The Pragmatic Programmers

TURING 图灵程序设计丛书

# C++ 程序设计实践与技巧

## 测试驱动开发

【美】 Jeff Langr 著
余飞 秦涛 译

- 敏捷大师Bob大叔作序推荐
- 提供大量现代C++编程的实践和技巧
- 深入探讨测试驱动开发、设计原则和优秀软件开发工艺

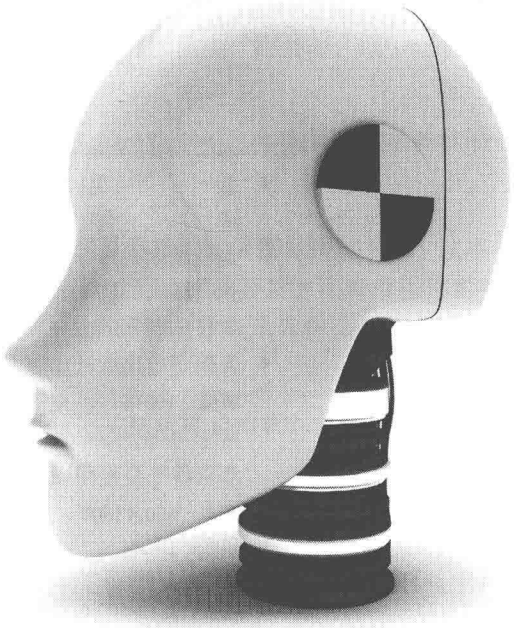
中国工信出版集团
人民邮电出版社 POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

# C++程序设计实践与技巧

## 测试驱动开发

【美】 Jeff Langr 著
余飞 秦涛 译



Modern C++ Programming
with Test-Driven Development
Code Better, Sleep Better

人民邮电出版社
北 京

**图书在版编目（CIP）数据**

C++程序设计实践与技巧 : 测试驱动开发 / (美) 杰夫·兰格 (Jeff Langr) 著 ; 余飞, 秦涛译. -- 北京 : 人民邮电出版社, 2017.1
(图灵程序设计丛书)
ISBN 978-7-115-43895-9

Ⅰ. ①C… Ⅱ. ①杰… ②余… ③秦… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2016)第264953号

## 内 容 提 要

本书是一本关于设计原则、编程实践、测试驱动开发的指南，旨在帮助C++程序员用测试驱动开发方法构建高性能解决方案。全书共11章，涵盖测试驱动开发的基本工作方式、潜在好处、怎样利用测试驱动开发解决设计缺陷、测试驱动开发的难点和成本、怎样利用测试驱动开发减少甚至免除调试工作，以及如何长时间维持测试驱动开发。

本书适合所有技术层次的C++程序员阅读。

◆ 著 [美] Jeff Langr
译 余 飞 秦 涛
责任编辑 朱 巍
执行编辑 杨 婷
责任印制 彭志环

◆ 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
北京昌平百善印刷厂印刷

◆ 开本：800×1000 1/16
印张：19.25
字数：449千字 2017年1月第1版
印数：1－3 500册 2017年1月北京第1次印刷
著作权合同登记号 图字：01-2014-0500号

定价：59.00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：京东工商广字第8052号**

# 本书赞誉

Jeff Langr又写了一本很棒的书。这一次他把测试驱动开发引入到C++世界。Jeff的示例让我们近距离地领略到好的测试驱动开发方法简约的一面。他解释了为什么要以这种方式工作，然后提出了重要的实践细节，涉及测试替身、与遗留代码打交道的方法、对付多线程代码，还有很多其他的内容。每个使用C++的开发者都应该拥有这本书。

——Ron Jeffries，极限编程方法学创始人之一，《软件开发本资论》作者

Jeff Langr写了近年来最好的一本C++图书。《C++程序设计实践与技巧：测试驱动开发》是理论与实践的完美结合。书中对抽象概念的解释清晰明了、妙趣横生，需要时，细节之处也是唾手可得。本书无疑是C++和测试驱动开发方面的经典之作。

