<<持续交付>>

图书基本信息

书名：<<持续交付>>

13位ISBN编号：9787115264596

10位ISBN编号：7115264597

出版时间：2011-10

出版时间：人民邮电

作者：Jez Humble　David Farley

译者：乔梁

<<持续交付>>

内容概要

软件发布是一个令人头痛的过程，非常耗时且风险很高。
本书独特而有条理地阐述了以快速、高效、可靠的方式向用户交付新功能的原则和技术实践。
通过实现自动化的构建、部署和测试过程，并改进开发人员、测试人员、运维人员之间的协作，交付团队可以在几小时（甚至几分钟）内发布软件变更，而这不受项目大小和代码复杂性的影响。

本书首先给出了实现快速、可靠、低风险交付过程的基础知识，然后介绍了部署流水线，即从签入到发布的过程中管理所有变更的一个自动化过程。
最后，书中探讨了支撑持续交付的“交付生态圈”，内容涉及基础设施、数据和配置的管理，以及组织治理。

作者为我们呈现了最新的技术，包括自动化的基础设施管理和数据迁移，以及虚拟化的使用，并分别探讨了各种技术中的关键问题和最佳实践，演示了降低风险的方法。
内容涉及：

?将软件构建、集成、测试和部署全面实现自动化
?在团队级别和组织级别实现部署流水线
?改进开发人员、测试人员和运维人员间的协作
?在大型分布式团队中增量开发软件功能
?实施高效的配置管理策略
?分析并实现自动化验收测试
?容量测试和其他非功能性需求的测试
?实现持续部署和零停机发布
?管理基础设施、数据、组件和依赖
?风险管理、符合度和审计

无论是开发人员、系统管理人员、测试人员，还是经理人员，本书都能前所未有地加速你将想法变成可发布软件的步伐，为企业迅速可靠地增添价值。

作者介绍：（其中两个作者，只有其一有照片，所以就只一个作者和一个译者放照片，另一作者不必放）

Jez Humble
ToughtWorks公司首席咨询顾问，致力于帮助企业快速、可靠地交付高质量软件，经常在各种敏捷技术大会上发表演讲，拥有牛津大学物理学学士学位和伦敦大学民族音乐学的硕士学位。
2000年至今，他曾在各行业和不同技术领域担任系统管理员、开发人员、培训人员、咨询师和经理人员。

David Farley
正在帮助构建伦敦多资产交易所（LMAE）。
他具有20年的大型分布式系统开发经验，是采用敏捷开发技术的先行者，曾作为技术负责人参加了ThoughtWorks公司许多极具挑战性的软件项目。

译者介绍：

乔 梁
拥有多年软件开发及管理经验，对敏捷开发管理及持续交付有深入的理解与丰富的实践经验，专注于提高软件企业的高质量交付能力，推广最佳实践。
为多个大型电信企业、互联网企业提供过专业的软件交付咨询服务。
曾在ThoughtWorks任职多年，现任百度项目管理部高级架构师。

