

持续集成实践

兰洋 温迎福 编著



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系QQ1779903665.

PDF代找说明：

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签索引和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ1779903665。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

声明：本人只提供代找服务，每本100%索引书签和目录，因寻找pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。

本书要点

第1部分 持续集成：介绍篇

介绍一些行业的持续集成解决方案，持续集成的特点及适用范围。

- ◎持续集成解决了什么问题
- ◎何谓持续集成
- ◎持续集成的核心价值
- ◎持续集成实践步骤
- ◎持续集成如何实施

第2部分 持续集成：玩转Jenkins

介绍持续集成如何进行一键式部署，如何配置，如何结合当前主流的工具进行应用。

- ◎持续集成工具Jenkins
- ◎搭建Jenkins 环境
- ◎Jenkins 的系统配置及使用说明
- ◎Jenkins 与Ant、Maven 结合
- ◎持续评审、持续部署与持续反馈

第3部分 自动化测试篇：Jenkins+Selenium

介绍持续集成与主流Web测试工具Selenium和TestNG的结合应用。

- ◎自动化测试工具之Selenium
- ◎Selenium入门
- ◎基于Selenium 封装的测试框架
- ◎自动化测试持续集成

第4部分 附录

- ◎技能储备
- ◎持续集成相关资源
- ◎名词解释

上架建议：软件工具 > 测试



博文视点Broadview



@博文视点Broadview



策划编辑：李 冰
责任编辑：郑柳洁
封面设计：李 玲

ISBN 978-7-121-26238-8



9 787121 262388 >

定价：49.00元

测试实践丛书

持续集成实践

兰洋 温迎福 编著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书分为三大部分，第一部分介绍一些行业的持续集成解决方案，持续集成的特点及适用范围；第二部分介绍持续集成如何进行一键式部署等；第三部分介绍持续集成与主流 Web 测试工具 Selenium 和 TestNG 的结合应用。

本书的预期读者主要为项目经理、CTO、开发经理、测试经理等。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目(CIP)数据

持续集成实践 / 兰洋，温迎福编著. —北京：电子工业出版社，2015.7

(测试实践丛书)

ISBN 978-7-121-26238-8

I. ①持… II. ①兰… ②温… III. ①软件质量—质量管理 IV. ①TP311.5

中国版本图书馆 CIP 数据核字(2015)第 120333 号

策划编辑：李 冰

责任编辑：郑柳洁

印 刷：三河市双峰印刷装订有限公司

装 订：三河市双峰印刷装订有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：15.25 字数：366 千字

版 次：2015 年 7 月第 1 版

印 次：2015 年 7 月第 1 次印刷

印 数：3000 册 定价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010) 88258888。

序

我，没有专家的光环，没有博士的高帽，没有国外知名软件公司的背景；我，就是那芸芸众生中的一分子；我，就是那众多读者中的一员。相信很多读者都曾这样想过：出书的人离我们生活的世界很远，很远……但一本好书对读者而言，离自己的心灵，真的很近，很近……

我相信很多读者都碰到过同样的问题：得知某某教授，某某专家，某某成功人士，出了某某书，于是我买，我搜索，我参考，希望能给工作中遇到的困难或瓶颈予指导帮助，但通常的结果是，理想很丰满，现实很骨感。此时我多么希望，能找到一些好的资料、好的案例，真正为有需要的读者提供帮助。

我及我们这个时代的人，以及我们的前辈，很多人都是这样痛苦地走过来的，只能靠自己琢磨钻研，但，我不希望我们的后辈，刚刚进入职场的新人，重新走我们的老路。在这里，我希望可以用我工作上的一些经验、积累和沉淀，帮助他们少走一些弯路。

我想起一件亲身经历的事情，2009年，我在一家金融公司做关于 OPENSSL 的性能测试，当时解决方案是封装 DLL 用 LR 调用。一年后，我和在另一家公司工作的朋友聊天时，发现他们也在做 OPENSSL 的性能测试，而且在找解决方案，我当时直接把我之前做过的脚本发给了他，他告诉我，参考我的脚本至少使他提前一个月完成任务。也许 2009 年我在做一件事情的时候，在某个地方，有一个或多个和我做一样事情的人，但我们不知道彼此；而我做过的事情，也许未来的某一天，某一个人或某些人会用到，我的及时分享，能帮到他们。

于是，我有了一个出书的种子埋在心里。2011 年，我在一家美国企业工作，那时我刚开始接触持续集成，我们可以快速构建、快速部署环境、快速测试。那时候的我们，尝到了甜头。原来构建可以如此简单，环境搭建可以如此快速，测试可以如此方便，原来懒人可以这样去玩。研究是在快乐和皱眉中前进的，当时除了看官网的帮助文档，就是看官网的帮助文档。当时我多想看到持续集成分享的实例啊，想告诉自己，自己不是一个人在战斗。

现在，我们的测试团队非常成熟地使用持续集成做测试，帮助开发人员快速构建，缩短项目周期。真心希望我们的实践经验能帮助更多的人，给更多的人提供我们的干货。

感谢我的好友陈能技，是他帮我联系出版社，使我从一个出版菜鸟了解了如何出书；感谢我的测试团队，是我们一起努力将持续集成应用到我们的项目团队中，一起解决问题一起前行；感谢电子工业出版社，给我们这样一个好的分享平台；最后，感谢家人和朋友的大力支持。希望本书能真正帮助到在持续集成道路上有困惑的人。

关于本书

持续集成，我相信这个概念大家并不陌生，但关于持续集成的实践，并没有想象中那

么多。

本书分为三大部分，第一部分介绍一些行业的持续集成解决方案，持续集成的特点及适用范围；第二部分介绍持续集成如何进行一键式部署，如何配置，如何结合当前主流的工具进行应用；第三部分介绍持续集成与主流 Web 测试工具 Selenium 和 TestNG 的结合应用。

这本书写给技术性读者。也许你并没有持续集成的经验，也许你有应用持续集成的想法，并想了解它可以给你的团队带来哪些好处。或者，你可能已经开始使用 Hudson 或者 Jenkins，并且要找一些资料把你的持续集成做得更进一步。相信选择这本书可以给你带来一些有用的答案。

本书的预期读者主要为项目经理、CTO、开发经理、测试经理等，特别是做快速迭代的团队，希望本书能给他们带来帮助。

目录

第 1 部分 持续集成：介绍篇

第 1 章 持续集成解决了什么问题	2
1.1 提高软件质量	2
1.2 节约时间，缩短项目发布周期	5
1.3 便捷部署	7
1.4 增强项目的可见性	8
1.5 建立团队对开发产品的信心	9
1.6 解决项目管理的困惑	10
1.7 总结	11
第 2 章 何谓持续集成	12
2.1 持续集成的定义	12
2.2 持续集成的特点	12
2.3 原则	15
2.4 总结	17
第 3 章 持续集成的核心价值	18
3.1 价值点	18
3.2 减少风险	19
3.3 根据变更构建软件	20
3.4 总结	22
第 4 章 持续集成实践步骤	23
4.1 如何选取最佳解决方案	23
4.2 持续集成实践计划	26
4.3 持续集成实践风险	31
4.4 总结	32
第 5 章 持续集成如何实施	33
5.1 场景一：Jenkins+版本控制	33
5.2 场景二：Jenkins+Selenium	37

5.3 场景三: Jenkins+Android	43
5.4 场景四: Jenkins+GitHub	52
5.5 总结	58

第 2 部分 持续集成: 玩转 Jenkins

第 6 章 持续集成工具 Jenkins	62
----------------------------	----

6.1 持续集成工具介绍	62
6.2 为什么选用 Jenkins	68
6.3 Jenkins 简介	69
6.4 总结	71

第 7 章 搭建 Jenkins 环境	72
---------------------------	----

7.1 Jenkins 的官网地址	72
7.2 安装环境	72
7.3 在 Windows 系统中安装 Jenkins	78
7.4 在 Linux 系统中安装 Jenkins	79
7.5 Jenkins 的目录结构	80
7.6 总结	81

