how to make the implementation and use of RUP a good experience. We will see how it can be used effectively for a small project, and specifically how to incorporate XP practices into the broader scope of a RUP-based project. This month's installment will examine *the areas where the two come together. Next month we will look at how RUP differs from XP, when and why you need to consider the differences, and what benefits accrue with the added strengths of RUP.*

在这个两部分的系列中，我们将看看怎样使RUP地实现和使用成为好的体验。我们会看到它怎样在小项目中被有效地使用，并具体地看到怎样把XP实践应用到一个基于RUP的项目的更广的范围中。这个月的连载将会关注这二者共同的领域。 下个月我们将看看RUP有什么与XP不同，什么时候以及为什么你需要考虑这些不同，以及RUP增加的力量会带来哪些好处。

If you're not familiar with the two processes, then start with my overviews of XP and RUP.

如果你对这两个过程不熟悉，可以先从我对XP与RUP的概述开始。

How do you decide on the right process for a software development project? If you are considering using XP as your process, the first question you need to ask and answer is: Can I actually use XP for this project? This may not be as easy as it sounds. There are lively debates on what makes a project an eXtreme project (see http://c2.com/cgi/wiki?AreYouDoingXp for one view). If you don't refactor your code continuously and write tests before you code, for example, then are you doing XP? In truth, many, if not most, so-called XP projects do not follow XP in an orthodox way. Most use some XP techniques; few use them all.

你要怎样为一个软件开发项目决定正确的过程呢？如果你正在考虑使用XP作为你的过程，你需要提出并回答的第一个问题是：我真的可以在这个项目中使用XP吗？这也许不像听上去那么简单。关于什么样项目是极端项目正有一场激烈的辩论（可以到http://c2.com/cgi/wiki?AreYouDoingXp去看一看）。举例说，如果你并不持续地重构你的代码或者在写好代码之前写测试，你是在进行XP吗？事实上，许多--可能还不是大多数--所谓的XP项目并不以保守的方式遵守XP。大多使用一些XP技术；而很少使用全部。

If you're thinking of using XP straight up, here are some more specific questions you should ask yourself:

Overview: eXtreme Programming
概述：极端编程

eXtreme Programming (XP) is a software development discipline developed by Kent Beck in 1996. It is based on four values: communication, simplicity, feedback, and courage. It stresses continual communication between the customer and development team members by having an on-site customer throughout the development lifecycle. The on-site customer decides what will be built and in what order. The development team keeps things simple by continually refactoring code and producing a minimal set of non-code artifacts. Many short releases and continual unit testing are the feedback mechanisms. Courage means doing the right thing, even when it is not the most popular thing to do. It means being honest about what you can and cannot do.

极端编程（XP）是Kent Beck于1996年发展的一种软件开发方法学，它主要基于四个核心价值：交流，简单化，反馈和勇气。它强调在整个开发周期中，开发人员和用户之间需要不断连续地交流。这种用户称为现场客户，他决定了开发什么样的产品和开发的顺序。其简

如果你正想不打折扣地使用XP，这里有一些你应该问自己的更具体的问题：

- Is the project team small (ten people or less)?
- 这个项目团队很小吗？（十个人或更少）？
- Is the team co-located, and willing and able to do pair programming?
- 这个团队驻扎在一起，并且愿意并有能力做成对编程吗？
- Do we have a commitment for an on-site customer?
- 我们有一个现场客户的承诺吗？

If you answered "no" to any of these questions, then you may not be a candidate for a full-fledged XP project. You may, however, be able to use the RUP and incorporate selected XP techniques into it.

如果你对这些问题的任何一个回答是"不"，那么你也许不能把一个完整的XP作为项目的候选。但不管怎样，你也许可以使用RUP并把选定的XP技术合并进来。

Unlike XP, which focuses narrowly on small, co-located teams with on-site customers, RUP is broader and more flexible. It addresses many styles of software development projects and urges users to adapt its elements to suit their specific projects. It is not hard to imagine an adaptation of RUP that matches XP very closely. In fact, this is exactly the claim that Robert Martin makes for his dX Process.[1]

与严格强调小规模、驻扎在一起的、有现场客户的团队的XP不同，RUP更宽容也更灵活。它准备应对多种风格的软件开发项目并促使用户改造它的成分来配合他们特定的项目。设想一个与XP紧密匹配的RUP改编并不困难。实际上，这完全正是Robert Martin为他的dX过程所做的要求。[1]

单化体现在开发小组不断的重构代码并生成一个不包括代码因素的最小代码集合。许多短期版本发布和不间断的单元测试构成了它的反馈机制。勇气表示做该做的事，即使是不受欢迎的。这意味着要对你能做和不能做的事情都要保持诚实。

Twelve XP practices support the four values. They are:

由以下12个XP实践活动支持这4个方面：

- *The planning game*. Determine the features in the next release through a combination of prioritized stories and technical estimates.
- 规划策略。通过综合考虑具有不同优先级的事情和技术评估决定下一个版本的特性。
- *Small releases*. Release the software often to the customer in small incremental versions.
- 小发布版本。经常向用户发布小型增量的版本。
- *Metaphor*. The metaphor is a simple shared story or description of how the system works.
- 隐喻。隐喻是一种对系统如何运作的简单分配过程和描述。
- *Simple design*. Keep the design simple by keeping the code simple. Continually look for complexity in the code and remove it at once.
- 简单设计。保持代码简单以保持设计简单。不断的检查代码中的复杂之处并立即简化之。
- *Testing*. The customer writes tests to test the stories. Programmers write tests to test anything that can break

XP Practices and the RUP
XP实践活动与RUP

The focus of XP is code: writing the code, keeping it simple, and getting it correct. This is a good thing, as far as it goes. If you build software, ultimately it comes down to delivering executable software to your customers.

