编写可测试的代码，避免 bug，提高软件质量！

Pearson

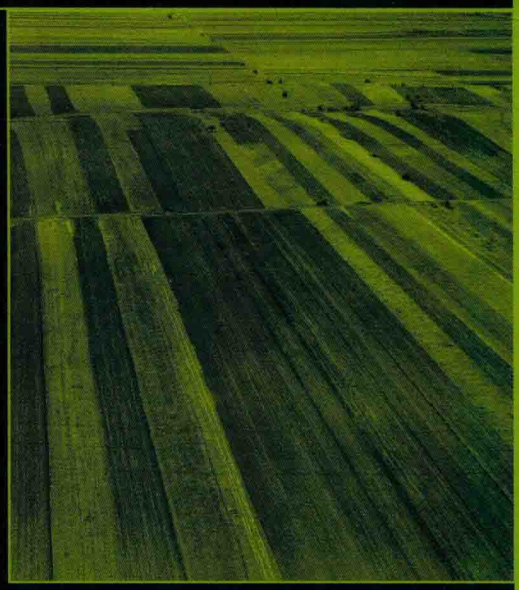
程序开发人员测试指南

构建高质量的软件

DEVELOPER TESTING

BUILDING QUALITY INTO SOFTWARE

【瑞典】亚历山大·塔林德（Alexander Tarlinder）著
朱少民 杨晓慧 欧阳辰 曾乐天 译



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

Pearson

程序开发人员测试指南

构建高质量的软件

DEVELOPER TESTING

BUILDING QUALITY INTO SOFTWARE

【瑞典】亚历山大·塔林德（Alexander Tarlinder）著
朱少民 杨晓慧 欧阳辰 曾乐天 译

人民邮电出版社
北京

图书在版编目（CIP）数据

程序开发人员测试指南 ：构建高质量的软件 /
（瑞典）亚历山大·塔林德（Alexander Tarlinder）著 ；
朱少民等译. -- 北京 ： 人民邮电出版社，2018.5
ISBN 978-7-115-48008-8

Ⅰ. ①程… Ⅱ. ①亚… ②朱… Ⅲ. ①软件－测试－
指南 Ⅳ. ①TP311.55-62

中国版本图书馆CIP数据核字(2018)第041733号

版权声明

◆ 著　[瑞典] 亚历山大·塔林德（Alexander Tarlinder）
译　朱少民　杨晓慧　欧阳辰　曾乐天
责任编辑　张　涛
责任印制　焦志炜

◆ 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
北京市艺辉印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：15.25
字数：365 千字　2018 年 5 月第 1 版
印数：1－2 400 册　2018 年 5 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-7881 号

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

译者简介

朱少民，国内软件测试界的领军人物和资深专家，三十多年来一直从事软件测试、质量管理和过程改进等工作，过去五年帮助了近百家企业提升其质量保证与测试能力，先后获得安徽省、原机械工业部、青岛市、合肥市等多项科技进步奖，出版了十多部著作，包括测试方面的畅销书《全程软件测试》《软件测试方法和技术》《完美测试》和译作《自动化测试最佳实践》等，经常在国内外会议上发表演讲，第一个在国内开设软件测试MOOC课程。之前曾任思科－网迅（中国）软件有限公司QA高级总监，目前是同济大学软件学院教授、中国科技大学软件学院教指委委员。

杨晓慧，前华为技术有限公司－软件公司首席测试专家，1999年进入华为公司，先后参与和主持过多项产品测试、测试流程改造、测试工程师职责定义等工作。工作覆盖测试策略、测试设计、测试评估和过程管理等软件测试工程的各个方面，在自动化、可靠性验证、可服务性验证、可测试性设计等领域上都有丰富的经验。2007年以后主管软件公司的测试技术架构设计、实现、应用，通过帮助产品持续积累和提升测试技术能力，实现研发效率和质量的提升。

内容提要

本书分为 19 章，主要内容为开发者测试，测试目标、方式和角色，测试术语，开发人员眼中的可测试性，契约式编程，可测试性的驱动者，单元测试，基于规格说明的测试，依赖关系，数据驱动和组合测试，准单元测试，测试替身，模拟框架，测试驱动开发——经典风格，测试驱动开发——Mockist 风格，使用测试代码，超越单元测试，测试思路等开发人员和测试人员必知必会的知识。