第 8 章 Jenkins 的系统配置及使用说明	82
--------------------------------	----

8.1 Jenkins 的系统配置	82
8.2 插件管理	89
8.3 权限设置	89
8.4 Jenkins 中 slave 节点的应用	92
8.5 新建一个构建	93
8.6 控制台操作	100
8.7 例子	104
8.8 Jenkins 维护之升级	110
8.9 Jenkins 维护之备份	111
8.10 总结	113

第 9 章 Jenkins 与 Ant、Maven 结合	114
------------------------------------	-----

9.1 Ant 简介	114
9.2 在 Jenkins 中配置 Ant 环境	114
9.3 用 Ant 构建项目	115
9.4 Ant 的常用命令	116
9.5 Maven 介绍	118

9.5.1	Maven 简介	118
9.5.2	Maven 的安装	119
9.5.3	Maven 坐标详解	120
9.5.4	Maven 的生命周期与命令行	122
9.6	在 Jenkins 中配置 Maven 环境	124
9.7	用 Maven 构建项目	125
9.8	总结	127
第 10 章	持续评审、持续部署与持续反馈	128
10.1	在 Jenkins 中配置 Checkstyle	128
10.2	在 Jenkins 中配置 FindBugs	129
10.3	在 Jenkins 中配置 Publish over SSH	132
10.4	在 Jenkins 中配置 Weblogic 项目的部署	134
10.5	在 Jenkins 中配置 Tomcat 项目的部署	135
10.6	Jenkins 中邮件的配置	136
10.7	配置构建完成后自动发送邮件	142
10.8	总结	144
 第 3 部分 自动化测试篇：Jenkins+Selenium		
第 11 章	自动化测试工具之 Selenium	146
11.1	Selenium 的定义	147
11.1.1	自动化测试的定义	147
11.1.2	Selenium 是优秀的 Web 测试工具	148
11.2	Selenium 1.0 与 Selenium 2.0	148
11.3	浏览器的支持	149
11.4	Selenium RC 的原理	150
11.5	Firefox 的安装	150
11.6	Selenium IDE 的安装	151
11.7	Firebug	152
11.7.1	Firebug 简介	152
11.7.2	Firebug 的安装	153
11.7.3	Firebug 定位页面元素	154
11.8	Java 开发环境的配置	156
11.9	Eclipse	158
11.9.1	Eclipse 简介	158
11.9.2	Eclipse 的安装	159
11.9.3	Eclipse 的常用快捷键	159

11.10 Eclipse 插件安装	160
第 12 章 Selenium 入门	163
12.1 Selenium IDE 的用法	163
12.1.1 Selenium IDE 脚本的录制与回放	163
12.1.2 Selenium IDE 脚本的调试	164
12.1.3 Selenium IDE 脚本的导出	166
12.2 XPath 的简介与应用	169
12.2.1 XPath 简介	169
12.2.2 XPath 中节点的定位	171
12.3 Selenium 2.0 基础	172
12.3.1 下载 Selenium lib 包	172
12.3.2 打开浏览器	173
12.3.3 打开测试页面	173
12.4 如何在 Selenium 中查找与定位页面元素	174
12.4.1 By ID	174
12.4.2 By Name	174
12.4.3 By className	174
12.4.4 By XPath	174
12.5 Selenium 如何操作页面元素	175
12.5.1 输入框	175
12.5.2 按钮	175
12.5.3 下拉选择框	175
12.5.4 弹出对话框	176
12.5.5 导航	176
12.5.6 上传文件	176
12.5.7 拖曳	177
12.5.8 双击	177
12.5.9 右键菜单	177
12.6 高级应用	177
12.6.1 读取 Cookie	177
12.6.2 调用 JavaScript	178
12.6.3 截图	178
12.6.4 页面的隐式等待	178
12.6.5 页面的显式等待	178
12.6.6 设置 profile 属性	179
12.7 其他	179

第 13 章 基于 Selenium 封装的测试框架	180
13.1 框架简介	180
13.1.1 框架特色	181
13.2 浏览器支持	181
13.2.1 Firefox	182
13.2.2 IE	182
13.3 Maven 管理	183
13.4 TestNG 工具	184
13.4.1 监听	187
13.5 关键字驱动	192
13.6 报告	193
13.6.1 日志	193
13.6.2 结果统计	197
13.7 其他功能	201
13.7.1 高亮	201
13.7.2 智能提醒	202
第 14 章 自动化测试持续集成	204
14.1 持续集成的基础配置	204
14.1.1 选择 JDK 的版本	204
14.1.2 配置源码管理方式	204
14.1.3 测试频率	205
14.1.4 配置 Maven	206
14.1.5 Windows 批处理命令设置	206
14.2 分布式测试执行	208
14.3 测试报告集成	210
附录 A 技能储备	227
附录 B 持续集成相关资源	228
附录 C 名词解释	230
后记	234

第 1 部分 持续集成：介绍篇

一个程序员，可以轻松地进行编译、测试和发布，而他的痛苦之处在于他庞大的编码工作量；当多个程序员协同工作时，开发 Leader 会将程序分解，并分到每个程序员手中。每个程序员的编码工作量减少了，但所有的代码都开发完成之后需要再集成到一起进行测试，工作量同样不少。同时，由于很多 BUG 在项目的早期就存在，到最后集成时才发现问题导致开发者需要在集成阶段花费大量的时间寻找 BUG 的根源，加上软件的复杂性，问题的根源很难定位，甚至出现不得不调整底层架构的情况，在这个阶段的“除虫会议”（BUG Meeting）特别多，会议的内容基本都是讨论 BUG 是怎么产生的，最后往往发展成为不同模块的负责人互相推诿责任。就在这样的背景下，极限编程、持续集成相继诞生。

为什么我们要提到极限编程。“持续集成（Continue Integration）”一词其实来源于极限编程（Extreme Programming），作为它的 12 个实践之一出现。Matthew Foemmel 将极限编程中的指导思想转换为具体的行动，从而让我们看到了项目从少而繁杂的集成进步到如今我们轻松愉快地集成。

在本书第 1 部分里，我们提供一个简单的读书路径，我们相信读者在读书前都会带着自己的问题。

持续集成到底可以解决哪些问题？

到底什么是持续集成？

持续集成的核心价值是什么？

持续集成具体怎么使用？

如何玩转持续集成工具？

持续集成和 Selenium 在一起，碰撞出了什么火花？

接下来，我们就来看看持续集成都解决了哪些问题。

第 1 章

持续集成解决了什么问题

1.1 提高软件质量

如何提高软件的质量已经不再是一个纯粹的技术问题，而是一个工程问题。

自计算机诞生以来，相应的软件开发就存在。由于早期的计算机运行性能较低，软件的可编程范围也较狭窄，因此质量问题就没有那么突出。20 世纪 50 年代后期到 20 世纪 60 年代，高级语言相继诞生并得到了广泛的应用，随之而来的是软件规模越来越庞大，越来越复杂。伴随着软件得到越来越广泛的应用，软件的质量问题变得越来越突出。根据美国国家宇航局 NASA 的统计，在 20 世纪 80 年代初，软件引起的故障与硬件引起的故障比率约为 1.1:1.0，到了 20 世纪 80 年代末，这一比率已达到 2.5:1.0。20 世纪 90 年代及 2000 年以后，质量已经被所有的公司重视。软件 BUG 产生的问题会让公司损失很多客户，互联网公司就是其中的典范，它们的目标是将用户体验做到极致，让软件覆盖越来越多的用户使用场景，让用户用着爽。如果 BUG 多，那么何谈优秀的用户体验呢？恐怕连极致体验的边都触碰不到。

因此，如何提高软件的质量成为软件工程研究的一个重点。自从软件危机产生以来，出现了很多提高产品质量的理论和方法，有的从技术角度出发，例如面向对象技术的产生和推广，第四代语言的诞生等；有的从自动化工具入手，例如 CASE 工具、过程控制软件、自动化管理平台等；有的从过程模型角度出发，例如迭代模型、螺旋模型、RUP、IPD、净室软件工程等；也有从管理角度出发的，例如团队管理、绩效管理、PSP、TSP 等；也有从测试角度出发的，例如加强全流程的测试等；一些相应的规范和标准也孕育而生，例如 ISO9000 系列、CMM、QMS 等。

