利用角色扮演和用例卡片进行需求复审

Robert Biddle 著，caorui 译

摘要

本文提出了一种利用角色扮演和索引卡片进行需求复审的方法，该方法能够更好地指导用例开发过程。这种方法建立在已经成功应用的面向对象设计技术——CRC 卡片技术之上。它使用索引卡片记录本质用例，同时利用角色扮演的方法对用例进行开发和复审。本文详细说明了这种方法，同时描述了这种方法的应用过程。

关键字

FA11 面向对象观点，FB0301 信息需求判定活动，FC28 面向对象分析

简介

我们已经探索出如何完善在面向对象开发方法的前期阶段使用技巧的方法。特别地，我们对如何更好地确定那些捕获需求并驱动设计的用例进行了研究。为了确认用例集合是否已经捕获需求，需要对它们进行复审，因此我们关注如何有效对用例进行复审。

用例描述了用户与系统的一系列交互。用例的概念已经在开发过程中得到普遍接受，在软件开发中寻找并使用用例已经是很寻常的事情。但正如面向对象分析方法的其它方面一样，要发挥用例的潜能，需要理解和经验。用例复审过程可以加深新手对用例的理解并增加他们的经验。

据我们所知，还没有专用于用例复审的技巧。我们寻求这样一种方法，它能保证团队成员的积极参与，提供一些操作性较强的指导，同时这种方法应该对于初学者和系统利益相关者开放。另外，这种技术应该是轻量级、完备而灵活的，以保证团队成员能够洞悉目标系统的行为，同时对系统有一致的理解。本文提出一种复审用例的方法，我们认为满足以上各点要求。该方法基于已经获得成功应用的 CRC 卡片技术，同时还使用了索引卡片和角色扮演。

本文是按如下方式组织的：下节提供一些背景知识，讨论面向对象开发中的用例及其作用；同时对我们应改编用于用例的 CRC 卡片设计技术做了一定的回顾。随后几节对方法本身进行描述，总结经验并解释本质用例的应用及推断。最后一节给出了结论。

背景

用例

Ivar Jacobson 等（1992）将用例定义为“与系统对话行为相关的一系列事务”。其基本思想是以用例来表达系统与外部世界间的一系列交互——甚至系统还没有构建出来。这种思想具有不可忽视的作用，有几个原因：

在前期开发阶段，用例可以有助于集中于交互，并借此引出需求或预期系统行为，从而捕捉需求并帮助确定规范。这种技术是十分有效的，因为可以按照容易唤起回忆或想象的形式描述交互，如同叙述或对话一样。

在其后的开发阶段，用例仍然很有帮助，因为它侧重于交互。在这个阶段交互就是系统必须满足的功能规范的具体体现。在设计和实现阶段，必须确认并建立一种能够满足这些规范的结构。在复审和测试阶段，用例可以驱动系统测试行为。

用例以一系列交互为基础，而系统需要的交互通常遵循一致的结构——即在一定程度上面向同一个目标或子目标。这就允许对需求规范分类，这是十分有益的。这种分类对于开发过程中的整体管理有很大帮助，因为用例可以通过选择、分组、过滤、优先级排队等方法来重新组织。

CRC 卡片

在寻找更加方便应用用例的方法时，我们从另一种技术中得到了灵感，在面向对象开发过程的稍后阶段需要这种技术。这种技术就是 CRC（类—职责—协作），它使用卡片和角色扮演的方法，帮助人们将系统设计为一系列协作的对象（Beck 和 Cunningham 1989， Bellin 和 Suchman Simone 1997， Wilkinson 1996）。以索引卡片代表每个类，设计组的成员扮演类的角色，以话语的形式模拟系统的行为。卡片上记录了类的职责和协作者。职责是一种概念的抽象，是类应该知道或做的事情。职责概念被当作一种直观推断试探法来分配系统中的智能。

