HZ BOOKS
华章科技

PEARSON

软件工程技术丛书

软件工程领域泰斗、极限编程之父Kent Beck经典力作，荣获第14届Jolt大奖，畅销不衰

不仅以案例的形式生动地呈现了测试驱动开发的原则和方法，而且详尽地阐述了测试驱动开发的模式和最佳实践

测试驱动开发

实战与模式解析

Test-Driven Development By Example

（美）Kent Beck 著　白云鹏 译

机械工业出版社
China Machine Press

软 件 工 程 技 术 丛

测试驱动开发

实战与模式解析

Test-Driven Development By Example

（美）Kent Beck 著　白云鹏 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

测试驱动开发：实战与模式解析 /（美）贝克（Beck, K.）著；白云鹏译. —北京：机械工业出版社，2013.7

（软件工程技术丛书）

书名原文：Test-Driven Development: By Example

ISBN 978-7-111-42386-7

I. 测… II. ①贝… ②白… III. 软件开发 IV. TP311.52

中国版本图书馆CIP数据核字（2013）第093739号

本书是测试驱动开发领域的开山之作，由软件工程领域泰斗、极限编程之父 Kent Beck 撰写，荣获第 14 界 Jolt 大奖，10 余年畅销不衰，具有里程碑意义。书中不仅以案例的形式呈现了测试驱动开发的原则和方法，而且详尽地阐述了测试驱动开发（TDD）的模式和最佳实践。

本书共 32 章，分为三大部分。第一部分（第 1 ~ 17 章）从简单问题入手，介绍了 TDD 的概念、优势与设计方法，再逐步深入到解决复杂问题的方式；细致讲解了如何在编写程序代码前编写自动化测试，如何先塑造一个设计再通过重构逐渐添加设计上的构思，如何为更复杂的逻辑创建测试等。第二部分（第 18 ~ 24 章）讲解用 xUnit 创建测试的实例，介绍如何利用 xUnit 框架创建自己的测试用例，便于高效地进行测试。第三部分（第 25 ~ 32 章）介绍 TDD 的设计模式，包括部分经典的设计模式以及如何将这些模式与 TDD 相结合，还介绍了重构的方法，以及 TDD 中的特殊问题等。本书从始至终贯穿了两个 TDD 项目，展示了如何轻而易举且卓有成效地编写优质代码的技术。

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：吴 怡

北京瑞德印刷有限公司印刷

2013年9月第1版第1次印刷

170mm × 242mm · 12.5印张

标准书号：ISBN 978-7-111-42386-7

定 价：59.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

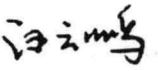
译 者 序

本书早在2002年由培生教育出版集团（Pearson Education）首次出版以来，十余年间，敏捷方法爱好者、业内专业人士将它口口相传，经久不衰。原因何在？我认为这是人们追寻更好的方法，期以获得更丰厚回报的天性使然。测试驱动开发（Test-Driven Development，TDD）始于20世纪90年代，历经岁月，成于2003年，也即本书首次出版之时。

从早期的面向过程程序开发到后来的面向对象软件开发，随之诞生了软件构件化、工程化的开发方法。在软件开发方法学中，测试驱动开发较之其他开发方法学可谓历久弥新，自诞生之日就一直受到业内高度关注，也从不乏勇于尝试和探索的有识之士。正如本书作者Kent Beck先生所说，测试驱动开发是一种正确的做事方法。这种方法打破了传统软件开发流程，提出了测试优先（Test First）的编程模式，其中的优点无须赘言。

本人也在实践当中尝试了书中的理论。整个过程轻巧安全，步步推进，给编码工作带来了耳目一新的感觉。译竣之后，也深感此书犹如测试驱动开发之蓝本，不负盛名。

本书由我负责翻译并整理，此过程中的辛酸苦辣一言难尽，但也感到与Kent Beck这样一位出色的业界领袖巨笔际会是一个享受的过程。在成书过程中，机械工业出版社的吴怡编辑给予了莫大的支持和鼓励。承荷兰威科集团（Wolters Kluwer）庚戈华工程师的悉心帮助，解决了不少翻译过程中遇到的困惑。微软美国总部的香钦柏工程师、美国密歇根大学安娜堡分校的葛瑾女士也审阅了书稿，提出了不少中肯建议。借此机会谨致谢忱。还要感谢给予了我很多支持和关心的家人，希望本书的出版，给你们带来快乐！