——Michael D. Hill，极限编程方法教练和作家

Jeff Langr是软件开发方面的专家。他在这本书中分享了关于构建优秀软件的智慧。这本书不是讲测试的，但是你依然能学到重要的测试技巧。本书是通过测试驱动开发方法来改善你的技术、代码、产品和生活。无论你是哪个层次的C++程序员，Jeff都将向你展示为什么使用测试驱动开发能构建出更好的C++软件产品，以及如何做到。

——James W. Grenning，《测试驱动的嵌入式C语言开发》作者

# 译 者 序

回想当初，其实我刚开始打算翻译的是一本关于调试的图书，后因其他缘由翻译了此书。当时想的是，唔，测试驱动开发在平常的开发过程中也会用到，应该不算陌生。看到本书原稿后，我便坚定了翻译此书的决心。一方面，本书就像一本关于测试驱动开发的《一万个为什么》，它回答了一系列的问题，比如什么是测试驱动开发？怎样进行？难点在哪？它为什么能有助于构建好的解决方案？你需要什么样的测试？测试该怎么写？测试需要维护吗？测试驱动开发中所编写的单元测试和非测试驱动开发中所编写的单元测试有何区别？总之，几乎你遇到的疑惑，本书中都有阐述。另一方面，它是以一个个示例的方式来阐述的，代入感非常强。

回头看测试驱动开发本身，它是早期XP（eXtreme Programming）运动的一部分，并且当下的众多敏捷开发团队也大多采用测试驱动开发或其变种。测试驱动开发的增量开发和敏捷流程相契合，能够提供更快的反馈，而快速的反馈意味着开发将更加精准和灵活。另外，测试驱动开发中所编写的测试会让你在不断调整代码的过程中不轻易破坏已有的行为。敏捷专家Bob大叔曾声称他的Fitness有将近6.4万行代码，而测试占据了2.8万行。这些测试可在90秒内运行一遍。后来Fitness中又加入了2万行代码，而整体代码的缺陷仅为10多个，这些测试显然为代码的持续修改和快速发布夯实了基础。

测试驱动开发的固有周期可以轻松地将你带入一个有序的开发轨道。在每个周期中，你将专注于思考真正的需求、解决方案及设计重构。它将你置身于一个框架下，不断锤炼你在各个周期所需要的技能。在测试的编写阶段，你会更加细致地思考需求，考察代码覆盖率等；在编写产品代码阶段，你可以专注于初始架构设计和编程实践；在代码重构期，除了重新考量架构，你还可以做任何你想做的优化。当然，有序的开发轨道并不意味着一路顺畅，你依然会在每个时期遇到各类挑战，这时你需要根据形势作出正确的判断和决定。

刚开始接触测试驱动开发的开发者可能会觉得用这种开发方式节奏有点慢。可能慢在写测试（下一个测试是什么，测试的命名，怎么写测试），也可能慢在审阅和重构。一开始，你花在测试上的时间可能要多于开发产品代码的时间，但坚持住，你所作出的努力都会得到回报的。不知道读者朋友们有没有过练习一门武术的经历。译者有幸曾跟随过一位旅居上海的华人练习咏春直至他离开上海，师傅经常提醒我，有的动作要慢慢练，慢到别人几乎看不出来你在动。所谓天下武功，唯快不破！这个快其实是无数慢速训练量变后所呈现的质变。在修习测试驱动开发时，有时候你也要慢慢做。在这个慢的过程中你一定会看到以前没有看到过的风景。

测试驱动开发易学难精，其原因在于各类软件或其模块的行为复杂度不一。此外，测试驱动开发的内容繁杂，在总体呈优的情况下也存有一些缺点。在运用测试驱动开发时，必要时需结合所开发软件的特点，做到扬弃地使用。最后，测试驱动开发符合事物发展的一般规律，其自身也在不断发展，本书只能算是引领你登堂入室的第一课，所以不要止步于此！要不断学习、实践和总结。当然，除了技术上的因素，处于乐于推行测试驱动开发的工作环境也是十分必要的。坦白讲，我第一次在生产环境下试图使用真正的测试驱动开发时，过程还是挺狼狈的。