最初，写作 CRC 卡片只是用于学习设计技术的方法，卡片使得对象的思想更加具体，角色扮演则促进了对对象协作的认识。但现在 CRC 卡片已经是得到广泛承认的技术，而不是仅仅面向初学者。Bellin 和 Suchman Simone（1997）认为 CRC 卡片技术是一个“元认知”过程，该技术具有可操作性强的本性，这有助于在设计阶段思考一些关键问题：

“团队中的每个人承担了一个类的职责，以 CRC 卡片为脚本，扮演设计的系统。这一方法的价值在于，扮演类并指出类应该做什么这一过程能够同灵机一动一样，触发相同的响应。把玩这些卡片能够引发难以预料的洞察力。角色扮演能够如此成功是因为它促进了团队成员积极的参与。”

用例卡片与角色扮演

我们已经使用过一段时间的 CRC 卡片技术，并十分赞赏其效果。在寻求改善用例方法的途径时，我们决定采取类似的方法。其基本想法是使用用例的索引卡片，同时引入角色扮演。

用例描述了与系统的交互，但有不止一种方法可以描述这些交互。我们选择用户与系统之间对话的形式。因为对话形式的用例描述能够构成一部剧本，我们希望这样能够有助于角色扮演。剧本有两个角色，用户和系统。这样用例角色扮演可以只有两个人参与，每人负责剧本的一部分。

我们决定，每张用例卡片只代表单个用例，卡片上要记录用例的名称和对话脚本步骤。为了清楚区别两个角色，用居中的一条竖线将卡片分为两个部分，左侧写用户脚本，右侧写系统脚本，如图 1 所示。这种布局正是 Wirfs-Brock（1993）采用的两栏式格式。这种格式对于功能性需求很适用，因为可以很清楚看出如何使用系统以及系统又是如何运作的。

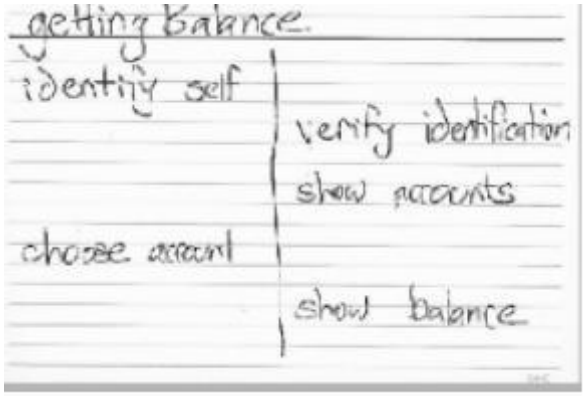

图 1：描述银行系统本质用例的卡片

通常，用例对于陈述非功能性需求没有什么帮助。然而，有时非功能性需求确实与特定的用例相关，这时可以在用例卡片的底部或者背面写下简短说明。

使用用例卡片及角色扮演的方式主要有助于详尽阐述用例的“主体”，确定交互的步骤。在此之前，首先应该识别用例。关于用户与系统交互的各种不同方式的信息等初期工作需要通过背景分析得到。在大型系统开发中，即使是用例识别的工作也可能需要付出大量的代价而且影响范围广泛。我们用各种分析技术识别大量的备选用例，确定它们的优先级，然后选择一些“重点”用例，这些用例代表了最重要的交互，即使是在早期原型中它们也是必需的。

然后，开始在这些重点用例上工作，首先利用卡片和角色扮演详细阐述它们的步骤。如 CRC 卡片方法一样，我们一般推荐 3 到 6 个人的小组来做这些工作。卡片上工作的第一步是写下重点、用例的简短名称。例如，银行系统的重点用例可能名为：*存款*、*检查帐户余额和取款*。