每一种解决质量问题的方法都不是独一无二的，软件质量的提高应该是一个综合的因

素，需要从每个方面进行改进，同时还需要兼顾成本和进度。作为软件产品的销售人员、市场人员或维护人员，经常会受到客户的指责或抱怨，客户说：你们产品的质量太差，不稳定等。那么什么是质量呢？我们该如何衡量质量呢？质量具有三个维度：符合目标、符合需求，以及符合实际需求。目标是客户定义的，符合目标即判断我们是不是在做需要做的事情。实际需求包括用户明确说明的和隐含的需求。符合用户习惯，这其中包含了我们要引导用户如何操作或者去适应用户本身的操作习惯。

ISO 关于质量的定义如下：“一个实体（产品或服务）的所有特性，基于这些特性可以满足明显的或隐含的需要。”

注意，在这个定义中包含明显的需求和隐含的需求。我们往往会忽略隐含的需求。因此，一方面，在控制一个产品的质量的过程中必须关注这些隐含的需求，并给予应有的验证。另一方面，因为我们的产品是为客户提供服务的，因此凡是不满足客户需求的，我们都认为是失败的（failure）。所以我们的产品必须始终围绕着客户的需求进行开发和验证。

其实在一个软件的需求收集过程中需要关注客户和用户，而我们经常会忽略客户与用户之间的区别。那么谁是客户？谁是用户呢？简单来说，客户是真正能够决定是否购买你的软件的人，而用户是实际使用软件的人。了解了这个区别，你在分析需求的重要性时可以参考。同时，在产品质量验证时也可以做出不同的权衡。另一方面，我们在考虑用户需求时，往往只考虑了实际使用软件的人员，而忽略了其他人员对软件的要求或对软件造成的潜在竞争，这包括维护人员的要求，系统管理人员的要求，软件上下游人员的要求，先前版本的情况，市场上竞争对手的软件情况等。

提到质量的时候，我们经常会遇到下列矛盾，在这些矛盾中隐含着对质量的承诺：质量需要一个承诺，尤其是高层管理者的承诺。但为了得到质量，高层管理者必须和其雇用的员工进行紧密合作；许多人相信没有 BUG 的产品和服务是不可能的，但是将缺陷数控制在一定级别是正常并可接受的；质量经常和成本紧密联系在一起，一个高质量的产品同时也意味着高投入，这是设计的质量和一致性质量的一个矛盾。一个高质量的产品的需求规格说明书要足够详细，以便产品可以根据这些规格说明书进行定量分析。

目前流行的流程方法有很多种，不同的过程模型适合于不同类型的项目。瀑布模型是应用最广泛的一种模型，也是最容易理解和掌握的模型，然而它的缺陷也是显而易见的。遗漏的需求或者不断变更的需求会使该模型无所适从。然而，对于那些容易理解但很复杂的项目，采用瀑布模型会是比较适合的，因为你可以按部就班地处理复杂的问题。在质量要求高于成本和进度要求时，该模型的表现尤其突出。

螺旋模型也是一个经典模型，它关注于发现和降低项目的风险。项目从小规模开始，然后探测风险，制定风险控制计划，接着确定项目是否继续，若继续则进行下一个螺旋的反复。该模型的最大优点就是随着成本的增加，风险程度随之降低。然而螺旋模型的缺点是它比较复杂，且需要管理人员有责任心、专注，以及有管理方面的经验。RUP 工作流程示意图和 IPD 工作流程示意图如图 1-1 和图 1-2 所示。

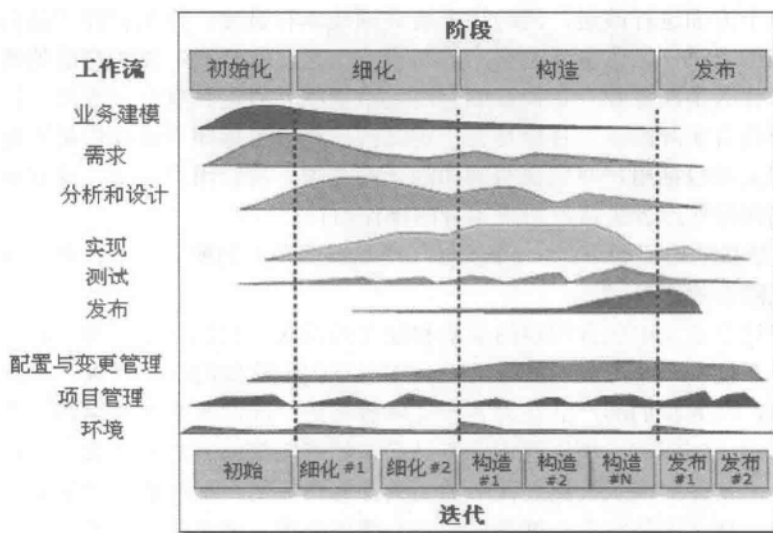


图 1-1 RUP 工作流程示意图



图 1-2 IPD 工作流程示意图

从以上两个模型中我们不难发现，所有的流程图例中都有开发一环，开发都有构建这一环，而在构建及发布环节上，持续集成帮你节约的时间，以及你根据你的实际项目需要而设定的构建频率，可以很好地帮助你在不断解决自己缺陷的同时，不影响团队其他人的代码质量。

BUG 的增加与时间并不是线性增长的关系，而是和时间的平方成正比，两次集成间隔

的时间越长,BUG 增加的数量就越多(超过你的预期),解决 BUG 的工作量也越大,而你越觉得工作量大,你就越想推迟到以后去集成。你甚至企图到最后一次性解决问题,结果 BUG 产生得更多,导致下一次集成的工作量更大,你越觉得集成痛苦,就越将集成的时间推后,最后形成恶性循环。

高效的持续集成模式,即在分批提交代码的时候,就已经进行了测试,测试一直持续到程序开发完毕,一直延续到功能开发完,测试完,再开发完,再测试完,避免了在代码累积到一定量进行集成时,造成的代码质量低下的风险。

如果出现问题,项目成员马上就会被通知,问题会在第一时间被修复。不采用持续集成的情况下,这些问题有可能到交付前的集成测试时才发现,这有可能导致延迟发布产品,而在急于修复这些缺陷时又有可能引入新的缺陷,最终可能导致项目失败。有人不禁会问,没有持续集成,我们的项目不是一样交付么?这么想当然没错。如果持续集成可以让你交付的代码质量更高,你,何乐而不为呢?

1.2 节约时间,缩短项目发布周期

快速开发和上市一个新产品,并快速取得预期的投资回报是每个企业孜孜以求的目标。但是事与愿违,很多新产品项目因盲目地追求开发进度而中途夭折,有些项目即使按期上市了也并未取得预期的投资回报。那么,如何在保证项目质量的前提下,尽可能地加快项目进度呢?有效的方法有很多,制作如图 1-3 所示的项目计划示意图就是其中一种。

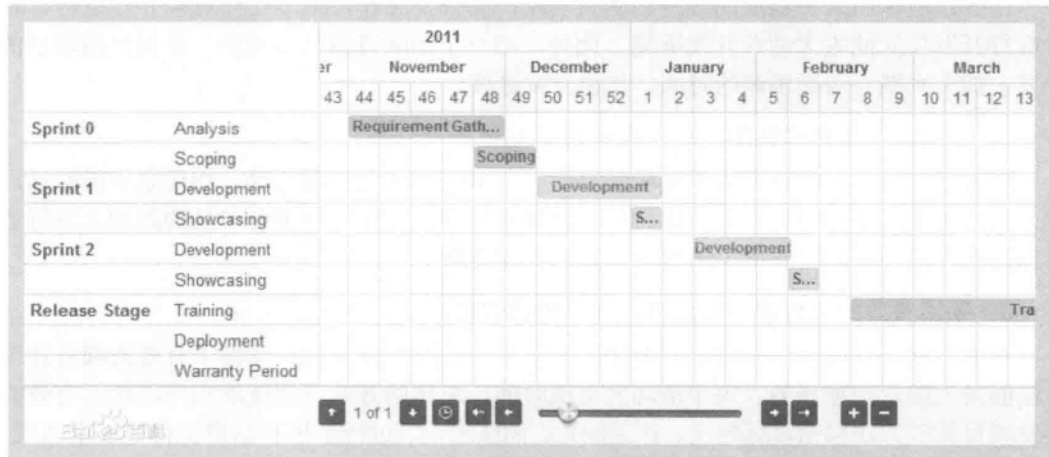


图 1-3 项目计划示意图

1. 深入了解顾客需求,减少开发过程中的需求变更与返工

顾客需求是新产品开发的输入,如果输入不正确、不完整,就必然导致开发过程中大量的变更,这对开发进度影响极大。根据国际最佳实践,改进型的项目在正式开发前应该有 80% 的需求被确定,突破性的新产品在开发前也至少应有 50% 的需求被确定。进行顾客需求调研时最好请市场人员与技术人员参与,不同背景的工作人员能从客户方了解到更为完整的信息。