如果你是一个希望所写的代码质量更高、缺陷更少的开发人员，那么这本书很适合你。本书介绍了如何用可测试性提升软件质量，在各种开发模式中，可测试性都是软件的主要质量属性之一。阅读本书，你可以成为更好的开发人员，学习到更多的软件测试知识，不必再苦于没有时间做测试、也无法从同事或团队那里获得相关的支持。

译者序

一年前，也是端午节，很巧合，本书的一个译者为另一个译者的新书《软件测试价值提升之路》写序。一年之后，还是端午节，我们一起为不一样风格的软件测试译著《程序开发人员测试指南：构建高质量的软件》（后简称《程序开发人员测试指南》）写序，依旧充满诗意，享受着成功的喜悦，并郑重推荐本书给所有的软件开发者和测试人员。

回想当初，朱少民从人民邮电出版社接下这本书的翻译任务，邀请3位测试界朋友杨晓慧、欧阳辰、曾天乐组成翻译团队，开启了本书的翻译旅程。我们虽是测试老兵，但从接下任务之后，还是感到较大的压力，始终怀有一颗谦逊的心来做这项工作。一方面它是全球第一本以“开发人员测试”命名的专业图书，希望中译本能够得到大家的长久喜爱。另一方面，翻译不是一件很容易的事，比写书还难。写书可以按照自己的想法、意愿去写，翻译则是“戴着脚链在跳舞”，经常需要仔细地揣摩原作者的写作思路或所表达的具体含义。就拿我们所整理的一个术语表为例，其中收集了218个术语，几经修改，先后出了好几个版本。多数术语很容易统一起来，有些术语（如quirk、circular argument、code bloat等）会有争议，不容易达成一致，查证各种资料，最终才达成一致。甚至面对个别术语（如Mockist、Test Double、Stub），我们觉得多数开发人员更容易理解英文术语，没必要翻译成中文，但为了图书规范，尽量把各种术语翻译成中文，只是在其第一次出现时，注明原英文术语。

十年前或更早，许多优秀的软件公司都有独立的测试团队，更强调测试的独立性、客观性，开发的质量很大程度上依赖于独立测试的质量。那时，微软公司拥有大约一万名专业的测试人员（Software Development Engineer for Test，SDET），成为全球为数不多的测试大军团，因此，那个时代微软是许多公司的测试标杆，大家学习微软如何做测试，参加由微软资深人士开设的培训讲座。那个时候，我们曾经亲身经历的项目，开发人员所做的测试很少，更多的测试是由专业的测试团队完成。即使大家都认为“单元测试应该是由开发人员来做”，单元测试的覆盖率也非常低，其效果依旧不理想，可圈可点的项目很少。 不少项目推行过开发者测试，成功者寥寥，甚至个别项目想摆脱独立的测试团队就直接上线，结果也是铩羽而归。

然而，最近几年敏捷开发席卷而来，到处开花结果，所有开发者越来越关注质量，开始做越来越多的测试。微软测试军团从2010年开始瓦解，到2014年烟消云散，绝大多数的SDET快速融入开发团队之中，成为开发工程师的一员，而剩余的SDET要么改行，去干运维、技术支持等工作，要么辞职，去其他公司继续专职的测试工作。今天，Google、Facebook等公司成为新的标杆，人们开始推崇非常简单的工程师文化、推崇开发与测试的融合。在这样的环境下，开发人员不仅需要完成代码，还需要全力保证代码的质量，要求开发人员做足够的测试。但是，开发人员不是天生下来就会做测试，测试能力还比较弱，甚至有些开发人员在今天敏捷开发的环境下，依旧排斥测试，开发者测试在国内不容乐观，这里面有客观因素，也有主观因素，但无论如何需要改变。

如果按照过去那种瀑布模型做测试，先开发，后测试，开发人员会遇到心理上、思维上的障碍。而从理论上看，测试驱动开发（TDD）则彻底解决了这个问题，因为测试在前，开发在后。在开发前，测试的思维不会受到实现思维的影响；实现的代码还没有，自然也不存在心理上的障碍。理想很丰满，现实很骨感，TDD 的应用还是凤毛麟角，因为 TDD 的具体实施会面对各种困难，如给开发人员带来额外的工作量、如何摆脱过去写代码的习惯等。例如，在 TDD 实施时，我们经常能够听到开发人员说，“再给我加一倍的时间”。由于增加较大的工作量，在进度压力下就很难实施 TDD，或者说，许多团队不知如何实施 TDD，如何更高效地完成软件的开发且提高质量，不仅让客户满意，而且也带来生产力。如果管理层有坚定的决心，并敢于在组织、流程、策略上做出相应的改变，TDD 可以带来“质量”和“生产力”的双收益，不靠事后检验，可以大大降低质量劣化带来的成本。