XP的关注点是代码：写代码，保持它简单，并使它正确。对于代码而言，这是件好事。如果你构建软件，最终要落实到向你的客户交付可执行的软件。

Let's look at nine specific XP practices to see how they complement or overlap with the RUP. For each practice, we will briefly discuss benefits as well as limitations and hidden assumptions. See the XP overview for a brief description of each specific practice.

让我们从九个具体的XP实践活动来看看它们是怎样与RUP重叠或补充的。对每个实践活动，我们会简要讨论一些好处、限制以及所隐含的假设。在XP概述中可以看到对每个具体实践活动的简要描述。

Everything Starts with Planning
一切从规划开始

When it comes to planning, RUP and XP agree: plans change, and you cannot, practically speaking, plan a complete project in detail. The best approach is to anticipate changes and ensure that you control the associated risks. According to XP, you should prioritize the "stories" you want your system to fulfill, and get technical estimates for the effort required to implement them. Is prioritizing stories any different from prioritizing use cases? Not really, if you equate stories with use cases. Some of the example stories from XP literature are not really use cases, so it may not make sense to equate the two.

in the code. These tests are written before the code is written.

- 测试。用户编写测试用例来测试功能；程序员则写一些可以找出程序中Bug的测试代码，但是这两种测试代码一般都应该在实际程序编写之前就准备好。
- *Refactoring.* This is a simplifying technique to remove duplication and complexity from code.
- 重构，这是一种从代码中除去重复编码和复杂实现的简单化技术
- *Pair programming.* Teams of two programmers at a single computer develop all the code. One writes the code, or drives, while the other reviews the code for correctness and understandability.
- 成对编程。成对编程小组在同一台计算机上开发所有的代码。一个编码，或实现，同时另一个复审代码的正确性和可理解性。
- *Collective ownership.* Everyone owns all of the code. This means that everyone has the ability to change any code at any time.
- 代码共享。每个人都拥有代码。这表示任何人都有能力在任何时间更改任何代码。
- *Continuous integration.* Build and integrate the system several times a day whenever any implementation task is completed.
- 持续集成。在一天之内构造并集成系统数次，一旦有某个任务完成就可以进行系统集成。
- *Forty-hour week.* Programmers cannot work at peak efficiency if they are

A story describes a unit of work, and XP assumes that the story's context is obvious. A use case provides a complete set of operations that provide value to a system user. I believe stories and use cases complement each other, and that a use case can be realized through multiple stories. A use case speaks to all stakeholders, whereas stories speak in more detail to developers. You can produce use-case realizations (according to the RUP) by filling in a complete, more detailed context for the stories.

说到规划的时候，RUP和XP都同意：规划都在变，而且现实地说，你不能够规划出整个项目的所有细节。最好的近似是预计到变化并保证你可以控制相关的风险。根据XP，你应该优先考虑想要系统完成的那些"事例"，并对实现它们所需的工作量作出技术上的估计。优先考虑的事例与优先考虑的用况有什么不同吗？实际上并不能把事例跟用况等同起来。某些XP中的范例事例并不真的是用况，所以等同二者也许是没有道理的。一个事例描述了一个工作的单元，而且XP假设事例上下文环境是显而易见的。一个用况提供了能够为系统用户提供价值的完整的一组操作。我相信事例和用况是互相补充的，一个用况可用多个事例来实现。用况是针对使用系统的用户的，而事例是针对开发人员的并且更详细。你可以通过为事例添加完整的更详细的上下文环境来设计用况（根据RUP）。

Alistair Cockburn says that stories are promises for conversations between the on-site customer and the programmer. These conversations are of great value, and the RUP specifically asks you to consider capturing their results in use cases and other requirements artifacts. XP *implies* that you should capture the results but provides little guidance on how to do it. In XP, the final resting place for requirements or design decisions is the code. Unfortunately, code is not an effective communication medium for all stakeholders.

tired. Overtime is never allowed during two consecutive weeks.
- 40小时工作周。程序员如果疲劳将不能在低效率中进行工作，不允许连续两周的超时工作。
- *On-site customer.* A real customer works in the development environment full-time to help define the system, write tests, and answer questions.
- 现场客户。一个在开发环境中工作的实际客户有助于定义系统，编写测试并回答疑问。
- *Coding standards.* The programmers adopt a consistent coding standard.
- 编码标准。程序员采用一致的编码标准。

For more information on XP, see:
要查找更多关于XP的信息，请参考：

Kent Beck, *Extreme Programming Explained*. Addison-Wesley, 2000.

Ron Jeffries, et al., *Extreme Programming Installed*. Addison-Wesley, 2001.

Kent Beck and Martin Fowler, *Planning Extreme Programming*. Addison-Wesley, 2001.

