软 件 工 程 技 术 丛 书

HZ BOOKS

PEARSON
Addison Wesley

分析系列

用例
通过背景环境获取需求
（原书第2版）

Use Cases
Requirements in Context, Second Edition

（美） Daryl Kulak
Eamonn Guiney 著

韩柯 杨柳青 等译

机械工业出版社
China Machine Press

软 件 工 程 技 术 丛 书

分析系列

用例
通过背景环境获取需求
（原书第2版）

Use Cases
Requirements in Context, Second Edition

（美） Daryl Kulak
Eamonn Guiney 著

韩柯 杨柳青 等译

机械工业出版社
China Machine Press

本书介绍了通过用例来采集用户需求，为实际的需求问题提供解决方案，从而产生满足用户要求的高质量的系统。本书作者极具实践经验，他们将自己的经历融入书中，从需求中遇到的问题谈起，阐述了使用用例的必要性，用例的基本知识，通过用例确定需求的三步法等内容，并对用例驱动的生命周期、跟踪等方面给出了有价值的建议。书中还包含大量的实例以供读者参考。本书主题鲜明，结构清晰，实用性强，适合软件需求工程师、设计人员、项目经理阅读，也适合作为相关专业的本科生、研究生的参考读物。

本书版权登记号：图字：01-2003-6198

图书在版编目（CIP）数据

用例：通过背景环境获取需求（原书第2版）/（美）库拉克（Kulak, D.）等著；韩柯等译. -北京：机械工业出版社，2004.1

（软件工程技术丛书　分析系列）

书名原文：Use Cases: Requirements in Context, Second Edition

ISBN 7-111-13467-2

Ⅰ.用…　Ⅱ.①库…②韩…　Ⅲ.①软件工程-系统分析②软件工程-系统设计　Ⅳ.TP311

中国版本图书馆CIP数据核字（2003）第112677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：王高翔　朱　劼

北京昌平奔腾印刷厂印刷·新华书店北京发行所发行

2004年1月第1版第1次印刷

787mm×1092mm 1/16·14.5印张

印数：0 001 - 4 000 册

定价：35.00 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

与软件危机一起诞生的软件工程方法和建模理论已经发展了几十年。但现实的情况是，软件项目存在的问题仍然很严重，我们经常会看到有头无尾的工程，用户不满意的工程，难以投入实际使用的工程，或者严重超支和拖延进度的工程。而这种现象往往是需求问题造成的。虽然人们提出了解决需求获取问题的很多方法和技术，但是仍然难以从根本上解决需求问题。

UML作为一种建模语言标准已经得到广泛应用，市场上有许多介绍这一主题的书籍。本书的特点是，专门论述通过基于用例的过程采集和定义软件需求，为系统分析人员和设计人员展示怎样运用用例为最具挑战性的需求问题提供解决方案，产生满足用户需要的高质量的有效系统。此外，作者还提出了需求获取的三步法，即一种渐进地产生细化需求的三轮迭代过程。书中给出大量详细的例子，有助于读者根据自己的具体情况，运用用例获取和描述需求。本书适合各类软件需求工程师、开发人员、测试工程师和软件项目经理阅读，对于大专院校计算机应用和系统工程专业的研究生和高年级学生，也是不错的补充材料。

在翻译过程中，除了对原文个别明显文字错误进行了相应更正外，我们力求忠实原文。但由于译者的知识水平和实际工作经验有限，不当之处在所难免，恳请读者批评指正。参加本书翻译、审校等工作的还有：黄慧菊、屈健、刘芙蓉、王威、李津津、原小玲、韩文臣、孟海军等。

译 者

前　言

本书第1版出版至今，已经过去了三年。三年前，用例还是一种“有趣的手段”，还没有被广泛采纳。在今天的软件开发市场中，用例已经成为采集需求的标准实践，甚至已经移植到其他应用中，包括业务过程和服务提供。但是在三年前，即使从最乐观的角度预计，也没有料到用例会如此流行。

当然在过去几年中，本书并不是惟一一本介绍用例的专著，不过它在一定程度上满足了软件行业人士了解这种新手段的要求。在这种软件技术发展趋势条件下，我们决定出版本书的第2版，把第1版出版以来我们得到的众多经验教训补充到本书中。我们根据开发多个用例驱动的项目的经验创建了本书第1版提出的方法，不过这种过程还很稚嫩。自从本书第1版出版以来，这种过程已经在很多其他项目中得到应用，我们也有机会把软件业务中很多好的想法融合到过程中，并对过程进行精心调整，使之成为更有效、更实用和更具扩展性的方法。我们还吸收了其他新兴领域的思想，包括“**混沌机构**”的思想（Dee Hock 2000；Margaret Wheatley 2001等）和Purdue大学Theodore Williams博士和Hong Li博士的“**Purdue企业参考体系结构**”的思想。这两项成果对我们将用例用于项目的方式和重新形成新版本中的许多新构想有着巨大影响。

