PEARSON

持续交付

发布可靠软件的系统方法

（英文版）

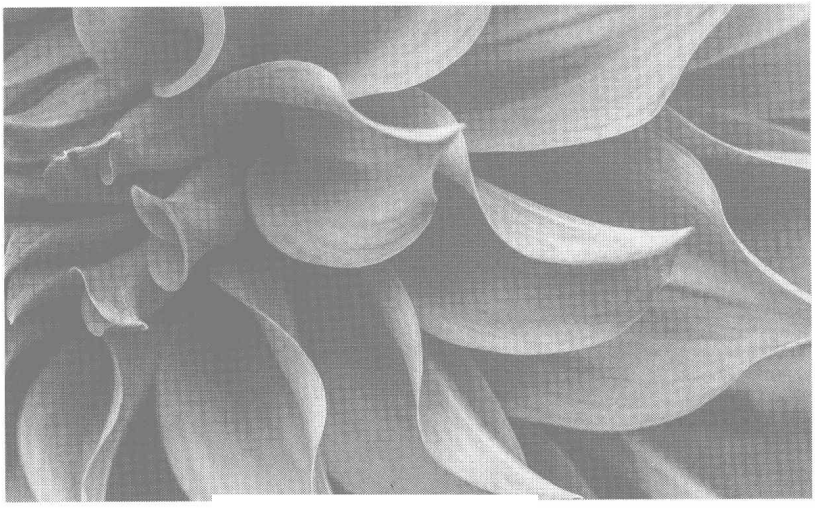
[英] Jez Humble David Farley 著

Continuous Delivery

Reliable Software Releases through Build, Test, and Deployment Automation

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS



持续交付

发布可靠软件的系统方法

（英文版）

[英] Jez Humble David Farley 著

Continuous Delivery

Reliable Software Releases through Build, Test, and Deployment Automation

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 : 英文 / (英) 亨布尔 (Humble,J.) , (英) 法利 (Farley,D.) 著. -- 北京 : 人民邮电出版社, 2015.10
书名原文: Continuous Delivery:Reliable Software Releases through Build,Test,and Deployment Automation
ISBN 978-7-115-40375-9

Ⅰ. ①持… Ⅱ. ①亨… ②法… Ⅲ. ①软件可靠性－研究－英文 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2015)第217124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。

◆ 著 [英] Jez Humble David Farley
责任编辑 杨海玲
责任印制 张佳莹 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 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：29.5
字数：646 千字 2015 年 10 月第 1 版
印数：1－2 000 册 2015 年 10 月北京第 1 次印刷
著作权合同登记号 图字：01-2015-6172 号

定价：89.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

版权声明

This book is dedicated to my dad, who has always given me his unconditional love and support.

—Jez

This book is dedicated to my dad, who always showed me the right direction.

—Dave

Martin Fowler 序[1]

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，而一件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

1. 本序由乔梁翻译。

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。

前言[1]

概述

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？”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

1. 本前言由乔梁翻译。

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。”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。

书中的链接

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。

版本记录

本书直接用 DocBook 写完。David 使用 TextMate 编辑器，而 Jez 使用 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。

Acknowledgments

Many people have contributed to this book. In particular, we'd like to thank our reviewers: David Clack, Leyna Cotran, Lisa Crispin, Sarah Edrie, Damon Edwards, Martin Fowler, James Kovacs, Bob Maksimchuk, Elliotte Rusty Harold, Rob Sanheim, and Chris Smith. We'd also like to extend special thanks to our editorial and production team at Addison-Wesley: Chris Guzikowski, Raina Chrobak, Susan Zahn, Kristy Hart, and Andy Beaster. Dmitry Kirsanov and Alina Kirsanova did a fantastic job of copyediting and proofreading the book, and typesetting it using their fully automated system.

Many of our colleagues have been instrumental in developing the ideas in this book, including (in no particular order) 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, Hu Kai, Qiao Yandong, Qiao Liang, Derek Yang, Julias Shaw, Deepthi, Mark Chang, Dante Briones, Li Guanglei, 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, and Chris Briesemeister.

Jez would like to thank his wife, Rani, for being the most loving partner he could wish for, and for cheering him up when he was grumpy during the writing of this book. He also thanks his daughter, Amrita, for her babbling, cuddles, and big gummy smiles. He is also profoundly grateful to his colleagues at ThoughtWorks for making it such an inspiring place to work, and to Cyndi Mitchell and Martin Fowler for their support of this book. Finally, a big shout out to Jeffrey Fredrick and Paul Julius for creating CITCON, and to the people he met there for many great conversations.

Dave would like to thank his wife Kate, and children Tom and Ben, for their unfailing support at every point, in this project and in many others. He would also like to make a special mention of ThoughtWorks, who, although no longer his employer, provided an environment of enlightenment and encouragement for

the people that worked there, thus fostering a creative approach to finding solutions, many of which populate the pages of this book. In addition, he would like to thank his current employer, LMAX, with a special mention for Martin Thompson, for their support, trust, and willing adoption of the techniques described in this book in an intensely challenging technical environment of world-class high-performance computing.

About the Authors

Jez Humble has been fascinated by computers and electronics since getting his first ZX Spectrum at age 11, and spent several years hacking on Acorn machines in 6502 and ARM assembler and BASIC until he was old enough to get a proper job. He got into IT in 2000, just in time for the dot-com bust. Since then he has worked as a developer, system administrator, trainer, consultant, manager, and speaker. He has worked with a variety of platforms and technologies, consulting for nonprofits, telecoms, financial services, and online retail companies. Since 2004 he has worked for ThoughtWorks and ThoughtWorks Studios in Beijing, Bangalore, London, and San Francisco. He holds a BA in Physics and Philosophy from Oxford University and an MMus in Ethnomusicology from the School of Oriental and African Studies, University of London. He is presently living in San Francisco with his wife and daughter.

Dave Farley has been having fun with computers for nearly 30 years. Over that period he has worked on most types of software—from firmware, through tinkering with operating systems and device drivers, to writing games and commercial applications of all shapes and sizes. He started working in large-scale distributed systems about twenty years ago, doing research into the development of loose-coupled, message-based systems—a forerunner of SOA. He has a wide range of experience leading the development of complex software in teams, both large and small, in the UK and USA. Dave was an early adopter of agile development techniques, employing iterative development, continuous integration, and significant levels of automated testing on commercial projects from the early 1990s. He honed his approach to agile development during his four-and-a-half-year stint at ThoughtWorks where he was a technical principal working on some of their biggest and most challenging projects. Dave is currently working for the London Multi-Asset Exchange (LMAX), an organization that is building one of the highest-performance financial exchanges in the world, where they rely upon all of the major techniques described in this book.

目录