You can also find information on XP at:

http://c2.com/cgi/wiki?ExtremeProgramming
http://www.extremeprogramming.org/
http://www.xprogramming.com/

程序员间的谈话的允诺。这些交谈非常有价值，RUP明确要求你考虑在用况和其他需求制品中捕获它们的结果。XP暗示你应该捕获这些结果但对如何做到没有提供什么指导。在XP中，需求和设计决定最后体现的地方就是代码。很不幸，代码并非对所有使用者都是有效的通讯媒介。

Getting technical estimates from the developer who will implement a feature is good practice. RUP does not go into detail about how to obtain these estimates, but if you have confidence in the developer, then adopt the practice as part of your planning process. In fact, go beyond this practice: When you get into the details of project deliverables, do estimates for documentation, training, support, and manufacturing.

从将要实现一个特性的开发人员那里获取技术预估是一个好惯例。RUP没有详细描述如何获得这些估计，但如果你对开发人员有信心，那么就采用这一惯例作为你的规划过程的一部分。实际上，可以超越这一惯例：当你深入到项目可交付使用的时候，为你的文档、培训、支持和制造作预估。

Simple Design: No Arguments
简单的设计：没有争论

Every technical discipline preaches simplicity. XP tells us to build the simplest system that meets current requirements, recasting this principle as *You Aren't Going to Need It*. What this means is that you should implement things when you actually need them, not when you realize that you might need them in the future. The RUP says almost the same thing using different words and at different levels: Manage your requirements, continually prioritize, and assess progress. Well-defined, prioritized requirements simplify the developer's decision making about what to do. The RUP also encourages the use of components and the Unified Modeling Language to help manage design complexity.

每种技术总则都强调简单。XP告诉我们建立满足现有需求的最简单的系统，而重建系统这个规则就意味着你不再需要它。这个意思就是在确实需要的时候才去实现，不是你认识到将来你可能需要就去实现。RUP几乎可以在不同的层次用不同的表达方式做同样的事情：管理需求、不断优化，进展评估。定义完善的、优化的需求简化了开发者所需做的决定。RUP也鼓励应用构件和UML来帮助对付复杂的设计。

It is easy to misinterpret the XP advice and mistakenly assume that you do not have to pay attention to infrastructure and architecture. But simple design does not mean that you can ignore required infrastructure or architecture. There is a sharp difference between RUP and XP in this area, which we will discuss next month under the XP practice of "metaphor."

XP 的建议很容易被误解，使人们错误地认为我们不必注意基础结构和体系结构。但简单设计并不意味着我们可以忽略必要的基础结构和体系结构。在这方面RUP和XP之间有一最大的不同，下个月我们将讨论XP的隐喻这个实践活动。

Testing: The Last Word?
进行测试：最后决定性的说明

Test first, then code! This is wisdom from XP, and it is good. The other testing pearl from XP is that the customer provides the acceptance test. Programmers write unit tests to ensure that the code does what they think it does. Customers write acceptance tests to ensure that the system does what it is supposed to do. RUP has a general framework for testing and provides guidance on how to write effective tests. In addition to unit and customer written tests, others may be required: for example, load tests for Web sites. Combine RUP and XP, and you get an excellent quality focus for your team.

先测试，后编码！这是XP的高明之处，它区别于其它测试的精华就是客户提供接收测试。程序员写单元测试以确保代码作的是程序员想完成的事情。客户写的接收测试确保系统功能的正确性。RUP有一套通用测试框架，提供一些有效的关于写测试的指导。除了单元和客户写的测试，还需要别的测试方式：例如装载测试到Web站点。我们把RUP和XP结合起来，就可以得到高质的产品。

In XP, the development team uses the test results to decide whether the system is ready for the customer. If the system passes all acceptance tests, then the software is ready. RUP suggests other acceptance criteria in addition to testing. Depending upon the project, you might consider including customer training, on-site installation, documentation, and several other items in your product acceptance criteria. Simply because a system passes the tests does not ensure that a programmer (or programming pair) has not inserted a trap door or some other time bomb in your software. Sometimes, depending upon the type of system, you need more rigorous code inspections by independent auditors.

在XP中，开发队伍用测试结果来确定系统是否满足客户的要求。如果系统通过了所有的接收测试，则软件就成功了。RUP除了依赖于项目的测试,还提出了其它的接收准则，要考虑包括客户培训、站点安装、文档和其它的产品接收准则。简单的说，因为一个系统通过了这些测试，并不保证一个程序员（或者编程伙伴）没有在软件中插入一个陷阱门或者一些别的时间炸弹。有时依照不同类型的系统，需要更严格的独立的审核者做代码检查。

Refactoring: A Little Goes a Long Way
重构：任重道远

Refactoring is the act of rewriting code to improve it. It is also a technique for keeping the design simple. RUP does not address code refactoring, but that does not mean you should not consider it. Be aware, however, that refactoring may create a risk for your team. What's simple to one programmer may be complex to another. If you do too much refactoring, then the team may thrash around and lose valuable time developing the code.

