

《高效团队开发》

图书基本信息

书名：《高效团队开发》

13位ISBN编号：9787115295948

出版时间：2015-7

作者：[日] 池田尚史, [日] 藤仓和明, [日] 井上史彰

页数：320

译者：严圣逸

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu000.com

《高效团队开发》

内容概要

本书以团队开发中所必需的工具的导入方法和使用方法为核心，对团队开发的整体结构进行概括性的说明。内容涉及团队开发中发生的问题、版本管理系统、缺陷管理系统、持续集成、持续交付以及回归测试，并且对“为什么用那个工具”“为什么要这样使用”等开发现场常有的问题进行举例说明。本书适合初次接手开发团队的项目经理，计划开始新项目的项目经理、Scrum Master，以及现有项目中返工、延期问题频发的开发人员阅读。

作者简介

作者简介：

池田尚史

DeNA软件开发工程师。曾做过IT顾问、程序员，从事过软件包开发、Web服务开发。Java的Web应用框架Play Framework 1的提交者。负责本书第1章~第5章，其中第2章的案例分析都是基于自身的实际经验编写的。

Twitter @ikeike443

藤仓和明

想能（SHANON）基础设施工程师。负责公司内部基础设施及服务环境的安全保障，致力于推动应用部署的自动化，并基于这方面丰富的实践经验，完成了本书第6章。喜欢OpenVZ、LXC等容器型虚拟化技术。

Twitter @fujya

井上史彰

想能（SHANON）软件工程师、QA工程师，现为想能信息科技（上海）有限公司总经理。开发经验丰富，致力于推动高效的自动化测试。负责本书第7章。

E-mail fu.inoue@gmail.com

译者简介：

严圣逸

毕业于上海交通大学。8年软件开发经验，期间赴日本工作。现就职于想能信息科技（上海）有限公司，从事基于云平台的客户关系管理及各类营销自动化系统的开发，侧重于对持续集成、自动化部署、自动化测试以及相关的开源工具的研究。本书所介绍的即是译者日常工作中所应用的开发流程以及工具。

书籍目录

第1章 什么是团队开发	1
1.1 一个人也能进行开发	2
1.2 团队开发面临的问题	3
1.3 如何解决这些问题	4
1.4 本书的构成	5
1.4.1 第2章：案例分析	5
1.4.2 第3~5章：基础实践	5
1.4.3 第6~7章：持续交付和回归测试	6
1.5 阅读本书前的注意事项	7
1.5.1 最好的方法是具体问题具体分析	7
1.5.2 没有最好的工具	7
第2章 团队开发中发生的问题	9
2.1 案例分析的前提	10
2.1.1 项目的前提条件	10
2.2 案例分析（第1天）	11
2.2.1 问题1：重要的邮件太多，无法确定处理的优先顺序	11
2.2.2 问题2：没有能用于验证的环境	11
2.2.3 问题3：用别名目录管理分支	12
2.2.4 问题4：重新制作数据库比较困难	14
2.3 案例分析（第1天）中的问题点	16
2.3.1 问题1：重要的邮件太多，无法确定处理的优先顺序	16
邮件的数量太多，导致重要的邮件被埋没	16
无法进行状态管理	17
直观性、检索性较弱	17
用邮件来管理项目的课题	17
2.3.2 问题2：没有能用于验证的环境	18
2.3.3 问题3：用别名目录管理分支	18
2.3.4 问题4：重新制作数据库比较困难	19
2.4 案例分析（第2天）	22
2.4.1 问题5：不运行系统就无法察觉问题	22
2.4.2 问题6：覆盖了其他组员修正的代码	22
2.4.3 问题7：无法自信地进行代码重构	24
2.4.4 问题8：不知道bug的修正日期，也不能追踪退化	25
2.4.5 问题9：没有灵活使用分支和标签	26
2.4.6 问题10：在测试环境、正式环境中无法运行	28
2.4.7 问题11：发布太复杂，以至于需要发布手册	28
2.5 案例分析（第2天）中的问题点	30
2.5.1 问题5：不运行系统就无法察觉问题	30
2.5.2 问题6：覆盖了其他组员修正的代码	31
2.5.3 问题7：无法自信地进行代码重构	31
2.5.4 问题8：不知道bug的修正日期，也不能追踪退化	33
2.5.5 问题9：没有灵活使用分支和标签	35
2.5.6 问题10：在测试环境、正式环境中无法运行	35
2.5.7 问题11：发布太复杂，以至于需要发布手册	36
2.6 什么是理想的项目	37
2.6.1 使用缺陷管理系统对课题等进行统筹管理	38
2.6.2 尽量使用版本管理系统	38