开发者测试不仅仅局限在 TDD、单元测试和集成测试，组件之间的交互性测试、调用系统进行更高层次的测试也会出现在开发者测试中，开发者测试也不仅仅使用测试技术，可测试性、依赖关系、复用和契约式编程、防御式编程等以构建高质量的代码为目的的技术也与开发者测试息息相关。测试不能真正保证质量，软件质量是在设计、编程过程中慢慢形成的。从这个角度看，开发者测试更为重要，在开始构造功能时就要思考怎么测试它，相对于找到代码错误，他们更关注于如何避免错误。在成功的团队中，团队的每个成员都拥有这样的理念：构建高质量的软件（正是本书的副标题）。他们与客户、交付团队协作，试图理解什么才能帮助客户获得成功，如何找到最有效、最简单的解决方案。这本书正是从这个角度展开讨论，以帮助开发人员正确地理解和掌握开发者测试，解决开发人员从准单元测试开始，到测试替身、模拟框架、不同的 TDD 模式等测试中遇到的各种障碍。本书还有其他一些特点，下面就让我们逐一介绍。

首先，这不是“测试专家写的开发者测试”，而是“开发专家写的开发者测试”。书中并没有花太多篇幅介绍测试的概念、测试设计技术、单元测试工具（这些可能是我们之前推行开发者测试的重点），而是把重心放在了可测试性、影响测试的编码风格、实现开发者测试的方式、测试环境和条件的构造、开发者测试在全部测试活动中的位置和作用等方面（这些是真正影响开发者测试效率的问题）。因此，这本书对于开发人员具有很好的实用价值。

其次，这本书不是一座“大山”，而是若干“甜点”组成，除了前 3 章介绍测试的基本概念和术语，其他各章相对独立，一章基本是一个主题，阐述开发者测试所遇到的问题、解决方法、注意事项等。即使隔了很长时间我们才读另一章，或者跳过没有兴趣的个别章节，也完全不影响我们阅读的体验或收获。“甜点”还隐藏在每一章中——每章穿插着一些“小窍门”“经验之谈”或“注意事项”等，点拨读者，读者获得启发或警醒。

再者，本书实例丰富，循序渐进，例如 14.1 节就用了“一个简单的搜索引擎”的 8 个实例，一步一步地介绍经典风格的 TDD 是如何实施的。本书的内容安排得当，有主有次，主次分明，例如许多测试书籍都有“基于需求的测试方法”的详细介绍，本书则用较少篇幅快速带过。而对于重点内容，如可测试性、Mock 技术和 TDD，分别用了两章阐述其不同的方面。在 Mock 技术中，逐一介绍了如何应用不同的 Mock 对象——桩对象、伪对象、模拟对象、监听器、哑对象，更体现其专业水准。

最后，这本书中的开发者测试不是“孤立”的，而是“在上下文中”的（上下文是软件工程中最重要的概念之一）。书中将开发人员与测试人员放在一个场景中，让读者更好地理解问题发生的前因后果。将问题放在代码中（很多还是来自于实际产品的代码），方便读者映射到自己的产品和

代码中，并设想解决问题的方法是否适合自己，留给读者更大的思考空间。将测试活动放在实际的研发项目中，单元测试和模块集成、独立模块测试的区别并不那么明显，书中也不回避这些现实问题，而是帮助读者看到这些测试的真正过程，使读者可以根据项目的具体情况做出策略选择。

我们非常高兴参与本书的翻译，翻译过程虽然很辛苦，但也是快乐的学习过程，能解开我们对开发者测试的一些疑惑。唯一的遗憾是，相见恨晚矣！如果早几年读到此书，在之前的工作中，很多事情会有更好的方式做，比如 TDD、单元测试，比如代码质量检测、敏捷一体化团队……相信本书对开发者们会有更大的帮助，会逐步提升开发者的测试能力。开发者测试做好了，在未来交付项目代码时，会感到很轻松，会更加充满自信。