下面选择一张卡片并写出对话。选出扮演用户和系统的人，使用探讨的方式进行排练。团队需要集体工作以决定对话的步骤。当某一想法听起来很合理时，演员按脚本将之“表演”出来。团队的其它成员则审查表演，之后再讨论，并在必要时进行改进。文档投影仪的应用越来越广泛，这样卡片可以很容易的投影到大屏幕上，供更大的团队查看。以上过程反复进行，直到整个团队认为对话已经能够描述用户与系统交互的方式。有时可以先放下某一用例而去审核另一用例，晚些时候再回头改善前一个用例。

角色扮演的作用之一就是以探索和迭代的方式进行用例开发。我们同样利用角色扮演来进行用例复审。这种复审可以由合作开发者、项目其它方面的工作人员、系统利益相关者或系统领域专家来进行。

在利用索引卡片和角色扮演确定了用例的主体后，其详细信息可以记录在需求文档或者 CASE 工具的数据库中。用例可以用于驱动系统设计和复审，在晚些时候用于系统测试和演示。在一些后期活动中，卡片和角色扮演可能还有用途，例如对于某一交互的替代方案进行快速的审查和探讨，卡片和角色扮演直接和轻量的自然本性有很大帮助。

经验

到目前为止，我们已经观察过许多团队使用用例卡片和角色扮演技术。我们在与超过 20 家企业团队合作中使用过该技术，典型情况下，这些团队都包括一些非技术人员，如业务分析人员或基层经理。在教学和指导中我们也介绍了这些方法，特别是在大学的面向对象开发与软件工程课程中。这些经验是鼓舞人心的，在工业领域和教育领域，所有的参与人员都已经熟悉了面向对象系统和 UML 表示的一些基本概念。

我们寻求团队成员的主动和直接参与，结果很令人满意。与 CRC 卡片相同，索引卡片的具体性和角色扮演的行为性结合起来，吸引了团队成员的注意力，同时也要求积极参与。仅仅这一点就有足够的价值，因为它确保在确定需求时整个团队都参与进来了。

与 CRC 卡片一样，索引卡片在另一方面也有所帮助，这与卡片的本性有关：它们既可以分散又可以结为一体，同时只需要很少的投资。卡片可以在桌面上排列、分级，需要的卡片可以抽取出来，在角色扮演时又可以拿着。一张卡片可以很容易被改动或抛弃而不会影响其它卡片，同时可以快速地进行改变。

与卡片相比，角色扮演的优势影响更加深远。团队成员，甚至非开发人员，都能够很好的理解对话。另外，他们很熟悉表演道具，同时乐于怀疑戏剧依赖的基础。人们通常都能够观看角色扮演并在真正理解的基础上想象完成的系统的交互。这是快速原型法的一种形式，它允许快速复审和反馈，允许迭代改良。

Bellin and Suchman Simone（1997）指出，CRC 的角色扮演方法可以引出出人预料的洞察力，对于用例角色扮演也是这样的。在 CRC 的角色扮演中，这种洞察力常常表现于如何在几个协作的对象间分配智能。在用例的角色扮演中，这种洞察则与如何更好在系统边界进行通信有关。

卡片与角色扮演本身并不总是足够确定用例是否合理。开发者应该清楚，有很多相关因素影响用例。特别地，开发者应该研究用例对功能性需求的建模（Armour 和 Miller 2001, Rosenberg 和 Scott 1999），以及用例得以高效的种种因素（Cockburn 2001）。Wirfs-Brock（1994）讨论了用例怎样很好地模拟“有意义的人机对话”，这与我们的技术有关。角色扮演使人机对话更加生活化，从而易于评价它是否真正有意义。

但是我们必须承认，系统还需要满足非功能性需求（如响应时间、事务处理速度、信息容量、可靠性等）。由于用例侧重于交互，所以在发现非功能性需求时并不理想。这意味着非功能性需求必须使用其他方法来进行确定。但无论如何，用例构造了一个很好的需求复审起点，并且通常会引起对非功能性需求的讨论。

