《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

《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。

《高效团队开发》

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。

《高效团队开发》

书籍目录

《高效团队开发》

《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jekins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

《高效团队开发》

精彩书评

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，看下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