2.6.3	准备可以反复验证的CI系统	38
2.6.4	将环境的影响控制在最小限度，并随时可以发布	39
2.6.5	保留所有记录以便日后追踪	39
2.7	本章总结	40
第3章	版本管理	41
3.1	版本管理系统	42
3.1.1	什么是版本管理系统	42
3.1.2	为什么使用版本管理系统能带来便利	42
	能够保留修改内容这一最基本的记录	43
	能够方便地查看版本之间的差异	43
	能够防止错误地覆盖他人修改的代码	43
专栏	锁模式和合并模式	44
	能够还原到任意时间点的状态	48
专栏	基于文件和基于变更集	49
	能够生成多个派生（分支和标签），保留当时项目状态的断面	49
3.2	版本管理系统的发展变迁	51
3.2.1	没有版本管理系统的时代（20世纪70年代以前）	52
3.2.2	RCS 的时代（20世纪80年代）	52
3.2.3	CVS 的诞生（20世纪90年代）	52
3.2.4	VSS、Perforce等商用工具的诞生（20世纪90年代）	53
3.2.5	Subversion 的诞生（2000年以后）	54
3.2.6	分布式版本管理系统的诞生（2005年以后）	54
3.2.7	番外篇：GitHub的诞生	55
3.2.8	版本管理系统的导入情况	57
3.3	分布式版本管理系统	59
3.3.1	使用分布式版本管理系统的5大原因	59
	能将代码库完整地复制到本地	59
	运行速度快	59
	临时作业的提交易于管理	59
	分支、合并简单方便	59
	可以不受地点的限制进行协作开发	60
3.3.2	分布式版本管理系统的缺点	60
	系统中没有真正意义上的最新版本	60
	没有真正意义上的版本号	60
	工作流程的配置过于灵活，容易产生混乱	61
	思维方式的习惯需要一定的时间	61
3.4	如何使用版本管理系统	62
3.4.1	前提	62
3.4.2	版本管理系统管理的对象	62
	代码	63
	需求资料、设计资料等文档	64
	数据库模式、数据	64
	配置文件	64
	库的依赖关系定义	65
3.5	使用Git顺利地推进并行开发	66
3.5.1	分支的用法	66
	什么是分支	66
	什么是发布分支（release branch）	66
	克隆和建立分支	67

提交和提交记录	67
分支的切换	68
修正bug后的提交	69
合并到master	70
向master进行Push	71
分支使用方法总结	72
3.5.2 标签的使用方法	72
什么是标签	72
新建标签	72
标签的确认	73
标签的取得	73
专栏 避免使用相同的标签名和分支名	74
标签使用方法总结	75
专栏 什么是Detached HEAD	76
3.6 Git的开发流程	77
3.6.1 Git工作流的模式	77
中央集权型工作流	77
GitHub型工作流	78
3.6.2 分支策略的模式	79
git-flow	79
github-flow	82
笔者的例子（折衷方案）	83
3.6.3 最合适的流程和分支策略因项目而异	84
3.7 数据库模式和数据的管理	85
3.7.1 需要对数据库模式进行管理的原因	85
由数据库管理员负责对修改进行管理的情况	85
修改共享数据库的模式的情况	85
3.7.2 应该如何管理数据库模式	86
版本管理的必要条件	86
什么是数据库迁移	86
数据库迁移的功能	87
3.7.3 数据库迁移工具	88
Migration (Ruby on Rails)	88
south (Django)	88
Migrations Plugin (CakePHP)	89
Evolution (Play Framework)	89
3.7.4 具体用法 (Evolution)	89
规定	89
SQL文件的执行	90
开发者之间数据库模式的同步	91
一致性问题的管理	93
3.7.5 数据库迁移中的注意点	94
3.8 配置文件的管理	96
3.9 依赖关系的管理	97
3.9.1 依赖关系管理系统	97
JVM 语言	97
脚本语言	98
管理依赖关系的优点	98
3.10 本章总结	100

