

# RUP 实施之夺命七招

Rational 统一过程(Rational Unified Process, RUP)提供了一个极有价值的软件开发业务框架，它正在成为一个广受欢迎的当代软件开发过程的事实标准——它整合了公认的最佳实践，例如适应性的、迭代的和风险驱动的开发模式；它是由在大、小型系统开发中均具有丰富经验的世界级领导者设计的；它在应用和扩展上都很灵活，而且被正式的出版物以及相应产品很好地记录了下来。然而作为框架，必须根据每个项目团队及其环境的需要进行调整。RUP 本来是用一种轻型和敏捷的方式来开发项目的，而不是被当作一个“万能尺码”的开发过程。若干因素妨碍了 RUP 的成功实施，其导致的结果往往非常糟糕。

本文用略带一点俏皮的口吻，与大家分享一组常见的 RUP 应用陷阱。如果您的目标是让 RUP 实施的结果一塌糊涂，那么我们向您推荐以下七招。

## **第一招： 在 RUP 之上叠加瀑布思维**

您的开发过程是否有点像：(1)企图定义和稳定绝大部分的需求，然后进行签署；(2)基于需求，进行详细设计；(3)基于设计，进行实现；(4)进行集成、系统测试和部署。这是一个线性的、串行的瀑布型生命周期的典型例子，而且是一个最优先的、最棒的，也是最常见的让 RUP 实施完全失败的策略。如果您的开发过程看上去与这很像，