<<持续交付>>

InfoQ特约编辑，主持“ 持续集成 ”专栏。

作者简介

作者:(英)Humble

书籍目录

章节摘录

软件交付的问题　1.1　引言　作为软件从业人员，我们面临的最重要问题就是，如果有人想到了一个好点子，我们如何以最快的速度将它交付给用户？
本书将给出这个问题的答案。
我们将专注于构建、部署、测试和发布过程，因为相对于软件生产全过程的其他环节来说，这部分内容的论著较为稀少。
确切地说，我们并不认为软件开发方法不重要，如果没有对软件生命周期中其他方面的关注，只把它们作为全部问题的次要因素草率对待的话，就不可能实现可靠、迅速且低风险的软件发布，无法以高效的方式将我们的劳动成果交到用户手中。
现在有很多种软件开发方法，但它们主要关注于需求管理及其对开发工作的影响。
市面上也有很多优秀的书，它们详细讨论了在软件设计、开发和测试方面各种各样的方法，但它们都仅仅讲述了将软件交付给作为客户的人或组织这一完整价值流的一部分。
一旦完成了需求定义以及方案的设计、开发和测试，我们接下来做什么？
我们如何协调这些活动，尽可能地使交付过程更加可靠有效呢？
我们如何让开发人员、测试人员，以及构建和运维人员在一起高效地工作呢？
本书描述了软件从开发到发布这一过程的有效模式。
书中讲述了帮助大家实现这种模式的技术和最佳实践，展示了它与软件交付中其他活动是如何联系的。
本书的中心模式是部署流水线。
从本质上讲，部署流水线就是指一个应用程序从构建、部署、测试到发布这整个过程的自动化实现。
部署流水线的实现对于每个组织都将是不同的，这取决于他们对软件发布的价值流的定义，但其背后的原则是相同的。
部署流水线的示例如图1-1所示。
部署流水线大致的工作方式如下。
对于应用程序的配置、源代码、环境或数据的每个变更都会触发创建一个新流水线实例的过程。
流水线的首要步骤之一就是创建二进制文件和安装包，而其余部分都是基于第一步的产物所做的一系列测试，用于证明其达到了发布质量。
每通过一步测试，我都会更加相信这些二进制文件、配置信息、环境和数据所构成的特殊组合可以正常工作。
如果这个产品通过了所有的测试环节，那么它就可以发布了。
图1-1　一个简单的部署流水线　部署流水线以持续集成过程为其理论基石，从本质上讲，它是采纳持续集成原理后的自然结果。
部署流水线的目标有三个。
首先，它让软件构建、部署、测试和发布过程对所有人可见，促进了合作。
其次，它改善了反馈，以便在整个过程中，我们能够更早地发现并解决问题。
最后，它使团队能够通过一个完全自动化的过程在任意环境上部署和发布软件的任意版本。
1.2　一些常见的发布反模式　软件发布的当天往往是紧张的一天。
为什么会这样呢？
对于大多数项目来说，在整个过程中，发布时的风险是比较大的。
在许多软件项目中，软件发布是一个需要很多手工操作的过程。
首先，由运维团队独自负责安装好该应用程序所需的操作系统环境，再把应用程序所依赖的第三方软件安装好。
其次，要手工将应用程序的软件产物复制到生产主机环境，然后通过Web服务器、应用服务器或其他第三方系统的管理控制台复制或创建配置信息，再把相关的数据复制一份到环境中，最后启动应用程序。
假如这是个分布式的或面向服务的应用程序，可能就需要一部分一部分地完成。

如上所述，发布当天紧张的原因应该比较清楚了：在这个过程中有太多步骤可能出错。
假如其中有一步没有完美地执行，应用程序就无法正确地运行。
一旦发生这种情况，我们很难一下子说清楚哪里出了错，或到底是哪一步出了错。
本书其他部分将讨论如何避免这些风险，如何减少发布当天的压力，以及如何确保每次发布的可靠性都是可预见的。
在此之前，让我们先明确到底要避免哪类失败。
下面列出了与可靠的发布过程相对应的几种常见的反模式，它们在我们这个行业中屡见不鲜。
1.2.1　反模式：手工部署软件　对于现在的大多数应用程序来说，无论规模大小，其部署过程都比较复杂，而且包含很多非常灵活的部分。
许多组织都使用手工方式发布软件，也就是说部署应用程序所需的步骤是独立的原子性操作，由某个人或某个小组来分别执行。
每个步骤里都有一些需要人为判断的事情，因此很容易发生人为错误。
即便不是这样，这些步骤的执行顺序和时机的不同也会导致结果的差异性，而这种差异性很可能给我们带来不良后果。
这种反模式的特征如下。
· 有一份非常详尽的文档，该文档描述了执行步骤及每个步骤中易出错的地方。
· 以手工测试来确认该应用程序是否运行正确。
· 在发布当天开发团队频繁地接到电话，客户要求解释部署为何会出错。
· 在发布时，常常会修正一些在发布过程中发现的问题。
· 如果是集群环境部署，常常发现在集群中各环境的配置都不相同，比如应用服务器的连接池设置不同或文件系统有不同的目录结构等。
· 发布过程需要较长的时间（超过几分钟）。
· 发布结果不可预测，常常不得不回滚或遇到不可预见的问题。
· 发布之后凌晨两点还睡眼惺忪地坐在显示器前，绞尽脑汁想着怎么让刚刚部署的应用程序能够正常工作。
相反，随着时间的推移，部署应该走向完全自动化，即对于那些负责将应用程序部署到开发环境、测试环境或生产环境的人来说，应该只需要做两件事：（1）挑选版本及需要部署的环境，（2）按一下“部署”按钮。
对于套装软件的发布来说，还应该有一个创建安装程序的自动化过程。
我们将在本书中讨论很多自动化问题。
当然，并不是所有的人都热衷于这个想法。
那么，我们先来解释一下为什么把自动化部署看做是一个必不可少的目标。
· 如果部署过程没有完全自动化，每次部署时都会发生错误。
唯一的问题就是“该问题严重与否”而已。
即便使用良好的部署测试，有些错误也很难追查。
· 如果部署过程不是自动化的，那么它就既不可重复也不可靠，就会在调试部署错误的过程中浪费很多时间。
· 手动部署流程不得不被写在文档里。
可是文档维护是一项复杂而费时的任务，它涉及多人之间的协作，因此文档通常要么是不完整的，要么就是未及时更新的，而把一套自动化部署脚本作为文档，它就永远是最新且完整的，否则就无法进行部署工作了。
· 自动部署本质上也是鼓励协作的，因为所有内容都在一个脚本里，一览无遗。
要读懂文档通常需要读者具备一定的知识水平。
然而在现实中，文档通常只是为执行部署者写的备忘录，是难以被他人理解的。
· 以上几点引起的一个必然结果：手工部署过程依赖于部署专家。
如果专家去度假或离职了，那你就有麻烦了。
· 尽管手工部署枯燥且极具重复性，但仍需要有相当程度的专业知识。