首先最值得注意的是，第2版只有三个“F”迭代，即外观（facade）、填充（filled）和聚焦（focused）迭代。在应用到诸多项目后，最后一个“F”（结束，finished）迭代被证明是有问题的。首先，在迭代方法中，没有什么是能够真正结束的，事物永远都要进化。此外在一次迭代中，用例和用户界面设计总要交织在一起。我们把用户界面思想转移到外观迭代中，因为用户界面的进化应该与创建早期用例并行进行，而不是在其之后进行。

我们方法的另一个大的变化出现在管理这一章中。虽然和以前的内容并没有直接矛盾，但是第2版对迭代/渐进式用例驱动的项目管理的解释做了很大扩展。我们把它叫作整体迭代/渐进法，也就是HI/I（读作hi-eye）。我们相信生命周期的这个领域还有很多工作要做，因为多年以来沿用的瀑布式项目管理过程无法适应今天节奏更快、更加“混沌”的软件生命周期。我们在本书中给出管理一章只是为了抛砖引玉，非常希望其他作者能够扩展这些思想，并提出解决这个

重大问题的新方法。此外，项目管理研究所（PMI）也提出了一些采用某些新的软件生命周期思想的方案。

我们了解到有很多读者认为本书第1版的附录既是书中的最好部分，也是最差部分。那是我们第一次尝试给出部分完整的用例例子，这也是理解创建用例的迭代本质的至关重要的一个步骤。但是，我们的表达不够清晰，因为我们通过四次迭代重复了该用例，有时用例有变化，有时没有变化，很难知道出现了什么变化以及什么时候出现的变化。这一次我们采用极为不同的方法。我们仍然要解释如何在软件需求采集中运用用例，但是所采用的方法不那么正式。在附录的各个部分中，我们选择不同风格的应用系统（大型业务应用系统、技术子系统、包评估等），并展示用例和其他制品不断演化的过程。我们希望这样做既能够保持第1版的优点，又能够反映出用例版本的演化。

我们发现，在很多的项目中，用例层次结构的思想除了引起混乱之外，没有起到任何作用。首先创建“高层”用例，然后创建“细节”用例，这种方法会损害需求获取过程。层次结构越高、越复杂（有的书建议采用4层或更多层次的结构），距离最初的业务需求就会越远。即使我们的初始过程只有两层结构（系统背景环境级用例和下层用例构成），但是当团队想要增加层次时仍然会遇到麻烦，并会引起混乱。类似地，对用例关联使用<<包含>>（include）和<<扩展>>（extend）构造型，也会产生不必要的层次问题，因此除了非常特殊的情况，一般不使用构造型。为解决这个问题，我们在大家都很熟悉的工具和过滤器包中补充了一种新的工具，即“**层次结构杀手**”。希望你在每次精简层次结构时都能够得到乐趣。

用例在很多方面都与其他类型的需求获取手段不同，其中一个方面就是可跟踪性方面的不同。用例对业务需求有很强的可跟踪性，对软件开发制品也有很强的可跟踪性，可以跟踪UML分析和设计制品，跟踪测试、编写文档记录、培训、安全，甚至跟踪体系结构的各个部分。我们决定专门设立一章来介绍用例的可跟踪性现象，以说明如何确保团队“做正确的事”。

最后，为了反映需求获取工具的最新动态，我们列出在本书出版时可以得到的工具，并提出一些充分利用这些工具的建议。由于这些工具的更新速度很快（但我们写书的速度很慢），因此我们只列出主要的工具。

我们希望读者能够喜欢本书的第2版。我们很高兴能够与Pearson Education公司再次合作，出版本书的更新版本。如果你有什么看法、经验和建议，请随时与我们联系。我们的电子邮件地址列在本书最后一章的末尾。

致谢

我们要感谢使本书第2版得以顺利出版的每个人，特别感谢Bill Banze、Jordan Antonelli和Lara Hoekstra。我们还要感谢Addison-Wesley出版公司的编辑和生产人员：Peter Gordon、Bernard Gaffney、Lynda D'Arcangelo和Tyrrell Albaugh。

第1版前言

可能与大多数书一样，写作本书的初衷来自抱怨。因为我们觉得没有一本好的著作专门讨论用于需求采集的用例。很多人都赞同用例是解决需求问题的完美工具，但是却没有人把详细过程写下来，帮助大家理解如何使用用例。

需求采集已经成为我们所参与的所有项目中的重要问题。需求的模糊特性常常使大多数软件分析人员感到困惑并且无从下手。用例是我们看到的用于解决通常与需求采集相关的规格说明和沟通问题的第一种工具。