第4章 缺陷管理	101
4.1 缺陷管理系统	102
4.1.1 项目进展不顺利的原因	102
4.1.2 用纸、邮件、Excel进行任务管理时的问题	103
4.1.3 导入缺陷管理系统的优点	104
具有任务管理所需的基本功能	104
直观性、检索性较强	104
能够对信息进行统一管理 & 共享	104
能够生成各类报表	105
能够和其他系统进行关联，具有可扩展性	105
4.1.4 什么是缺陷驱动开发	106
缺陷驱动开发的具体步骤	106
专栏 彻底贯彻缺陷驱动开发的情况	107
4.2 主要的缺陷管理系统	108
4.2.1 OSS产品	108
Trac	108
Redmine	109
Bugzilla	110
Mantis	111
4.2.2 商用产品	112
JIRA	112
YouTRACK	113
Pivotal Tracker	113
Backlog	114
GitHub	115
4.2.3 选择工具（缺陷管理系统）的要点	116
专栏 缺陷管理系统的应用事例	117
4.3 缺陷管理系统与版本管理系统的关联	118
4.3.1 通过关联实现的功能	118
从提交链接到问题票	118
从问题票链接到提交	118
提交的同时修改问题票的状态	119
4.3.2 关联的配置方法	119
4.3.3 GitHub	119
GitHub的issue	119
Service Hooks	120
GitHub和Pivotal Tracker的关联	121
GitHub和JIRA的关联	123
4.3.4 Trac/Redmine	124
4.3.5 Backlog	124
Backlog和Git的关联	125
Backlog和GitHub的关联	126
4.3.6 Git自带的Hook的使用方法	127
4.4 新功能开发、修改bug时的工作流程	128
4.4.1 工作流程	128
A建立问题票	128
B指定负责人	129
C开发	129
D提交	129

E Push到代码库	129
4.5 回答“那个bug是什么时候修正的”的问题	131
4.5.1 Pivotal Tracker的例子	131
用记忆中残留的关键字进行检索	131
检索	131
通过问题票查找代码修改	132
4.5.2 Backlog的例子	133
检索	134
4.6 回答“为什么要这样修改”的问题	136
4.7 本章总结	137
专栏 缺陷管理、bug 管理以及需求管理	137
第5章 CI (持续集成)	141
5.1 CI (持续集成)	142
5.1.1 什么是CI (持续集成)	142
集成 (integration)	142
持续地进行集成就是CI	142
5.1.2 使开发敏捷化	143
瀑布式开发的开发阶段	143
敏捷开发的开发阶段	144
5.1.3 为什么要进行CI 这样的实践	147
成本效益	147
市场变化的速度	148
兼顾开发速度和质量	148
5.1.4 CI的必要条件	149
版本管理系统	149
build 工具	149
测试代码	151
CI 工具	151
5.1.5 编写测试代码所需的框架	151
测试驱动开发 (TDD) 的框架	151
行为驱动开发 (BDD) 的框架	152
5.1.6 主要的CI 工具	154
Jenkins	154
TravisCI	155
5.2 build工具的使用方法	157
5.2.1 新建工程的情况	157
建立工程雏形	158
依赖关系的定义	160
执行测试	161
导入Eclipse	162
5.2.2 为已有工程添加自动build 功能	162
5.2.3 build工具的总结	163
5.3 测试代码的写法	164
5.3.1 作为CI的对象的测试的种类	164
5.3.2 何时编写测试	165
新建工程的情况	165
已有工程中没有测试的情况	165
修改bug或添加新功能的情况	166
5.3.3 棘手的测试该如何写	166

