HZ BOOKS

PRENTICE HALL

IBM PRESS

IBM RSA和UML可视化建模指南

Visual Modeling with IBM Rational Software Architect and UML

(美) Terry Quatrani
Jim Palistrant 著

许杰星 苏敬凯 金振林 等译

机械工业出版社
China Machine Press

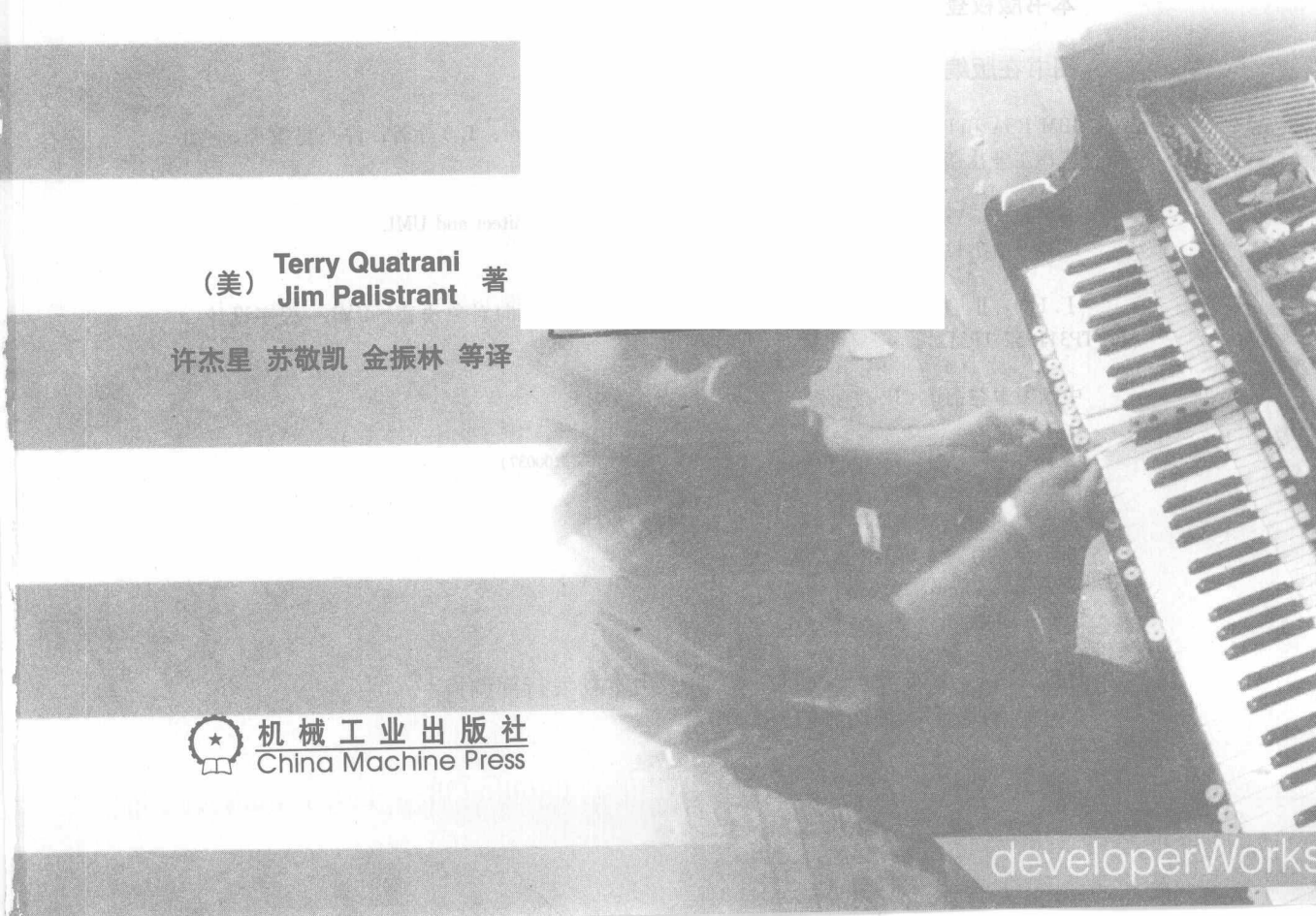
developerWorks

IBM RSA和UML可视化建模指南

Visual Modeling with IBM Rational Software Architect and UML

（美）Terry Quatrani
Jim Palistrant 著

许杰星 苏敬凯 金振林 等译



机械工业出版社
China Machine Press

本书以通俗、简洁的语言介绍可视化建模的方法，具体讲解如何使用 IBM Rational Software Architect 进行基于 UML 2.0 的可视化建模。全书以一个大学课程注册系统为例，通过一步步的操作，让读者学会利用 RSA 技术进行分析、设计和实现的方法，学会通过一种过程、一种语言和一个工具创建自己的软件系统蓝图。

本书适合软件开发人员、架构师、项目管理等参考。

本书版权登记号：图字：01-2007-1794

图书在版编目（CIP）数据

IBM RSA 和 UML 可视化建模指南/（美）奎特尼（Quatrani，T.）等著；许杰星等译. －北京：机械工业出版社，2007.6

书名原文：Visual Modeling with IBM Rational Software Architect and UML

ISBN 978-7-111-21555-4

Ⅰ.I… Ⅱ.①奎… ②许… Ⅲ.①软件开发 ②面向对象语言，UML—程序设计 Ⅳ.TP311.52 TP312

中国版本图书馆 CIP 数据核字（2007）第 076515 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）

责任编辑：刘立卿

北京京北制版厂印刷·新华书店北京发行所发行

2007 年 6 月第 1 版第 1 次印刷

186mm×240mm·9.75 印张