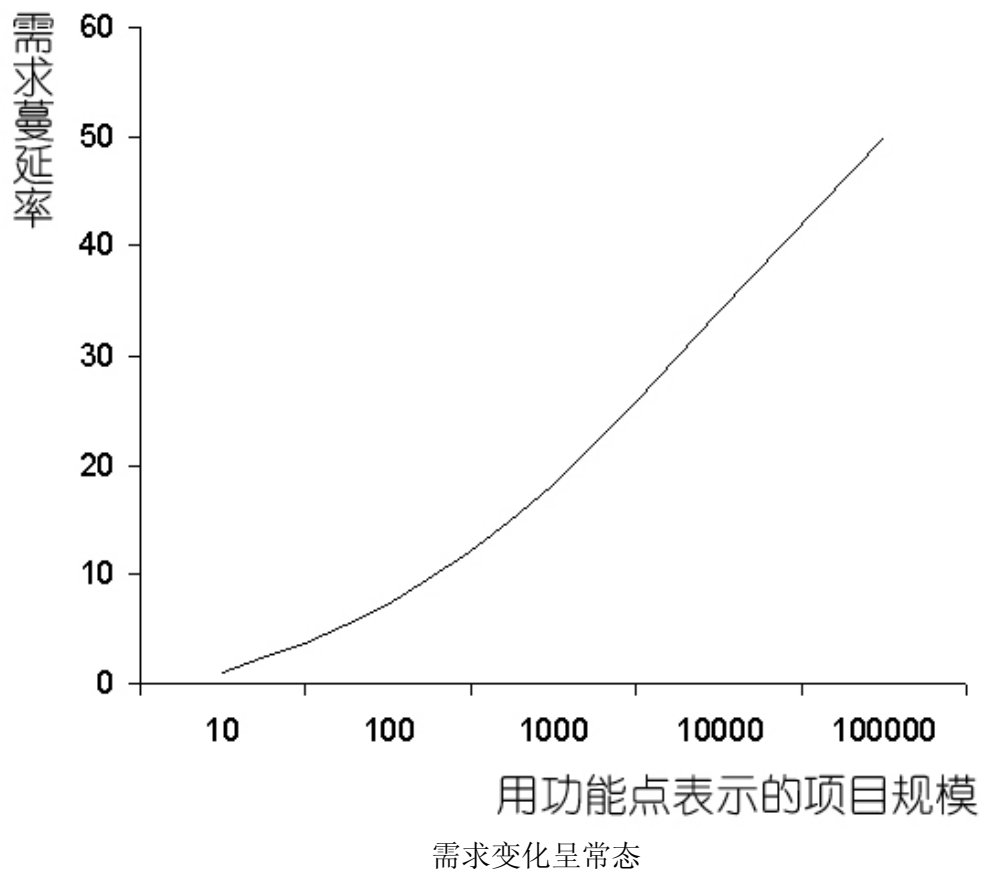
那么祝贺您，您成功地没有采用 RUP！

关于瀑布型生命周期与迭代式开发最重要的一件事是，我们在 20 世纪 60 年代和 70 年代被误导了：许多人当时接受到的教育是瀑布式做法是聪明老练的。然而，历史研究表明，这一建议的出现并没有得到任何统计意义上的证据的支持，而且更为重要的是，当前的软件项目失败研究结论性地表明，瀑布型是风险最高、极易失败、低生产率以及高缺陷率的软件构建方法。如果您要一个软件项目失败，您应该遵循瀑布型做法！而与之相反，现在有极具说服力的证据表明，迭代和演进式方法(如 RUP)能带来较低的失败率、更低的风险和更高的生产率。值得称道的是，美国国防部原本提倡瀑布过程和观点，在发现那么多采用了该方法的失败之后，不但放弃了对它的要求(如 1988 年的 STD-2167A 标准)，而且从 1994 年的报告开始，积极地鼓励采用更加现代化的迭代式生命周期来取代瀑布型做法。

从发展的角度，瀑布型过程与 20 世纪 60 年代开发软件的随心所欲方式相比，它是一个相对合理的策略。这一做法借鉴了其他领域的工程和建造，但遗憾的是，未经对其应用于软件开发的真正适应性进行认真和严肃的研究，它就被广泛地传播和采用了，于是几代师生都不假思索地学习、不断地照搬它。不幸的是，如今仍有许多软件过程“专家”、咨询顾问、项目经理和过程工程顾问(例如某些带有瀑布型偏好的 CMM 实施顾问)似乎一点儿不知道已有的研究证据，仍然继续根据自己的主观意愿或信念而非统计意义上的证据来行事。

有些东西必须像建筑那样被建造，然而人们发现软件通常不属于

这一类。这有许多原因，最具有说服力的是一个错误的假定，即可以在项目的第一个阶段中定义绝大部分的需求。Capers Jones 等人的研究粉碎了这一神话。如图 1 所示，在这项含有 6700 个项目的庞大研究中，蔓延的需求(在项目开始时没有预见到的需求)是软件开发业当中非常显著的一个事实，在普通项目中它大概占到了 25%，而在大型项目中则占到 50%。



瀑布式价值观和做法是：(1)完成绝大部分的需求，隐含着假定那些以前没见过该产品的用户们能很好地定义这些需求；(2)根据能够精确地为一个定义很差的问题制定解决方案这一假设，进行详细设计；(3)尽管设计还没有被证实，而且往往不可能被证实，仍然进行系统的实现；(4)集成、测试和部署。

瀑布型竭力回避需求变化的现实，它假定需求和设计能够正确地被指明和冻结，这与项目的现实严重不符。软件开发在设计和实现之前无法固定需求有许多原因，但不管什么原因，高明的应对方法不是去“对抗变化”，竭力固定需求，而正相反，应像 Kent Beck 积极主张的那样“拥抱变化”，并把这当作软件过程的一个核心驱动力。

于是，在 RUP 当中，开发的行进表现为一系列的迭代，每个迭代都被时间盒限定在一个固定的工期(例如正好 4 周)当中，每次迭代结束时都将产生最终系统某个子集的一个稳定内部发布。在迭代式开发中，时间盒限定是一个关键的概念：它意在固定迭代的结束日期，而且通常不允许结束日期的推迟。如果无法满足所有的目标，那么就要从迭代中去掉一些需求，而不是延长当前迭代的工期。

在一个迭代中，有一种类似微型瀑布的情况。首先挑选一小部分需求，相对全面地对其进行分析；用几天时间进行设计，然后迅速地对系统的这一部分开始实现、集成和进行实际的系统测试与压力测试。每次迭代的结束将产生一个可运行的部分系统，它能产生反馈，并引发未来迭代中对需求和设计的调整。随着时间的流逝，这些反馈-适应迭代周期揭示了一组合适的需求和一个健壮的、经过验证的设计与实现。这里，一个串行的瀑布方法在周(而不是月或年)的尺度上得到了应用，而且存在一个有机的反馈和适应机制。在一个数周的时间尺度上，一个串行的生命周期是可行的，然而当迭代长度增加时，这种方式将不再有效。