若要求专家做这些无聊、重复，但有技术要求的任务则必定会出现各种我们可以预料到的人为失误，同时失眠，酗酒这种问题也会接踵而至。
然而自动化部署可以把那些成本高昂的资深高技术人员从过度工作中解放出来，让他们投身于更高价值的工作活动当中。
· 对手工部署过程进行测试的唯一方法就是原封不动地做一次（或者几次）。
这往往费时，还会造成高昂的金钱成本，而测试自动化的部署过程却是既便宜又容易。
· 另外，还有一种说法：自动化过程不如手工过程的可审计性好。
我们对这个观点感到很疑惑。
对于一个手工过程来说，没人能确保其执行者会非常严格地遵循文档完成操作。
只有自动化过程是完全可审核的。
有什么会比一个可工作的部署脚本更容易被审核的呢？
· 每个人都应该使用自动化部署过程，而且它应该是软件部署的唯一方式。
这个准则可以确保：在需要部署时，部署脚本就能完成工作。
在本书中我们会提到多个原则，而其中之一就是“使用相同的脚本将软件部署到各种环境上”。
如果使用相同的脚本将软件部署到各类环境中，那么在发布当天需要向生产环境进行部署时，这个脚本已经被验证过成百上千次了。
如果发布时出现任何问题的话，你可以百分百地确定是该环境的具体配置问题，而不是这个脚本的问题。
当然，手工密集型的发布工作有时也会进行得非常顺利。
有没有可能是糟糕的情况刚巧都被我们撞见了呢？
假如在整个软件生产过程中它还算不上一个易出错的步骤，那么为什么还总要这么严阵以待呢？
为什么需要这些流程和文档呢？
为什么团队在周末还要加班呢？
为什么还要求大家原地待命，以防意外发生呢？
1.2.2　反模式：开发完成之后才向类生产环境部署　在这一模式下，当软件被第一次部署到类生产环境（比如试运行环境）时，就是大部分开发工作完成时，至少是开发团队认为“该软件开发完成了”。
这种模式中，经常出现下面这些情况。
· 如果测试人员一直参与了在此之前的过程，那么他们已在开发机器上对软件进行了测试。
· 只有在向试运行环境部署时，运维人员才第一次接触到这个新应用程序。
在某些组织中，通常是由独立的运维团队负责将应用程序部署到试运行环境和生产环境。
在这种工作方式下，运维人员只有在产品被发布到生产环境时才第一次见到这个软件。
· 有可能由于类生产环境非常昂贵，所以权限控制严格，操作人员自己无权对该环境进行操作，也有可能环境没有按时准备好，甚至也可能根本没人去准备环境。
· 开发团队将正确的安装程序、配置文件、数据库迁移脚本和部署文档一同交给那些真正执行部署任务的人员，而所有这些都没有在类生产环境或试运行环境中进行过测试。
· 开发团队和真正执行部署任务的人员之间的协作非常少。
每当需要将软件部署到试运行环境时，都要组建一个团队来完成这项任务。
有时候这个团队是一个全功能团队。
然而在大型组织中，这种部署责任通常落在多个分立的团队肩上。
DBA、中间件团队、Web团队，以及其他团队都会涉及应用程序最后版本的部署工作。
由于部署工作中的很多步骤根本没有在试运行环境上测试过，所以常常遇到问题。
比如，文档中漏掉了一些重要的步骤，文档和脚本对目标环境的版本或配置作出错误的假设，从而使部署失败。
部署团队必须猜测开发团队的意图。