翻译是一个学习、积淀的过程，当然也是艰辛的。感谢我的家人，特别是我的妻子从一开始就非常支持我，翻译的过程占据了大量本可以陪伴家人的时间，但是你们却一如既往地给我支持。

在翻译本书的后期，由于工作和生活上的事情使得翻译时间缩减，因此我邀请了好友秦涛帮忙翻译第9章和第10章。多谢你在繁忙之中拔冗相助，使得翻译得以及时完成。

感谢朱巍老师从一开始就给予的信任和支持。感谢各位编辑老师为本书付出的辛勤汗水。没有你们的帮助和支持，很难想象出版本书是怎样的过程。

由于时间和水平有限，文中肯定存有一些瑕疵，或者读者朋友们有什么问题，都可以发邮件至fei_faith@outlook.com作进一步讨论。

最后以Langr为原书提供的副标题与各位程序设计同仁共勉：Code Better, Sleep Better!

余飞

2016/10/10，上海

# 序

不要被书名误导。

我的意思是，这是一本关于设计原则、编码实践、测试驱动开发和技艺的非常非常好的书，却起了个“Modern C++ Programming with Test-Driven Development:Code Better,Sleep Better”的名字。唉！

不要误会，这本书的确是关于现代C++编程的。如果你是C++开发者，你会喜欢上书中所有的代码。这本书中充满了有趣的、编写良好的C++代码。事实上，我认为代码略多于文字描述。翻阅一下这本书吧。你能找到没有代码的一页吗？我敢打赌没多少！所以，如果你正在找一本现代C++实践方面配以大量示例的好书，那么你算是找对了！

但是，这本书讨论的内容远不止现代C++编程，而是讲了很多很多！

首先，这是我见过测试驱动开发方面内容最完整、论述最好的书（我看过的书太多了）。几乎在过去十五年里，未经讲述的每个测试驱动开发问题在这本书中都有讨论。从脆弱性测试到模拟（mock），从伦敦流派[①]到Cleveland流派，从Single Assert到Given-When-Then[②]，这本书中都讨论了，而且还不止这些。此外，这本书并非泛泛地概述一些没有关联的问题。相反，它详述每个问题，配以示例和讨论。可谓是问题现于代码，也解于代码。

需要先成为C++程序员才能理解这本书吗？当然不需要。书中的C++代码十分整洁，概念也十分清晰，所以，Java、C#、C甚至Ruby程序员理解起来都没问题。

其次，书中讲述了设计原则！谢天谢地，这本书是一本设计教程！它带你遍及令人目不暇接的原理、问题和技巧。从单一功能原则到依赖倒置原则，从接口隔离原则到简洁设计背后的敏捷开发原则，从DRY[③]到Tell-Don't-Ask[④]。本书收纳了各种软件设计方法和方案，而且这些方法就

---

① 测试驱动开发中有多种流派，其中伦敦流派起源于创立于伦敦的ETC（Extreme Tuesday Club），这个俱乐部提倡敏捷开发。——译者注

② Give-When-Then是一种指导编写测试的模板。可以参考http://guide.agilealliance.org/guide/gwt.html。——译者注

③ DRY是Don't Repeat Yourself的缩写，此原则倡导消除各式各样的信息冗余。更多内容请参考https://en.wikipedia.org/wiki/Don%27t_repeat_yourself。——译者注

④ 这个原则倡导一个对象应该命令其他对象该做什么，而不是去查询其他对象的状态来决定做什么。遵循此原则会带来更好的封装性。——译者注

呈现在真实问题和真实代码解决方案之中。

之后，书中还讲述了编码实践和技巧。这本书通篇都在讲解这些内容，从小方法到结对编程，从编程招式到变量名。书中不仅有大量的代码供你一窥优秀的实践和技巧，作者还以恰当的讨论和演绎入木三分地解释了这些主题。