根据经验，确定需求时的一个关键问题就是决定系统的边界，将应该由系统完成的事与不应该由系统完成的事分开。因为牵涉到系统的人包括分析人员和各种利益相关人员，对于边界的区分工作达成一致可能很困难。

在决定系统边界时，用例卡片和角色扮演方法被证明是非常有效的。我们使用一种与设计阶段相似的手段，就是将系统看成“黑箱”。并不说明内部如何工作，但如何使用系统由用例指定。每张用例卡片上的两栏式对话表清晰地显示了用户职责和系统职

责，划分两个角色的线就是系统的边界。

在确定系统边界时，用例角色扮演有多方面的帮助作用。在前期的探索式表演中，很快可以看出团队成员对系统应该如何工作持有各种不同意见。人们阅读的是同一份分析文档，但却提出不同的解释，特别常见的是设计人员与业务分析人员或领域专家出现不同理解。及早发现和解决这种差异是很重要的，在确定角色扮演的实际交互序列时比较容易暴露这些差异。

探索式角色扮演同样也能暴露一些对于用户或系统角色的不合理假设。对于用户来说，如果用例指定的动作与可能的实际用户行为不一致，则很容易发现。对于系统来说，如果无法获得重要的信息，则指定的行为无法完成。图 3 显示了在角色扮演中能够发现异常现象。实际的表演看来比仔细阅读更容易暴露出这些异常问题。

第一幕

用户：

我指出我需要做的工作，系统显示该工作的细节。

系统：

喂！说明系统做什么是我的事情。

停！——这经常只是演员的错误，但也能揭示关于系统边界的迷惑。注意负责表演系统的人如何“保护自己的势力范围”

第二幕

用户：

我指定我要做的工作。

系统：

我显示这项工作的细节，并说明是否有座位。

观众：

等一下，还没有指定座位呢。

停！——这说明在用例中有遗失的信息。在此例中，观众保证演员诚实的表演。

第三幕：

用户：

我说明我要做的工作。

系统：

我显示这项工作的细节。

用户：

我指定我要哪一个座位。除非我不想这样做……，如果我这样做，但没有座位，我就得不断的选择座位，直到找到。要是系统能

告诉我哪些座位可用就好了。

停！——在这个例子中，用户已经深入到角色中去了，并且发现了当前版本用例的不足。

第四幕：

用户：

我说明我要做的工作。

系统：

我显示这项工作的细节，并列出可用的座位。

用户：

我选择我想要的座位。

系统：

我将这些座位标记为不可用，并将票打印出来，并且……嗯，也许我应该先算一算需要多少钱并收了钱后再打印这些票？

停！

第五幕：如此等等…

图 3：剧院订票系统用例的角色扮演对话示例，描述角色扮演如何在早期发现用例中的难点

要解决这些异常，可能需要修改用例的步骤，也可能需要以不同的方式来组织用例。这些变化都需要改动其它用例，甚至可能要新建用例。但重要的一点是，可以在早期发现并修正问题，而不是在后期开发才发现从而付出更大的代价。

角色扮演方法也使开发队伍之外的人易于理解用例交互。在给那些事务繁忙的利益相关者演示用例时犹其有用。角色扮演方法快捷、现场感强、易于为人接受。此方法不需要太多投资，能对问题进行讨论，对比几种不同的方案，并可能发现存在的缺陷。

在寻求对用例角色扮演和系统边界确认工作的支持时，我们发现用例图有很大帮助。我们使用的是 UML（Rumbaugh 等 1998, OMG, 2000）用例图，它以线条小人表示角色（在 UML 中表示用户），用椭圆表示用例。我们同样明确地标出系统边界，表示为用例周围的方框，角色与用例之间的线穿越边界，如图 4 所示。在用例图中表示出系统边界的约定由 Jacobson 等人（1992）倡导，虽然它是 UML 用例图的一个可选部分，但它并不是 UML 的典型特性。这样描述系统边界的好处在于将系统可视化地表示为一个单元，边界线与用例卡片上划分用户与系统的线是一致的。