定价：19.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

如果在看过了一些厚厚的 UML 书籍之后，你仍然不知道该如何在软件开发实践中使用 UML，那么这本书会对你有所帮助。

运用 UML 进行可视化建模，离不开软件开发过程，尤其是以 RUP 为代表的迭代式的增量软件开发过程。因此，结合软件过程来学习 UML 和可视化建模才是一种有效的方式。

本书以 Rational 统一过程(RUP)为线索，通过完成一个具体案例的分析、设计和实现的方式，向读者介绍如何使用 Rational Software Architect(RSA)这一工具来进行可视化建模。这是一本简单而实用的书，是将 UML 和可视化建模技术运用到实践中的一个很好的开端。

在本书每一章的最后，作者列出了 IBM developerWorks(开发者社区)上的一些值得一读的相关文章的地址。通过阅读这些文章，读者还能对书中所讲述的概念和问题进行更全面深入的学习和研究。

关于 UML 术语的翻译，很多译著都不统一。我们参照 RSA 6.0 中文版对本书中的 UML 术语和 RSA 的操作步骤进行了翻译，同时也注明了英文原文，以便于读者学习和对照。

参加本书翻译的除许杰星、苏敬凯、金振林外，还有史红伟、祝国志、张峰峰和周宝华。由于时间仓促，译者水平有限，翻译的错误和不妥之处在所难免，欢迎广大读者批评指正。

2007 年 3 月

序

当 Jim Rumbaugh、Ivar Jacobson 和我开始定义最初一版的统一建模语言(UML)时，我们说：UML 的角色是“对软件密集系统的工件进行可视化、规范定义、构建和文档化”，现在我们依然这样说。请注意我们对可视化的强调：可视化是为了对复杂结构和行为进行推断。通过选择正确的可视化，我们能够不受特定的实现语言和其他技术等细节的影响，从而使我们可以对这些模式和大量的决定了系统架构的其他设计决策进行规范定义、构建和文档化。

现在，“架构”是一个见仁见智的词。有些人认为它代表了那些注重“仪式”的软件过程的一个重量级的工件；而另外一些人认为它和设计没什么两样。然而，我认为软件架构代表了形成软件密集系统的那些重要设计决策；所有软件架构都是设计，但不是所有设计都是软件架构。此外，我的经验是每个这样的系统(软件密集系统)都有一个架构，尽管有些是有意识的，有些是无意识的。不过，每个成功的组织往往都会让其系统的架构增量地和迭代式地成长。