是的，书名是彻底的败笔。这不是一本讲C++的书，而是一本论述优秀软件开发工艺的书，只是恰巧使用C++来书写示例罢了。真的，书名应该叫“软件技艺：现代C++示例”（Software Craftsmanship: With Examples in Modern C++）。

所以，如果你是一名Java程序员，或者C#程序员，抑或是Ruby、Python、PHP、VB甚至COBOL程序员，都会有阅读此书的冲动。不要被封面上的C++字样吓到。不管怎样，先读一读。阅读的时候，也读一读附带的代码。你会发现它很容易理解。在学习好的设计原则、编码技巧、技艺和测试驱动开发时，你或许会发现学一点点C++其实也无妨。

“Bob大叔”Martin

Object Mentor公司创始人

# 前　言

尽管当下程序设计语言呈现爆炸式发展，C++却一如既往地坚挺。在2013年7月Tiobe最受欢迎的编程语言排行榜上，C++位居第四（最新的排名可以参见http://www.tiobe.com/index.php/content/paper-info/tpci/index.html）。2011版ISO标准（ISO/IEC 14822:2011，即C++11）定义了一些新的C++特性，这些特性可以提升C++的接受度，至少能减弱反对使用C++的声音。

对于构建高性能解决方案，C++依然是最好的选择之一。如果你所在公司的产品要与硬件集成，那么很有可能公司已经拥有了一个基于C++的庞大系统。如果你的公司在20世纪90年代或更早之前就存在，很有可能拥有一个使用了很长时间的基于C++的系统，而且它在未来的几年内也不会消失。

假设在工作中使用C++，你可能会思考以下几件事情。

- 现在是2013年，为什么我会回头使用这一难用（但好玩）并且早在几年前就想摈弃的编程语言？我该怎样做才不会自讨苦吃？
- 我是一个C++老手，对这门功能强大的语言了如指掌。多年来我一直使用它顺利地进行开发工作。我为什么要改变工作方式呢？
- 我的职业前景在哪里？

谈谈我自己吧！我从20世纪90年代初期开始使用C++，那个时候还没有模板（和模板元编程）、RTTI、STL和Boost这样的东西。从那时起，我有几次必须使用这门强大的语言，但是结果有点令人沮丧。和众多编程语言一样，C++也时常会让你搬起石头砸自己的脚，但使用C++的不同之处是，有些时候你不会意识到即将要砸到脚，等真正发生时已为时过晚。与使用其他编程语言相比，你可能会经受更多的痛苦。

如果你已经使用C++工作了很多年，可能采用了许多惯用法来提高代码的质量。在所有编程语言开发者中，铁杆的C++老手相对更加仔细，因为长时间使用C++开发而不出问题，需要对代码给予一丝不苟的关注和照料。

你或许会想，如此精心照料，C++系统的代码质量应该很高吧！但是，大部分C++系统都会暴露出如下一些相同的问题，而且会不断地出现。通常，这些问题和编程语言无关：

- 几千行的巨量源代码；

- 包含成百上千行晦涩代码的成员函数；
- 大量的无用代码；
- 编译时间超过几个小时；
- 居高不下的代码缺陷数目；
- 快速修复导致逻辑复杂晦涩而难以安全地管理；
- 散落在多个文件、类和模块间的重复代码；
- 使用早已废弃的编程实践。

这些问题不可避免吗？答案是可以避免，测试驱动开发（Test-Driven Development，TDD）正是这样的工具，你可以掌握并运用它来降低系统的熵[①]。它甚至可以重振你对编程的热情。

倘若你只是想找份工作，大量的C++工作机会能让你不愁就业问题。但是，C++是一门高度讲究技巧、很微妙的编程语言，粗心大意地使用会导致代码缺陷、间歇性故障和旷日持久的调试工作。这些也会使你的工作没有保障。好在测试驱动开发帮得上忙。

往一个庞大且存在了很长时间的C++系统中加入新的功能通常很耗时，而且无法估计进度。只是理解一段代码然后修改几行，有可能要花费几个小时甚至几天。开发者需要花上数小时等待代码改动编译完成，并且要等待更长的时间去检测这些改动是否和现有的系统完好集成，这会导致生产力进一步下降。