若不良协作使得在试运行环境上的部署工作问题重重，就会通过临时拨打电话、发电子邮件来沟通，并由开发人员做快速修复。
一个严格自律的团队会将所有这类沟通纳入部署计划中，但这个过程很少有效。
随着部署压力的增大，为了能够在规定的时间内完成部署，开发团队与部署团队之间这种严格定义的协作过程将被颠覆。
在执行部署过程中，我们常常发现系统设计中存在对生产环境的错误假设。
例如，部署的某个应用软件是用文件系统做数据缓存的。
这在开发环境中是没有什么问题的，但在集群环境中可能就不行了。
解决这类问题可能要花很长时间，而且在问题解决之前，根本无法完成应用程序的部署。
一旦将应用程序部署到了试运行环境，我们常常会发现新的缺陷。
遗憾的是，我们常常没有时间修复所有问题，因为最后期限马上就到了，而且项目进行到这个阶段时，推迟发布日期是不能被人接受的。
所以，大多数严重缺陷被匆忙修复，而为了安全起见，项目经理会保存一份已知缺陷列表，可是当下一次发布开始时，这些缺陷的优先级还是常常被排得很低。
有的时候，情况会比这还糟。
以下这些事情会使与发布相关的问题恶化。
· 假如一个应用程序是全新开发的，那么第一次将它部署到试运行环境时，可能会非常棘手。
· 发布周期越长，开发团队在部署前作出错误假设的时间就越长，修复这些问题的时间也就越长。
· 交付过程被划分到开发、DBA、运维、测试等部门的那些大型组织中，各部门之间的协作成本可能会非常高，有时甚至会将发布过程拖上“地狱列车”。
此时为了完成某个部署任务（更糟糕的情况是，为了解决部署过程中出现的问题），开发人员、测试人员和运维人员总是高举着问题单（不断地互发电子邮件）。
· 开发环境与生产环境差异性越大，开发过程中所做的那些假设与现实之间的差距就越大。
虽然很难量化，但我敢说，如果在Windows系统上开发软件，而最终要部署在Solaris集群上，那么你会遇到很多意想不到的事情。
· 如果应用程序是由用户自行安装的（你可能没有太多权限来对用户的环境进行操作），或者其中的某些组件不在企业控制范围之内，此时可能需要很多额外的测试工作。
那么，我们的对策就是将测试、部署和发布活动也纳入到开发过程中，让它们成为开发流程正常的一部分。
这样的话，当准备好进行系统发布时就几乎很少或不会有风险了，因为你已经在很多种环境，甚至类生产环境中重复过很多次，也就相当于测试过很多次了。
而且要确保每个人都成为这个软件交付过程的一份子，无论是构建发布团队、还是开发测试人员，都应该从项目开始就一起共事。
我们是测试的狂热者，而大量使用持续集成和持续部署（不但对应用程序进行测试，而且对部署过程进行测试）正是我们所描述的方法的基石。
1.2.3　反模式：生产环境的手工配置管理　很多组织通过专门的运维团队来管理生产环境的配置。
如果需要修改一些东西，比如修改数据库的连接配置或者增加应用服务器线程池中的线程数，就由这个团队登录到生产服务器上进行手工修改。
如果把这样一个修改记录下来，那么就相当于是变更管理数据库中的一条记录了。
这种反模式的特征如下。
· 多次部署到试运行环境都非常成功，但当部署到生产环境时就失败。
· 集群中各节点的行为有所不同。
例如，与其他节点相比，某个节点所承担的负载少一些，或者处理请求的时间花得多一些。

