TURING 图灵程序设计丛书

Addison Wesley

第21届
Jolt大奖
获奖作品

Continuous Delivery

Reliable Software Releases through Build, Test, and Deployment Automation

持续交付

发布可靠软件的系统方法

[英] Jez Humble
David Farley 著

乔梁 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Continuous Delivery

Reliable Software Releases through Build, Test, and Deployment Automation

持续交付

发布可靠软件的系统方法

[英] Jez Humble
David Farley 著

乔梁 译

人民邮电出版社
北京

图书在版编目（CIP）数据

持续交付 ：发布可靠软件的系统方法 /（英）亨布尔（Humble,J.），（英）法利（Farley,D.）著 ；乔梁译. -- 北京 ：人民邮电出版社，2011.10
（图灵程序设计丛书）
书名原文：Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation
ISBN 978-7-115-26459-6

Ⅰ. ①持… Ⅱ. ①亨… ②法… ③乔… Ⅲ. ①软件开发 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2011)第197388号

内 容 提 要

本书讲述如何实现更快、更可靠、低成本的自动化软件交付，描述了如何通过增加反馈，并改进开发人员、测试人员、运维人员和项目经理之间的协作来达到这个目标。本书由三部分组成。第一部分阐述了持续交付背后的一些原则，以及支持这些原则的实践。第二部分是本书的核心，全面讲述了部署流水线。第三部分围绕部署流水线的投入产出讨论了更多细节，包括增量开发技术、高级版本控制模式，以及基础设施、环境和数据的管理和组织治理。

本书适合所有开发人员、测试人员、运维人员和项目经理学习参考。

图灵程序设计丛书

持续交付：发布可靠软件的系统方法

◆ 著 [英] Jez Humble David Farley
译 乔 梁
责任编辑 卢秀丽
执行编辑 毛倩倩 杨 爽

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：24
字数：512千字 2011年10月第1版
印数：1－4 000册 2011年10月北京第1次印刷
著作权合同登记号 图字：01-2011-0727号
ISBN 978-7-115-26459-6

定价：89.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

谨以此书献给我的父亲，他总是给予我无条件的爱与支持。

——Jez

谨以此书献给我的父亲，他一直为我指明正确的方向。

——David

对本书的赞誉

如果你需要频繁地部署软件，那么本书就是你所需要的。采用本书所描述的实践会帮助你降低风险，克服工作的乏味，并增强信心。我会在我所有的项目中使用本书所描述的原则与实践。

——Kent Beck，三川研究室

不管你的软件开发团队是否已经明白持续集成就像源代码控制一样必不可少，本书都是必读之物。本书不可多得地将整个开发和交付过程放在一起进行诠释，不仅提到了技术与工具，而且提供了一种理念和一些原则。作者讲述的内容从测试自动化到自动部署不一而足，能够满足读者的广泛需求。开发团队中的每个人，包括编程人员、测试人员、系统管理员、DBA和管理者，都应该读一读这本书。

——Lisa Crispin，*Agile Testing: A Practical Guide for Testers and Agile Teams*的作者之一

对于很多组织来说，持续交付不仅仅是一种部署方法，它对于开展业务也是至关重要的。本书展示了如何在具体环境中让持续交付成为现实。

——James Turnbull，*Pulling Strings with Puppet: Configuration Management Made Easy*的作者

这是一本清晰、准确、精心编写的书，力求让读者明白发布过程究竟应该是什么样子。作者以渐进的方式一步步地阐述了软件部署中的理想状态与障碍。本书是每位软件工程师的必备读物。

——Leyna Cotran，加利福尼亚大学欧文分校软件研究所

Humble和Farley阐明了是什么使快速成长的Web应用取得成功。曾经颇具争议的持续部署和交付已经成为司空见惯的技术，而本书出色地讲述了其中的方方面面。在很多层面上，这都是开发和运维的交点，而他们正是瞄准了这一点。

——John Allspaw，Etsy.com技术运营副总裁
*The Art of Capacity Planning*和*Web Operations*的作者