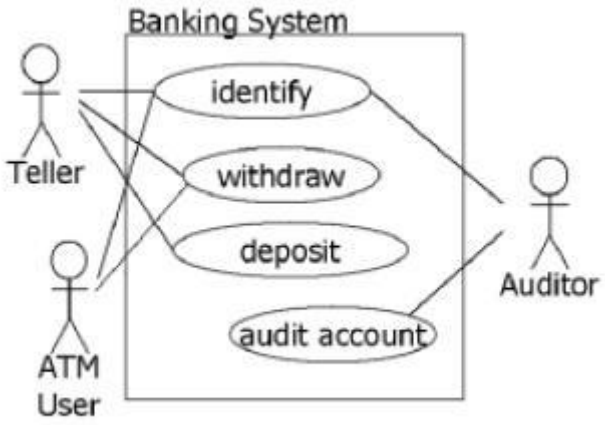

图 4：明确表示出系统边界的用例图

本质用例

尽管卡片与角色扮演可以涵盖任何用例，事实上，我们主要将它们应用在精选过的用例上，这些用例称为本质用例。Constantine 和 Lockwood（1997）发展了这种精选方法，它是“以使用为中心设计”（Usage-Centered Design）的一部分，是用于开发用户界面的过程。他们观察到一个重要现象：

“特别地，常规用例一般包含太多针对尚需设计的用户界面的假设，这些假设多数是隐藏或者暗示的。”

Constantine 和 Lockwood 将本质用例定义为“用例的简化和泛化形式”。他们使用“本质”这一词，因为这些用例：“希望通过非技术性的、理想化和抽象化的描述来捕获问题的本质”

本质用例的抽象与用例没有直接关联，而是更多地与用例的步骤相关。从这一点上讲，一个本质用例确实说明了一系列交互，但只是一些抽象步骤的序列。例如，图 5 显示了银行系统的一个普通用例，该用例详细指定了对话中的具体步骤，例如“插入卡片并读磁条”。图 6 则是一个与之等价的本质用例，它以一个更抽象的“确认自己的身份”开始。再有，需要注意的是，普通用例对各角色的标识是“用户行为”和“系统响应”，这无疑强调了实际的具体交互。而本质用例则在用户部分使用“意图”，系统部分使用“职责”，这样更加抽象些，但内容更丰富。

取款

用户动作	系统响应
插入卡	
	读取磁卡
	提示输入 PIN
输入 PIN	
	校验 PIN
	显示交易菜单

按键	
	显示总额菜单
按键	
	提示总额
输入总额	
	显示总额
按键	
	退卡
取卡	
	吐钞
取钞	

图 5：从自动柜员机取现金的普通用例（引自 Constantine 和 Lockwood.）

取款	
用户意图	系统职责
身份识别	
	验证身份
	提供选择
选择	
	吐钞
取钞	

图 6 从自动柜员机取现金的本质用例（引自 Constantine 和 Lockwood）

在卡片上使用这种形式的用例有很多原因。抽象保证了本质用例的简洁性，这样一张卡片就可以容纳用例描述。同时，这种抽象可以避免对于无关的实现细节的不必要争论，从而加快进度。关注于“意图”和“职责”的方法具有重大意义。

我们的经验证实本质用例确实有简明的优点，同时还具有更深远的影响。正如我们希望的，抽象使得用例适于在索引卡片上使用，同时也避免过早陷入实现细节的讨论。

从更深层次上讲，将用户角色表述为“意图”，而系统角色表述为“职责”的做法具有启发的优点。二者的结合形成的用例对话能够更好的反映用户的真实动机，更好的捕捉系统需求，同时避免不成熟的设计决策。