我很高兴与 Terry 一起工作了很多年。她对于使用 UML 进行可视化开发的重要性有着深刻的体会，并有丰富的实践经验。在本书中，她专注于随着系统成长的生命周期而对系统架构的不同方面进行可视化，从初始阶段的用例到细化和实现阶段的分析、设计、实现甚至测试用例。Terry 的风格总是那么直截了当、亲切自然并注重实效。抽象很难，可视化抽象同样是一个难题，而本书将指导你使用 Rational Software Architect 来完成这两件事。

Grady Booch

IBM 院士

前　言

当我开始写本书的第一版时，我认为："这相当容易……我以此为生嘛"。但是我错了。把日常所做的事情变成文字是我所做过的最难的事情之一。但是我坚持花了很多个夜晚和周末坐在电脑前，最终写出《Visual Modeling with Rational Rose and UML》一书。我必须承认，当我第一次看见自己的书摆在书架上时，我激动得都要发抖了。我还发现，在阅读书评时要有很好的心态。我的书很独特，因为有的人喜欢它(5 星)，有的人对它没什么印象(1 星)。由于某些原因，我很少得到这二者之间的评分。

我还体会到，写一本与某一工具相关的书就像是养育孩子，需要持之以恒地关心它。因此，我不断地润色、更新我的书，以使它能够与 Rational Rose 2002 中的一些新特性同步。

后来，随着开发的继续，事情又有所改变，一个全新的工具和进展的乐趣一起出炉了。而一个新工具就意味着又一本新书。这次我变得聪明了，我增加了一名合著者。就像我第一次写书时那样，Jim 也认为这一工作很容易。但他吃惊了！事情并非他想像的那样。

目标

尽管本书被评为截然不同的两种分数，但我并不打算作出什么改变。如果你喜欢我的其他书，你也会喜欢这本书，因为本书的目标没有改变，依然是对可视化建模领域的一个简单介绍。实际上，你甚至可能会更喜欢这本书，因为这次我们真正把它深入到了代码开发阶段。如果你对前两本书没有什么印象，你可能也会不喜欢这本书。我这样说是因为本书的目标没有改变，它不是一个完整的 UML 指南(Grady 和 Jim 写过

这样的书，我还不打算和这些权威专家竞争），它不是一个完整的 Rational 统一过程的指南（很快会有其他人来写这样的书），它甚至不是一本完整的关于代码开发的书。正如我所说的，本书的目的是对如何使用一个过程、一种语言和一个工具来为你的系统创建一个蓝图进行简单的初次介绍。

方式

本书采用了一种实践的方式来讲解可视化建模和 UML，使用了案例研究的方法来展示如何分析、设计以及（部分）实现一个系统。书中的应用是一个用于大学的课程注册系统，选择这个问题域是因为它容易理解并且没有针对任何一个计算机科学领域，这样你可以专注于领域建模的细节，而不必花时间理解一个不熟悉的问题域。

对待问题时，只要认真到足以能够给你带来可视化建模方面的实际练习和解决一个实际问题，这就足够了，不要完全按照现实来对待问题，以免陷入到细节的泥潭中。因此，要把很多有趣并且可能很必要的需求、考虑和约束放在一边，这样才能产生一个简化的、有用的、适合本书范畴的案例研究。

developerWorks 网站上的更多内容

访问 developerWorks 网站的 Rational 区（www. ibm. com/developerWorks/rational），可以找到很多关于可视化建模和 UML 以及如何将这些技术应用到应用系统中的更详细内容。在本书每一章末都列出了指向相应文章的链接。