前　言

“可运行的简洁的代码”是Ron Jeffries关于测试驱动开发（TDD）目标的精辟概括。有一大揽子理由能够说明，可运行的简洁的代码是值得实现的目标：

- 它是一种可预测的开发方式。你会知道自己何时可以完成，而不必担心因缺陷而拖得太久。
- 它会给你全面认识代码的机会。如果你草率地采用了自己最初的想法，那就再也没有时间来考虑另外更好的想法了。
- 它让使用我们软件的用户生活变得更加美好。
- 它使成员之间相互信赖。
- 它本身的编写过程也让人感觉很棒。

但我们如何才能得到简洁的可运行的代码呢？很多因素使我们与简洁的代码渐行渐远，甚至导致代码无法运行。其实，不用为我们的担心浪费太多笔墨，这样做就可以：使用自动化测试驱动开发，一种叫做“测试驱动开发（简称TDD）”的开发方法。在测试驱动开发中，我们：

- 仅当自动化测试失败的时候才编写新的代码。
- 去掉重复的部分。

这是两项比较简单的原则，但这会产生出一个或一组技术方面复杂的行为准则，例如：

- 我们必须参考每次修改后代码运行状况的反馈，逐渐完成设计。
- 我们必须自己编写测试，因为我们不能指望其他人编写测试。
- 我们的开发环境必须对细微的修改迅速做出响应。
- 我们的设计必须遵从高内聚、低耦合的原则，这样便于实施测试。

那两项简单的原则预示了编程任务的先后顺序：

1. 红色指示条——编写一个无法工作的简单测试，当然，这个测试起初甚至都无法通过编译。

2. 绿色指示条——迅速使测试工作[⊖]起来，这个过程可谓想方设法，甚至不择手段。

⊖ JUnit中红色指示条表示测试未通过，绿色指示条表示测试无编译错误且全部通过。在本书中描述测试“工作”或“运行”表示测试可通过编译。——译者注

3. 重构——去掉单纯由于使测试工作起来而产生的重复部分。

红色指示条—绿色指示条—重构——此乃 TDD 的经典三部曲。

先假设这样的编程方式是成立的，进一步讲，这种方式可使代码缺陷的密度降低，还可以使所有相关人员对工作主旨若网在纲。如果这样的话，只在测试失败时才需要编写的代码还有其社会意涵：

- 如果可以将缺陷的密度降到足够低的话，那么，质量保证人员的工作状态就能够变被动为主动。
- 如果可以尽可能地减少令人不悦的意外发生，那么，项目管理人员就能够非常精确地估算出日常开发工作所需的人手。
- 如果可以将技术讨论的主题构思得足够清晰，那么，软件工程师就可以亲密无间地合作，而非拉锯式地工作。
- 同样，如果可以将缺陷的密度降到足够低，那么，我们每天都可以发布带有新功能的软件，这样会与客户建立新的业务往来。

诚然，这个理念并不复杂，但其背后的动机又是什么呢？为什么一位软件工程师要插手编写自动化测试以外的工作呢？为什么一位软件工程师能够给出高屋建瓴的设计却以细微的步伐前行呢？勇气使然。

勇气（Courage）

测试驱动开发是一种在编程过程中管理担忧的方式。这不是杞人忧天之虑，而是合理的担忧，但是否合理的问题是很难从一开始就看出来的。如果疼痛的自然反应是喊“停！”，那么担忧的自然反应则是喊“当心！”。当心历来是好事，但担忧却有不少其他的副作用：