译者

于 2018 年

杰夫·朗格尔（Jeff Langr）的序

10年前，我曾在当地一个小创业公司就职，经过几个月的开发工作，我成了这个开发团队的经理和技术主管（technical lead）。当时团队写的代码几乎充斥着各种典型的费解的逻辑和难以解决的缺陷。作为团队的领头人，我开始推广测试驱动开发（test-driven development，TDD）的理念来尝试提升我们代码的质量。最终，大部分开发者都愿意听从这个建议，也有几个开发者最终欣然接受了TDD。

然而，有一位开发人员仅过了两天就一声不吭地退出了团队。有人告诉我，他说过类似“我永远不会去写测试，那不是一个开发人员要做的事情”的话。我一开始担心是不是我对这件事操之过急了（虽然我从来没有执意要求什么，只是尝试着让他们了解TDD），但是，当我看到了他写的如同噩梦般的代码后，我就不再觉得愧疚了。

之后有一位测试人员向我抱怨：团队中有个开发人员——拥有多年经验的顾问——一直在提交充满缺陷的代码给我们的测试团队，并称“写代码是我的工作，找出代码问题是他们的工作”。即使再多的商讨也无法说服这位先生能够测试他自己的代码。

后来，还是在这套代码上，我发布了一个带有严重缺陷的版本，尽管我尽了各种努力去确保所有单元是被很好测试过的，这个缺陷还是没有被测试团队发现。对服务器代码的一点点修改和对一个布尔变量的轻易改动导致了客户端（一个高安全聊天应用）不再对新消息进行铃声提示。当时，如果我们做了全面的端到端测试（end-to-end testing）就能发现这个缺陷。

开发者测试只是一种手段，但不是为了让你的老板高兴而已。如果它仅仅是这样的话，那我宁愿不去创建开发者测试。无论是将软件发布给最终用户还是发布给测试团队，测试都是建立自信的一种手段。

值得庆幸的是，10年过去了，大部分的开发者都能意识到“测试自己的代码确实是开发工作的一部分”。在面试中，很少有人会遇到不讨论任何有关开发者测试的情况。我们都期望成为一名非常专业的软件开发人员，而写出高质量的代码是不可或缺的。10年后的今天，如果应聘者认为他们不需要对自己的代码做测试，我就会打消聘用他们的念头。

然而，开发者测试不仅仅是TDD，也不再是简单地写一些集成测试。为了交付正确且高质量的软件，一个真正的开发人员必须要接受很多方面的测试。尽管你可以找到介绍TDD或者组合测试的好书，但《程序开发人员测试指南：构建高质量的软件》（以下简称《开发人员测试》）则涵盖了这两个方面的必要内容。为了阐明各式各样的开发者测试，作者（亚历山大）纵览了测试的世界，权衡各种测试方法之间的优势，为你提供了成功路上不可或缺的诀窍。

在本书中，亚历山大首先展示了那些需要关注的测试。他介绍了一些被人们忽视但很实用的观念，例如契约式编程（Programming By Contract）。他教会我们如何设计出容易被测试的代码。他强调了两个我喜欢的目标：

- 构建具有高可读性的、基于规格说明的测试，依旧保持文档的价值；

- 消灭高质量系统的最大敌人之一——各种坏味道的复制。

他通过实用的、平衡的方法做好 TDD，并呈现了传统的 TDD 和 Mockist TDD 的应用技巧，从而向我们全面介绍了单元测试的内容。

等等，我还要再说说第 18 章：“超越单元测试”。开发者测试是一个模糊不清的世界，这一章对单元测试之外的内容讨论之广超越了我们的期待。设计稳定、有效、可持续发展的测试是一个挑战。本书提供了丰富的、独特的智慧来处理这个问题，一定不会让你失望。

我享受经历本书完成的过程，随着亚历山大完成内容丰富的代码部分，我觉得书变得越来越棒了。想出让读者参与到书中并且感受不到挫折的例子是一件很难的事，但是亚历山大用他写的例子成功地做到了这点。我想你会喜欢这本书，并且庆幸在你事业的成长路上拥有关键的测试技巧的基础。

丽莎·克里斯平（Lisa Crispin）的序

本书的副标题昭示了一切。我们知道不能通过编程“结束”之后的测试来测试所构建的软件质量，软件质量必须被融入到编程过程中。因此，整个交付产品的团队，包括开发者，在开始构造功能时就要思考怎么测试它。在成功的团队中，团队的每个成员都拥有敏捷测试的思维。他们与客户、交付团队协作，试图理解什么才能帮助客户获得成功。相对于找到代码错误，他们更关注于如何避免错误。他们能够找到提供合适价值的最简单的方案。

