

池田尚史
[日] 藤仓和明 著
井上史彰
严圣逸 译

TURING 图灵程序设计丛书

提高项目质量、加快开发速度、降低运维成本

高效团队开发 工具与方法



版本管理 · Git · GitHub · 缺陷管理 · 持续集成 · Jenkins
Build工具 · 测试代码 · 部署 · 持续交付 · Vagrant
Chef · serverspec · Capistrano · 回归测试 · Selenium



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

高效团队开发 工具与方法



池田尚史
[日] 藤仓和明 著
井上史彰
严圣逸 译



人民邮电出版社
北 京

图书在版编目(CIP)数据

高效团队开发:工具与方法/(日)池田尚史,
(日)藤仓和明,(日)井上史彰著;严圣逸译.--北京:
人民邮电出版社,2015.6

(图灵程序设计丛书)

ISBN 978-7-115-29594-1

I. ①高… II. ①池… ②藤… ③井… ④严… III.
①程序设计 IV. ①TP311.1

中国版本图书馆CIP数据核字(2015)第109228号

内 容 提 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。

本书适合所有想要系统性地学习团队开发工具的人阅读。

◆ 著 [日]池田尚史 藤仓和明 井上史彰
译 严圣逸
责任编辑 乐 馨
执行编辑 杜晓静
责任印制 杨林杰

◆ 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>
北京隆昌伟业印刷有限公司印刷

◆ 开本:880×1230 1/32
印张:10
字数:298千字 2015年6月第1版
印数:1-4 000册 2015年6月北京第1次印刷

著作权合同登记号 图字:01-2014-7807号

定价:49.00元

读者服务热线:(010)51095186转600 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

致中文版的读者

感谢您购买了《高效团队开发：工具与方法》。同样要感谢已经决定购买本书的读者。还在犹豫是否购买本书的读者，如果您看了这篇序后决定购买，那将没有比这更令人高兴的事情了。

本书是由我、想能（SHANON）时期的同事藤仓和明先生与井上史彰先生共同写作的。这次是受到想能上海分公司总经理井上先生的委托来写的这篇序。

其实我并没有去过中国大陆。因此，从真正意义上来说，我并不知道中国软件行业的真实情况。当然，像阿里巴巴和腾讯这样的大公司还是知道的，但我没有在中国工作的经历。

中国的软件工程师极为优秀。我所在的公司就有很多非常优秀的中国工程师。OSS 社区也经常能看到中国的工程师，他们都十分令人敬佩。

中国工程师无疑是非常出众的，但中国的软件开发环境是怎样的呢？缺陷管理和分布式版本管理的运用，测试代码的覆盖和 CI 的配备，部署的自动化等机制的组合应用，这些我听说都还刚刚起步。

您所在的开发现场又是怎样的呢？

如果上述情形并未出现在您的开发现场中，那么非常抱歉，您不需要这本书。请将这本书放回书架，回去继续工作。如果存在上述情形，那么本书将对您有所帮助。

在日本，能够构建本书中所写的高效开发环境的公司和无法构建这样的环境的公司之间有着很大的差距。究其原因，其一是完备的环境有助于提高开发效率，能够迅速地发布优秀的产品或服务；其二是因为工程师注重团队开发环境是否完善，开发环境完善的公司能够吸引到优秀的工程师，而优秀的工程师越多开发效率自然就越高。这样一来，公司之间的差距就越来越大，这就是日本的真实情况。

同样的情况在中国也会发生，或许可能早就已经在发生了。

团队开发环境的完善就像“减肥”一样。明明知道只要去做就会有

效果、有益处，但却迟迟没有付之于行动。往往会以太忙了、有其他优先度更高的工作为由来应付过去。

本书第2章讲述了如果怠慢这个“减肥”会变成什么情形。那是我过去的真实体验，为了避免重复那样的经历，只要是能够提高团队开发效率的事情，我都会去尝试、实践，而本书就是这些尝试和实践的结晶。

请大家务必阅读、学习本书，避免陷入第2章中描述的悲惨境地。已经陷入上述境地的各位，更应该阅读本书，以便能够从上述境地中解脱出来。

上文已经提到，在日本是否能够实践本书的内容，决定了公司间的差距。各位读者也请学习、实践本书的内容，以使自己所在的公司能和竞争对手拉开差距。

如果您的实践一切顺利的话，请告知我一下，我们可以一起去喝一杯。只要有您的邀请，我随时都可以去中国！我非常喜欢绍兴酒，白酒也想尝试一下 :-)

作者代表 池田尚史

2014年11月15日 于千叶县自家

译者序

《高效团队开发：工具与方法》并不是以实际的项目带你体验多人开发项目的整体流程，而是告诉你使用哪些工具和方法能够实现高效的团队开发。从版本管理系统、缺陷管理系统到 CI 工具、虚拟化、自动化测试等，无论你使用哪种语言、框架、软件开发模式，无论你是负责开发、测试，还是负责运维、项目管理，都会涉及这些工具。这些工具也直接影响着开发和运维的效率、项目成本以及公司的日常开销。

随着 SaaS（软件即服务）的普及，越来越多的项目已经不是经过一段时间的密集开发就结束的了。后期的开发，包括集成、测试、运维（部署、发布等），从重要性以及成本的角度来看都已经成为项目中的重要部分。本书后半部分介绍的持续集成、自动部署（持续交付）以及回归测试，都能有效地帮助这样的项目提高质量、加快开发速度、降低运维成本。