- 担忧使你举棋不定。
- 担忧使你疏于交流。
- 担忧使你回避反馈。
- 担忧使你郁郁寡欢。

这些作用对于编程都是无益的，尤其是在编写有难度的程序时。因此，你该如何面对困难的局面呢？并且：

- 尽快开始具体地了解一些东西而不是试探。
- 更加清晰地交流而不是闭口不言。

- 找出具体的有用的反馈而不是回避。
- 克制负面情绪。

把编程想象成使用曲柄从一口井中摇上一桶水的过程。当水桶不太大时，自由旋转的曲柄还不错。当水桶比较大并且装满水，再把水桶提上来的时候，你就会感到累了。你需要一个在转动曲柄的间隙能够使你休息的棘齿装置。越重的桶，就需要棘齿装置的齿越密。

测试驱动开发当中的测试就像棘齿装置的齿。一旦测试运行起来，你就知道它会这样子运行下去了。比起测试坏掉的情况，你已经朝着一切运行如仪的目标又迈进了一步。现在，还可以使下一个，下下个和再下一个测试运行。以此类推，越是棘手的编程问题，每个测试的覆盖面要越小。

阅读本人著作《Extreme Programming Explained》[⊖]的读者会注意到描述极限编程（XP）和测试驱动开发在语气上的差异。TDD 与 XP 有所不同。XP 会要求你必须准备这样那样很多东西。但 TDD 却没有那样地泾渭分明。TDD 很明确地知道，设计和现实之间肯定存在很多差异，还告诉了你控制这种差异的方法。“我要是用一周时间做书面上的设计，然后用测试驱动代码如何？这算是 TDD 吗？”当然算啦！这就是 TDD。你意识到了设计与现实之间的差异，并且有意识地掌控它。

即便如此，大多数学习 TDD 的人发现他们的编程行为被永远地改变了。测试感染（Test Infected）是 Erich Gamma 发明的用来描述这一转变的说法。你可能会发现自己比以前会写更多的测试，会发现使用更小的步伐推进要比曾经自己想象的要合理。另一方面，一些软件工程师学习 TDD 后保留了他们原有的编程行为，在一些特殊的情况下，当常规的编程方法不奏效的时候，他们才拿出 TDD 救急。

当然存在不能仅仅（或者根本不能）由测试来驱动的编程任务，例如，软件安全性和并发问题，使用 TDD 去一板一眼地证明这两方面与需求的吻合度就会显得捉襟见肘。尽管事实上软件的安全性本质上依赖于无缺陷的代码，但也依赖于使软件变得安全所使用方法的人为判断。微妙的并发问题是不能通过运行代码就能可靠再现的。

一旦你阅读完这本书，你应该准备：

- 从简单的地方着手。
- 编写自动化测试。

⊖ 中文版书名为《解析极限编程——拥抱变化》，由机械工业出版社引进出版。——编辑注

• 通过每次重构添加一回设计上的构思。

本书分为三部分。

第一部分，货币实例——一个典型的使用 TDD 编写的示例代码。这个示例是数年前我从 Ward Cunningham 那里得到的，而且在多币种算法的例子中使用过多次。这个例子会使你学到在编码前就编写测试，并且逐渐地完成设计。

第二部分，xUnit 实例——一个为自动化测试开发的框架，测试包含反射和异常且拥有比这更复杂逻辑的例子。这个例子还会给你引入 xUnit 架构，这是众多面向程序员的测试工具之关键所在。在第二个例子中，你会学到用比前一个例子中更小的步伐来工作，包括那种计算机科学家们所钟爱的自我提醒式的念叨。

第三部分，测试驱动开发的模式——包括了决定可以编写哪些测试这样的模式，如何使用 xUnit 编写测试，以及精选的设计模式和在示例中使用的重构方法。

我将示例描述为结对编程的场景。如果你喜欢在闲逛前先查看地图的话，那么也许你要直接去看第三部分的那些模式，并且尝试其中所举的那些例子。如果你倾向于只是闲逛，事后再去看看地图你都到过哪里的话，那就试着逐个阅读书中的示例吧，当你想知道某个技术的更多细节时可以查阅这些模式，把这些模式作为参考。本书的几位审阅者说，当他们启动编程环境，输入代码并运行他们所读到的程序时，最大的收获来自于这些示例之外。