在我的经验中，即使团队里已经有了经验丰富的专业测试工程师，也需要让开发人员懂得测试。开发人员要能够与设计师、产品专家、测试人员及其他的团队成员交流，每个功能都应该这么做；需要设计可测试的代码；要懂得如何从单元直到系统都能用测试来指导编码；要像懂得生产代码那样懂得如何设计测试代码，甚至更要了解，因为测试代码是我们活的文档，是我们的安全帽；要懂得怎样去探索他们开发的每一个功能，去了解这些功能是否带给客户相应的价值。

我遇到过许许多多这样的团队：开发者被雇用去写产品代码并不断被施加压力追赶一个又一个的最后期限。他们的经理认为将时间花在测试上是一种浪费。若这样的团队中有任何测试工程师，他们会被认为是团队中价值不高的贡献者，并且他们发现的 bug 只是记录在缺陷跟踪系统中，但往往被忽略。这些团队开发出大量的没人能看懂的代码，这些代码很难在不破坏其他功能的情况下进行修改。随着时间的推进，他们通常在技术债的重压之下慢慢停止下来。

多年来，我很幸运地能够和几位真正懂得测试的开发者们一起工作。他们积极与领域专家、设计师、测试工程师、大数据专家等人沟通，从而对于各种特性该如何表现能够达成共识。他们与测试工程师相处融洽，就算在代码提交到测试环境之前也会愉快地对自己的代码进行测试。这些团队是快乐的，能够开发可靠的、有价值的功能给用户，他们能够迅速改变方向来适应新的业务优先级。

测试是一个广阔的领域，而繁忙的我们应该从何下手？这本书通过一种让你迅速上手的方式传递了测试的重要原则和实践，帮助你及团队向用户交付他们真正需要的东西。你将会学到测试的语言，然后和测试工程师、客户及其他团队成员高效合作。更重要的是（至少对我来说），你将会更享受你的工作并且对你做的产品感到自豪。

致谢

写书是一项团队工作。虽然作者是写下这些文字并与它们相处时间最多的人，但其实很多人都做出了贡献，这本书也不例外。首先我要感谢拥有独特视角的软件领域专家、我的好友乔金（Joakim Tengstrand）。在这本书的开始阶段到书的完成，他不断地提供了许多有深刻见解的反馈。

另一个我要特别感谢的人是 Stephen Vance。他很热情，从技术角度来帮我对本书进行二次校对。他不仅提供了很有价值的反馈，还找到了很多（即使不是全部）我想要偷懒的地方。此外，他给了我一些更多选择和观点来帮我拓宽这本书的内容。

事实上，如果没有丽莎·克里斯平的帮助，这本书将不会以现在的形式出现。她帮我出版了这本书，整个过程中，在我需要帮助的任何时刻她都在支持我。我很荣幸她能为本书写序。谈及此，同样深深感谢为本书写另一序言的杰夫·朗格尔，他还激发了我重写已拖延了很久但重要的章节。我无法表达对 Mike Cohn 有多感激，虽然我从未有幸与他谋面，他却将我的书收入他的系列，这对我来说意义真的重大。

在这本书的出版方面，我真的要感谢艾迪生韦斯利出版社（Addison-Wesley）的克里斯（Chris Guzikowski）。他在整个过程中表现得非常专业，最重要的是他突破各种所限以支持本书的出版。我不知道我写了多少开头类似于“非常感谢您的耐心，但在我提交手稿之前我还有一件事要做……”的 E-mail。在本书后期定稿过程中，我很荣幸能够和乐于助人的非常专业的人士一起工作，是他们让这段旅程变得有趣、富于挑战。在此非常感谢 Chris Zahn、Lisa McCoy、Julie Nahil 和 Rachel Paul。

本书的审稿者——Mikael Brodd、Max Wenzin、Mats Henricson——为本书的第一遍技术校对做了巨大贡献。

特别感谢 Carlos Ble 带我参加了一个 TDD 活动，由此产生了一个和本书中有关 TDD 章节很不一样的解决方案。这件事引发了我的进一步思考，最终促进我重写了整个章节。Ben Kelly 很好地帮助我弄清楚书中一些测试术语的细节，使我不得不区分开发人员和测试人员某些不同的工作。Dan North 帮助我弄明白 BDD 和 ATDD。Frank Appel 在单元测试及其相关资料上帮助了我。他所做的评论不仅详尽，而且有根有据，有时候让我不得不停下来进行深刻的思考。对此，我感激不尽。Alex Moore-Niemi 提供了关于类型的补充内容，拓宽了本书的范围，而我对此的理解是很肤浅的。