市场人员与技术人员一起进行产品定义,可以减少信息传递过程的失真,减少开发过程工作的返工。

新产品项目的顾客需求输入不应该由市场人员“提供”给技术人员,而应该请市场人员与技术人员坐在一起进行多次沟通与交流。比较复杂的产品可以采用 QFD 方法进行产品定义。清晰的产品定义能大幅减少开发过程的返工工作。

采用跨职能团队合作方式进行项目立项分析工作,以节省开发、测试、制造和上市等的周期时间。

影响新产品上市时间的不止是开发过程,测试、制造和上市准备等过程中如果出现大量反复,也会严重影响上市时机。例如,一些企业在新产品样机出来后发现开模很难、关键部件难以采购、生产工艺达不到要求等,这将大大延长新产品的上市周期,甚至导致新产品项目失败。从立项分析阶段开始就采用跨职能团队合作模式一起协同工作,能有效缩短样机出来后到上市的时间。

2. 对新产品项目进行开发优先顺序排列和合理资源分配,确保重要的项目得到优先开发

在资源有限的情况下,同时开发过多项目的结果是所有项目都会延期。国际最佳实践研究表明,一个开发工程师同时进行两个项目的开发时效率最高,同时开发 3 个项目时效率开始下降,同时开发 4 个项目时效率将显著下降。所以,一个工程师最好不要同时安排两个以上的开发项目,以保证重点项目的开发进度。

3. 采取跨职能团队组织模式进行新产品开发

由各职能部门人员组成的开发团队负责新产品开发工作,很多工作可以并行进行,相对串行开发模式能大大缩短开发周期。此外,由一个团队自始至终负责一个新产品项目的开发,能大大减少沟通协调的时间,加快开发进度。

4. 建立技术平台和共用模块,缩短开发周期

有研究发现,一个新产品开发项目中平均有 40% 以上的重复劳动。如果企业能够通过建立技术平台,使一些技术模块化,使一些模块标准化,在一个平台产品的基础上进行更多的同类产品开发,则能大大缩短新项目的开发周期。

5. 采用有效的项目管理方法进行开发项目的管理

有效的项目管理方法是每个开发团队成员都应该熟练掌握的,掌握了有效的项目管理方法能大大缩短每项任务、每个活动的完成时间,总体的开发周期就能相应缩短。有效的开发项目管理方法包括目标定义、计划制定、团队组织、过程监控和结果交付等 5 个步骤。

采用适当的 IT 工具提升开发效率。比如应用 PDM 工具能有效管理各种产品信息,减少重复劳动,降低信息沟通成本,加快新产品的开发速度。其他如 CAD/CAM/CAE 等软件的应用也能大大加速新产品的设计和开发。

减少重复的过程可以节省时间、费用和工作量。说起来简单,做起来难。这些浪费时间的重复劳动可能在我们的项目活动的任何一个环节发生,包括代码编译、数据库集成、测试、审查、部署及反馈。通过 Jenkins 工具进行的持续集成可以将这些重复的动作变成自动化的,无须太多人工干预,让人们将更多时间投入到需要深入钻研的事情上。从而缩短项目总体耗时,将项目发布时间提前。

1.3 便捷部署

显而易见，部署包需要包含应用程序的所有组件，不仅仅是你自己的二进制包——EARs、WARs 之类。通常，这些包由集成构建产生，还应包含静态内容、配置文件、共享库、防火墙配置，等等。特别还应包括应用服务器的参数和设置，比如消息队列、连接工厂、数据源或类环境变量。实际上，软件包应包含软件生命周期中的所有内容，也就是那些需要一起被部署、升级和取消部署的内容，图 1-4 所示为部署包示意图。

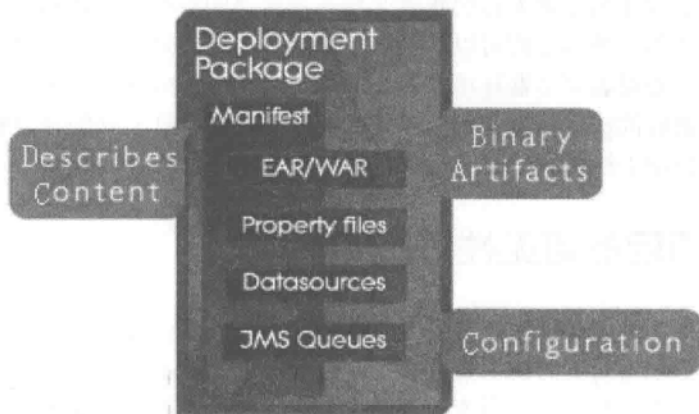


图 1-4 部署包示意图

确保软件包是“完备的可部署单元”对于一次可靠的部署来说是至关重要的，特别是在大规模环境中。指望目标中间件栈（Middleware Stack）已经用“正确值”设置好是导致部署失败的一个常见原因，这通常导致耗费很多时间找出不同环境中的细微差异，这往往是令人沮丧的过程。

部署包中只包含配置信息，比如共享“服务”，例如，为多个应用所共享的消息队列或数据源，这种情形不常见。这些配置显然应该按照有版本管理的、可部署的对象来处理，这意味着这些配置文件按照和“普通”应用程序一样严格的流程来发布。实际上，平稳、可靠的部署共享服务通常更为关键。

除了确保软件包最完整地描述了包含什么使特定版本的应用程序或配置正确工作外，软件包还需要指出它们能够运行所需的其他东西。换句话说，部署软件包需要明确界定他们的先决条件和依赖，以便在匹配的环境上部署。

准确地说，哪个部门负责部署和组织结构有很大关系。多数情况下，开发团队通常能使用持续构建工具使某些发布流程变为自动化，但发布过程还可能涉及其他团队。举例来说，让 DBA 在发布前确认 SQL 脚本，或要中间件竞争中心（Center of Competence）检查中间件配置的情况并不少见。

顺畅和可靠的部署流程从简单的第一步开始：以一种自动的、可检查校验的方法整合并发布一个结构化的、完整的部署包，该部署包定义了特定版本的应用程序中所有的组件、配置和依赖关系。这能显著减少因不合法或缺少定制值、组件或必需的服务而产生的错误。

计算机可读的包也是引入部署自动化的第一步，这里提到的自动化不仅是检查校验自动化，还包括按照自定义的标准流程部署这些包，并按照环境自动定制。除了确保一致性和提高可靠性之外，还可以缩短项目发布的周期，显著提高效率并可免除开发和维护生命周期内的一个最常见的瓶颈。

持续集成可以让你在任何时间发布可以部署的软件。在外界看来，这是持续集成最明显的好处，提起改进软件品质和减少风险我们可能会滔滔不绝，但对客户来说，可以部署的软件产品是最实际的资产。利用持续集成，你可以经常对源代码进行一些小改动，并将这些改动和其他代码进行集成。

举个例子，就像作为部署人员的我在北京，而客户的发布服务器却在杭州一样。其实只需要北京和杭州的服务器之间可以访问且可以通信，具有复制读写权限；不需要北京的部署人员到杭州异地部署服务器环境。对公司来说，节省了餐补、住酒店的费用，以及该员工在路上耗费的时间成本等。北京的部署人员仅仅需要将开发好的包发布到杭州客户的发布环境即可，自动上线后进行回归测试，想一想就觉得很美妙！