虽然用例本身相当直观，但是围绕用例制定的过程却常常不能很好地实施。人们的问题是：我要进行多少次迭代？用例的粒度应该有多细？大多数著作都没有回答甚至没有涉及这些问题。这可能是因为这些问题很难回答，因为在不同情况下答案会有很大不同。但是这些都是很重要的问题，所以我们决定把自己的想法写出来，目的是抛砖引玉，希望能够得到同行关于如何产生能够描述用户需要的需求这一问题的看法。

本书是技术工作者的一本实践参考。与信息技术业的咨询公司一样，我们也把使用用例描述业务系统作为自己日常工作的一部分。我们认为自己理解人们在使用像统一建模语言和用例这样的工具设计软件时所面临的问题。我们的主要目的不是描述用例符号，尽管我们确实也要讨论这个问题。我们的主要目的是，说明以一种可以产生高质量结果的方式采集需求的需求过程。

在编写本书时，我们研究了引起需求采集问题的因素，提出了一种交付面向需求的可交付产品集合的用例方法。这种方法论将产生需求的活动分解为一系列步骤，它可以解决人们在使用用例时通常会遇到的问题。本书与交付规格说明、管理团队工作和激励下属等实际工作相关。

我们希望读者能够喜欢本书。本书对于我们而言是爱的付出。本书所述的过程是很适合我们的，如果这个过程也适合你，那再好不过了。如果不适合，那么读者可以根据自己的实际情况，对本书提供的工具、思想或建议进行修改以解决自己的需求问题。

致谢

衷心感谢帮助本书出版的很多人。他们提供了物质帮助和建议，在本书整个编写过程中都提供了热情的支持和鼓励。很难在这里列出我们要感谢的每一个人，不过我们要特别感谢Bill

Banze、Craig Larson、Steve Frison、Peter Gordon、Mat Henshall、Erin Lavkulich、John Lavkulich、Sam Starr、Terry Noreault、Jeanne Baker、Steve Ulrich、Ken Richetts、Hong Li、Ted Williams、Lisa Austin、Naomi Kunkel、Jennifer Werth、Tracie Bennett、Vishal Saboo、Bill Klos、Karen Sander、Tamara Oakley、Debbie Ard、Paul Reed、Steve Jackson、Sarah Ward和Ogden Weary。

目　录

第 1 章

需求中的问题

当面临他们所认为的难题的时候，大多数工程师都忙于提出解决方案。

——Alan M. Davis

1.1 首要问题与至少要解决的问题

图1-1 技术人员往往更关注实体关系图或类图，而不是需求清单

每次系统开发人员着手向业务人员提供某个计算机系统时，都要完成一组相当一致的活动：

- 需求获取
- 分析

- 设计
- 构建
- 测试
- 部署
- 维护

开发团队给予每个阶段的强调程度，决定开发方向和最终计算机系统的质量。如果有一种活动没有按时开展，项目和最终产品就会出现能够预见到的问题。不过在现实世界中，某些活动通常会得到比其他活动更多的关注。要解释为什么会出现这种现象并不容易，但的确会出现这种现象。通常会被忽视或只得到空头允诺的活动是：

- 需求获取
- 测试
- 部署
- 维护

在传统上，只有少数提供商提供用于完成这些活动的华而不实的工具，也许这是这些工具不那么有意义、不那么吸引实践者的原因。当然，每种活动都需要很高的创造性和范围很宽的技能，但是可以感觉到，除了分析、设计和构建这三种主要活动外，其他活动都不需要多少关注和想像力。

这种感觉正在缓慢而全面地发生变化，因为提供商正在构建用于管理需求（Rational RequisitePro、Borland Caliber RM、Telelogic DOORS）、进行自动测试（Segue SilkTest/SilkPerformer、Mercury Interactive WinRunner/LoadRunner、Rational Robot/Team Test/TestManager）和为部署提供帮助（Borland Deployment Server/Java、Marimba Desktop/Mobile、BEA WebLogic）的工具。不久前的Y2K问题使对维护的关注也出现了一个小高潮。我们对提高这些没有得到应有重视的活动的重要性、吸引力和有效性方面有自己的观点，将在我们的下一本书中专门讨论。

我们写这本书是因为我们重视需求。首先，有效的需求是产生满足用户需要的系统的关键。毫不夸张地说，需求本身就是用户需要。

不仅如此，我们越来越重视需求，因为我们看到更多的项目主要由于需求不

好，而导致进展不利或失败。由于需求是推动其他系统开发工作的手段，因此到系统部署时，需求方面任何小的疏漏都会放大为严重缺陷。弥补这些缺陷会极为耗时（意味着成本很高），因为有这么多的工作已经被引导到错误的方向上。但是，需求并没有很好地翻译为已编程实现的计算机系统的分立模块。有时，需求嵌入在贯穿很多代码库和组件的原则中。由于需求这么抽象，并且与计算机程序有这样大的差别，因此对于技能以具体的计算机程序设计为主的人们来说，恰当地处理需求相当困难。