如果你的业务就是构建和交付基于软件的服务，你一定会从本书清晰阐述的理念中受益。而且，除了这些理念以外，Humble和Farley还为快速可靠地进行软件变更提供了一份卓越的“剧本”。

——Damon Edwards，DTO Solutions总裁，dev2ops.org网站主编之一

我相信，做软件的人拿起这本书，翻到任意一章，都会很快得到有价值的信息。如果从头到尾仔细阅读，你就能根据所在组织的具体情况对构建和部署过程进行简化。我认为，这是一本关于软件构建、部署、测试和发布的必备手册。

——Sarah Edrie，哈佛商学院质量工程总监

对于现代软件团队来说，显然持续交付就是持续集成的下一步。本书以不断为客户提供有价值的软件为目标，通过一套明确且有效的原则和做法使这一目标的实现成为了可能。

——Rob Sanheim，Relevance公司技术骨干

译 者 序

十年前，敏捷软件开发方法在国内还少有人知，现在却已渐成主流。持续集成作为一个敏捷开发最佳实践，近年来也被广泛接受。然而，它们并没有很好地解决所谓的“最后一公里问题”，即如何让软件从“开发完成”迅速实现“上线发布”。

本书的问世让这个问题有了答案。通过将敏捷和持续集成的理念应用到整个软件生命周期中，利用各种敏捷原则与最佳实践打破了用户、交付团队及运维团队之间的壁垒，让原本令人紧张、疲惫的软件发布过程变得轻松了，令原本枯燥易错的部署操作已变得只需轻点鼠标即可完成。

本书首次从业务视角阐述了持续交付的必要性，并从现实问题出发，对软件交付过程进行了彻底的剖析，指出了各交付环节所需遵守的原则与最佳实践，列举了各种常见的反模式。书中的案例详实，贴近生产一线，读过之后，你一定会产生强烈的共鸣。

本书为所有人带来了曙光：

作为IT部门的主管，当你发现这本书后，一定会觉得眼前豁然开朗；

作为项目经理，当你读完本书的前五章后，一定会觉得手中的项目不再令你望而却步；

作为开发人员，当你在多个分支之间痛苦地解决着合并冲突时，本书中的配置管理实践一定会让你觉得看到了希望；

作为测试人员，当你在各类测试间疲于奔命时，本书中的自动化测试章节一定会令你觉得神清气爽；

作为运维人员，当你在为各类环境的维护而苦恼不休时，本书中的环境管理内容一定会让你觉得心旷神怡。

持续交付以全面的版本控制和全面自动化为核心，通过各角色的紧密合作，力图让每个发布都变成可靠且可重复的过程。

作为Cruise[①]的业务分析师，我暗自庆幸能和Cruise团队的其他成员一起见证这本书的问世，并在Cruise整个研发过程里采用书中的诸多实践，为本书提供了素材。

---

① Cruise是ThoughtWorks Studios在2008年发布的一款持续集成与发布管理工具，现已更名为Go。

——译者注

正因了解本书对软件行业的重大意义，在其出版之前，我就向作者之一Jez Humble提出，希望将这本书引入中国。也正因如此，在此书英文版出版后的短短一年内，中文版就能与国内读者见面。同时，也感谢图灵教育以专业的态度，制订了完善的出版计划，本书才得以尽早出版。

此外，谨以此书献给我的妻子兆霞和儿子皓天。他们的支持和鼓励让我在过去的十个月中，利用业余时间顺利地完成了这数百页的翻译工作。

昨日获悉，本书获得了2011年Jolt图书大奖，这足以证明它值得一读。希望读完之后，你也和我一样有所收获，并把它介绍给更多还在痛苦中艰难前行的软件从业者。

乔 梁

于2011年8月

马丁·福勒序

20世纪90年代末期，我拜访了Kent Beck，当时他正在瑞士的一家保险公司工作。他向我介绍了他的项目，他的团队有高度的自律性，而一个很有趣的事情是每晚他们都将软件部署到生产环境中。这种具有规律性的部署带给他们很多好处，已写好的软件不必在部署之前无谓地等待，他们对问题和机会反应迅速，周转期很短使他们与其业务客户以及最终用户三方之间建立了更深层次的关系。