即使到了今天，在第一轮警报响起的十多年之后，仍有不少咨询

公司、管理者、教师和作家们把瀑布型生命周期或观点当作一门优越的技术来提倡。克服有意识或无意识地在 RUP 和迭代式开发之上叠加瀑布价值观，是一个项目团队面临的最大挑战之一。真正掌握了 RUP 的一个最深刻的变化不在于诸如写用例、对象建模等技能上，也不在于在学习 RUP 提供的许多工件和活动上(所有这些是可选的)，而在于对瀑布思维的根本态度和期望的转变。下面是 RUP 事实中四个典型的错误。

### **错误一：在 RUP 的阶段上叠加瀑布型阶段**

RUP 开发周期由四个阶段组成(起始、细化、构建和移交)，导致 RUP 实施失败的最常见策略就是把这四个阶段与瀑布阶段(需求、分析设计、实现和部署)直接等同起来。在此对 RUP 的四个阶段作一简短描述。

1. 起始：制定系统的业务依据；探查一小部分(大概占 10%)但很重要的需求，以便对项目范围的大致规模和关键风险有比较准确的把握，并决定是否值得为细化阶段拨付资金。

2. 细化：迭代地构建核心架构，消除项目的主要技术风险。所谓的构建架构，是指真正地编程、集成和测试——决不是纸面上的练习或可抛弃的原型。为了达到这一目的，可能需要迭代地详细探查绝大部分的需求(可能占到 80%)，并同时并行地实现系统中有风险的核心部分。在这一阶段中，通过反馈-适应周期不断地评估部分实现，需求则可能发生显著的变化。请注意与经典的瀑布式需求定义不同，RUP 的绝大部分需求是在与开发核心架构的工作并发的过程中精化

的，而且从实际的开发当中获得了反馈。在此阶段，还要决定是否项目的最终完成拨付资金。

3. 构建：迭代地构建在细化阶段尚未完成的内容，进行迭代集成和质量保证，为部署作准备。在这一阶段需求变化较少，因为大部分需求的不稳定性已经在前面阶段得到了迭代式的澄清。

4. 移交：完成 Beta 测试，解决遗留问题并部署系统。

### **错误二：迭代不是太长就是太短**

如果把计划的平均迭代长度定为 6~12 个月，在绝大多数情况下是很不适宜的。RUP 有一条惯例：“在理想情况下，一个迭代的工期应该是 2~6 周。”迭代式开发与 RUP 方法的精华在于通过小规模的工作步骤逐步获得一个可能不完善的实现，接着快速集成，进行 QA 和测试，快速获得反馈，然后基于该反馈调整需求、设计和实现。短小的步骤、反馈和调整是关键的思想。过长的迭代则达不到这种效果，因而是使 RUP 实施失败的一个良好策略。

### **错误三：在设计前磨亮需求**

迭代式方法允许有指导方向的试验和创造，而瀑布方法迫使我们从一开始就必须做到聪明绝顶，这种在没有反馈的情况下，把绝大部分需求定义并稳定下来的想法与软件开发的现实不符。

假设用瀑布方法建议我们购买软件的方式来购买衣服，那么，我们必须要在看不见能得到什么的情况下来确切地描述我们需要什么，甚至还不能先试穿一下，看看穿上这件衣服效果如何。我们必须要在真正收到这件衣服的几个甚至几年之前，精确地指明每一个测度。

一旦我们改变了主意，或者长胖了、变瘦了，那只有老天来帮忙了。不错，这恰恰就是瀑布方法的解决之道——在设计或实现之前确定绝大部分的需求。如果我们连买衣服都不想用这种办法，那么，凭什么让我们相信它对软件开发也是有效的呢？

#### **错误四：项目刚开始就期望获得可信的估算和详细的计划**

一位石油业高层主管对您说：“我听说 FooBarKhan 那里有石油，我太想要了！请您赶紧用一周的时间写一份报告，告诉我那里有多少石油，在那里建油田需要花多少钱、多少时间、多少资源。”在石油业中，这样的场景是很滑稽的，为什么同样的做法反而被软件工业接受了呢？理智的石油人士知道：没有预先在试验性钻探上大量投入是不可能获得答案的。只有通过调查发现了储量和地质情况的真相，作出估算或初步的计划才有可能。可是，在同样的不确定性和高复杂度的情况下，我们却奢望没有投入实际的“试验性钻探”调查就能对软件项目进行估算和计划，这不是很可笑吗？