重构是一项重新编码达到改善软件目的的实践活动。它也是保持设计简单的一种技术。 RUP 没有涉及代码重构，但是并不意味着我们可以不考虑它。 值得注意的是重构可能对团队产生风险，一个程序对于一个程序员来说是简单的,而对于另一个程序员却不尽然。如果作了过多重构，开发团队可能会遭受损失，失去开发这些代码的宝贵时间。

Pair Programming: Are Two Heads Better Than One?
成对编程：两个头脑优于一个？

There is evidence that pair programming is an effective way to improve programmer productivity. Programmers remain focused, and because they get immediate feedback, quality improves. Pair programming is one way to avoid code reviews, but it places some constraints on a project team:
　　成对编程是一个已经得到证明的能完善编程者产品的有效途径。成对编程可以使程序员保持清醒，因为他们立刻就得到了反馈并做保质的改进。成对编程是避免代码检查（走查）的一种方法，但它给项目开发团队增加了一些限制：

- The team must be in one location.
- 开发队伍必须在一起；
- Paired team members must have compatible personalities plus well-matched programming skills.
- 成对编程队伍必须有亲和的个性，此外还需要有配对者两人相配合的编程技巧。

Continuous Integration: Do One Build or More Per Day
持续集成：每天完成一个或多个版本

Every programmer on an XP project must be able to change code and ensure that it works, not just for unit tests but also for acceptance tests. This requires frequent builds: one or more a day. This practice is an excellent one. In order for it to really work, however, you need powerful configuration management tools and an effective process for using them. The RUP provides general guidelines for continuous integration as well as specific information on using Rational ClearCase. In fact, at Rational, we perform daily integrations on the complete Rational Suite product family.
　　在XP 项目中每个编程者都能修改代码，并且保证所改代码能正常运行，不仅要进行单元测试，而且要进行接收测试。这就要求频繁的建立系统：一天可能一次或多次建立系统。这个实践是非常优秀的做法。然而，为了保证这种方式能真正工作，需要功能强大的配置管理工具和一个有效的应用过程。RUP 提供持续集成的一般指导，以及运用Rational ClearCase的特定信息。事实上，在Rational ，我们在完整的Rational Suite产品家族上实施日常的集成。

Forty Hour Work Week: No Sleeping Under the Desk
周工作40小时：工作时不能打盹

What a great idea! Is it practical for your organization? Studies indicate that most people experience rapidly diminishing returns when they invest more than forty hours in their work -- especially when it is habitual. An XP project forbids two consecutive weeks of overtime.
　　多好的主意啊！你的组织能实行吗？研究表明：大多数人当他们每周花在工作上的时间多于40小时，就会导致收效的降低，当形成习惯的时候这种情形尤其突出。一个XP项目禁止两周连续加班。

On-site Customer: A Must-Have?
现场客户：非常必要？

Originally, XP said, "A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities." This has been further refined to, "An XP project is steered by a dedicated individual who is empowered to determine requirements, set priorities, and answer questions as the programmers have them."

最初，XP说"一个真正的客户必须与开发队伍待在一起，回答开发者的问题，解决分歧，设置小规模的优先级"。目前，这已经被提炼为："通过授权一个专用的个体来确定需求、设置优先级、回答问题，就象程序员拥有他们一样，来指导XP项目"

The RUP is more flexible. Although it has always maintained that the customer, in fact all stakeholders, must be adequately represented in steering a project, the RUP also acknowledges that it is not always possible or desirable to have a real customer co-located with the development team. Instead, RUP defines several roles that are responsible for determining the project goals, scope, and so on, and says that a customer (on-site or not) or some other appropriate person in the organization can perform the activities mapped to these roles. It's not important whether the person is an actual customer, or whether he or she is actually on site. What *is* important is that the person be available to clarify issues, and capable and responsible enough to produce the information necessary for the team to progress as quickly as possible -- including feedback in a form the team can understand.

RUP 更灵活。虽然要经常与客户保持联系，事实上是所有风险保管人必须在指导项目中充分地体现其代表性，RUP 也承认真正要客户和开发队伍经常在一起是不可能的或不可取的。取而代之RUP 定义了几个角色，这些角色负责决定项目的目标、范围等等，进一步说就是一个客户（在现场与否）或组织中一些合适人选可以对应于这些角色进行活动。这些人是不是客户或是否在现场并不重要，重要的是这些人能澄清问题，并能尽可能快地处理问题，包括用能让开发团队明白的形式给出反馈。

Coding Standards: Have Them and Use Them
编码标准：有并且要应用

No one is going to argue against having coding standards. But what is *really* important? To use them! It doesn't matter what the standards *are* as long as everyone uses them. The RUP provides three coding standard examples to get you started: Ada, C++, and Java. As a complement to these coding standards, the RUP also encourages that you define architectural mechanisms, which standardize not only the language of the code, but also its structure and usage (error handling and transaction locking are examples of common mechanisms).