其实这都是可以避免的。早在20世纪90年代末，TDD就被用来帮助不断往系统中加入新的特性，同时保持C++系统在可控的范围内。（先于代码写测试的观念已经存在很长一段时间了，但是，正式的TDD周期是Ward Cunningham和Kent Beck在《测试驱动开发》一书中提出的。）

本书的主要目的是教你实际使用TDD的系统方法。你将会学到：

- TDD的基本工作方式；
- TDD的潜在好处；
- TDD怎样帮助解决设计缺陷；
- TDD的难点和成本；
- TDD怎样减少甚至免除调试工作；
- 怎样长时间维持TDD。

## 它适合我和我的系统吗

“这和单元测试有什么关系？似乎并不能帮到我太多。”

或许你已经尝试过单元测试，抑或正在努力为一个旧系统编写单元测试。TDD似乎对于工作

① 熵是热力学的概念，是在物质微观热运动时，用以表达其混乱程度的量。最初由香农引入信息论中，用以衡量信息的不确定性或随机性。这里亦可以表示软件系统混乱、不稳定的程度。——译者注

在新系统上的幸运儿是好东西，但它能解决C++系统上长期存在的棘手的日常问题吗？

的确，TDD是一个有用的工具，但不是对付旧系统的良方。虽然可以在开发系统新特性时使用TDD，但依然需要清理系统中多年积累的障碍。为了持续这样的清理工作，需要额外的战略和战术。你需要学习战术性的依赖消除法和安全的代码修改方法，这些可以在Michael Feathers的《修改代码的艺术》一书中找到。你还需要理解在不引起很多问题的情况下，怎样进行大规模的重构。这方面的内容可以参考*Mikado Method* [BE12]。本书也会传授此类实践和其他技术。

通常，只为努力阐明“系统就是这样的”而为现有代码添加单元测试[我称之为开发后测试（Test-After Development，TAD）]，收效甚微。可能投入了大量人力编写此类测试，但对系统的质量影响却很小。

若要让TDD帮助塑造系统，你的设计要从定义上是可测试的。这么做会有所不同，在许多方面要好于未采用TDD。越懂得什么是好的设计，TDD就越能帮你达到此目的。[①]

为了协助你转变思考设计的方式，本书强调潜藏于优秀代码管理方法背后的原则，如面向对象设计中的SOLID原则，这在《敏捷软件开发：原则、模式与实践》一书中有所讲述。本书将讨论优秀的设计理念怎样保持持续的开发和生产力，还有TDD怎样帮助有心的开发者达到更一致、可靠的结果。

## 本书目标读者

本书旨在帮助所有技术层次的C++程序员，无论是对C++语言有基本理解的新手，还是长期浸淫在语言秘籍中的老手。如果已经有一段时间没用C++了，那么你会发现TDD的快速反馈周期会帮助你快速地重新拾起这门编程语言。

尽管本书的目的是传授TDD，但是无论使用TDD的经验是多还是少，你都会有所收获。如果你完全对编写单元测试没有概念的话，我会带着你逐步地了解TDD的基础知识。如果你是第一次听说TDD，会在本书中发现很多专家建议，这些建议以简单的方式呈现，配以直截了当的例子。甚至经验老到的测试驱动开发者也能找到一些有用的智慧结晶、有关实践的可靠理论基础，以及一些有待探索的新主题。

如果你是怀疑论者，则可以从多个视角探索TDD。我会在整本书中解释为什么我认为TDD能够很好地工作，我也会分享一些它不能很好工作的经历以及原因。本书不是一本宣传手册，而是革命性技术的探索旅程，令人大开眼界。

各类读者也会找到在团队中发展和维持TDD的办法。TDD很容易上手，但是你的团队可能会遇到很多挑战。怎样避免你的变革不被这些挑战折损？怎样避免灾难？第11章会提供一些我认为行之有效的方法。