· 运维团队需要较长时间为每次发布准备环境。
· 系统无法回滚到之前部署的某个配置，这些配置包括操作系统、应用服务器、关系型数据库管理系统、Web服务器或其他基础设施设置。
· 不知道从什么时候起，集群中的某些服务器所用的操作系统、第三方基础设施、依赖库的版本或补丁级别就不同了。
· 直接修改生产环境上的配置来改变系统配置。
相反，对于测试环境、试运行环境和生产环境的所有方面，尤其是系统中的任何第三方元素的配置，都应该通过一个自动化的过程进行版本控制。
本书描述的关键实践之一就是配置管理，其责任之一就是让你能够重复地创建那些你开发的应用程序所依赖的每个基础设施。
这意味着操作系统、补丁级别、操作系统配置、应用程序所依赖的其他软件及其配置、基础设施的配置等都应该处于受控状态。
你应该具有重建生产环境的能力，最好是能通过自动化的方式重建生产环境。
虚拟化技术在这一点上可能对你有所帮助。
你应该完全掌握生产环境中的任何信息。
这意味着生产环境中的每次变更都应该被记录下来，而且做到今后可以查阅。
部署失败经常是因为某个人在上次部署时为生产环境打了补丁，但却没有将这个修改记录下来。
实际上，不应该允许手工改变测试环境、试运行环境和生产环境，而只允许通过自动化过程来改变这些环境。
应用软件之间通常会有一些依赖关系。
我们应该很容易知道当前发布的是软件的哪个版本。
发布可能是一件令人兴奋的事情，也可能变成一件累人而又沉闷的工作。
几乎在每次发布的最后都会有一些变更，比如修改数据库的登录账户或者更新所用外部服务的URL。
我们应该使用某种方法来引入此类变更，以便这些变更可以被记录并测试。
这里我们再次强调一下，自动化是关键。
变更首先应该被提交到版本控制系统中，然后通过某个自动化过程对生产环境进行更新。
我们也应该有能力在部署出错时，通过同一个自动化过程将系统回滚到之前的版本。
1.2.4　我们能做得更好吗　当然可以，本书就是来讲如何做好这件事的。
即使是在一个非常复杂的企业环境中，我们所说的这些原则、实践和技术的目标都是将软件发布工作变成一个没有任何突发事件且索然无味的事情。
软件发布能够（也应该）成为一个低风险、频繁、廉价、迅速且可预见的过程。
这些实践在过去的几年中已经被使用，并且我们发现它们令很多项目变得非比寻常。
本书所提到的所有实践既在具有分布式团队的大型企业项目中验证过，也在小型开发组中验证过。
我们确信它们是有效的，而且可以应用在大项目中。
自动化部署的威力　曾经有个客户，他们在过去每次发布时都会组建一个较大的专职团队。
大家在一起工作七天（包括周末的两天）才能把应用程序部署到生产环境中。
他们的发布成功率很低，要么是发现了错误，要么是在发布当天需要高度干预，且常常要在接下来的几天里修复在发布过程中引入的问题或者是配置新软件时导致的人为问题。
我们帮助客户实现了一个完善的自动构建、部署、测试和发布系统。
为了让这个系统能够良好运行下去，我们还帮助他们采用了一些必要的开发实践和技术。
我们看到的最后一次发布，只花了七秒钟就将应用程序部署到了生产环境中。
根本没有人意识到发生了什么，只是感觉突然间多了一些新功能。
假如部署失败了，无论是什么原因，我们都可以在同样短的时间里回滚。
本书的目标是描述如何使用部署流水线，将高度自动化的测试和部署以及全面的配置管理结合在一起，实现一键式软件发布。
也就是说，只需要点击一下鼠标，就可以将软件部署到任何目标环境，包括开发环境、测试环境或生产环境。