仅仅因为我们能够作出计划显示出我们正取得进展，并不意味着我们真的能够做到。在没有获得任何关于未完成工作量的真实信息的情况下，在日程表上填上精确的日期是容易的。如果我们依赖完美的计划来交付一个项目，那么失败将是注定的。迭代方法允许我们边干边学，随着迭代的进行，我们获得了有关真实需求的更多信息，对真正的风险有了更好的了解，对我们应对项目的各项挑战的能力也有了更为清晰的认识。

有时固定价格交付要求的存在被当成瀑布方法的一个理由，因为

在没有真实信息的条件下作出决策也是合理的。迭代方法对初始的估算可能并没有什么改善，不过您能很快知道您做得怎么样，然后您可以更加容易地选择必要的时机启动协商，设定干系人的期望和管理项目范围。正如 RUP 所倡导的，一个处理固定价格投标与合同的理智方法是一种两阶段的策略。在第一阶段中，针对初始短暂(比如 4 周)的固定时间和固定价格签订合同，以便开展实际的调查和试探性编程，以产生更加成熟的范围、风险和需求规约。这一经改进的规约随后可被用作第二阶段(完整的开发)投标的基础。第一阶段调查的投入并没有浪费，其结果将节省第二阶段的工作量，并有助于产生第二份(最终的)更加符合实际的合同。

## **第二招： 把 RUP 当作一个重型的、先断性的过程来用**

重型或轻型、先断性或适应性的过程常常被人提及，而一个“重型过程”的称谓通常具有贬义，它具有以下性质：刚性和控制；许多活动和 RUP 工件是在一种官僚主义的氛围当中创建的；大量的文档；细致入微的长时间的详细计划；在基本工作之上存在大量的过程开销；以过程为本，而不是以人为本；以一种机械的方式把人视为可插拔的部件；先断性而非适应性。一个先断性的过程企图在一段相对长的时间段里(比如几乎在项目的整个周期中)计划和预测活动与资源(人员)的分配，其价值观隐含着对瀑布型的偏好。

相反，一个轻型或敏捷的过程意味着“精简与平实”，它具有如下特点：除去了所有不必要的官僚主义的过程开销，尽量减少创建低



价值或无意义的文档；专注于工作环境中人的本性之现实，让软件开发成为一种乐趣；它是适应性的。

为了促成 RUP 实施的失败，请采取以下重型、先断性的做法：

- I 对整个迭代项目进行详细的计划。从一开始就定义所有迭代的编号和日期，规定每个迭代中将要发生的事情。
- I 创建绝大部分甚至所有的 RUP 工件。
- I 增加大量项目和过程的正规程式，甚至建立几个项目委员会。
- I 在项目中追求一种机械的时钟驱动感，把人当作整个项目机器中的专用齿轮。

把 RUP 设计成一个重型或先断性的过程并非其创始人的本意，造成这一假象主要是由于人们误添了错误的过程观点或对 RUP 本身有误解，而 RUP 产品提供的庞大的详细过程文档又加剧了错误的印象，极容易导致 RUP 被错误地实施。RUP 作者们的本意是鼓励人们以一种轻型的、敏捷的、适应性的过程实质来运用它。比方说：

- I 应该创建 RUP 活动和工件的一个最小集，即只包含那些真正有附加值的東西；随着项目的进展，如果任何的过程开销活动没有附加值，那么就果断地抛弃它。
- I 对所有的迭代，都不预设详细的计划。有一个高层的计划（叫作阶段计划）估计项目的完工日期和其他主要的里程碑，然而它并不详细指定通向这些里程碑的路径。一个细化的计划（叫作迭代计划）只是提前对一个迭代（比如下一

个两周的迭代)进行较细致的计划。从一个迭代到另一个迭代的详细计划是适应性地完成的。

### **第三招： 忽视对象技术技能**

RUP 的直接目标在于开发面向对象系统。多年来，我们观察到许多对象技术项目失败或遇到严重的挫折，一个普遍的问题是缺乏真正地能以对象方式思考，熟练掌握对象设计、对象模式和面向对象编程的人员。拥有技艺精湛的 OT 开发人员是一个绝对优先的关键成功因素，而采用 RUP 或其他过程则相对次要。正如 Grady Booch 所言，“人远比过程重要”。

因此，如果要想让 RUP 实施真正失败，就可忽视聘用或培养那些拥有出色对象技能的人员。真正有实效的 OT 培养并非指先有一个星期的 Java 技术课程，然后是一个星期的面向对象分析设计课程，而是需要向软件工程师提供半年内大至八周的、有老师精心指导的培训，之后还需要一年左右的专家辅导巩固期。