① 一个好的设计，其内部行为规整有序。TDD有助于保持这种好的设计行为不变性。——译者注

## 阅读前提

为了使用随书附带的所有例子，你需要一个编译器和一个单元测试工具。有些例子会用到第三方程序库。下面将概览这三种要素。可以参见第1章进一步了解细节。

### 单元测试工具

在众多的C++单元测试工具中，我选择Google Mock（基于Google Test）作为书中大部分例子的工具。目前来说，它是关注度最高的，但是我选择它的主要原因是，它支持Hamcrest表示法[一种基于匹配器（matcher）的断言形式，用以提供具备较强表达力的测试]。第1章提供的信息将帮助你快速使用Google Mock。

但是，本书既不是Google Mock的专著，也不是其宣传手册，而是教授TDD的。你只会学到足够的Google Mock知识，用以有效地实践TDD。

对于有些示例，还需要另外一个单元测试工具——CppUTest。如果没有使用过Google Mock或者CppUTest，也大可放心，因为你会发现学习另外一个单元测试工具是非常容易的。

倘若你正在使用不同的单元测试工具，诸如CppUnit或者Boost.Test，也不用担心！这些工具和Google Mock在概念上很像，在实现方式上也类似。你可以轻松跟上，并且几乎可以用任何其他的C++单元测试工具来做TDD的例子。可以参见附录A，了解在选择单元测试工具时什么是最重要的。

本书中的大部分例子使用Google Mock用以mocking和stubbing，参见第5章。当然，Google Mock和Google Test是一起工作的，但是也能将Google Mock成功地集成进你所选的单元测试工具。

### 编译器

需要一个支持C++11标准的编译器。本书中的例子最初是使用gcc编译的，在Linux和Mac OS上可以直接使用。参见第1章获取如何在Windows上编译的信息。所有的例子都使用了STL，它是许多平台上现代C++开发中的基本组成部分。

### 第三方程序库

一些例子使用了免费的第三方程序库。参见第1章获取需要下载的程序库列表。

## 怎样使用本书

我在设计本书章节时尽量使它们功能独立，你可以随机挑选一章阅读，而不需要依赖其他章

节。如果使用电子阅读器的话，我也提供了充足的全文交叉索引，可方便、轻松地在章节间跳转。每一章都有一个总览和一个总结，并配以下一章预览。这些概要性小节的名称，与许多单元测试框架的初始化代码段和清理代码段一样，取名为Setup和Teardown[①]。

## 源代码

本书包含大量的代码示例。大部分代码与一个特定的文件名联系。你可以在本书的网站[②]上找到所有代码：http://pragprog.com/book/lotdd/modern-c-programming-with-test-driven-development。或者访问我的GitHub主页：http://gitbug.com/jlangr。

示例代码按照章节组织。在每一章的目录下，有很多以数字命名的目录，每个数字对应一个版本号。在各章的学习过程中，我们会引用这些版本号说明代码的演变。举个例子，以c2/7/SoundexTest.cpp为标题的代码，在第2章代码目录下第7个版本目录的SoundexTest.cpp文件中。

## 本书讨论

请加入本书论坛：https://groups.google.com/forum/?fromgroups#!forum/modern-cpp-with-tdd。论坛的目的是讨论书中内容，以及和C++测试驱动开发有关的方面。我也会发布一些和本书相关的有用信息。

## 如果你第一次接触测试驱动开发：本书包含什么

虽然本书适合所有人，但其主要受众是第一次接触TDD的程序员，所以书中的章节安排有相应的先后顺序。我强烈建议你完成第2章的练习。完成这样一个示例，你会对TDD背后的思想有深刻的领悟。切忌只是阅读，要尽量动手去做，并保证该通过的测试都能通过。

接下来的第3章和第4章是必读的。这两章详述了什么是TDD（和什么不是TDD）以及怎样构建测试。在学习mock之前（参见第5章），务必保证理解这两章的内容。mock是构建大多数产品级系统的必备技术。