没有人认为不需要编码标准，但是什么是真正重要的？当然是运用标准。什么标准没有关系，只要每个人用它就行。RUP 提供了三种编码标准样本：Ada 、C++ 和Java，可以使开发者顺利启动。作为对这些编码标准的一个补充，RUP 鼓励开发者定义体系结构的机制，不仅要标准化编码的语言，而且要标准化它的结构和用法（错误处理和处理事务加锁是公共机制的例子）。

Summary
结束语

As you can see, there is significant agreement between RUP and XP on nine of the twelve XP practices. Typically, RUP provides complementary guidance for doing more to address specific risks.
　　如上所说，表明在十二个实践中有九个实践活动，RUP和XP 有显著的一致性。类似的，RUP 为对付特别的风险提供了补充指南。

As you evaluate both processes *vis ?vis* your next project, it's important to keep in mind this primary rule:
　　象你评估两种过程， 对于(Vis?Vis) 你的下一个项目，记住这些基本规则很重要：

Ask yourself, "If we don't perform a specific activity, produce an artifact, or adopt a practice, will anything bad happen?"
If the answer is "no," then don't do it!
　　问你自己，"如果我们不实行一个特定活动，生产一个产品，或采用一个惯例，将会有不利的事情发生吗？"
　　如果回答是"否"，就不要做。

Next month we will look at the three remaining XP practices and discuss additional pitfalls associated with XP practices. And finally, we'll examine what important areas XP does *not* address.
　　下一个月我们将研究三个其余的XP实践，并讨论与XP时间相关的附加陷阱。最后，我们将考查XP没有涉及的重要部分。

---

[1] See http://www.objectmentor.com/publications/RUPvsXP.pdf, a chapter from Martin's forthcoming *Object Oriented Analysis and Design with Applications, Third Edition*, from Addison Wesley.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!

Part II: Valuing Differences
第二部分：本质区别

In the last issue of The Rational Edge, we looked at the common ground between the Rational Unified Process (RUP) and eXtreme Programming (XP). They certainly have a lot in common. This month, in Part Two of our comparison, we look at the last three XP practices and at some areas of RUP not covered by XP.
　　我们在上一期《The Rational Edge》考察了RUP和XP的通用基本概念。知道两者之间有很多的相通之处。在本月，我们将对两者进行对比，同时考察XP实践剩下的三条以及RUP独有的一些地方。

There are three XP practices we deferred discussing until this issue. They are:

在本期将要讨论的三条XP实践是：

- *Small releases*
- *Collective code ownership*
- *Metaphor*
- 小版本发布
- 代码共享
- 隐喻

We will discuss each of these. But first, I'd like to point out that the set of XP practices we are talking about is the original twelve practices set forth by Kent Beck in his book Extreme Programming Explained: Embrace Change, *published by Addison-Wesley in 1999. As of March 15, 2001, there were several additional supporting practices listed on the Extreme Programming Roadmap page: http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap. This indicates the dynamic and somewhat experimental nature of XP, which is not necessarily a bad thing. Any process needs to be dynamic and keep up-to-date with proven best practices. At the end of this article we will also look at some ways RUP and XP can work together to provide a good experience for software development project members.*

我们将逐一讨论，但首先我要指出的是：我们所讨论的这些XP实践，最初来自Kent Beck的著作《Embrace Change》（1999年 Addison-Wesley出版）中的XP十二个实践。在2001年3月15日，"XP路标：http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap."的主页上，又补充了几条。这表明XP正在实践中发展，这是一件好事，因为任何方法都需要在实践中去检验和更新。在文章的最后，我们还将RUP和XP结合起来考察，并给开发人员提供一个好的实例。

Small Releases: How Small and Released to Whom?
小发布版本

What is a release? Depending upon how you answer this question, RUP and XP can seem quite similar in their concepts of a release. RUP defines a release as: "...a stable, executable version of product, together with any artifacts necessary to use this release, such as release notes or installation instructions."[1] Furthermore, according to RUP, releases are either internal or external. By this definition, a "release" creates a forcing function that ensures a shippable product, rather than a system that is only 80 percent complete. XP defines a release as "...a pile of stories that together make business sense."[2] In much of the discussion about small releases on some XP Web pages, the practice of small releases seems to coincide with the practice of continuous integration.[3] If you interpret the stories to mean the code as well as any artifacts necessary to use the release, and you accept the release as either internal or external, then the RUP and the XP concepts of a release are almost identical.

　　什么是发布？每人的看法都不一样，RUP和XP在这一点上看起来很相似。在RUP中，发布被定义为"...一个产品稳定的可执行版本，同时包括和这个发布相关的一些东西，例如发布说明和安装说明。"[1] 根据RUP的进一步定义，发布分为内部发布和外部发布。根据此定义，发布就是生成推出产品时的完整版本，而不是所谓的一个完整系统的80%内容。在XP中，发布被定义为"...能够体现事务而组合在一起的许多事例。"[2] 在XP的主页中有好多关于小发布版本的讨论，这项实践看起来和持续集成的实践联系紧密。[3] 如果你把事例理解为代码和发布相关的一些东西，同时同意发布有内部和外部两种，那么RUP和XP关于发布的定义就几乎是一样的。

RUP invites you to consider more than just code. A release, especially an external one to the customer, may prove useless unless accompanied by release notes, documentation, and training. XP addresses code and assumes the rest will appear. Since code is the primary artifact of XP, the others need to be derived from it. This implies certain skills that may not be obvious. For example, technical writers might need to be able to read the code to understand how the system works in order to produce the documentation.