在过去10年里，我一直在ThoughtWorks工作。我们所做的项目有一个共同的主题，那就是缩短从想法到可用软件之间的生产周期。我看到过很多项目案例，几乎所有项目都在设法缩短周期。尽管我们通常不会每天发布，但现在每两周发布一次却是很常见的。

David与Jez就身处这场巨大变革之中。在他们所致力的项目中，频繁且可靠地进行交付已然成为一种文化。David、Jez和我们的同事已经将很多每年才能做一次软件部署的组织带到了**持续交付**的世界里，即让发布变成了常规活动。

至少对开发团队来说，该方法的基础是持续集成（CI）。CI使整个开发团队保持同步，消除了集成问题引起的延期。在几年前，Paul Duvall写了一本关于CI的书。但CI只是第一步。软件即使被成功地集成到了代码主干上，也仍旧是没有在生产环境中发挥作用的软件。David和Jez的书对从CI至“最后一公里”的问题进行了阐述，描述了如何构建部署流水线，才能将已集成的代码转变为已部署到生产环境中的软件。

这种交付思想长期以来一直是软件开发中被人遗忘的角落，是开发人员和运维团队之间的一个黑洞。因此毫无疑问的是，本书中的技术都依赖于这些团队的凝聚，而这也就是悄然兴起的DevOps运动的前兆。这个过程也包括测试人员，因为测试工作也是确保无差错发布的关键因素。贯穿一切的是高度自动化，让事情能够很快完成而且没有差错。

为了实现这些，需要付出努力，但所带来的好处是意义深远的。持续时间长且强度很高的发布将成为过去。软件客户能够看到一些想法快速变成他们可以每天使用的可工作软件。也许最重要的是，我们消除了软件开发中严重压力的一个重大来源。没有人喜欢为了让系统升级包能够在周一的黎明前发布而在周末紧张加班。

在我看来，如果有这样一本书，能够告诉你如何做到无压力且频繁地交付软件，那么显然它是值得一读的。考虑到你们团队的利益，我希望你能同意我的观点。

致　谢

很多人为本书作出了贡献。特别要感谢我们的审校者David Clack、Leyna Cotran、Lisa Crispin、Sarah Edrie、Damon Edwards、Martin Fowler、James Kovacs、Bob Maksimchuk、Elliotte Rusty Harold、Rob Sanheim和Chris Smith。同时也要感谢Addison-Wesley的编辑和制作团队，包括Chris Guzikowski、Raina Chrobak、Susan Zahn、Kristy Hart和Andy Beaster。Dmitry Kirsanov和Alina Kirsanova很好地完成了本书的编辑和校对工作，并使用完全自动化的系统完成了本书的排版工作。

本书中很多想法都得益于众多同事给我们的启发，这些人包括（排名不分先后）Chris Read、Sam Newman、Dan North、Dan Worthington-Bodart、Manish Kumar、Kraig Parkinson、Julian Simpson、Paul Julius、Marco Jansen、Jeffrey Fredrick、Ajey Gore、Chris Turner、Paul Hammant、胡凯、乔彦东、乔梁、杨哈达、Julias Shaw、Deepthi、Mark Chang、Dante Briones、李光磊、Erik Doernenburg、Kraig Parkinson、Ram Narayanan、Mark Rickmeier、Chris Stevenson、Jay Flowers、Jason Sankey、Daniel Ostermeier、Rolf Russell、Jon Tirsen、Timothy Reaves、Ben Wyeth、Tim Harding、Tim Brown、Pavan Kadambi Sudarshan、Stephen Foreshew、Yogi Kulkarni、David Rice、Chad Wathington、Jonny LeRoy、和Chris Briesemeister。

Jez要感谢他的妻子Rani，他的挚爱。她的鼓励让他走出了本书写作过程中的低谷。他还要感谢他的女儿Amrita，她的呀呀学语、拥抱和灿烂的笑容让他坚定了信心。同时，他还要深深地感谢他在ThoughtWorks的同事，是他们造就了一个鼓舞人心的工作环境，并感谢Cyndi Mitchell和Martin Fowler对本书的支持。最后，还要特别感谢Jeffrey Fredrick和Paul Julius（他们俩创建了CITCON），以及他有机会在CITCON遇见并与之畅谈甚欢的人们。