以前的需求获取常常出现以下问题：

- 要花很长时间。
- 把错误的东西写入文档。
- 对尚未发生的活动做出假设。
- 由于业务的变化太快、太大，常常刚刚完成一稿需求，马上就要推倒重来。

不久以前，我们遇到过一份需求定义文档，包含160多页的“需求”。这么厚的需求清单，使我们一想到要仔细消化并将各个部分串联起来，就感到很恐慌（或至少有些担心）。表1-1给出了一个需求清单的样本（与我们看过的很多其他需求清单相比并没有太多不同点）。

表1-1 需求清单举例

需 求	定 义
6.7.1.4.2	系统必须提供获取本财政年度所有客户事务的能力
6.7.1.4.3	系统必须提供受限制的单独或同时远程访问（通过拨号接入）查询视图图像和数据的能力
6.7.1.4.4	系统在分发文档之前，要自动地在文档上打印条形码。条形码至少可以在文档被返回时，标识机构内部文档应该送入的队列
6.7.1.4.5	当启动工作流后，系统必须能够根据文档类型预获取以电子图像格式存储的文档，或根据过程预获取相关的文档
6.7.1.4.6	每当生成信件时，系统都必须在日志文件中创建一条记录
6.7.1.4.7	系统必须维护当前开放工作过程的列表，并标识要被执行的工作过程和该过程的工作流队列。当扫描特定的文档时，系统要确定是否有向“社会安全号”（SSN）开放的过程。如果有这样的一个开放过程，则系统将该文档送到相应的工作流队列中，显示工作过程脚本，突出显示当前工作过程

请别忘了，这样的清单还有159页！

我们稍后再剖析这些以及其他需求描述，不过读者可以想阅读这个分层组织的、远没有把真正需求与细碎细节剥离开的大量信息有多困难。如果读者正在

读这本书，那读者很可能也遇到过同样的问题。

需求文档常常很糟的原因之一是，需求获取常常采取了没有成果的路线。例如，需求文档化过程会被忽略。在这种情况下，开发团队直接寻找解决方案，根据非书面并且没有达成一致的有关系统运行方式的假设进行设计。或者需求获取本身就变成终点，收集了大量“需求”，形成文档、交叉引用并进行发布，导致分析困难，甚至在生命周期后续部分还没有开始之前就取消了项目。或者需求是用户、系统拥有者或高层经理一个人确定的，使系统只为他一个人服务，不能使其他人满意。这些方法都不能为分析活动产生令人满意的输入信息。

从各种各样方法论和过程中得到的几百种应用系统生命周期活动定义，在我们的书架和因特网中到处都是。表1-2给出了我们对这些术语的定义，因为我们要通过这些定义解释最佳的需求获取方法。

表1-2 活动定义

活动名称	描 述
需求获取[1]	以用户的语言，从用户的角度，收集应用系统应该为用户执行的功能，并形成文档
分析	从需求入手，确定满足需求的逻辑解决方案，但不一定考虑物理约束条件
设计	从逻辑解决方案入手，使方案有效地适应物理约束条件（网络延迟、数据库性能、用户界面、高速缓存、可用性等），产生指导构建工作的规格说明
构建	使用物理解决方案产生能够执行的代码，包括渐进地做出最低层的设计决策、编写代码、编译、调试和测试
测试	通过系统测试、检查并记录问题，修改错误，使用已构建的应用系统产生完整的运行工作系统，让用户验收系统
部署	通过把代码库部署到目标计算机上，培训用户，精细调整新系统外围的业务过程，使经过测试的应用系统适合生产环境
维护	在生产环境中，对正式运行的工作系统进行管理和修改，以适应不断出现的业务变更（法律、竞争环境）、技术变更（硬件、软件和通信）、物理变更（地点、配置）、人员（信息技术、用户）和系统问题（代码错误、设计问题）甚至办公方针

注：① 我们把需求获取看作是与分析分开的活动。这与主张将这两种活动合在一起的著名行业领导者的观点不同。没有哪种方法绝对正确，或绝对错误，我们把这两种活动分开，只是为了强调这些活动的重要性。

这里给出的生命周期活动定义并不来自任何具体的方法论，我们只是试图选择每个术语的最常使用的名称和定义。请注意，我们使用的是“活动”而不是“阶段”。这样做是有意义的。在瀑布模型中，生命周期、活动（也就是工作流）和阶段都是同义词。但是，“活动”与一个或一组人所做的事有关。“阶段”含