不要自以为了解设计和重构而跳过第6章。践行TDD的重要原因是，它使你在通过持续重构改进设计的同时，能够保持代码干净整洁。大部分系统有着糟糕的设计和晦涩的代码，部分原因在于，开发者不愿意做足够的重构或者不知道怎么去做。你将学到足够的知识，了解怎样获益于一个更小、更简单的系统。

---

① 一般而言，每个测试会提供Setup和Teardown函数，用于测试的初始化和清理操作。作者特意为书中每章配备这一头一尾两小节，与测试框架的结构相呼应，颇具创意。相比于测试代码，为了便于书面阅读，Setup和Teardown在文中分别译作“开场白”和“结束语”。——译者注

② 本书在图灵社区上的地址是：http://www.ituring.com.cn/book/1303。——编者注

第7章将总结TDD的核心技巧。这一章考察了许多方法，它们能提高你在TDD上投资的回报。学习其中的一些技巧将能够决定TDD成功与否。

你肯定会被卷入一场与已有系统的斗争，而这个已有系统可能没有使用TDD。可以阅读第8章了解一些应对遗留代码的简单技术。

第9章专门讨论多线程下的TDD。TDD方法或许会让你眼前一亮。

第10章将深入一些特定的领域和关注点。你会发现一些TDD的最新思路，包括与本书不同的一些方法。

最后，你想知道怎样使你的团队采用TDD。当然，你也一定想确保持续使用TDD。第11章会提供一些你想要收入囊中的点子。

### 如果你有一些测试驱动开发经验

你可以随机阅读各个章节，但也会发现许多来之不易的智慧之言贯穿全书。

### 排版约定

书中会穿插一些长度不一的代码段。当正文引用这些代码时，将使用下面的约定。

- 类名和测试名采用大驼峰式命名法（UpperCamelCase）。
- 其他的代码内容以代码字体呈现。下面是一些例子：
  - 函数名（显示空参数表，即便所指的函数有一个或多个参数。有时候，我称成员函数为*方法*）；
  - 变量名；
  - 关键字；
  - 其他代码片段。

为了保持简洁并节省版面，代码清单通常会省略与当前讨论不相干的代码。大括号后的注释代替了省略掉的代码。如下所示，`for`循环体被省略。

```
for (int i = 0; i < count; i++) {
    // …
}
```

## 关于“我们”

本书是我们之间进行交流的载体。一般而言，我是在与读者谈话。当我特指“我”的时候（不是很经常），通常是在表述经验之谈或个人偏好。隐含之意是，这些表述可能并没有被广泛接受

（虽然有可能是个好想法）。

具体到编码，我希望不是你一个人在做，原因在于你仍在努力学习。我们将共同完成书中的编码练习。

## 关于我

我从1980年就开始编写程序了，当时我在读高二。我的编程职业生涯是从1982年开始的，当时我在马里兰大学工作，同时也在攻读计算机科学学士学位。2000年，我完成了从程序员到咨询顾问的角色转变，期间我很享受为Bob Martin工作，以及偶尔与Object Mentor的优秀人才一起工作的时光。

2003年，我创办了Langr Software Solutions，提供敏捷开发相关的咨询和培训服务。我的大部分工作是与开发者一起实践TDD，或者教授TDD。在真实的开发团队中，我坚持在咨询师/培训师和一线程序员两种角色间来回切换，以保证自己一直参与其中不掉队。从2002年起，我作为全职程序员为四个不同的公司工作了相当长一段时间。

我喜欢从事软件开发方面的写作。这是我深入学习事物的一种方式，我也乐于帮助其他开发者快速编写出高质量代码。这是我的第四本书，其他三本是*Essential Java Style: Patterns for Implementation* [Lan99]、*Agile Java: Crafting Code With Test-Driven Development* [Lan05]和*Agile in a Flash* [OL11]，其中最后一本是与Tim Ottinger合著的。我为Bob大叔的《代码整洁之道》撰写了几章。除了我自己的网络站点，我还在其他站点上写过一百多篇文章。我会定期更新自己的博客（http://langrsoft.com/jeff），也为Agile in a Flash项目（http://agileinaflash.com）写过或参与写作上百篇文章。