我们发现使用本质用例需要做一些调整。调整之一，用例角色扮演本身应该是抽象的，以避免过于具体而与特定的接口细节相关。按照我们的经验，这并不是问题，尽管发现团队有时候考虑用例的一些具体条例有助于检查他们的理解。

本质用例起源于用户界面设计。通常的情况下，系统职责仅限于与直接交互相关的职责。暂时并不分析系统行为更深层的职责语义。然而我们发现，在有利的时候可以将这一部分较深层的职责包含进来。例如，图 5 中的用例可能包含维护账户余额一致性

和审计的职责。

关注对话中系统部分，如记录其职责，对于系统开发具有更深远的意义。确定用例之后，面向对象开发过程后一步是将系统设计为一系列协作的类和对象。关注于职责正是确定这些关系的主要方法之一，这也是 CRC 卡片的主要用途，它在职责驱动的设计方法中得到了更加细致的应用（Wirfs-Brock 和 Wilkerson 1989, Wirfs-Brock 等，1990）。

本质用例将系统的角色记录为职责的集合。这意味着，当所有的用例都已经确定时，我们也就有了整个系统的职责集合。我们可以将系统看作单个对象，而职责则是这个对象的规范。我们可以设计一系列协作的对象，如同分解系统对象，并将系统职责恰当地分配给系统各部分一样。

CRC 之类的职责驱动设计技术已经阐述了如何分配职责。但是这些技术都是从领域分析得到对象和类关联的职责开始。本质用例交付了正在设计的系统特有的职责集合，从而提供有价值的机会。

从本质上说，系统职责必须与对象集合职责相匹配。这有两个主要含义：首先，CRC 或职责驱动设计方法依据的职责可以来自本质用例，也可以来自领域分析；其次，我们可以在任何时候检查对象职责的集合，以确认它们是否真的符合系统需求。不管是快速评价其它备用设计方案，还是考察其它现有的可重用设计资源，如模式、框架或组件，我们都能够确认这些对象能够满足用例规定的系统职责。前者意味着在设计初始能够有更好的操作指南。后者意味着设计与需求之间具有更好的可追溯性。

相关工作

本节综述对我们的方法有影响的公开出版物。

用例方法是面向对象软件工程（OOSE）引入的一个关键概念（Jacobson 等, 1992）。用例有很多种形式，对我们影响最大的是 Wirfs-Brock 称为“会话”的那一种（“Wirfs-Brock，1993 及 1994”）。在实践中我们采用的是“本质用例”，它是从用户界面设计中发展起来的（Constantine 和 Lockwood，1999）。Cockburn（2001）提供了与用例相关的论述和方法的综述。

OOSE 建议如何在系统开发语境中应用用例。用例也是更现代化开发过程的特征，如 Rational（瑞理）统一方法（Unified Method）（Jacobson，Booch，和 Rumbaugh， 1999）。在 UML（Jacobson，Booch， and Rumbaugh， 1998）等图示语言中也处理了用例。影响我们的工作还有，关于如何利用用例对需求建模的研究（Rosenberg 和 Scott, 1999, 以及 Armour 和 Miller, 2001），极限编程（Beck，1999）中对“用户故事”（该概念很象用例，但是写得很具体）的应用，以及强调开发者和利益相关者参加复审的重要性。OPEN 技术工具箱（ Henderson-Sellers, Simons, 和 Younessi, 1998）提供了一个详尽的调查，包括关于用例和 CRC 卡片的讨论和指南。

CRC 卡片技术，包括卡片本身和角色扮演，对我们的方法影响很大。CRC 最初只是一个教学手段）（Beck 和 Cunningham, 1989）。但最近的研究和实践表明，CRC 卡片作为一个通用设计技术有很大好处（Wilkinson, 1996, 以及 Bellin 和 Suchman-Simone, 1997)）。