关于这些示例需要注意的一点是，多币种的计算和测试框架这两个例子显得比较简单。但却也存在着（我曾经见过）使用复杂，不雅且愚蠢的方法解决相同问题的情况。我本来可以选择其中某个复杂、不雅、愚蠢的方法来给本书以“真实感”。但是，我的目标是编写可运行的简洁代码，希望你的目标也是如此。在开始这些被认为非常简单的例子之前，花费 15 秒钟想象一下，一个所有代码都简洁明了，不存在复杂解决方法的编程世界，仅有复杂的问题需要认真思考。那么，TDD 完全可以帮助你得偿所愿。

致　谢

感谢广大直言不讳给予批评与建议的读者。我为本书所有内容负责，然而，没有他们的帮助，本书则会在可读性和实用性方面大打折扣。他们分别是：Steve Freeman、Frank Westphal、Ron Jeffries、Dierk König、Edward Hieatt、Tammo Freese、Jim Newkirk、Johannes Link、Manfred Lange、Steve Hayes、Alan Francis、Jonathan Rasmusson、Shane Clauson、Simon Crase、Kay Pentecost、Murray Bishop、Ryan King、Bill Wake、Edmund Schweppe、Kevin Lawrence、John Carter、Phlip、Peter Hansen、Ben Schroeder、Alex Chaffee、Peter van Rooijen、Rich Kawala、Mark van Hamersveld、Doug Swartz、Laurent Bossavit、Ilja Preuß、Daniel Le Berre、Frank Carver、Justin Sampson、Mike Clark、Christian Pekeler、Karl Scotland、Carl Manaster、J.B.Rainsberger、Peter Lindberg、Darach Ennis、Kyle Cordes、Patrick Logan、Darren Hobbs、Aaron Sansone、Syver Enstad、Shinobu Kawai、Erik Meade、Dan Rawshorne、Bill Rutiser、Eric Herman、Paul Chisholm、Asim Jalis、Ivan Moore、Levi Purvis、Rick Mugridge、Anthony Adachi、Nigel Thorne、John Bley、Kari Hoijarvi、Manuel Amago、Kaoru Hosokawa、Pat Eyler、Ross Shaw、Sam Gentle、Jean Rajotte、Phillipe Antras 和 Jaime Nino。

曾经与我一同进行过测试驱动开发实践的战友们，我衷心感谢你们的坚韧，特别是在早期，忍受我那听起来无比疯狂的想法。而我从你们身上学到了比自身领悟到的多得多的东西。我无意冒犯任何人，但是，Massimo Arnoldi、Ralph Beattie、Ron Jeffries、Martin Fowler 以及最后提及但很重要的成员 Erich Gamma 却都深刻在我的记忆中，他们就像测试的驱动者，使我受益良多。

我还要感谢 Martin Fowler 提供了关于 FrameMaker 方面适时的帮助。他一定是这个星球上报酬最高的排版顾问，而我（目前）却有幸享受着这样的免费服务。

我真正的程序员生涯始于 Ward Cunningham 耐心的指导和不断的合作。有的时候，我看到测试驱动开发作为一种尝试被推荐给任何软件工程师并在任意环境中使用，就像我们所使用的 Smalltalk 环境和程序那种亲密无间的感觉那样。当两

个人考虑同一件事情的时候，你没有什么办法分清这是谁的点子。如果你认为书中所有好点子都出自 Ward，这么想也不为过哟。

当某个家庭成员煞费苦心地将自己的奇思妙想表述成书的时候，对此谈论家人所做出的牺牲未免有点儿旧调重弹的味道。那是因为家人所做的牺牲就像写作需要纸张一样不可避免。感谢我的孩子们，他们只有等到我完成某个章节的撰写后才能吃上早餐，特别是我的妻子，两个月来事事都要对我提醒再三，在此聊表寸心。

感谢 Mike Henderson 对我循循善诱和 Marcy Barnes 雪中送炭。

最后，要感谢一位不知名的作者，我在彼时 12 岁的怪诞年华时读过他的书。该作者建议使用来自实际输入结果的期望结果，然后运行代码直到实际结果与期望结果吻合。谢谢，谢谢你们，谢谢大家。