　　RUP不仅仅局限于代码，特别是，如果给客户的外部发布中不包含说明，文档，和培训，那发布将是毫无价值的。XP包含了代码，并且假定其他内容是非常明显的。对于一个XP的制品，代码是最基本的，其它都是它的延伸。这就意味着有些必需技术并不是那么明显，例如，技术撰写者也许要有能力阅读代码并理解系统是如何工作的，这样才能写出相应的文档。

I have talked with several people who assume the frequent releases in XP are all to be delivered to an external customer. In fact, XP is not clear about this. In *Extreme Programming Installed,* the authors urge you to get the code into the customer's hands as frequently as possible.[4] The fact is, in many organizations customers cannot accept frequent software updates. You need to weigh the benefits of frequent delivery against the impact on the customer's ability to be productive. When you are unable to deliver a system to the customer, you should consider other ways of getting feedback, such as usability testing. On a RUP-based project, you typically deliver to the customer in the last construction iteration as well as in the transition phase iterations.

　　我曾经和一些人讨论过XP中的小发布版本，他们认为这些发布是给外部客户的。实际上，XP并没有明确说明。在《Extreme Programming Installed》一书中，作者强调要尽可能把代码交到客户手中。[4] 但很多的客户都不可能接受软件的频繁更新。你必须在快速发布的好处和影响用户使用产品的弊处间进行均衡。当你不能把产品交给客户使用时，你应该考虑其他途径发布小版本制品，例如使用测试的方式。在RUP的项目中，通常只是把

有重大改变的版本和最后的版本交给客户。

Collective Code Ownership: Yours, Mine, and Ours
代码共享：你的，我的，我们的

XP promotes "collective code ownership," which means that when you encounter code that needs to be changed, you change it. Everyone has permission to make changes to any part of the code. Not only do you have permission to make the changes -- you have the responsibility to make them.

XP鼓励代码共享，这意味着当你觉得有必要修改代码时，就修改吧！每个人都有权利修改代码的任何部分，不仅仅有权，也有责任这样做。

There is an obvious benefit with this practice. When you find code that needs to be changed, you can change it and get on with your work without having to wait for someone else to change it. In order for this practice to work, however, you need to also practice "continuous integration" and maintain an extensive set of tests. If you change any code, then you need to run the tests and not check in your code changes until all tests pass.

这样做的好处是明显的，但你发现代码必须修改时，你可以立即修改而继续你的工作，而不必等别人来做。当然，为使这条实践有效，你必须记住持续集成这条实践。一旦你修改了代码，你就必须测试，直到测试全部通过，否则不能提交你的修改。

But will collective ownership work everywhere? Probably not. Large systems contain too much content for a single person to understand it all at a detailed level. Some small systems often include code that is complex due to its domain or the function it performs. If a specialist is required, then collective ownership may not be appropriate. When a system is developed in a distributed environment, it is not possible for everyone to modify the code.

In these cases, XP offers a supporting practice called "code stewardship."[5]

With code stewardship, one person, the steward, has responsibility for the code, with input from other team members. There are no guidelines when to apply code stewardship instead of collective code ownership.

但是不是代码共享到处都可行呢？不大可能。对于大系统，一个人是没有能力理解所有的细节的。即使是小系统，也可能由于范围或它的功能的特殊，而变得很复杂。如果是专业系统，代码共享可能就不适合。当系统是分布式开发时，任何人都修改也不可能。在这些情况下，XP提供另一种实践"代码管理"。[5] 用这种方法时，有一人，即管理者，对代码负责，他从其他成员那得到代码。使用代码共享还是使用代码管理没有明显的标准。

Collective code ownership provides a way for a team to change code rapidly when it needs changing. Are there any potential drawbacks to this? If you consider all the ways code is changed, then there are some things you may want to control less democratically, in a centralized way -- for example, when code is modified because it lacks some functionality. If a programmer is implementing a story (or a use case, or a scenario), and requires behavior from a class, collective code ownership allows the class to be modified on the spot. As long as the system is small enough for a programmer to understand all of the code, this should work fine. But as the system gets larger, it is possible that the same functionality might be added to code that exists somewhere else. This redundancy might be caught at some point and the code refactored, but it is certainly possible for it to go unnoticed and for the functionality to begin diverging.

代码共享提供了一条快速修改代码的途径。那么有没有缺点呢？你可以考虑代码修改的各种情况，你就会发现有些情况需要用集中而牺牲自由的办法来控制。例如，当为增加功能而修改代码时，如果一个程序员为实现某个功能（或使用特性，或场景），需要某个类的方法，就需要改动代码的所有权以便修改。只要系统够小，程序员能理解所有的代码，这样做没问题。但是如果系统很大，有可能相同的功能别处已经实现了。这种冗余也许会被捕获从而进行代码重构，有时有可能因为新的实现和旧的不同而被忽略。

You may want to start a project using collective code ownership to allow your team to move quickly. As long as you have good code management tools and effective testing, then it will work for you -- up to a point. As a project leader or manager, however, you need to be on the lookout for the point when the code base becomes too large or too specialized in places. When this happens, you may want to structure your system into an appropriate set of components and subsystems and ensure that specific team members are responsible for them. RUP provides guidance and other help on how to structure your system.