对象技术开发技能绝非小菜一碟，成功的对象设计与编程需要受过良好训练的开发人员。确保项目失败的绝好办法可以是：不必大量投入来聘用或培养熟练的对象技术人才，或者让技能生疏的工程师勉强过关以减少这方面的投入。我们还要强调加强对象设计与设计模式技能和培养的必要性，不能仅仅关注对象编程。

## 第四招： 贬低适应性、迭代式开发

在 RUP 的核心理念和最佳实践(管理需求、迭代式开发、控制变更、校验质量、基于架构和构件的设计)中，迭代式开发的地位是最突出的。对于正从瀑布式价值观和实践上开始转变的组织而言，迭代式开发就像一场革命。事实上，在对待开发软件思考的许多层面上，如果向 RUP 转变的一个组织没有经历一场深刻甚至痛苦的变革，那么通常说明他们没有“把握”迭代式开发并真正地采用它。

### 错误一：信奉刚性和先断性观点

这种观点的含义即企图冻结需求和设计，而非拥抱变化；试图计划未来所有的迭代，而不是适应性地调整。瀑布模型对如何处理变化存在着尤为明显的偏见。在某种程度上，整个瀑布方法可以被视为大体上是反对变化的——一旦需求被“批准”(冻结)，即便后来发现某个需求是错误的，仍然需要特别费力地改变它。

### 错误二：颠倒迭代式开发的做法

项目经理似乎喜欢瀑布型生命周期那种表面上“明显”的稳定性和确定性，问题在于它们具有欺骗性——只有通过测试，才能评估对事物真伪的预先假定；只有通过实现，才能知道一个任务到底需要多少工作量。在项目当中，我们其实都在玩一个游戏——我们作出假设，猜想事物会是什么样子，解决方案将如何。然而事实上，我们的工作是基于不完善的信息。

聪明的项目经理通过设置许多检查点来验证先前的假定：布置了收集信息的任务，想方设法地抓住各种机会，利用新获取的信息来调

整计划。迭代方法允许人们收集信息，通过迭代来改进计划，甚至改变方向。瀑布方法最大的错误可能在于，计划的制定脱离了现实，而且即使当收集到新的信息时，这种计划也不能被更新，改变计划被认为是预测未来行为的一种失败。然而调整计划并非预测的失败，相反是一次改进的机会。

### **错误三：没有让干系人了解迭代式开发的意义**

客户认为转交了项目的需求，他们就可以袖手旁观，直到完整的产品被交付。管理者也已经习惯性地认为，他们可以期待在项目的第一天就能制定完美的计划，然后按部就班地执行，而且几乎不需要什么调查、试验编程或概念验证，就可以获得可靠的估算。如果一个项目失败了，他们会认为，不是因为计划错误，而是因为计划没有得到执行，或者是由不称职的计划或估算人员造成的，而这种思维恰恰是让管理者招致恶名的一个原因。

为了让迭代开发发挥作用，客户必须参与其中。迭代式开发的精髓是根据反馈进行及时调整，而不是预先揣测。如果客户不感兴趣，不积极参与以确保构建了他们要求和需要的系统，那么他们必将自食其果。软件开发最难的部分在于构建“对”的东西，让功能和易用性真实有效。开发人员很少是领域专家，他们需要获得帮助来理解解决问题需要些什么，以及如何来构建“对”的系统，而瀑布方法剥夺了开发团队与客户之间有意义的交流。客户应该在定义需要开发些什么的任务当中扮演积极的角色。最佳的做法是在开发人员与用户/客户的协作之下，从理解需要解决的问题开始，并以一种演进的方式来探

查问题(包括机遇和挑战)。迭代方法提供了对于这类开发来说非常关键的适应性反馈。

#### **错误四：过度建模并创建了很多低价值的工件**

RUP 是一个能够解决许多不同问题的框架。然而，您不太可能在项目中使用 RUP 全部的内容。迭代式开发的观点是为了获得实际的反馈，当只获得一部分需求或设计时，就尽快开始编程，而不是停滞于对需求和设计的百般揣测。因此，一种有效的让 RUP 和迭代失败的方法是创建所有的 RUP 文档，并试图用最大的精度来细化它们，在编程之前至少画上 N 页的 UML 图。