1.4 增强项目的可见性

在传统项目中，当一个项目经理或一名开发者说自己已经完成了 80% 的任务，你必须保持审慎的态度。因为剩下的 20% 可能还需要 80% 的时间，甚至永远都不能完成。为什么数据如此恐怖？因为项目中唯一不变的就是变化，我们每个人都需要拥抱变化，而变化让我们不知道将软件做到什么程度才算是做完全了。软件开发项目，往往在项目进度和软件质量方面缺少可见性，项目越缺少可见性，项目就越难控制，越有可能失败。我们可以通过迭代开发、技术评审、持续集成增强项目的可见性。

1. 迭代开发

采用迭代的开发模型，将产品的交付过程分为多个阶段，按照功能递增式交付。以下是一些典型的迭代方式。

- (1) 一次简短的先期迭代，以建立规模和前景并确定商业理由。
- (2) 一次精化迭代，其间将为稳定的构架划定基线。
- (3) 一次构建迭代，其间将实现用例并充实构架。
- (4) 几次产品化迭代，将产品转移到用户群。

每次迭代，都要充分接收用户的评审意见，以便自我纠正。渐近式的功能交付，有利于降低开发人员的压力，增加用户的满意度，有利于增强项目的可见性，是最好的进展报告。

2. 技术评审

技术评审是确保软件质量的重要环节，技术评审包括代码走审、会议评审和同行专家评审。代码走审可以是开发人员之间的交叉审查，也可以是高级开发人员对普通开发人员的审查；会议评审一般应至少每两周进行一次，每次评审的时间不宜太长；同行专家指技术和业务两方面的专家，经常性地让精通业务的用户专家参与项目评审，是项目成功的重要保证。

另外,充分利用质量审查的工具软件,也有助于提高代码质量。例如,在 Eclipse 开发环境中,可以集成 FindBugs、Checkstyle、PMD 插件检查代码编写质量。

3. 持续集成

持续集成能够把最终的一次大规模的集成调试过程分散到项目开发时间表的每一周、每一天、甚至每个小时。让项目中的各个人员都能够随时掌握当前的整体进度,并迅速发现集成过程中出现的问题并进行解决。

开发小组应制定持续集成的制度,一般情况下每日构建一次,可以利用 Ant 等构建工具进行 Java 应用程序的构建。小组成员应在每个功能开发完成后,及时向版本控制系统(如 SNV)提交代码,而且不应该向版本控制系统提交有问题(编译通不过)的代码。

每日构建、持续集成,让项目进度跟踪工作更加容易。当项目小组每天重新编译系统时,已完成与未完成的功能清楚可见,小组成员能够简单地从软件的表现知道距离整体完成还有多远。

持续集成让我们能够注意到项目进展的趋势并对其进行有效的决策。如果没有真实或最新的项目指标数据提供支持,那么项目就会遇到麻烦。通常,项目成员通过手工收集这些信息,这增加了项目成员的工作负担,也很耗时。持续集成可以带来以下两点积极效果。

(1) 有效决策:持续集成系统为项目构建状态和品质指标提供了及时的信息,有些持续集成系统可以报告功能完成度和缺陷率。

(2) 注意项目进展的趋势:由于经常集成,我们可以看到一些趋势,如构建成功或失败、总体品质及其他的项目信息。

1.5 建立团队对开发产品的信心

在竞争不断加剧的商业社会,能否开发出消费者真正需要的产品,往往决定了企业的成败。企业的领导人通常认为,在产品开发方面,企业有三个误区,而在这方面犯错误往往是致命的。

第一个误区是一劳永逸。很多失败的企业过去曾经很成功,它们之所以失败,很重要的一个原因是总盯着过去的荣誉。一些企业在开发出一项新产品后就不思进取,认为可以凭借这项成就过一段时间的安生日子,然而市场不会给你休息的时间。

第二个误区是闭门造车。一些企业耗费巨资推出了新产品,投向市场才发现消费者根本不喜欢。可口可乐在开发运动饮料时就犯过这个错误,该公司新推出的能量饮料遭受了连续失败,在第4次推向市场时才成功。究其原因,该公司认为自己没有遵循消费者至上的原则,一味改进饮料的口味,忘了运动饮料最基本的功能是解渴。

第三个误区是盲目追随。在新产品开发方面,企业面临的一个挑战来自竞争对手的全新创意。这时,很多企业容易陷入盲目追随的误区。

在创新方面有两个基本选择:跟着市场走或是带着市场走。索尼公司主张做引领者,从新技术中汲取灵感;但坎贝尔公司认为,如果不愿受市场驱使,引领市场就是一句空话。

默克药业认为,关于创新的这两个选择听起来南辕北辙,实际上没有本质区别:消费者可能无法用准确的语言形容他们的需求,但会通过他们的行为和态度表现出来,灵敏的

产品开发人员会从中找到灵感。

持续集成可以建立开发团队对开发产品的信心，因为他们清楚地知道每一次构建的结果，他们知道他们对软件的改动造成了哪些影响，影响的结果怎么样。

接下来我们深入学习何谓持续集成。

1.6 解决项目管理的困惑

L 先生曾负责一个航空领域的大型数据仓库项目，在此项目一期的时候，各类问题层出不穷，项目组疲于应付，四处救火。

在项目中，L 先生没有与各项目干系人建立有效的联系，根本无法让他们了解项目的进展情况，甚至项目开发人员自身对项目整体的情况都没有清楚的认识，只管自己那一部分，对其他工作则不闻不问。一旦项目开始，直到项目结束才能准确知道产品情况。整个开发过程完全是一种黑盒模式，项目组成员无法把握准确进度，无法保证项目质量。

到了项目后期才发现销售模块开发进度过慢，不得已加班加点，仓促交工，连自己都不放心项目质量，大量的 BUG 遗留在这部分，产生许多隐患，维护的工作量甚至超过了开发，导致系统维护成本过大。用户抱怨颇多，维护人员更是怨声载道。

在项目交工时，客户提出运输模块提供的信息无法满足制作报表的要求，并抱怨这个变更早就通知过项目组，可 L 先生作为项目经理竟然全然不知，结果是双方来回扯皮。

开发人员在设计对内开账模块时描述了实现方式，但为了节省时间，只是粗粗地写了设计，就去编码。等编码结束时，发现和 L 先生原来所理解的出入甚大，只得推翻重来，不但开发人员的工作量增加了，而且成本超支严重。

在此项目的二期时，公司开始采用 CMM 的开发思想，加强对需求、计划的管理，采用配置管理工具 VSS 管理文档后情况才逐步好转，但前期仍然出现了许多问题：不少项目成员对文档敷衍了事，认为只是走走过场而已。

L 先生遇到的现象绝非偶然，稍加分析我们会发现，L 先生的项目组在开始引用 CMM 的开发思想时，缺乏一种项目组各个层面间的沟通机制，而加强沟通管理应是实施成功项目管理的必经之路。

持续集成在开发者和开发者之间、开发者和测试之间、开发者和配置管理之间形成了一条“无缝焊接”之路。

当团队进入项目中时，对于项目管理者来说，真的不是一件容易的事情。由于我是从一线技术人员逐步走到管理岗位的，所以我很清楚当我是开发者、测试者的时候，即使那时我技能再突出，我也更倾向于从自己角色的角度看问题。

打个比方，每个人都希望自己的职责明确，但职责本身就是人划出来的，要么会有重复，要么会有交互的丢失，总之存在问题是非常常见的，这就是项目管理中常常困扰项目经理的问题——人是最难管理的。

都说项目管理有三宝：进度、流程、做报表。可是在你不断跟踪、反馈、风险评估、持续协调的时候，有一个减轻沟通成本的工具或者模式，对项目经理来说不亚于雪中送炭。

在项目实施过程中，软件开发到什么程度，当前进度是否存在风险、是否合理，开发者彼此之间的程序接口定义及配合情况是否清晰、顺畅等，这些都决定了项目的进度和项

目的成败。持续集成让项目经理在项目开发过程中，不必跟在程序员后面，不断地记录每天的开发状态。项目管理者只需要把控项目进度，不需要再关注开发和测试间的接口问题，因为在持续集成实践中，可以认为开发和开发者之间，测试和测试者之间的工作协作都是集成到一起的，持续集成的频率是非常快的，项目管理者只需要花时间关注开发/测试流程之后的结果，不必关注开发结果后再关注测试结果。