除了C++，我们还使用过许多其他编程语言，如Java、Smalltalk、C、C#和Pascal等。目前，我在学习Erlang，也能够使用Python和Ruby进行编程。除此之外，我学习过至少十几种编程语言来了解它们（有时候也是为了一时之需）。

## 本书中的C++代码风格

虽然我使用C++开发过各种规模的软件系统，但我并不认为我是一个语言专家。我读过Meyers和Sutter的一些重要著作，当然还有其他一些。我知道怎样让C++代码工作，以及如何使代码更具表达力和可维护性。我知晓这种语言的大部分深奥的用法，却尽量避免使用它们。我在本书中对聪明代码的定义是“难以掌控”。我会引导你另辟蹊径。

我写C++代码的风格偏向于面向对象（这当然是受到Smalltalk、Java和C#的影响）。我喜欢大部分代码以类的方式存在。本书中大部分代码遵从这一风格。例如第一个示例中的Soundex就是以类的方式构建的（参见第2章），当然不是必须这样做。我喜欢这种方式，如果你不喜欢，可以

用自己的方式来实现。

TDD的价值本身与C++编程风格无关，所以不要受我的风格影响而否定其潜力。过多强调面向对象有助于对测试替身（参见第5章）的理解，这个时候需要拆除难缠的依赖关系。如果置身于TDD中，时间久了或许会让你的编码风格向面向对象转变。这倒不是坏事！

我有点懒。鉴于示例规模较小，我尽量不用命名空间，但在实际的产品代码中我一定会使用它。

我喜欢尽量保持代码精简，这样会避免我认为的视觉混乱。因此在大部分实现文件中，你会发现use namespace std;，尽管许多人认为这是不好的风格。（保持类和函数小巧且功能单一，比“所有的函数应该有唯一的返回值”这样的指南更有用。）不必担心，TDD不会影响你坚持自己的标准，我也不会。

最后关于C++要说的是，这门语言博大精深。我确信有更好的方法来实现本书中的示例，而且我敢肯定有一些我没使用过的库。测试驱动的优点是，你可以重新以多种方式完成一个实现，却不用担心损害其他功能。无论如何，请写信告诉我一些改善的建议，前提是你愿意使用TDD的方式。

## 电子书

扫描如下二维码，即可购买本书电子版。

# 致　　谢

感谢我的编辑Michael Swaine和PragPorg的伙伴们，他们提供了指导和书中所用的资源。

感谢Uncle Bob为本书精彩作序！

非常感谢我的技术编辑Dale Stewart，他在整个成书过程中提供了非常有价值的帮助，尤其是对书中示例的反馈和帮助。

在写作过程中，我总是寻求真诚的反馈，而Bas Vodde针对全书提供了大量反馈。他是一个真诚而有力的幕后搭档。

特别感谢Joe Miller，他不辞劳苦地将示例移植到Windows下。

非常感谢提供想法和重要反馈的人们：Steve Andrews、Kevin Brothaler、Marshall Clow、Chris Freeman、George Dinwiddie、James Grenning、Michael Hill、Jeff Hoffman、Ron Jeffries、Neil Johnson、Chisun Joung、Dale Keener、Bob Koss、Robert C. Martin、Paul Nelson、Ken Oden、Tim Ottinger、Dave Rooney、Tan Yeong Sheng、Peter Sommerlad和Zhanyong Wan。如果漏掉了哪位，请接受我的道歉。

感谢对PragProg勘误页面提供反馈的人们：Bradford Baker、Jim Barnett、Travis Beatty、Kevin Brown、Brett DiFrischia、Jared Grubb、David Pol、Bo Rydberg、Jon Seidel、Marton Suranyi、Curtis Zimmerman，等等。

再次感谢Tim Ottinger，他撰写了前言的一部分，也为本书出谋划策。我怀念与你共事的日子！

谢谢你们帮助我让本书变得更好，只靠我自己是做不到的。