内容概述

第 1 章“可视化建模简介”。该章介绍书中全程使用的技术、语言和过程，讨论可视化建模的优点、UML 的历史和所使用的软件开发过程。

第 2 章“开始一个项目”。该章简单介绍与本书全程使用的“课程注册系统”案例相关的信息。

第 3 章“用例模型”。该章讨论在以用例的方式检查系统行为时所使用的技术，展

示如何可视化地捕获和文档化系统的功能需求。

第4章“分析模型”。该章讨论在创建分析模型时所使用的技术。分析模型是系统实现的第一步，这是开始能看到将“如何”实现这一系统的第一个模型。

第5章“设计模型”。该章讨论在创建设计模型时所使用的技术。设计模型是分析模型的实现，也是实现模型及其源代码的抽象。

第6章“实现模型”。这里讨论的是用于创建实现模型的技术。实现模型从实现子系统和实现元素(目录和文件，包括源代码、数据和可执行文件)的角度表现了实现的物理组成。正如Jim不断提醒我的那句话：“事情就要不断有所进步。”

附录A“UML元模型”。该附录列出了UML 2.0对UML元模型做了哪些变更。

附录B“表示法汇总”。该附录用图形演示了UML 2.0的表示法。

致谢

我们要感谢很多人，感谢他们对于本书的内容、风格、表现形式以及编写所作出的贡献。

特别感谢下列人员：Grady Booch、Jim Conallen、Maria Ericsson、Kurt Bittner、Ivar Jacobson、Philippe Kruchten、Peter Luckey、Walker Royce、Jim Rumbaugh、Tom Schultz、Anthony Kesterson、George DeCandio、Rick Weaver、Eric Naiburg。我们还要感谢IBM出版社的Bill Zobrist、Mary Kate Murray和Chris Zahn，没有他们的帮助，本书就不会出版。

关于作者

Terry Quatrani是IBM公司的一位UML讲师。Terry在全世界宣讲Grady Booch、Jim Rumbaugh和Ivar Jacobson的可视化建模，她是《Succeeding with the Booch and OMT Methods》一书(1996)的合著者，并且是畅销书《Visual Modeling with Rational Rose and UML》(1998)、《Visual Modeling with Rational Rose 2000 and UML》(2000)和《Visual Modeling with Rational Rose 2002 and UML》(2003)这三本书的作者。这些书都由

Addison-Wesley 公司出版。

在为 IBM 公司工作之前，Terry 就职于 Rational 软件公司，在那里她就是 UML 讲师。她还曾就职于通用电气(GE)公司，作为一名程序员和分析师，是 GE 高级概念中心的元老之一。她刚开始工作时是新泽西州 Pennsauken 市的一名八年级数学老师。Terry 毕业于宾夕法尼亚州费城圣约翰大学，拥有数学专业理学学士学位。

Jim Palistrant 在 IBM 工作了近 25 年，其中大部分时间，他所从事的工作是开发和测试工具，他编程经验丰富，从汇编语言的系统编程到面向对象语言的编程，他都了解。1995 年，他开始从事 IBM 的 Java 和 Web 开发工具的工作，帮助将各种 IBM 的开发工具引入市场。他最近从事的工作是与 SOA 相关的工具。他拥有北卡罗来纳大学的信息系统学士学位和计算机科学的硕士学位。

目　录

第 1 章

可视化建模简介

可视化建模是一种思考问题的方法，它使用由真实世界的思想组织而成的模型来思考问题。模型有助于：理解问题、参与项目中每个人（顾客、领域专家、分析师、设计师等等）之间的沟通、为企业建模、准备文档，以及设计程序及数据库。建模能够使需求更易于理解，使设计更清晰，以及使系统更具可维护性。

模型是一种抽象，它过滤掉了非本质的细节，从而描绘出复杂问题或结构的本质，这样就能使问题更易于理解。抽象是人类的一种基本能力，它使我们能够处理复杂事物。几千年来，工程师、艺术家和工匠们已经建立了很多模型，为的就是能够在执行设计之前试验设计。软件系统的开发也不例外。若要构建复杂系统，开发人员必须抽象出系统的几个不同的视图，使用精确的表示法来构建模型，验证模型是否满足系统的需求，然后逐步增加细节从而将模型转换为实现。