David要感谢他的妻子Kate及其孩子Tom和Ben在这个项目及其他方面无尽的支持。同时，尽管他不再就职于ThoughtWorks，但仍感谢它为在那里工作的人们提供了一个开放、进取且鼓励创新的环境，并因此促成了寻求解决方案的创新型方法，而其中很多方法又丰富了本书内容。另外，他还感谢现在的公司LMAX以及Martin Thompson，感谢他们的支持、信任，他们采纳了本书中所描述的技术，将其应用到了世界级高性能计算这个具有强烈挑战性的技术环境。

前　言

引言

昨天，老板让你给客户演示一下产品很好的新特性，但你却什么都演示不了。所有新功能都只开发了一半，没人能让这个系统运行起来。你能拿到代码，可以编译，所有的单元测试在持续集成服务器上都能跑过，但是还要花几天才能把这个新版本发布到对外公开的UAT环境中。这种临时安排的商务演示活动算不上不合理吧？

在生产环境中发现了一个严重缺陷，它每天都在让你的公司蒙受损失。你也知道怎么修复它：只要在这个三层架构的系统中，修改那个被这三层都用到的库上的一行代码，然后再修改一下数据库中对应的表即可。但是，上次发布新版本到生产环境时，你花掉了整个周末的时间，而且直到凌晨三点才完活儿。另外，上次执行部署的那个家伙在不久之后因厌倦这样的工作辞职了。你清楚地知道，下次发布肯定不是一个周末就能搞定的。也就是说，该系统在工作日也会停机一段时间。唉，要是业务人员也能理解我们的问题就好了。

虽然这些事都很常见，但这些问题并不是软件开发过程不可避免的产物，它们只是在暗示我们：某个地方做错了。软件发布应该是一个快速且可重复的过程。现在，很多公司都会在一天内发布很多次。甚至对于那些代码非常复杂的代码库来说，这样做也是可能的。我们在本书中就会告诉你如何做到这一点。

Poppendieck夫妇（Mary和Tom）问道："在你的公司里，仅涉及一行代码的改动需要花多长时间才能部署上线？你的处理方式是否可重复且可靠呢？"[①] 从"决定做某种修改"到"该修改结果正式上线"的这段时间称为周期时间（cycle time）。对任何项目而言，它都是一个极为重要的度量标准。

在很多组织中，周期时间的度量单位是周或者月，而且发布过程也是不可重复或不可靠的。部署常常是手工操作过程，甚至将软件部署到测试环境或试运行环境都需要一个团队来完成，更不用说部署到生产环境了。我们遇到过同样复杂的项目，它们曾经也是这种状态，但是经过深入的业务流程重组后，对于某一关键的修改，团队做

---

① *Implementing Lean Software Development*第59页。

到了小时级别甚至分钟级别的发布。之所以能做到，就是因为我们创建了一个完全自动化、可重复且可靠的过程，让变更顺利地经过构建、部署、测试和发布过程。在这里，自动化是关键，它让开发人员、测试人员和运营人员能够通过一键式操作完成软件创建和部署过程中的所有常见任务。

本书将讲述如何缩短从想法到商业价值实现的时间，并使之更安全，从而彻底改变软件交付方式。

软件交到用户手上之后才会为个人或公司带来收入。这是显而易见的事，但在大多数组织中，将软件发布到生产环境的过程是一种手工密集型的、易出错且高风险的过程。虽然几个月的周期时间很常见，但很多公司的情况会更糟糕：发布周期会超过一年。对于大公司，从一个想法到用代码实现它的时间每延迟一周就意味着多出数百万美元的机会成本，而且这些大公司每次发布所经历的时间往往也是最长的。