引　言

某个周五的早晨，老板来找 Ward，并将他介绍给了 Peter，Peter 是公司正在出售的债券组合管理系统的潜在客户。Peter 说道：这套系统的功能给我留下了深刻的印象，但是，我也注意到你们的系统只限于处理美元证券，我新开设了一家证券投资基金，并且，我的战略需要处理不同币种的证券”。老板转向 Ward，问道：“呃，我们可以做吗”？

对于任何一位软件设计者来说这无疑是梦魇般的一幕。你按照设想的那套方案满怀欣喜且成功地步步推进，突然，一切都变了。梦魇不只是对于 Ward 而言，就连他的老板，这位指导软件开发经验丰富的老手也是不知所措。

WyCash 系统由一个规模不大的团队开发了两年。该系统可以处理美国证券市场大多数常见的各类固定收益有价证券，还可以处理少量国外的新型金融票据，比如担保投资合同（Guaranteed Investment Contract），而同类型的产品则无法做到这些。

WyCash 系统始终使用面向对象语言和面向对象数据库进行开发。作为计算的抽象基础，Dollar 的计算处理起初被外包给了一群聪明的程序员。开发出来的东西兼顾了格式和计算两方面的要求。

在过去的六个月里，Ward 和其他团队成员有条不紊地剥离出了符合要求的 Dollar 类。Smalltalk 语言的数值（numerical）类在计算方面表现还不错。所有关于三位小数四舍五入的微妙的代码都可以得出精确的结果。由于结果变得更精确，测试框架中存在着一定误差的、复杂的比较机制也被期望结果和实际结果精确的匹配取而代之了。

管理数据格式的职责实际上被划归到了用户接口类。由于测试代码，特别是生成报告的框架[⊖]被写在了用户接口类层面，这些测试不用去适应这样的改进。经过六个月悉心的剥离，结果是 Dollar 类的职责所剩不多了。

系统中最复杂的算法之一，加权平均，同样也经历着缓慢的转变。先前在系统中散布着许多不同形式的加权平均处理代码。由于生成报告的框架是从

⊖ 关于生成报告的框架更多信息，请参见 c2.com/doc/oopsla91.html。

那些最初的对象组合而成，因此加权平均算法显然需要有一个归宿，那就是AveragedColumn。

Ward现在要把注意力转向AveragedColumn了。如果加权平均算法可以处理多币种货币的话，那么系统其余部分应该是没问题的。加权平均算法的核心是将钱的数额存到列中。事实上，加权平均算法已经高度抽象，以致可以计算该算法所及的任何对象的加权平均值了。

周末像往常一样过去了，周一的早上，老板过来问："怎么样，我们能做吗"？

Ward答道："再给我一天时间，我给您确切的答复"。

因为Dollar在加权平均算法中扮演着计算器的角色，为了处理多币种运算，他们需要一个为每种货币备有计算器的对象，这有点儿像多项式。但多项式的因子不是$3x^2$和$4y^2$，而是15USD和200CHF这样的表达方式。

一个短暂的实验表明，使用一个泛化的Curency对象来代替Dollar对象做计算是可行的。当两类不同币种相加时，返回一个PolyCurrency对象。目前，棘手之处在于不能破坏任何已正常工作的部分而为新功能誊出空间。如果Ward运行那些测试的话，会发生什么事情呢？

为Currency类添加了一些尚未实现的操作后，那些测试整体上获得了通过。一天下来所有的测试都通过了。Ward将代码签入后跑到老板那里肯定地说："我们可以做！"

让我们回顾一下这个故事。在两天的时间里，潜在的市场被成倍地拓展了，还有WyCash的价值也是一样地倍增。但具有短时间内创造出如此巨大商业价值的能力绝不是偶然。几个因素发挥了作用：

- 方法——Ward和WyCash系统开发小组其他成员需要不断地实践来一点一点地塑造系统的设计，因此，系统的改进机制很好地被实施。
- 动机——Ward和开发小组其他成员完全理解开发WyCash系统多币种处理功能的重要商业价值，并有勇气接手这一看起来不可能完成的任务。
- 契机——全面且增加自信的测试，完美重构的程序和可以与设计独立的编程语言，所有这些事物的共同作用，意味着鲜有出现错误的根源，即使出现错误，也可以轻易地进行辨认。