为复杂系统建立模型，是因为我们不能整体地去理解这样的系统。在对复杂性的理解上，人的能力有限。在建筑领域中也可以看到这个概念。如果想在后院盖一个棚子，只要开始盖就好了；但如果想建一个新房子，可能需要一个设计图；如果要盖一个摩天大楼，必然需要一个设计图。在软件领域中也是这样。从几行源代码或者即便是用 Visual Basic 编写的分析表单开始，无法给程序员提供关于这个开发项目的全局视

图。建立模型能够使设计师专注于项目的总体视图从而了解组件之间是如何交互的，——而不必陷入到每个组件的具体细节中。

高度的竞争以及变动的业务环境导致了软件复杂性日益增加，这向系统开发者提出了极不寻常的挑战。模型能帮助我们组织、可视化、理解和创建复杂的事物。我们使用模型来帮助自己迎接当今和未来软件开发的挑战。

1.1 成功的三边关系

我曾经频繁地使用图 1-1 所示的三边关系来解释一个成功的项目所需的组件。表示法、过程和工具，所有这三个方面你都需要。你可以学习表示法，但如果不知道如何使用它（过程），那么你仍可能会失败。你可能有一个伟大的过程，但是如果不能在过程之间进行通信（表示法），那你同样会失败。而最后，如果不能把你的工作的工件文档化（工具），那么你仍会失败。

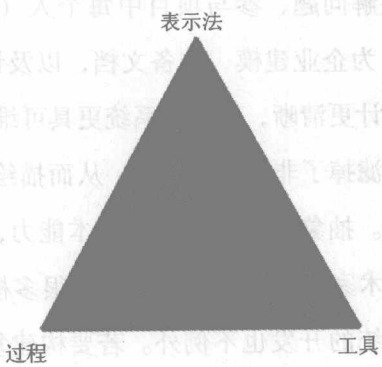


图 1-1 成功的三边关系

1.2 表示法的角色

在任何一个模型中，表示法都扮演了一个重要的角色，它是将过程组合在一起的粘合剂。表示法有三种角色：

- 作为一种语言，它沟通那些非显而易见的或不能从代码本身推论出的决策。

- 它提供了足够丰富的语法，可以捕获所有重要的战略和战术决策。
- 它提供了一种足够具体化的形式，可以满足人类推理和工具操作的需要[⊖]。

统一建模语言（UML）提供了一种非常健壮的表示法，它从分析延伸到设计。在分析期间，引入了表示法中的某些元素，比如类、关联、聚集和继承。而在设计期间，则引入了表示法中的其他元素，比如包含关系实现指示器和属性。

1.3 UML 的历史

20 世纪 90 年代，市场上涌现了很多方法和表示法。最流行的三种方法是 OMT（Rumbaugh）、Booch 和 OOSE（Jacobson）。每种方法都有其自己的价值和侧重点。OMT 的强项在分析，而弱项在设计。Booch 1991 的强项在设计，而弱项在分析。Jacobson 的强项在行为分析，而弱项在其他领域。

随着时间的推移，Booch 写了他的第二本书，其中吸收了很多 Rumbaugh 和 Jacobson 以及其他人提出的优秀的分析技术。Rumbaugh 发表了一系列的文章，其中接受了很多 Booch 方法在设计方面的优秀技术，这就形成了所谓的 OMT-2 方法。这些方法开始向一起集中，但是它们仍然各自有着专门的表示法。不同表示法的使用给市场带来了混乱，因为同样一个符号对于不同的人却意味着不同的东西。例如，一个实心环在 OMT 中是一个多重指示器，而在 Booch 中是一个聚集符号。你可能听说过“方法战争”这个词，它所描述的就是这段时期的争论：应该用云的形状还是用矩形表示一个类？哪一种表示更好？

直到采纳了 UML 作为表示法之后，方法战争才结束。“UML 是一种语言，用于规范定义、可视化和文档化所开发的面向对象系统的工件。它代表了 Booch、OMT 和 Objectory 表示法的统一，还包含了来自很多其他方法的最佳思想，如图 1-2 所示。通过对这些面向对象的方法所使用的表示法进行统一，UML 为在广泛的用户体验基础之上建

⊖ Booch, Grady. *Object Solutions*. Reading, MA: Addison-Wesley, 1996.