尽管如此，现代大多数的软件开发项目仍旧没有把低风险软件交付的机制和过程作为其组成部分。

我们的目标是改变软件交付方式，将其由开发人员的手工操作变成一种可靠、可预期、可视化的过程并在很大程度上实现了自动化的流程，而且它要具备易于理解与风险可量化的特点。使用本书所描述的方法，就有可能在几分钟或几个小时内把一个想法变成可交付到生产环境中的工作代码，而且同时还能提高交付软件的质量。

交付成功软件的成本绝大部分都是在首次发布后产生的。这些成本包括技术支持、维护、增加新功能和修复缺陷的成本。通过迭代方式交付的软件更是如此，因为首次发布只会包含能给用户提供价值的最小功能集合。因此本书的书名《持续交付》就来源于敏捷宣言的第一原则："我们的首要任务是尽早持续交付有价值的软件并让客户满意。"[bibNp0]这也反映了这样一个现实：对于成功的软件，首次发布只是交付过程的开始。

书中描述的所有技术都与交付软件新版本给客户相关，旨在减少时间和降低风险。这些技术的核心是增加反馈，改进负责交付的开发、测试和运维人员之间的协作。这些技术能确保当需要修改应用程序（也许是修复缺陷，也许是开发新功能）时，从修改代码到正式部署上线之间的时间尽可能短，尽早发现缺陷以便快速修复，并更好地了解与本次修改相关的风险。

读者对象及本书内容

本书的一个主要目标是改善负责软件交付的相关人员之间的协作，尤其是开发人员、测试人员、系统和数据库管理员以及经理。

本书内容广泛，包括经常提到的配置管理、源代码控制、发布计划、审计、符合性和集成，以及构建、测试和部署流程的自动化。我们也会讲述自动化验收测试、依赖管理、数据库迁移，以及测试和生产环境的创建与管理等技术。

参与过软件开发的很多人认为与写代码相比，这些活动不那么重要。然而，根据我们的经验，它们会消耗大量的时间和精力，而且是成功交付软件的关键因素。当与这些活动相关的风险管理没有做到位时，它们就可能耗费很多资金，甚至经常会超过构建软件本身的成本。本书会告诉你如何了解这些风险，而且更重要的是，会教会你如何降低这些风险。

这个目标很大，我们当然无法在一本书中面面俱到。事实上，我们很有可能会疏远各类目标受众，比如由于没有深入讨论架构、行为驱动的开发和重构等问题而疏远开发人员，或由于没花足够多的篇幅讨论探索性测试和测试管理策略而疏远测试人员，或由于没有特别关注容量计划、数据库迁移和生产环境监控等问题而疏远运维人员。

然而，市面上已经有一些分别详细讨论这些内容的书了。我们认为，真正缺少的是一本讨论如何把各方面（包括配置管理、自动化测试、持续集成和部署、数据管理、环境管理以及发布管理）融合在一起的书。精益软件开发运动告诉我们很多事情，其中有一件就是“整体优化是非常重要的”。为了做到整体优化，用一种整体方法将交付过程中各个方面以及参与该过程的所有人联系在一起是非常必要的。只有当能够控制每一次从引入变更到发布的整个过程时，你才能开始优化和改进软件交付的速度和质量。

我们的目标是提供一个整体方案，并给出这个方案涉及的各种原则。我们会告诉你如何在自己的项目中使用这些实践。我们认为不会有一种一刀切的解决方案可以应对软件开发中的各个方面，更不用说像配置管理和企业系统的运维控制这么大的主题了。然而本书所述的基本内容是可以广泛应用于各种软件项目的，包括大的、小的、高技术要求或短平快的快速收益项目。

在开始实际应用这些原则时你会发现，针对特定场景还需要关于这些方面的更详细信息。本书最后列有一份参考书目，以及一些在线资源链接，你可以在其中找到关于本书中各主题的更详细信息。

本书由三部分组成。第一部分阐述了持续交付背后的一些原则，以及支持这些原则所需的实践。第二部分是本书的核心——我们称为*部署流水线*（deployment pipeline）的一种模式。第三部分更详细地介绍了支持部署流水线的生态系统，包括：让增量开发成为可能的技术；高级版本控制模式；基础设施、环境和数据的管理；治理（governance）。