RUP 就像一个药箱或一家药店，我们大部分人可能不会走进一家药店，每一种药都买上一些然后全部服下，我们知道要谨遵医嘱，只服用我们需要的药。有人抱怨说 RUP 太庞大了，这就好比说药店里的药太多了。真正的问题在于：人们需要更好地了解如何知道自己需要些什么。把风险缓解作为判断依据是确定 RUP 究竟应该用多少的一个好办法：理解您面临的问题，然后挑选能够帮助解决这些问题的技术(药)。

### **第五招： 回避那些真正懂得迭代式开发的顾问**

有一些所谓的专家顾问并没有真正领会迭代式开发以及 RUP 对瀑布型价值观和先断性做法的根本抛弃。他们所传达的 RUP 信息是错误的，而且通常夹带着瀑布思维。

为您的第一个 RUP 迭代项目组建这样一个团队：他们从来没有从

事过基于短时间盒的适应性迭代式开发，尤其是那些项目领导者。找到那些执著于瀑布型观点和做法的人。坚决不要与知道如何进行迭代式开发的顾问签约。如果一位经验丰富的顾问不幸参加了这个项目，保证他只能扮演一个配角，对他的建议进行多番辩论、严厉质疑，最终让项目领导拒绝他的想法，而且还要不断奚落他。一定要找这样的人——他们曲解 RUP，不露声色地叠加先断性的瀑布观点，例如，认为细化阶段就是为了详绘模型，然后在构造阶段进行实现，或者在项目一开始就计划和分配所有迭代的工作。

## **第六招： 大张旗鼓地实施 RUP**

如果可能的话，在某一天向整个组织介绍 RUP，然后要求所有的项目第二天就照办而不告诉任何人实施的细节。如果这条办不到，那么就对组织进行强制教育，而且只培训那些开发人员，而不包括经理或 IT 执行主管。如果所有人都必须参加 RUP 学习，那么尽量在 RUP 实施前 6 个月这么做，否则人们就会全然忘了它，然后只好再找一位 RUP “导师” 提供短期的 1 天培训来强化瀑布理念。如果必须在培训后马上实施 RUP，那么至少应该在全组织内的所有项目上进行实施，同时让所有人立刻切换过来。

千万不要这样做：在一位有经验的教练指导下，通过一个小型的示范项目，尝试采用一批少量的、简单的 RUP 做法，从实践中学习，逐渐地增加实施内容，在取得第一个项目的基础上再启动第二个。如果某人提出相反意见，就立刻解雇他。请出那些怀疑 RUP/迭代、支

持瀑布型的人，让他们来负责管理 RUP 实施项目。

## 第七招： 误解清单

为了确保取得 RUP 实施中的彻底误解和失败，我们向您提供以下检查清单(评分表)。得分越高，RUP 的实施也就越失败：

- I 您认为起始阶段 = 需求；细化阶段 = 设计；构造阶段 = 实现。
- I 您认为细化阶段的目的是为了完整细致地定义模型，并在构造阶段将它翻译成代码。
- I 您认为在细化阶段只需创建原型(事实上，应该在细化阶段针对高风险的架构元素编程实现具有产品级质量的核心子系统)。
- I 您试图在开始设计或实现之前定义绝大部分的需求，或者在开始实现之前完成绝大部分的设计。
- I 您认为一个合适的迭代长度只能以月为单位来计算，而不是周。
- I 您认为编程之前的 UML 绘图和设计活动阶段是为了完整地、精确地定义极其详尽的设计与模型，认为编程只是简单、机械地把模型翻译成代码。
- I 您企图从头至尾详细地计划一个项目，然后把工作分配到每一个迭代；竭尽全力地企图预测所有的迭代以及在每一个迭代中发生的情况。

- I 一个组织在进入细化阶段之前,就要求获得可信的计划和估算。
- I 一个组织认为实施 RUP 就意味着从事尽可能多的活动或创建大量的文档,把 RUP 当作一个有许多步骤需要遵循的规范过程来运用。

我们相信遵照并运用了以上的夺命七招,您的 RUP 和迭代开发项目实施必将一塌糊涂。

## **作者简介:**

Craig Larman , 国际著名的对象技术顾问和软件过程专家, Val tech 咨询集团首席科学家。其经典教材《UML 和模式应用》和《敏捷与迭代式开发:管理者指南》已被翻译成多种语言在全球工业界和软件院校当中被广泛采用。