和外部系统有交互的测试	166
使用mocking框架进行测试	167
使用内存数据库进行测试	168
数据库变更管理和配置文件管理的测试	169
UI 相关的测试	169
棘手的测试要权衡工数	170
5.4 执行基于Jenkins 的CI	171
5.4.1 Jenkins的安装	171
使用本地安装包进行安装	172
5.4.2 Jenkins能干些什么	172
5.4.3 新建任务	173
5.4.4 下载代码	173
5.4.5 自动执行build 和测试	175
定期执行	175
轮询版本管理系统	175
专栏 从版本管理系统进行Push	176
build 的记述	177
5.4.6 统计结果并生成报表	178
专栏 以JUnitXML 的形式输出报表比较高效	179
5.4.7 统计覆盖率	179
覆盖率统计工具	180
Maven Cobertura插件的安装	180
专栏 Java 程序库的查找方法	182
Jenkins 插件的配置	183
5.4.8 静态分析	184
5.4.9 配置通知	185
5.5 CI 的运用	187
5.5.1 build 失败了该怎么办	187
Subversion 等中央集权型版本管理系统的情况	187
Git 等分布式管理系统的情况	187
专栏 造成build 失败后的惩罚游戏	188
测试后合并	189
5.5.2 确保可追溯性	193
关联build 和提交	193
关联缺陷管理	194
5.6 本章总结——借助CI 能够实现的事	198
第6章 部署的自动化（持续交付）	199
6.1 应该如何部署	200
6.1.1 部署自动化带来的好处	200
细粒度、频繁地发布可以使风险可控	200
能尽快地获得用户的反馈	200
团队的规模可控	201
6.2 部署的自动化	202
6.2.1 部署自动化方面的共识	202
6.2.2 部署流水线	203
通过自动化加快部署速度	204
任何人都能够实施部署是很重要的	204
6.2.3 服务提供工具链（provisioning tool chain）	204
6.3 引导（Bootstrapping）	206

6.3.1 Kickstart	206
Kickstart 的使用方法	206
使用时的注意事项	206
Kickstart 的配置示例	207
6.3.2 Vagrant	208
为每一位开发人员准备实体电脑比较困难	208
使用虚拟机时的注意事项	209
什么是Vagrant	209
Vagrant的安装及运行方法	209
6.4 配置 (Configuration)	212
6.4.1 不使用自动化时的问题	212
6.4.2 Chef	213
Chef的构成	213
目录构成和文件配置	215
node.json	215
setup.json	216
solo.rb	216
default.rb	217
virtualhost.conf.erb	218
Chef的运行方法和运行结果	218
使用Chef的优点	219
使用Chef时的注意事项	220
使用Chef的时间点	220
6.4.3 serverspec	221
什么是serverspec	221
serverspec的安装	221
测试文件的记述方式	222
httpd_spec.rb	222
git_spec.rb	223
serverspec的执行方法及执行结果	223
serverspec的优点	224
6.4.4 最佳实践 (其1)	224
Vagrantfile	226
default.rb	227
6.4.5 最佳实践 (其2)	227
6.4.6 实现物理服务器投入运营为止的所有步骤的自动化	229
6.5 编配 (Orchestration)	230
6.5.1 发布作业的反面教材	230
6.5.2 Capistrano	231
Capistrano的系统构成	231
Capistrano的安装	232
deploy.rb	232
Capistrano 的执行方法	233
6.5.3 Fabric	233
Fabric (串行执行) 的情况	234
Capistrano (并行执行) 的情况	234
理解本地服务器和远程服务器操作上的区别	234
Fabric的运行方法	236
6.5.4 Jenkins	237