其中很多技术看上去只适用于大型软件应用。尽管我们的经验多数来自于大型软件应用，但相信即便是极小的项目也可以从这些技术中受益，理由很简单，项目会变大的。当小项目开始时，你的决定会对其发展产生不可避免的影响，通过以正确的方式开始，你会使自己或者后继者在前进的路上减少很多痛苦。

本书作者都认同精益和迭代软件开发理论，即我们的目标是向客户快速并迭代地交付有价值且可工作的软件，并持续不断地从交付流程中消除浪费。我们描述的很多

原则与技术最早都是在大型敏捷软件项目中总结出来的。然而，本书中提到的技术都是通用的。我们的关注点更多的是通过更好的可视化和更快的反馈改善协作。这在每个项目中都会产生积极影响，无论项目是否使用迭代开发过程。

我们尽量做到每一章（甚至每一节）都相对独立，可以分开阅读。至少，我们希望你想了解的内容以及相关的进一步的参考信息是清晰且容易找到的，以便你可以把这本书当做一本工具书来用。

还要说明的一点是，我们并不追求所讨论主题的学术性。市面上与之相关的理论书籍非常丰富，其中很多都非常有趣，也不乏深刻的见解。尤其是，我们不会花太多时间在标准上，而是会关注那些对软件开发中的每个角色都很有用且经过实战检验的技能和技术，并尽量言简意赅地阐明它们，使其在现实中每天都能发挥作用。在适当之处，我们还会提供一些案例，以方便阐述这些技术。

内容简介

我们知道，并不是每个人都想从头到尾读完这本书。所以本书采用了特别的编写方式，使你一旦看过了介绍，就可以从不同的地方开始阅读它。为此，书中会有一定量的重复内容，但是希望不至于让逐页阅读的读者感到啰嗦。

本书包括三部分。第一部分从第1章到第4章，讲述有规律、可重复、低风险发布的基本原则和与其相关的实践。第二部分从第5章到第10章，讲述部署流水线。从第11章开始，我们会深入分析支撑持续交付的生态系统。

建议第1章必读。我们相信，那些刚接触软件开发流程的人，甚至是有经验的开发人员，都会从中发现很多挑战其对专业软件开发原有观点的内容。对于本书的其他部分，你可以在闲暇时翻看，或者在遇到问题时查看。

第一部分——基础篇

第一部分描述了理解部署流水线的前提条件。每章的内容都建立在上一章的基础之上。

第1章首先描述了在很多软件开发团队中常见的反模式，然后阐述了我们的目标以及实现方式。最后总结了软件交付的一些原则，本书的其他内容都以这些原则为基础。

第2章阐述了管理构建、部署、测试和发布应用程序所需的一些要素，包括源代码、构建脚本，以及环境和应用程序配置。

第3章讲述了在程序每个变更后执行构建和运行自动化测试相关的实践，以便确保你的软件一直处于可工作状态。

第4章介绍了每个项目中的各种手工和自动化测试，并讨论了如何确定适合自己项目的策略。

第二部分——部署流水线

本书的第二部分详细介绍了部署流水线，包括如何实现部署流水线中的各种阶段。

第5章介绍了一种模式，即本书的核心——每次代码修改后从提交到发布的一个自动化过程。我们也讨论了如何在团队级别和组织级别实现流水线。

第6章讨论了用于创建自动化构建和部署流程的脚本化技术，以及使用这些脚本的最佳实践。

第7章讨论了部署流水线中的第一个阶段，即任何一次提交都应该触发的自动化过程。我们还讨论了如何创建一个快速、高效的提交测试套件。

第8章展现了从分析一直到实现的自动化验收测试。我们讨论了为什么对于持续交付来说验收测试非常关键，以及如何创建一个成本合理的高效验收测试套件，来保护应用程序中那些有价值的功能。

第9章讨论了非功能需求，并重点介绍了容量测试，内容包括如何创建容量测试，以及如何准备容量测试环境。

第10章讲述自动化测试之后应做什么：一键式将候选发布版本部署到手工测试环境、用户验收测试环境、试运行环境，直至最终发布。其中包括一些至关重要的主题，如持续部署、回滚以及零停机发布。