你可能想使用代码共享而使你的团队开发加快。只要你有好的代码管理工具和有效的测试，这可以做的很好。但是作为一名项目主管要预见到代码可能会变得很大，有些地方有可能变得很专业。在这种情况下，你可能需要重构你的系统，分成合适的组件和子系统，让每个专门的成员各负其责。RUP也提供重构系统方面的一些指导和帮助。

System Metaphor: It's Like Architecture
系统隐喻：象一个体系结构

A metaphor is a figure of speech that allows us to make comparisons. It is one way that we learn: "A motorcycle is like a bicycle, but it has a motor attached." XP uses a system metaphor instead of RUP's formal architecture. This system metaphor is "...a simple shared story of how the system works. This story typically involves a handful of classes and patterns that shape the core flow of the system being built."[6]> Based on comparisons with familiar things, patterns help us understand new and unfamiliar things.

一个隐喻就是允许我们做类比的语言修辞。正如我们认为"一个摩托车象一个自行车，但是摩托车带有发动机"一样，XP用系统隐喻代替了RUP的形式上的系统结构。系统隐喻就是表示系统如何工作的一个简单的分配事例。这个事例包含一些类和模式，正是这些类和模式形成了将要建立系统的核心内容。[6] 通过与已经熟悉的情况的类比，这些模式帮助我们理解新的和不熟悉的事情。

And indeed, the XP system metaphor may be a suitable replacement for architecture in some cases, but usually only in small systems. For many, if not most, software systems, you need more than a simple shared story. How much more you need depends upon many factors.

并且事实上，XP的系统隐喻在某些情况下是RUP中的体系结构的一个最佳替代方案，但是通常都用于一些小的系统。而对一些软件系统，需要的并不是一个简单的分配事例，所需要的主要由很多的因素来决定。

By contrast, RUP is an architecture-centric process.[7] Architecture is more than a metaphor, although it may include several metaphors. Architecture is concerned with structure, behavior, context, usage, functionality, performance, resilience, reuse, comprehensibility, constraints, trade-offs, and aesthetics. It is usually not possible to capture all of this in a simple shared story. Architecture does not provide a complete representation of the whole system. It concentrates on what is architecturally significant and important in reducing risks.

与此相比，RUP是一个以体系结构为中心的过程。[7] 体系结构不再是隐喻，尽管它可能包含几个隐喻。体系结构所关心的可能是结构，行为，环境，用法，功能，性能，适应性，重用性，理解性，约束性，权衡以及美学等内容。通常，它不可能获取一个简单分配事例的所包含的所有情况，也并不能提供一个整个系统的完全表示。它注重于表明结构方面的重要意义和减少危险的重要性。

RUP provides a wealth of guidance on constructing and managing architecture. It helps the practitioner construct different views of the architecture for different purposes. [8] The different views are needed because there are different aspects that need to be highlighted and different people who need to view the architecture.

RUP提供了一个构造和管理体系结构的优良的指导。它帮助实践者为不同的目的构造体系结构的不同视图。[8] 这些不同的视图是必须的，因为，体系结构的不同方面需要被强调，以及不同的人需要理解体系结构。

A RUP-based project will address architecture early. Often an executable architecture is produced during the Elaboration Phase. This provides an opportunity to evaluate solutions to critical technical risks, and the architecture is built upon during subsequent construction iterations.

一个以RUP为基础的工程将预先发布体系机构。通常在详细设计阶段，产生一个可执行的体系结构。这样就可以有机会评估方案以评论技术危险，并且在反复的连续的构建中体系结构被实现。

An executable *architecture* is a partial implementation of the system, built to demonstrate selected system functions and properties, in particular those satisfying non-functional requirements. It is built during the *elaboration phase* to mitigate risks related to performance, throughput, capacity, reliability and other "ilities", so that the complete functional capability of the system may be added in the *construction phase* on a solid foundation, without fear of breakage. It is the intention of the Rational Unified Process that the executable architecture be built as an evolutionary prototype, with the intention of retaining what is found to work (and satisfies requirements), and making it part of the deliverable system.[7]

　　一个可执行的体系结构是一个系统的局部实施，在特定的满足非功能性需求的情形下，以演示所选择系统的部分的功能和属性。这些都是在详细设计阶段建立以减少关于性能，生产量，容量，可靠性等方面的危险，这样，系统的全部的功能将在构建阶段以一个坚实的基础上添加，没有被破坏的危险。RUP的目的是使可执行的系统结构以一个可进化的原型建立，并且使它成为可交付使用系统的一部分。[7]

What's Not Covered by XP That Is in RUP?
哪些是XP没有的而RUP具有的？

Your project may be able to use XP for developing the code. If not, then you may need to add some additional process, but just enough to reduce your risks and ensure that you are able to deliver the right product to your customers on time.
　　你的工程可能可以使用XP开发代码。如果不能，你可以加入一些额外的方法，但是这要能足够的减少危险以确信你能够将正确的产品及时的交付给你的客户。

However, when you look at a development project as a complete set of deliverables, code, documentation, training, and support, there are many things RUP addresses that are not considered in XP. Again, you need to determine whether they are needed for your specific project. The following list provides things you may need to consider. The list is not exhaustive. You can find additional information about these items in the Rational Unified Process.
　　然而，当我们将一个开发项目作为一个包含代码，文档，培训以及技术支持的完整的发布版本来看的时候，它们很多是用RUP完成的而不考虑XP。而且，你必须决定这些对于你的工程这是否是必须的。下面的列表列出了你必须要考虑的内容。这个列表并不是全面的。你可以在RUP中找到关于这些列项的其它信息。