立面向对象的分析和设计领域的事实标准提供了基础。”[⊖]

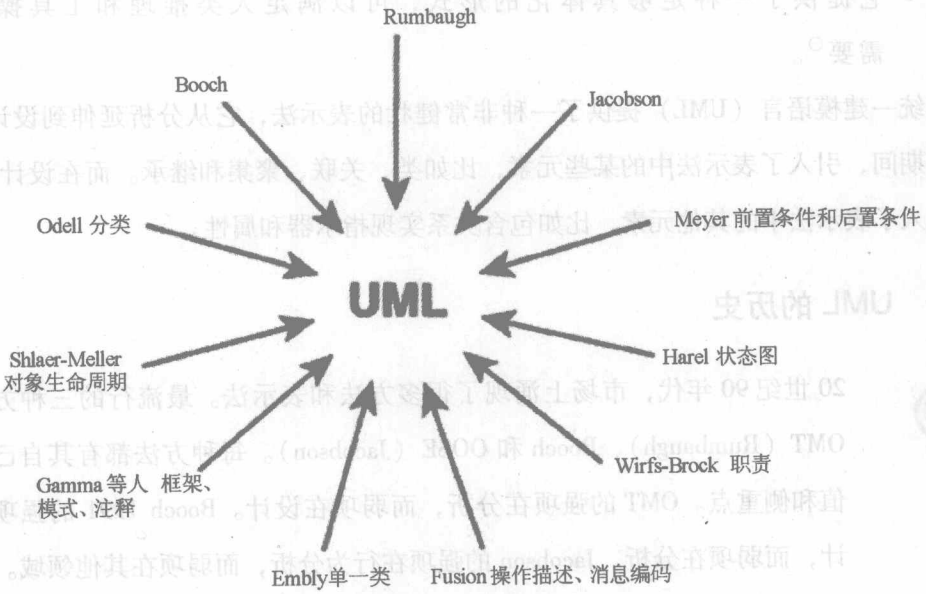


图 1-2 UML 的输入

UML 试图标准化分析和设计的工件：语义模型、语法表示法和图。它的第一个公开的草案（0.8 版）发布于 1995 年 10 月。在后续的两个版本（1996 年 7 月的 0.9 版和 1996 年 10 月的 0.91 版）中，包含了公众的反馈和 Ivar Jacobson 的思想。1997 年 7 月，1.0 版作为对象管理组（OMG）的标准而发布。UML 的 1.1 版得到进一步增强，它由 OMG 发表于 1997 年 9 月。在 1997 年的 11 月，UML 被 OMG 接纳为标准的建模语言。在编写本书时，UML 已被更新为 UML 2.0。这一工作分为 4 个部分：UML 2.0 超级结构、UML 2.0 基础结构、UML 2.0 对象约束语言（OCL）和 UML 2.0 图的交换。UML 2.0 超级结构（描述 UML 表示法的文档）已经被采纳，现在正处于最后的编辑阶段（不会对规范做出技术性的修改）。另外 3 个文档的工作也接近完成。读者可以通过访问 OMG 的网站 www.omg.org 来获取更多的信息。

⊖ *The Unified Method*, Draft Edition (0.8). Rational Software Corporation, October, 1995.