能否得到任何机缘来运用技术魔力使得项目的价值翻番，这是你所不能控制的；但另一方面，方法和契机却是你可以完全控制的。Ward和他的团队融合了他

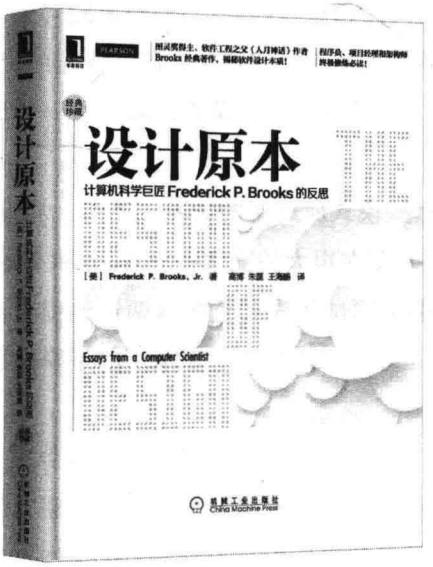
们的高超才智、经验和纪律而创造了方法和契机。这是否意味着，如果你不是这个星球上的十大杰出软件工程师之一，或在银行里有大笔存款，从而可以炒你老板的鱿鱼的话，你就非得花时间把事情做对不可，这么一来就永远也达不到这些目标呢？

当然不是。即使你是位技术一般的软件工程师，并且，当身处压力之中，有时会屈服于压力，寻找捷径，你也完全可以掌控项目，使其出现奇迹。测试驱动开发是一项支持简单的设计，支持激发自信的测试集，并且任何软件工程师都能够使用的技术。如果你是个天才，那就根本不需要这些条条框框。而如果你是个蠢材，这些规则也无济于事。对于我们这些介于两者之间的大多数人，下面这两条规则可以帮助我们发挥最大潜能去工作：

- 在你编写任何代码前先编写会运行失败的自动化测试。
- 去掉重复部分。

到底如何去做呢？如何巧妙地分阶段并张弛有度地使用这两条规则是本书的主题。我们就从 Ward 灵感迸发时创造的对象——多币种货币实例开始。

推荐阅读



设计原本（精装本）

如果说《人月神话》是近40年来所有软件开发工程师和项目经理们必读的一本书，那么本书将会是未来数十年内从事软件行业的程序员、项目经理和架构师必读的一本书。它是《人月神话》作者、著名计算机科学家、软件工程教父、美国两院院士、图灵奖和IEEE计算机先驱奖得主Brooks在计算机软硬件架构与设计、建筑和组织机构的架构与设计等领域毕生经验的结晶，是计算机图书领域的又一史诗级著作。

领域特定语言

本书是DSL领域的丰碑之作，由世界级软件开发大师和软件开发“教父”Martin Fowler历时多年写作而成。全面详尽地讲解了各种DSL及其构造方式，揭示了与编程语言无关的通用原则和模式，阐释了如何通过DSL有效提高开发人员的生产力以及增进与领域专家的有效沟通，能为开发人员选择和使用DSL提供有效的决策依据和指导方法。

目　录

第三部分　测试驱动开发的模式

第一部分

货币实例

在本部分中，我们将完全采用由测试驱动的方法进行典型样例代码的开发（除非我们出于教学目的而故意地漏掉某段代码）。本部分旨在让你掌握测试驱动开发的节奏，即：

1. 迅速地添加一个测试用例。
2. 执行所有的测试，并查看新添加的这个用例得到失败结果。
3. 对代码做少量修改。
4. 再次执行所有的测试，并查看所有的用例都得到通过结果。
5. 对代码进行重构，消除重复部分。

可能会让你感觉意外的是：

- 每个测试用例可以怎样去覆盖小型功能增量。
- 为使得新的测试用例得以通过，代码要做的修改是怎样地微观和拙劣。
- 测试的执行是怎样地频繁。
- 重构的结果是由怎样大量的微观步骤一步步达成的。