1.7 总结

持续集成能节约你的时间和成本，能让你的迭代的节奏更紧凑，让开发和测试之间的协作更完美协调。

但是，凡事一定要从实际情况出发，如果你所在的公司使用持续集成会伤筋动骨，会劳民伤财，那么不建议你使用。

当你所在的公司或者团队，存在以上问题需要解决，并且大家可以达成观念上的一致时，一起采用持续集成的方式解决现有的问题，这才是最理想的。

第2章

何谓持续集成

2.1 持续集成的定义

大师 Martin Fowler 是这样定义持续集成的：持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作。通常，每个成员每天至少集成一次，也就意味着每天可能发生多次集成。

根据对项目实践的理解，持续集成中的“持续”是指不间断的；“集成”可分为广义和狭义，广义的集成指软件各个过程的集成，包括开发、部署、测试等。狭义的集成即代码和代码之间的集成，从而保证代码合并不冲突。

每次集成都通过自动化的构建（包括编译、发布和自动化测试）来验证，从而尽快地发现集成错误。许多团队都发现这个过程可以大大减少代码集成的问题，让团队更快地开发内聚的软件。请注意，持续集成不等于持续编译。

2.2 持续集成的特点

持续集成是什么，怎么用它？其实关键点在于你想怎么用它。我先列举大家喜欢它的地方。

1. 开源，免费

从成本角度讲，如果从一个免费的、好用的软件和一个收费且好用的软件中择一使用，毫无疑问，我们一定会采用开源的软件。免费，何乐而不为？图 2-1 所示为安装包下载页面。



图 2-1 安装包下载页面

2. 入门便捷

如果你是一个初学者，那么你只要去官网下载一个 Jenkins 的最新版的 war 包，在本地环境装一个 JDK，就可以启动 Jenkins 服务了，就是如此简单。Jenkins 服务启动示例图如图 2-2 所示。

```

管理员: C:\Windows\system32\cmd.exe - java -jar jenkins.war

C:\jenkins>java -jar jenkins.war
Running from: C:\jenkins\jenkins.war
javaagent: $user.home/.jenkins
九月 22, 2014 9:34:55 上午 winstone.Logger logInternal
信息: Beginning extraction from war file
九月 22, 2014 9:34:55 上午 org.eclipse.jetty.util.log.JavaUtilLog info
信息: jetty-8.y.z-SNAPSHOT
九月 22, 2014 9:35:05 上午 org.eclipse.jetty.util.log.JavaUtilLog info
信息: NO JSP Support for ., did not find org.apache.jasper.servlet.JspServlet
Jenkins home directory: C:\Users\Administrator\.jenkins found at: $user.home/.jenkins
九月 22, 2014 9:35:07 上午 org.eclipse.jetty.util.log.JavaUtilLog info
信息: Started SelectChannelConnector@0.0.0.0:8080
九月 22, 2014 9:35:07 上午 winstone.Logger logInternal
信息: Winstone Servlet Engine v2.0 running: controlPort=disabled
九月 22, 2014 9:35:08 上午 jenkins.InitReactorRunner$1 onBtained
信息: Started initialization
九月 22, 2014 9:35:14 上午 jenkins.InitReactorRunner$1 onBtained
信息: Listed all plugins
九月 22, 2014 9:35:14 上午 jenkins.InitReactorRunner$1 onBtained
信息: Prepared all plugins
九月 22, 2014 9:35:15 上午 jenkins.InitReactorRunner$1 onBtained
信息: Started all plugins
九月 22, 2014 9:35:15 上午 jenkins.InitReactorRunner$1 onBtained
信息: Augmented all extensions
九月 22, 2014 9:35:22 上午 jenkins.InitReactorRunner$1 onBtained
信息: Loaded all jobs
九月 22, 2014 9:35:24 上午 org.jenkinsci.main.modules.sshd.SSHD start
信息: Started SSHD at port 56572
九月 22, 2014 9:35:24 上午 jenkins.InitReactorRunner$1 onBtained
信息: Completed initialization
九月 22, 2014 9:35:24 上午 hudson.WebAppMain$3 run
  
```

图 2-2 Jenkins 服务启动示例图

3. 易用性好

Jenkins 中有很多提示性的输入。如图 2-3 所示，清晰的操作步骤，所见即所得。

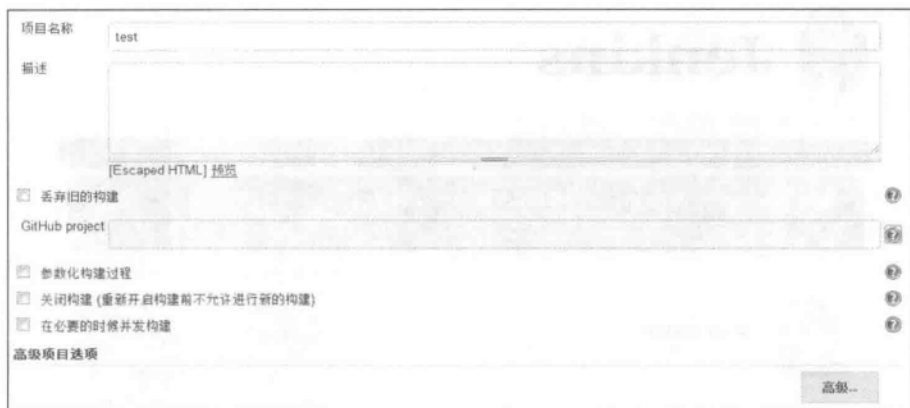


图 2-3 Jenkins 的任务面板

4. 用的人越来越多

由于持续集成优点显著，所以越来越多的团队开始采用持续集成工具构建代码、持续测试。

Google、苹果、BAT 公司都在使用持续集成工具构建代码，这就是最好的证明。

5. 有组织长期维护

Jenkins 是由 Jenkins 组织长期更新升级维护的，这对 Jenkins 的使用者来说，是一件非常舒服的事情。Jenkins 中的一些常见问题如图 2-4 所示。

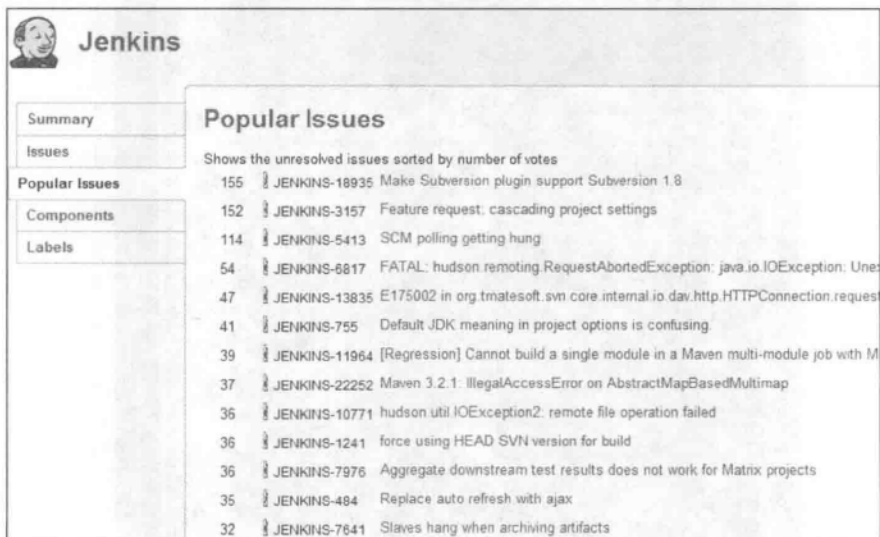


图 2-4 Jenkins 中的一些常见问题

6. 扩展性

Jenkins 的扩展性极好，因为其支持多种插件。目前为止，Jenkins 支持很多主流插件，例如 Shell、SSH、Maven、Ant 等，Android 的模拟器 Jenkins 插件也是支持的，图 2-5 所示为 Jenkins 可安装插件示意图。