1.4 过程的角色

一个成功的开发项目应满足或超出客户的预期，按照及时和经济的方式开发，具有可变更和自适应的弹性。开发的生命周期必须能够促进创造和创新。同时，开发的过程必须受控和可度量，从而确保项目真正完成。“创造性是所有结构良好的面向对象的架构的根本，但若允许开发者拥有完全的无限制的创造性，就会导致项目无法结束。类似地，在将开发团队的工作组织到一起的时候，纪律是必须的，但是过多的纪律会滋生丑陋的官僚主义，从而扼杀了所有创新上的尝试”[⊖]。管理良好的迭代式的和增量的软件开发生命周期应该提供必要的控制，但又不会影响到创造性。

1.5 什么是迭代式的和增量的开发

在一个迭代式的和增量的开发生命周期（如图 1-3 所示）中，开发过程由一系列迭代构成，最后进展到最终系统。每次迭代（iteration）由下列过程组件中的一个或多个组成：业务建模、需求、分析、设计、实现、测试和部署。开发人员不需要假设在开发生命周期开始时已经知道了所有的需求；实际上，所有的阶段都已预计到了变更。

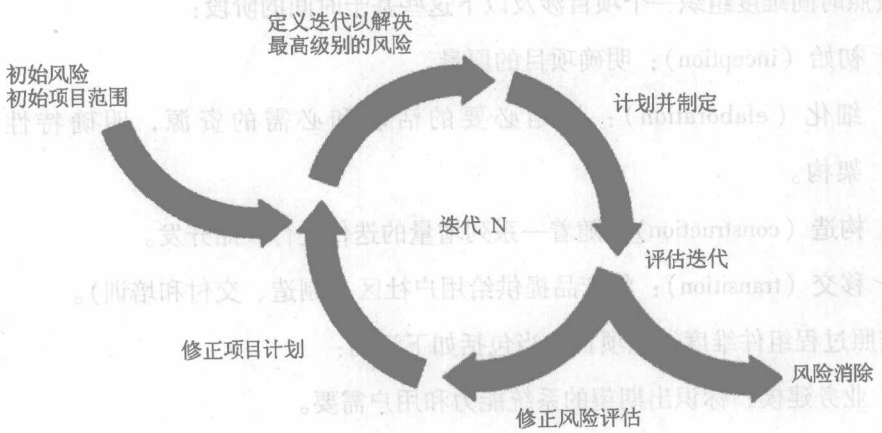


图 1-3 迭代和增量的开发

⊖ Booch, Grady. *Object Solutions*. Reading, MA: Addison - Wesley, 1996.

这种开发生命周期是一种缓解风险（risk-mitigating）的过程。在开发生命周期的早期，对技术风险进行了评估和优先分类，在每次迭代的开发过程中，又对技术风险进行了修正。风险被附加到每次迭代中，因此迭代的成功完成就缓解了附加到这次迭代上的风险。对于发布的进度加以计划也是为了保证首先处理最高级别的风险。按照这种方式来开发系统能够在生命周期的早期暴露和缓和系统的风险。采用这种开发生命周期方法的结果是风险更小且投资最少。⊖

1.6 Rational 统一过程

采用 Rational 统一过程可以对迭代式的、增量开发的生命周期进行控制。Rational 统一过程是一组广泛的准则，解决的是软件开发的技术方面和组织方面的问题，它专注于需求分析和设计。

在结构上，Rational 统一过程分成两个维度：

- 时间维度：将生命周期分为阶段和迭代。
- 过程组件维度：工件的特定集合与良好定义的活动的乘积。

为了使项目成功，对于这两个维度都要重视。

按照时间维度组织一个项目涉及以下这些基于时间的阶段：

- 初始（inception）：明确项目的愿景。
- 细化（elaboration）：计划必要的活动和必需的资源，明确特性并设计架构。
- 构造（construction）：随着一系列增量的迭代进行产品开发。
- 移交（transition）：将产品提供给用户社区（制造、交付和培训）。

按照过程组件维度组织项目应当包括如下活动：

- 业务建模：标识出期望的系统能力和用户需要。

⊖ 关于将迭代式的和增量的开发方法应用到软件开发中的更多信息，可以在 Philippe Kruchten 所写的《A Rational Development Process》一文中找到（*CrossTalk*，9（7），1996 年 7 月，第 11 ~ 16 页）。这篇文章也可以在 Rational 的网站上找到：http：//www. rational. com。