另外，CRC 卡片技术和本质用例都大量使用“职责”做为分析指南。我们对这种启发式方法的应用也受到“职责驱动设计”过程的影响。（Wirfs-Brock 和 Wilkerson, 1989, Wirfs-Brock, Wilkerson 和 Wiener, 1990）

做为一种复审技术，我们的方法也反映了其它软件质量保证工作的结果。过去需求检查或者很不正规，或者仅仅是一次明确而正式的“软件检查”（Fagon，1979）的一部分。我们的方法不是完全正式的，但提供了一个有用的整体结构，类似于 Yourdon（1989）的“排练”，但强调用例而不是设计或实现。

结论

本文提出使用索引卡片和角色扮演进行用例复审的技术，以帮助开发更好的用例。该技术基于面向对象设计中成熟的 CRC 卡片技术。在角色扮演中只有两类角色：用户和系统，这来自于使用对话格式的形式描述用例，如同本质用例中使用的一样。

该技术是轻量级的，并不拘泥于形式，这意味着较低的准入门槛，特别是对初学者和利益相关者来说。这一点很重要，因为它能够保证利益相关者很容易参与到复审过程中来，这无疑能够保证高质量。

我们发现该技术对于暴露用例中潜在的缺陷和异常很有效，在确定系统边界时更是有帮助，因为这种划分正好与角色划分相同。另外，利用本质用例方法对于面向对象设计还有其它好处，它可以带来设计和需求之间更好的可追溯性。

参考文献

Frank Armour and Granville Miller. (2001) *Advanced Use Case Modeling: Software Systems, Volume 1*. Addison-Wesley.

Kent Beck and Ward Cunningham. (1989) A laboratory for teaching object-oriented thinking. In *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pages 1-6.

David Bellin and Susan Suchman-Simone. (1997) *The CRC Card Book*. Addison-Wesley.

Alistair Cockburn. (2001) *Writing effective use cases*. Addison-Wesley.

Larry L. Constantine and Lucy A. D. Lockwood. (1999) *Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design*. Addison-Wesley.

Michael E. Fagan. (1986) Advances in Software Inspection. *IEEE Transactions on Software Engineering*, pages 744-751, July.

Brian Henderson-Sellers, Anthony Simons, and Houman Younessi. (1998) *The OPEN toolbox of techniques,* Addison-Wesley.

Ivar Jacobson, Grady Booch, and James Rumbaugh. (1999) *The Unified Software Development Process,* Addison-Wesley.

Ivar Jacobson, Mahnus Christerson, Patrik Jonsson, and Gunnar Overgaard. (1992) *Object-Oriented Software Engineering*. Addison-Wesley.

OMG Object Management Group. (2000) Unified Modeling Language (UML) 1.3 specification.

Doug Rosenberg and Kendall Scott. (1999) *Use case driven object modeling with UML: A practical approach*. Addison-Wesley.

James Rumbaugh, Ivar Jacobson, and Grady Booch. (1998) *The Unified Modeling Language Reference Manual*. Addison-Wesley.

Nancy Wilkinson. (1996) *Using CRC Cards - An Informal Approach to OO Development*. Cambridge University Press.

Rebecca Wirfs-Brock and Brian Wilkerson. (1989) Object-oriented design: A responsibility-driven approach. In Norman Meyrowitz, editor, *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pages 71-75.

Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. (1990) *Designing Object Oriented Software*. Prentice Hall.

Rebecca J. Wirfs-Brock. (1993) Designing scenarios: Making the case for a use case framework. *The Smalltalk Report*, 3(3).

Rebecca J. Wirfs-Brock. (1994) The art of meaningful conversations. *The Smalltalk Report*, 4(5).

Ed Yourdon. (1989) *Structured Walkthroughs*, Prentice Hall/Yourdon Press.

版权

Biddle, Noble, Tempero © 2001. The authors assign to ACIS an educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive licence to ACIS to publish this document in full in the Conference Papers and Proceedings. Those documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the authors.

摘自 umlChina