接下来，我们会描述这种模式及其所需的技术，并提供一些建议帮你解决将面临的某些问题。
实现这种方法，实在是磨刀不误砍柴工。
所有这些工作并不会超出项目团队的能力范围。
它不需要刚性的流程、大量的文档或很多人力。
我们希望，读完本章以后，你会理解这种方法背后的原则。
1.3 如何实现目标 正如我们所说，作为软件从业者，我们的目标是尽快地向用户交付有用的可工作的软件。
速度是至关重要的，因为未交付的软件就意味着机会成本。
软件发布之时就是投资得到回报之时。
因此，本书有两个目标，其中之一就是找到减少周期时间（cycle time）的方法。
周期时间是从决定进行变更的时刻开始，包括修正缺陷或增加特性，直至用户可以使用本次变更后的结果。
快速交付也是非常重要的，因为这使你能够验证那些新开发的特性或者修复的缺陷是否真的有用。
决定开发这个应用程序的人（我们称为客户）会猜测哪些特性或缺陷修复对用户是有用的。
然而，直到使用者真正使用之前，这些全是未经过验证的假设。
这也是为什么减少周期时间并建立有效反馈环如此重要的原因。
有用性的一个重要部分是质量。
我们的软件应该满足它的业务目的。
质量并不等于完美，正如伏尔泰所说“追求完美是把事情做好的大敌”，但我们的目标应该一直是交付质量足够高的软件，给客户带来价值。
因此，尽快地交付软件很重要，保证一定的质量是基础。
因此，我们来调整一下目标，即找到可以以一种高效、快速、可靠的方式交付高质量且有价值的软件的方法。
我们及我们的同修发现，为了达到这些目标（短周期、高质量），我们需要频繁且自动化地发布软件。
为什么呢？
· 自动化。
如果构建、部署、测试和发布流程不是自动化的，那它就是不可重复的。
由于软件本身、系统配置、环境以及发布过程的不同，每次做完这些活动以后，其结果可能都会有所不同。
由于每个步骤都是手工操作，所以出错的机会很大，而且无法确切地知道具体都做了什么。
这意味着整个发布过程无法得到应有的控制来确保高质量。
常常说软件发布像是一种艺术，但事实上，它应该是一种工程学科。
· 频繁做。
如果能够做到频繁发布，每个发布版本之间的差异会很小。
这会大大减少与发布相关的风险，且更容易回滚。
频繁发布也会加快反馈速度，而客户也需要它。
本书很多内容都聚焦于如何尽快得到对软件及其相关配置所做变化的反馈，这包括其环境、部署过程及数据等。
对于频繁地自动化发布来说，反馈是至关重要的。
下面关于反馈的三个标准是很有用的： · 无论什么样的修改都应该触发反馈流程；
· 反馈应该尽快发出； · 交付团队必须接收反馈，并依据它作出相应的行动。
让我们逐一审视一下这三个标准，考虑如何能达到这样的标准。
1.3.1 每次修改都应该触发反馈流程 一个可工作的软件可分成以下几个部分：可执行的代码、配置信息、运行环境和数据。
如果其中任何一部分发生了变化，都可能导致软件的行为发生变化。

所以我们要能够控制这四部分，并确保任何修改都会被验证。
　　当修改了源代码后，可执行代码当然也就会随之发生变化。
因此每当修改源代码后，都要进行构建和测试。
为了能够控制这个流程，构建可执行代码并对其进行测试都应该是自动化的。
每次提交都对应用程序进行构建并测试，这称作持续集成。
我们会在第3章详细描述它。
　　之后的部署活动中都应该使用这个构建并测试后的可执行代码，无论是部署至测试环境，还是生产环境。
如果你的应用软件需要编译，你应该确保在所有需要可执行代码的地方都使用在构建流程中已生成的这个，而不是再重新编译一次生成一个新的。
　　对环境的任何修改都应该作为配置信息来管理。
无论在什么环境下，对于应用程序配置的变更都应该被测试。
如果用户自己安装软件的话，任何可能的配置项都应该在各种具有代表性的环境上测试。
 配置管理将在第2章中讨论。
　　如果需要修改该应用程序所要被部署的运行环境，那么整个系统都应该在修改后的环境中进行测试。
这包括对操作系统配置、该应用程序所依赖的软件集、网络配置，以及任何基础设施和外部系统的修改。
第11章会讲基础设施和环境的管理，包括自动化地创建及维护测试环境和生产环境。
　　如果是数据结构发生了变化，这些变化也同样要经过测试。
我们在第12章讨论数据管理。
　　什么是反馈流程？
它是指完全以自动化方式尽可能地测试每一次变更。
根据系统的不同，测试会有所不同，但通常至少包括下面的检测。
　　· 创建可执行代码的流程必须是能奏效的。
这用于验证源代码是否符合语法。
　　· 软件的单元测试必须是成功的。
这可以检查应用程序的行为是否与期望相同。
　　· 软件应该满足一定的质量标准，比如测试覆盖率以及其他与技术相关的度量项。
　　· 软件的功能验收测试必须是成功的。
这可以检查应用是否满足业务验收条件，交付了所期望的业务价值。
　　· 软件的非功能测试必须是成功的。
这可以检查应用程序是否满足用户对性能、有效性、安全性等方面的要求。
　　· 软件必须通过了探索性测试，并给客户以及部分用户做过演示。
这些通常在一个手工测试环境上完成。
此时，产品负责人可能认为软件功能还有缺失，我们自己也可能发现需要修复的缺陷，还要为其写自动化测试来避免回归测试。
　　运行测试的这些环境应该尽可能与生产环境相似，从而验证对于环境的任何修改都不会影响应用程序的正常运行。
　　1.3.2　必须尽快接收反馈　　快速反馈的关键是自动化。
对于实现完全自动化过程来说，唯一的约束条件就是你能够使用的硬件数量。
如果是手工过程，我们可以通过人力来完成这个工作。
然而，手工操作会花更长的时间，可能引入更多的错误，并且无法审计。
另外，持续做手工构建、测试和部署非常枯燥而且有重复劳动，与人力资源利用率的准则相悖。
人力资源是昂贵且非常有价值的，所以我们应该集中人力来生产用户所需要的新功能，尽可能快速地交付这些新功能，而不是做枯燥且易出错的工作。
像回归测试、虚拟机的创建和部署这类工作最好都由机器来完成。