主节点 (master node) 和从节点 (slave node) 的协作	237
从节点的添加	238
任务的添加	240
任务的执行	242
6.5.5 最佳实践	243
结合Jenkins和Fabric	243
6.5.6 考虑安全问题	244
专栏 手动部署的例子	245
6.6 考虑运用相关的问题	247
6.6.1 不中断服务的部署方法	247
6.6.2 蓝绿部署 (blue-green deployment)	247
6.6.3 云 (cloud) 时代的蓝绿部署	250
6.6.4 回滚 (rollback) 相关问题的考察	251
随时准备好退路	251
数据库模式的版本管理	251
回滚的验证	252
只更新代码的发布时的回滚	252
数据库模式更新时的回滚	253
6.7 本章总结	255
专栏 PaaS的使用方式	255
第7章 回归测试	259
7.1 回归测试	260
7.1.1 什么是回归测试	260
7.1.2 测试分类的整理	261
支持团队的技术层面的测试 (第1象限)	262
支持团队的业务层面的测试 (第2象限)	262
评价产品的业务层面的测试 (第3象限)	262
使用技术层面测试的产品评价 (第4象限)	263
7.1.3 回归测试的必要性	263
退化 (degrade) 的发生	263
应该实现自动测试的原因	263
7.1.4 回归测试自动化的目标	265
7.2 Selenium	266
7.2.1 什么是Selenium	266
7.2.2 Selenium的优点	266
自动化测试用例制作简单	266
支持多种浏览器及OS	266
7.2.3 Selenium的组件	267
Selenium IDE	267
Selenium Remote Control (Selenium RC)	268
Selenium WebDriver	269
7.2.4 测试用例的制作和执行	271
Selenium IDE的安装和运行	271
Selenium的测试用例	271
什么是好的测试用例	274
用Selenium Server来运行测试	274
7.2.5 Selenium的实际应用	276
测试页面是否有改动	276
使Selenium测试稳定运行	278

7.3 Jenkins和Selenium的协作	282
7.3.1 关联Jenkins和Selenium的步骤	282
7.4 Selenium测试的高速化	287
7.4.1 利用Jenkins的分布式构建实现测试的并行执行	288
Jenkins的分布式构建的构成	288
分布式构建的配置	289
7.4.2 Selenium测试并行化中的难点	291
7.5 多个应用程序版本的测试	295
7.5.1 应用的部署	296
7.5.2 从版本管理系统下载测试用例	296
7.5.3 用Selenium测试	296

《高效团队开发》

精彩短评

- 1、讲的东西都用过，串起来讲了一遍还行
- 2、感觉很一般啊。很多都是粗略的介绍，而且作者翻译的也不是很流畅
- 3、明了
- 4、算是系统的讲解了开发团队提高工作效率和规范工作流程的理念，很多东西都需要展开了去实践，提供了一个不错的全局视野。
- 5、推荐，蛮好的书，在工程化的路上的工具都有介绍到
- 6、将团队开发的工具方法讲得比较浅显易懂，但是很多例子都只适用于web项目，对于单机或是内部环境的还需要调整适应。可以作为参考
- 7、最近越来越喜欢岛国的书，这本书条理清晰，对问题分析和方案选择的考虑都解释得比较清楚，图释也很棒，但后面过多的具体技术细节很无趣。
- 8、写得比较中肯，那种本国与硅谷的差距，是如此地相似
- 9、介绍大量的团队开发工具与方法，虽然许多工具面向的是WEB与桌面平台，但其方法还是相当值得借鉴与学习。
- 10、版本管理和缺陷管理很好，测试和部署不适合嵌入式系统！
- 11、相当于一篇综述吧
- 12、方方面面都讲到了，看完之后对这方面有更深刻的了解，同时作者还力图照顾到细节，虽然说这似乎不必要。
- 13、深度不够。只是简单的汇总几个工具串烧一下。团队自动化势必成为主流，但是自动化的界限在哪里，很难讲。在小团队来讲，git和jenkins已足够，自动化测试反而增加太大工作量。
- 14、走马观花看了一遍，适合知识普及和工具选型。
- 15、看起来很厉害的样子，但最终证明不咋滴