- Business modeling. The whole subject of business modeling is absent from XP. Systems are deployed into an organization. Knowledge of the organization can be important when identifying the requirements and for understanding how well a solution might be accepted.
  事务建模．事务建模的整个主题并不属于XP。系统被安排一个组织来完成。辩明用户需求和理解一个解决方案被接受的程度对这个组织的认识是很重要的。
- Project inception. XP assumes the project has been justified and does not address how that justification takes place. In many organizations, a business case must be made before a project begins in earnest. RUP helps a team make its business case by developing stakeholders' needs and a

vision.
项目开端. XP认为项目被证明，就不能公布这个证明是如何发生的。在许多组织中，在一个项目开始前，一个事务必须被制定。RUP通过完成客户 的需求和视图表示帮助团队制定它的事务案例。

- Deployment. The whole area of system deployment is missing from XP. Any system needs supporting materials, minimally online documentation. Most need more. Commercial software products require packaging, distribution, user manuals, training materials, and a support organization. The RUP Deployment discipline provides guidance for practitioners on how to create appropriate materials and then use them.
  部署。XP中缺少系统部署。任何系统需要提供支持材料，最小量的在线文档。绝大多数的商业软件产品需要内容的更多，包括软件包，发布，用户手册，培训材料和技术支持材料。RUP实施原则为实践者怎样建立合适支持材料和使用手册方面提供指导。

Mix and Match for Best Results
最佳结果的混合与匹配

Process diversity is important.[9] One size does not fit all projects. The process you use for your project should be appropriate for it. Consider what your project needs and adopt the right approach. Consider all aspects and risks. Use as much as you need, but neither too little nor too much.
过程的多样性是很重要的。一种方法并不能适合所有的项目。你的项目所用的过程应该是适合这个项目的。这要考虑到你的项目的需求，并采纳合适的方法，还需要考虑的各个方面和危险性。尽可能用你必需的，即不能太少也不能太多。

RUP and XP provide two different approaches to software development projects. They complement each other in several ways. XP concentrates on code and techniques for a small team to create that code. It stresses face-to-face communication and places minimal effort on non-code artifacts. RUP is a process framework that you configure for different types of projects. It invites you to consider risks and risk mitigation techniques. RUP is often misinterpreted as being *heavy* because, as a framework, it provides process information for many types of projects. In fact, a configured instance of RUP may be very light, depending upon the risks that need to be addressed. It may incorporate some of the excellent techniques of XP and other processes if they are appropriate for the project at hand.
RUP和XP为软件开发项目提供了两个不同的方法。它们在某些方面互相补充。XP适用于小型团队开发，关注于代码和技术。它强调面对面的交流和在非代码方面倾注很少的努力。RUP是一个框架性的过程，你可以为不同类型的过程设置不同的框架。它让你考虑危险和减轻危险的技术。由于RUP作为一种框架为许多类型的项目提供过程信息，因此，RUP经常在这种重压之下被曲解。事实上，一个RUP的配置实例可能非常轻量级，这主要由发布的危险所决定。它可以融合XP的优秀技术和过程，这些过程对目前的过程是相辅相成的。

If my project is only about creating code, I may use just XP. However, almost all of the projects I work on require initial business decisions and planning, complete documentation, support, and deployment to customers. For this reason, I would more likely start with RUP and use the appropriate XP practices that will help my team move ahead quickly and mitigate real risks the project faces.

如果我做的项目仅仅是创建代码，那么我可以用XP。然而，我说做的绝大多数项目需要最初的商业决定和计划，完整的文档，技术支持和用户的部署。正因为这个原因，我更愿意使用RUP，用适当的XP实践去帮助我的小组进度更迅速，还能减少项目所面对的危险。

As a software engineer, I try to have a well-stocked toolbox of techniques, processes, and tools that help me succeed. I'm glad to have both RUP and XP as part of my collection. More techniques in my toolbox means that I can provide better value to my project and my organization. In addition, as a project manager or process engineer, I can create an instance of a process for a project that addresses the organization's need for controls while providing individual project members with an environment that can be fun and satisfying.

作为一个软件工程师，我将试着使用各种技术、过程和工具组合在一起的工具箱帮助我成功完成项目。我很高兴将RUP和XP作为我的工具箱的一部分。在我的工具箱中的很多技术都能为我的项目和团队提供有益的价值。此外，作为一个项目经理或者过程工程师，我能够为我的项目创建过程实例，以支持团队的发布控制要求，并且为团队成员提供一个有趣和满意的环境。

---

[1] Rational Unified Process Glossary.

[2] Kent Beck, *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley 1999, p.178.

[3] http://c2.com/cgi/wiki?FrequentReleases.

[4] Ron Jeffries et al, *Extreme Programming Installed*. Reading, MA: Addison-Wesley, 2000, p.50.

[5] http://c2.com/cgi/wiki?CodeStewardship.

[6] http://c2.com/cgi/wiki?SystemMetaphor.

[7] This is described in Philippe Kruchten, "The 4+1 View of Architecture." *IEEE Software*, November, 1995.

[8] This definition is taken from the Rational Unified Process glossary.

[9] For more information on process diversity, see Mikael Lindvall and Iona Rus, "Process Diversity in Software Development." *IEEE Software*, July/August 2000.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!

?