图 2-5 Jenkins 可安装插件示意图

2.3 原则

持续集成告诉我们：你越投入，你获得的价值越多。

1. 维护唯一的代码仓库

一个软件会包括很多文件，很多版本，我们需要将它们像音符一样编制在一起才能生成产品。要想把握好力度是耗时耗力的，特别是当项目有多人参与时。

所以，一个基本的原则是确保你有一个相对最新版本的代码管理系统，像 SVN、Perforce、CVS 等。当前，互联网公司比较主流的模式是用 GitHub，这也是一个非常棒的选择，GitHub 不仅仅具有代码管理工具的一些功能，还可以提供一些开源代码的下载服务，以及一些在线开源代码项目的合并等令人振奋的功能。

有了代码工具，就要好好地规划，把构建的内容都放进去，不像将相同颜色的砖头垒起来那么简单。你最好把构建所必需的东西都放进资源控制系统里，你也可以放些其他开发者经常用到的东西，IDE Setups 也可以放进去，这样大家可以共享同一个 IDE。版本控制的一个特性就是能让开发人员创建多个分支，从而在各自不同的分支中提交并维护自己的代码。最后，代码从分支合并到主干，再到代码发布的一个过程。

但是过分使用代码分支同样会让开发人员陷入困境。我们需要根据我们产品和系统的特性规划我们的代码结构、代码分支和版本策略，从而制定出一个最符合公司产品发布节奏的代码分支策略。一般情况下，开发主干只有一个，它就是开发主线，从主干上分出来的代码分支上也可以定义很多代码版本。主干、分支、版本，各自的使用场景不同。有关这部分的详细描述请参考本书 5.1 节。

2. 让你的构建能自测

构建，传统意义上意味着编译、链接和一些程序执行的工作。一个程序能运行，不意味着它能做正确的事情。

获取 BUG 的最好方式是在代码构建过程中包含自动测试。测试不可能完美，但是它确实能捕获 BUG，这就够了。实际上，极限编程和测试驱动开发已经为自测代码的推广树立了好的榜样，越来越多的人看到了这项技术的价值。

经常看我文章的人都知道，我是极限编程和测试驱动开发的忠实粉丝，然而我要强调的是：不是所有这些方法都能获得自测代码的好处。这些方法都强调在你编码之前写测试，然后让你的编码通过这些测试，在这种模式下，测试工作跟设计系统所做的工作一样多。这是个好事情，但是对于持续集成来说不是必要的，我们没有那么多的自测代码需求（尽管测试驱动开发是我偏爱的生成自测代码的方式）。

对于自测代码，你需要一套自动测试来检查大段代码中的 BUG，这些测试需要能被从简单的命令中剔除并能够自测。测试的结果能够指示错误所在。自测的失败会造成构建的失败。

这些年来，测试驱动开发的广泛使用让开源工具 XUnit family 在测试中普及，Unit 工具在 ThoughtWorks 的工作实践中已经被证明非常有用，我总是建议人们去使用它们。这些工具（Kent Beck 是我们的先导）能帮助我们轻易地构建一个完整的自测环境。

XUnit 工具肯定是你进行代码自测的起点。你也应该看看其他更注重 end-to-end 测试的工具，例如 FIT、Selenium、Sahi、Watir、FITnesse 等。

当然，你别指望测试能找到一切。有句话叫作：测试不能证明 BUG 不存在，完美的测试是写不出来的，而不完美的测试却可以执行多遍帮你找到错误。

3. 日构建，每人每日提交代码

集成主要是沟通，它能让开发者知道彼此所做的改动。

开发者提交代码的前提是他们能正确构建自己的代码，当然也包括构建中的测试。提交的过程是：先把自己的工作备份与主线同步，找到并解决其中可能的冲突，然后在自己本地机器上构建，如果能够通过，就可以自由地提交代码到主线了。

频繁操作，开发者能很快地发现两个开发者间的冲突。解决这个问题关键是尽快发现冲突。若开发者几小时提交一次代码，那么冲突就能在几小时内被发现，这个时候开发者可能还没走远，问题就容易解决。而且这几个小时内变动的地方有限，BUG 的藏身之处也就容易发现。你可以用 diff-debugging 的技巧帮你找到 BUG。

我的原则是每个开发者每天都得提交他们的代码，当然如果大家提交得更频繁，那么就更容易发现冲突，但通常人们会发现自己无法在几小时内就完成一项有意义的变动。

4. 每次提交都在主线集成的机器上做构建

使用每日提交，团队会频繁地进行测试构建。如果大家在提交前没有更新就做了构建或者开发者的机器环境不一样，那么大家构建的结果会不一样。所以你得在集成了主线的代码的机器上做构建，这样你就需要一个持续集成的服务器。

持续集成的服务器就像数据仓库的监视器，每次提交代码都会自动触发 checkout 代码到集成服务器上，初始化构建并将结果通知提交者。提交工作直到收到邮件通知才算结束。

持续集成的引入会给大家的工作习惯带来冲击，大家会不习惯总是留意主线的构建状态。

2.4 总结

持续集成是一种做事不断持续，代码不断集成的工作思想，它是一种节奏。用得好，它可以指导你节约一些不必浪费的时间；用得不好，它会让你浪费时间。千万不要为了持续集成而去持续集成，一定要根据自己企业的实际情况去实施。

任何工具、理念都不是万能的，它们都有适用范围。持续集成的模式不是一成不变的，而是不断优化再优化的。本章列举的一些问题，是在一些企业的技术团队中常见的问题，若恰巧以上问题你也希望解决，那么请用持续集成去解决吧。至于如何解决，请看第3章。

第3章

持续集成的核心价值

3.1 价值点

事物总有其多面性，我们要多维地看待它，找到其最匹配、最利于自己的一面，这个非常重要。

1. 持续集成的价值

- 降低项目风险。
- 缺陷的检测和修复变得更快。
- 软件的健康程度可以测量。
- 减少假定。
- 减少重复过程。
- 减少重复过程的劳动，让人们有时间做更多需要动脑筋的、更高价值的工作。
- 通过对一些重要过程的（如测试盒数据库集成）自动化，消除项目中某些成员对实现中改进的抵制。
- 在任何时间、任何地点生成可部署的软件。

2. 增强项目的可见性

- 有效地决策：持续集成系统可以对当前的构建状态和品质指标提供及时的信息。
- 监控构建结果状态：根据观察构建成功和失败的走势图，了解代码合并的冲突次数和代码本身分工是否合理，并监控开发人员代码的质量。
- 对开发团队的软件产品建立更强大的产品信心。

3. 对于在项目中实现持续集成的团队和个人很有好处的 7 项实践

- 经常提交代码：进行小的变更；在每个任务之后进行提交。
- 不要提交无法构建的代码。
- 立即修复无法集成的构建。
- 编写自动化的开发者测试。
- 必须通过所以测试和审查。
- 执行私有构建。
- 避免签出无法构建的代码。

在软件开发中，持续集成实践能够解决的问题是尽早集成和尽早反馈。因此，尽管目前流行的所有版本的控制工具都提供了分支/合并功能，但在少于 20 人的团队中实现持续集成，推荐使用 Single Branch 开发策略。这样会减少多分支在合并时的开销。另外，由于理想情况下，每个分支都需要有专属的持续集成环境（包括持续集成服务器、构建环境和测试环境等），所以 Single Branch 也减少了对持续集成环境的需求量（当编译时间较长或测试用例较多时，这个因素的影响尤其重要）。

当团队完成持续集成服务器的最初搭建，编写好一键式编译和测试脚本工作后，就需要考虑如何利用持续集成环境进行团队协作开发。一定有人会问：“若多人同时在一个分支上开发，每个人提交时都要合并代码，不是更浪费时间吗？”

这个问题也正是持续集成期望解决的问题。每当开发人员提交代码时，就是其与其他开发人员工作成果的一次集成。如果每个人都能够频繁提交代码，那么代码集成的频率就会提高，在持续集成的有力支持下，代码中潜在的问题就会更早暴露出来（比如代码编译链接问题，自动化测试失败反映出来的代码功能问题，或对需求理解不一致等问题），以便团队尽早解决。