第三部分——交付生态圈

本书的最后一部分讨论了用于支撑部署流水线的各类交叉实践与技术。

第11章的内容包括环境的自动化创建、管理和监控，包括虚拟化技术和云计算的使用。

第12章讨论了在应用程序的生命周期中，如何创建和迁移测试数据和生产数据。

第13章首先讨论了如何在不使用分支的情况下让应用程序一直处于可发布状态。然后描述了如何将应用程序分解成多个组件，以及如何建立和测试这些组件。

第14章概述了最流行的一些工具，以及使用版本控制的不同模式。

第15章的内容是风险管理和符合度，并提出了配置和发布管理的一个成熟度模型。然后，我们讨论了持续交付带来的商业价值，和迭代增量交付项目的生命周期。

本书中的在线链接

由于外部网站的完整链接很长，所以我们用了短链接，其格式如[bibNp0]。有两种方法打开这个链接，可以使用bit.ly（其URL是http://bit.ly/bibNp0），也可以使用我们安装在http://continuousdelivery.com/go/上的一个短链接服务（Key值是一样的，所以上例中的URL是http://continuousdelivery.com/go/bibNp0），如果bit.ly因为某种原因停止了服务，你还可以用这个链接。如果网页更换了地址，我们会尽量保留http://continuousdelivery.com/go/这个短链接服务，所以如果bit.ly不可用，可以试一试这个。

关于封面的插图

Martin Fowler签名系列的所有书中，其封面都以“桥”为主题。我们原打算使用“英国大铁桥”的照片，但是它已经用在了本系列的另一本书上。所以，我们选择了英国的另一座大桥“福斯铁路桥”，这张美丽的照片由Stewart Hardy拍摄而成。

福斯铁路桥是英国第一座使用钢铁建造的大桥。其钢铁使用最新的西门子-马丁平炉工艺制造，并由在苏格兰的两个钢铁厂和威尔士的一个钢铁厂交付。钢铁是以管状桁架的形式运送的，这是英国首次使用大规模生产的零部件组装桥梁。与早期的桥梁不同，设计师John Fowler爵士，Benjamin Baker爵士和Allan Stewart计算了建筑压发生率，以便减少后续的维护成本，并计算了风压和温度对结构的影响，而这很像软件开发中的功能需求和非功能需求。他们还监督了桥梁建设，以确保这些要求都能得到满足。

当时有4600名工人参与建造该桥，其中不幸死亡约一百人，致残数百人。然而它仍是工业革命的一个奇迹，因为1890年建成时，它是世界上最长的桥，而到了21世纪初，它仍是世界第二长的悬臂大桥。就像长生命周期的软件项目一样，这座桥需要持续维护。这已在设计时考虑到了，大桥配套工程中不但有一个维护车间和场地，而且在Dalmeny车站还有一个50个房间的铁路“聚居点”。据估计，该桥的使用寿命还有一百多年。

版本记录

本书直接利用DocBook[①]写完。David使用了TextMate编辑器，而Jez使用了Aquamacs Emacs。图形是用OmniGraffle画的。David和Jez通常身居地球的不同地方，他们之间的协作是通过Subversion完成的。我们还使用了持续集成技术，工具是CruiseControl.rb，每次有人提交一个更改后，它就会运行dblatex产生本书的PDF。

本书印刷前一个月，Dmitry Kirsanov和Alina Kirsanova开始排版制作，与作者的合作都通过Subversion库、电子邮件和共享的Google Docs表进行协调。Dmitry用XEmacs做DocBook源文件的修编，Alina则完成了其他事情，包括使用定制的XSLT样式表和一个XSL-FO格式化工具进行页面排版，从源文件中的索引标签里编译并编辑索引，并做了本书的最终校订。

① DocBook是一种XML（和SGML）应用程序，可用于编写技术图书和文档，一度成为最流行的XML格式。它非常适合于编写关于计算机硬件和软件的书籍和论文，但绝不限于这些应用。——译者注