1、这本书花了2天时间粗略过了一遍，据说这本书还得了什么奖，看起来感觉并没有想象的那么好。书的内容可分为两部分，一部分是普及敏捷开发的概念，作者举出了一些开发中“常见”的例子，通过这些案例对比，很通俗直观的表现出了敏捷开发的优势所在。还有一部分就是讲解敏捷开发会用到的一些工具的用法。作者似乎非常的推崇敏捷开发，可实际上“敏捷开发”的一些实践还颇多争议，而有一些即便没争议也夸大了实际的效果。本书的前几章举了一个例子，说某个团队在开发过程中不用版本控制软件，代码都是拷贝粘贴备份这种方式，之后作者引出了svn和git这样的版本控制软件，说有了这个神器，就永远告别了前面这种代码和文档的原始的管理方式了。作者说法自然是没有任何问题的，可是这都2016年了喂！还有不用版本控制的团队吗？本书的前面的好多章都是类似这样的内容，且不说现在几乎没有团队还以如此原始的方式开发项目，即便真的像作者说的这样，我认为也没有必要用这么多的篇幅来说明这件事，完全可以直接把使用这些工具的好处列举出来就可以了。当然如果有人想详细的了解一下这些开发的流程，还是可以买来参考一下的。本书还用大篇幅介绍了另外一个概念：TDD。作者认为TDD能让我们的项目质量可控，代码质量可控。真是这样吗？就我个人的经验来看，实际开发却并非如此。换句话说TDD这种东西只能领会其精神，不能一成不变的直接套用在项目开发中的。如果严格的遵循TDD的标准去编写方法级别粒度的测试用例，覆盖所有逻辑分支，其代码量往往是项目代码的好几倍。如果还需要多个系统之间的集成测试，整个测试过程会变得更加不容易，加上测试数据与测试环境的准备，会让编写测试代码和测试代码的后期维护变成一个极为沉重的负担。而如果系统逻辑本身很简单，这又会使得测试代码的编写成为一种多此一举的行为，测试代码也会变成一种形式化的“自我安慰”。比如：假定代码遇到某些输入边界会抛出一个异常，程序员写了一大坨代码来触发去验证程序逻辑是否正确，这种测试代码写得会让人很抓狂！而且测试还有另外一个问题，同一个用例项目编码早期可能跟发布时的代码有天壤之别，项目代码都是边开发边优化。而测试却是很实现细节的表现，过早的编写测试就意味着后面每一次主体代码的修改都伴随着测试代码维护逻辑一致性的修改。一些极端的TDD甚至主张的实践团队甚至会要求程序员先编写测试用例，在写符合测试用例的代码，他们认为这样就能保证项目的质量。这种做法更是加剧这个问题的产生，诚然，测试先行确实能一定程度的保证项目逻辑的正确性，但是仅因为这一点好处，就让开发本身变得负累过重值得吗？一旦出现这些问题，就一定会导致另外一个问题，因为维护性的问题使得代码质量会急剧下降。试想，如果程序员还没有编写测试用例，可能当他发现实现代码可以优化时顺手就修改了。如果同时还有许多测试代码对实现代码的引用，那么他必然就要对测试代码做出修改，可能是自己写的代码还好，无非多动手多改两处就好，而如果是别人写的代码呢？他本可以优化实现代码，却由于有测试代码的同步修改而对测试代码的不熟悉使得他就不愿意去修改了！其实单方面的过于强调这些概念，都是十分愚蠢的掉书袋的行为！依据项目的情况作出合适的选择，这才是正确的项目开发之道。

《高效团队开发》

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:www.tushu000.com