本书让我印象较深的一点是贯穿全书的自动化意识，包括自动化环境构建、持续集成、自动化测试、自动部署和发布。点击鼠标提交代码和测试用例，借助 CI 和各类自动化工具，自动触发编译、集成、测试、部署，还会自动将版本管理系统中提交的信息关联到缺陷管理和 CI 系统中，几分钟后打开浏览器就能够“享受”自己的劳动成果了。这样的场景实在太美了。想来是因为日本长期的劳动力不足以及高昂的劳动力成本才让作者对于自动化如此执着。对于还能够享受人口红利的中国软件行业来说，自动化也是非常必要的。除了能够在开发、测试、运维等多方面降低成本之外，自动化环境构建、自动化测试这样的机制能够降低项目对于成员个体的依赖，在大规模的团队开发中以及在灵活调整团队规模方面都是必不可少的。

本书所介绍的内容对于公司来说不仅可以提高效率，降低成本，还可以成为公司的一张名片。持续集成、自动化测试、持续交付，加上 Github、Jenkins、Vagrant、Chef、serverspec、Selenium 这些工具，由此构筑起的技术堆栈，无论是对于开发、测试人员还是运维人员来说都是

非常具有吸引力的。对于个人来说，除了扩展自己的知识之外，还能作为你选择公司的重要参考依据，判断公司是否对技术敏感，推测项目大致的工作流程以及是否可能成为 Death march。更重要的是

员工：“老板，我要加工资！”

老板：“为什么？”

员工：“因为我长得帅！”

老板：“……”

员工：“因为我跟你 10 年了，没有功劳也有苦劳吧！”

老板：“好吧，加 5% 差不多了。”

员工：“这个项目交给我，我有办法只需要一半的人手就能完成！”

老板：“真的？好！工资翻倍！”

最后感谢在翻译过程中给予我支持及鼓励的各位。特别是我的妻子，翻译这段时间恰好是她怀孕和生产的时候。我们平安地迎来了家里的新成员滚滚，借此祝愿他健康成长。

严圣逸

2015 年 3 月于上海

序言

本书名为《高效团队开发：工具与方法》。

读者朋友们大多都知道，团队开发是一件复杂、困难的事情。

关于团队开发，现在已经有了各式各样的方法论和工具。方法论方面，除了 Scrum、XP 等敏捷开发以外，还有些更具体的设计开发方法，如 TDD（Test Driven Development，测试驱动开发）、BDD（Behavior Driven Development，行为驱动开发）、TiDD（Ticket Driven Development，缺陷驱动开发）等，以及具体实践，如 CI（Continuous Integration，持续集成）、CD（Continuous Delivery，持续交付）等。讲述这些方法论的书籍、杂志以及网站到处都是，甚至多得让人不知该从何处着手。

再看一下从技术上支持这些方法论的工具，缺陷管理系统、版本管理系统、自动化测试、静态分析工具、自动化部署工具等，仅是种类就有很多。即使是简单地列举每一类中具有代表性的工具，其数量就多得令人感到头晕。

并且和数年前相比，支持团队开发的工具已经变得非常易用，能方便地构建高效的开发流程。但由于信息量过多，并且很分散，所以想要系统性地学习或者对新人进行高效的培训都还是比较困难的。正是因为意识到了上述这些问题，笔者才有了写作本书的想法。

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重点，对团队开发的整体结构进行概括性的说明。并且对“为什么用那个工具”“为什么要这样使用”等开发现场常有的一些问题进行举例说明。

希望你能喜欢本书。

读者对象

本书适合的读者对象有：

- 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

- 现有项目中返工、延期问题频发，想了解能帮助自己解决这些问题的工具及其使用方法的人
- 测试是测试部门的事，与己无关；部署、发布是运维部门的事，与己无关——有如此想法的人
- 想了解最近能够在 Web 服务开发中发挥作用的工具的人

致谢

在写作本书的过程中有幸得到了诸多人士的帮助。在此特别感谢堀让治先生、冈村纯一先生、平田耕造先生、奥村康树先生、吉羽龙太郎先生、藤原大先生、中村知成先生、梅泽雄一郎先生、伊藤望先生、鹿糠秀俊先生、角田良太先生，他们极有耐心地审阅了我们粗劣的原稿，并提出宝贵意见。感谢 Vincent Driessen 先生和 Pablo Santos 先生提供的 Git-flow 图和 SCM 历史简图以及翻译许可。

还要感谢堀让治先生等想能（SHANON）株式会社的各位，在本书写作期间给予的各类帮助。

还有技术评论社的春原先生，从最初会面到现在的两年时间里，让您费心费力了。您一直极有耐心地等待着我那迟迟没有进展的原稿，实在是非常感谢。

最后要感谢我的妻子和两个孩子，对于休息日一直埋头写作的我，他们没有丝毫埋怨，给予了我百分之百的支持。还有陪着孩子玩耍的岳父岳母，以及我自己的父亲、母亲，谢谢你们！

作者代表 池田尚史

2014 年 3 月

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、正式环境上无法运行	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、正式环境上无法运行	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及共享	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
	① 建立问题票	128
	② 指定负责人	129
	③ 开发	129
	④ 提交	129
	⑤ 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词进行检索	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