当然，持续集成所鼓励的频繁提交并不是指那种仅将版本控制库当成备份工具，无约束地“随意”提交，而是指需要团队开发流程约束的提交。

3.2 减少风险

持续集成能够缓解的一些关键风险如下。

- 没有可部署的软件：利用持续集成系统随时构建可部署的软件。创建一个可重复的构建过程，使用版本控制库中的所有软件资产。
- 很晚才发现缺陷：每次变更时都执行开发者测试，这样就能够在软件开发生命周期中尽早发现缺陷。
- 缺少项目可见性：经常运行构建，随时了解项目的健康状况。如果有效地应用持续集成实践，项目的状态就不再是一个问题。
- 低品质的软件：在每次变更时执行测试和审查，这样就能通过了解复杂程度、重复情况、设计、代码覆盖率和其他因素发现可能进入代码中的潜在缺陷。

关于风险控制的幽默趣图如图 3-1 所示。

慌什么，此船已装上风险控制系统，
风浪何所畏惧



图 3-1 风险控制

3.3 根据变更构建软件

执行单命令构建。Martin Fowler 说：“把所需要的东西都放到源代码版本控制库中，以便你通过单个命令就能构建整个系统。”

将构建脚本从 IDE 中分离，因为：（1）每个开发者可能使用不同的 IDE，找出每个 IDE 中配置文件的差别会很困难；（2）持续集成服务器必须在无人干预的情况下执行自动化构建，因此开发者执行的自动化构建脚本，同样也应该能够由持续集成服务器执行。

集中放置软件资产。一种方法是使用版本控制库存放所有文件，除了代码，还包括：

- 组件，包括源文件和库文件。
- 第三方组件，如 JAR 文件、库、DLL 等。
- 配置文件。
- 初始化应用程序的数据文件。
- 构建脚本和构建环境设置。
- 某些组件的安装脚本。

让构建快速失败。好的构建知道如何快速失败。在构建的许多部分都成功之后再失败，会使人很恼火，而且你在确定失败的目标时也会失去宝贵的时间。创建快速失败构建的主要步骤为：集成组件、运行真正的单元测试，以及执行其他自动化的过程。

1. 构建类型

- 私有构建：开发者在向版本控制库提交代码之前，要先进行私有构建。
- 集成构建：集成构建集成项目团队向版本控制库中的主线提交的变更。
- 发布构建：准备好发布给用户的软件。

2. 构建触发机制

- 用户驱动：由某人手工发起构建。
- 定期执行：由时间驱动，不论是否发生了变更。
- 轮训变更：一个进程以固定的时间间隔唤醒，从版本控制库中签出变更。如果检测到变更，它就执行集成构建。
- 事件驱动：与轮训变更类似，但是是由版本控制库发出的，基于实现定义好的变更事件。

3. 构建常用功能

- 以特定的时间间隔轮训版本控制库中的变更。
- 定期执行某种操作。
- 标识出“安静期”，在这段时间内项目不进行集成构建。
- 支持不同的构建脚本工具，包括命令行工具。
- 向相应人员发送电子邮件。
- 显示以前构建的历史。
- 显示一个 Web 信息面板，这样每个人都可以查看集成构建的信息。
- 为不同的项目支持多个版本控制方法。

4. 构建效率统计

- 收集构建测量数据。
- 分析构建测量数据。
- 选择并实现改进。
- 重新评估，有必要则再次重复这个过程。

5. 构建测量指标

- 编译时间。
- 源代码行数（SLOC）。
- 审查的种类数。
- 平均组装生成时间。
- 测试执行时间（根据分类）。
- 成功构建和失败构建的比例。
- 审查时间。
- 部署时间。
- 重建数据库的时间。
- 集成构建计算机系统资源和使用情况。
- 版本控制系统的负载。

6. 构建改进方法

- 使用专门的集成构建计算机。
- 增强集成构建计算机的硬件能力。

- 改进测试性能。
- 安排集成构建的顺序。
- 优化基础设施。
- 优化构建过程。
- 单独构建系统组件。
- 改进软件审查的性能。
- 执行分布式集成构建。

3.4 总结

持续集成的初级阶段，可以以快速构建为核心；再深入可以以集成多种工具、提高构建结果和代码质量为核心；在高级阶段，利用持续再持续的方式将多角色的多工作结合到一起，形成快速迭代，实施、结果反馈、优化→再迭代→实施、结果反馈、优化的过程。

在这个过程中，最核心的是将所有看似散乱的工作有机地形成一个整体，明确定义阶段性产出物，将职责的定义清晰化。

第 4 章

持续集成实践步骤

4.1 如何选取最佳解决方案

想找到属于自己的最佳解决方案，我们可以从以下几个方面考虑。

1. 目标

一切都是为目标服务的，要用持续集成解决什么问题，就要以其为重点来实施。

在一个典型的持续集成过程中：

- 开发者将代码提交到 SVN 之前，必须运行本地测试。尽量保证不会破坏持续集成服务器的构建过程。
- 开发者每天进行多次提交：小步前进会大大减少服务器构建失败的概率，并使修复失败构建的时间大大缩短。
- 持续集成在一台服务器上不断运行：保证在稳定的环境中进行测试。
- 所有测试必须全部通过：保证软件现有的功能不被破坏并且没有引入新的缺陷。
- 生成构建结果：用以开展下一步工作，譬如 QA 的探索测试等。
- 生成报表：帮助管理人员评估开发状态。
- 修复失败的构建：失败的构建意味着软件现有的功能已经被破坏或者有新的缺陷被引入，修复的速度越慢，修复的难度就越大，并带来更大的损失。

2. 成本

(1) 硬件花费。

对大多数团队来说，持续集成服务器不需要运行在一台性能异常强大的机器上，使用一台开发机器加一个指示灯（在构建失败时变红）就足够了，硬件的成本在不断下降，与

开发团队不断修复回归测试发现的缺陷，编译失败，延迟发布比起来，这个投资应该是非常划算的。

(2) 管理开销。

目前的持续集成服务器已经比较成熟，并且有许多免费的产品（如 CruiseControl）可以选择。这些服务器不论是运行还是配置都非常简单。开发团队可以根据熟悉的编程语言选择相关产品，以 CruiseControl 为例，分别有 CruiseControl.rb（Ruby），CruiseControl.net（.NET），CruiseControl（Java）供选择。这样开发团队可以在没有系统管理员的情况下，自行维护持续集成服务器。

3. 耗时

(1) 提高开发者提交的速度。

开发者用 10 分钟进行测试，节省了整个团队修复构建的时间，免去了不必要的交流成本，节省了修复 BUG 的成本，大大降低了延迟发布的可能性。

(2) 学习自动化测试。

花时间学习 Ant、Maven、Shell 脚本比每天花半小时手工测试有趣得多，引入自动化测试，节省的不仅仅是个人的时间，整个团队的效率也因此提高。

4. 成效

为了享受持续集成带来的诸多好处，开发者需要做到如下几点。

- 小步前进，频繁提交。
- 不要提交本地测试失败的代码。
- 编写可以自动进行的测试。
- 编写可以快速运行的测试。
- 如果构建失败，第一时间进行修复。
- 如果构建失败，拒绝更新代码。

没有最好的，只有最合适的，这句话是解决方案选型的最佳诠释。读者可以参考以下案例选择最适合自己的实践。

(1) 持续编译。

[现象] 某些团队仅仅使用持续集成服务进行编译并生成最终的构建结果。

[影响] 持续集成无法给开发人员和管理人员带来有价值的快速反馈。

[原因] 开发团队可能缺乏编写易于测试的代码的能力，或者不了解现代软件开发中测试的流程和作用。

[解决方案] 测试优先（分别进行单元测试、功能测试和验收测试）。

(2) 构建长时间失败。

[现象] 没有开发者愿意修复失败的构建，持续集成工具上的构建已经持续失败很长时间了。

[影响] 开发者忽视持续集成服务器发布的结果，修复构建的成本和难度升高，开发团队和管理团队无法得到快速反馈，丧失安全感。

[原因] 长时间不进行代码更新，一次提交太多代码，构建时间太长导致开发者缺乏耐心运行本地构建、任务过于复杂。

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系QQ1779903665.

PDF代找说明：

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签索引和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ1779903665。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

声明：本人只提供代找服务，每本100%索引书签和目录，因寻找pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。