当然，实现这样的部署流水线是需要大量资源的，尤其是当有了全面的自动化测试套件之后。
部署流水线的关键目的之一就是对人力资源利用率的优化：我们希望将人力释放出来做更有价值的工作，将那些重复性的体力活交给机器来做。
对于整个流水线中的提交（commit）阶段，其测试应具有如下特征。
· 运行速度快。
· 尽可能全面，即75%左右的代码库覆盖率。
只有这样，这些测试通过以后，我们才对自己写的软件比较有信心。
· 如果有测试失败的话，就表明应用程序有严重问题，无论如何都不能发布。
也就是说，像检查界面元素的颜色是否正确这类测试不应该包含在这个测试集合当中。
· 尽可能做到环境中立。
这个环境没必要和生产环境一模一样，可以相对简单廉价一些。
相对而言，提交阶段之后的测试一般有如下这些特点。
· 它们通常运行更慢一些，所以适合于并行执行。
· 即使某些测试有可能失败，但在某种场合下，我们还是会发布应用程序。
比如某个即将发布的版本有一个不稳定的修复，会导致其性能低于预先定义的标准，但有时我们还是会决定发布这个版本。
· 它们的运行环境应该尽可能与生产环境相同。
除了测试功能以外，它同时还会对部署过程以及对生产环境的任何修改进行测试。
先经过一轮测试（在便宜的硬件上运行最快的那些测试）之后，再经过这种测试过程，会让我们对软件更有信心。
如果这些测试失败了，这个构建版本就不会再进入后续阶段，这样就可以更好地利用资源。
第5章中会详细介绍流水线技术，而第7、8、9章中会分别讲述提交测试阶段、自动化验收测试，以及非功能需求的测试。
这种方法的基础之一就是快速的反馈。
为了确保对变更的快速反馈，我们就要注意开发软件的流程，特别是如何使用版本控制系统和如何组织代码。
开发人员应该频繁提交代码到版本控制系统中，像管理大规模团队或分布式团队那样，将代码分成多个组件。
在大多数情况下，应该避免使用分支。
我们将在第13章讨论增量式交付以及组件的使用，在第14章中讨论分支与合并。
……

编辑推荐

Jez Humble编著的《持续交付(发布可靠软件的系统方法)》是一本软件工程师的职场指南，以大量虚构的名字和情景描述了极客的日常工作，对他们常遇到的各类棘手问题给予了巧妙回答。
作者以自己在苹果、网景等公司中面临的生死攸关的时刻所做的抉择为例，总结了在硅谷摸爬滚打的经验，旨在为软件工程师更好地规划自己的职业生涯提供帮助。
本书适合软件工程师以及所有职场人士阅读。

版权说明