我同样要感谢 Al Bogdonas，我的初稿校对员兼编辑对这个项目的奉献。

此外，我同样要感谢那些帮助过我或给我灵感的人：Per Lundholm、Kristoffer Skjutare、Bobby Singh Sanghera、Gojko Adzic、Peter Franzen。

最后，我要加入那些感谢妻子和家庭的作家队伍。写书需要热情、奉献、努力，最重要的是，需要牺牲家庭时间。Teresia，谢谢你的耐心和支持。

作者介绍

20世纪90年代初，10岁的亚历山大（Alexander Tarlinder ）写下了他的第一个程序——Commodore 64计算机上的简单的基于文字的角色扮演游戏。程序里面有很多的GOTO语句和大量重复冗余的代码。但对于当时的他来说，这仍然是他能够构想出的最棒的软件，也成为了他之后职业道路的起点。

25年过后，亚历山大仍然在写代码，内心还是把自己视为一位开发人员。15年的职业生涯中，他做过开发者、系统架构师、项目经理、测试工程师和敏捷开发专家及教练。在这些工作中，他保持可持续的开发节奏、匠人精神并关注质量，最终大约在2005年深受测试的影响。从某种程度上来说，这是不可避免的，因为他的项目很多代码都涉及了资金（比如为银行或者游戏行业做的产品），他总认为在将产品交给别人之前他可以做更多的事去保证代码的质量。

目前，亚历山大在寻找能让他对项目的实施过程有更大影响的角色。他将开发项目与培训、训练相结合，并在研讨会和一些当地用户见面会中分享他对开发者测试及质量保证的技术和非技术方面的理解。

前言

4 年前开始写这本书的时候，我已经明确了本书写什么内容，面向的是哪些读者。4 年是相当长的时间，期间我已经修正了我的某些想法和假设，这既是出于对本领域其他工作的呼应，也是由于对这个课题更深入的思考。近年来这个课题最明显的变化就是，关于它的争论变少了。最近出版的好几本书都与本书持有类似的观点，这让我感到宽慰，我认为这些观点都是朝着正确方向发展。

为什么写这本书

我写这本书，是因为我在 10 年前就想读到这样的一本书！10 年是一个相当长的时间，但是，无论你信或不信，今天我仍然需要这本书——尽管是出于不同的原因。

大约 10 年前，我开始了软件质量的研究旅程。那时我并不理解软件质量，我只知道我和我的同事们写的代码中充斥着缺陷，这些缺陷让我们沮丧，让客户不满。我相信，测试人员对我们的软件所做的手工的常规检查，并不能显著提高产品的质量——时间也证明了，我是对的。于是，我开始读所有我能找到的关于软件匠艺（craftsmanship）和软件测试的书，从中我有两个主要的发现。

首先，让我吃惊的是，这些书常常把课题完全割裂开来讨论！一方面，关于编写软件的书中，极少有关验证的内容。也许会提到一两种测试方法，但是却没有介绍相关的思想和概念框架，而这些内容对于“在不同情境下怎样使开发和测试开展有效协作”是必要的信息。至少，这些是我的印象。另一方面，与测试相关的书籍，常常从测试流程入手。类似地，测试驱动开发（TDD）的书，只聚焦于 TDD 如何应用于开发。不仅是书籍，博客和其他在线资料也是一样。

其次，写出可测试的代码比我们最初想象的要难，更别提改造遗留代码使之变得可测试了。为了真正了解软件测试，我花了大量精力，深入思考，学习了软件开发技术、重构、遗留代码、TDD 以及单元测试。

基于这些研究和我积累的经验，我将本书的目标设定为以下几项。

- 让软件测试的基础知识更容易被开发人员理解，这样，他们就能够为即将发布的代码选择最合适的验证类型和级别。在我的经验中，许多开发者不会去读测试类的书或博客，尽管他们总是在问自己：测试是否足够？需要写多少测试用例？应该测试什么？我希望他们读完这本书会使这些问题变得十分容易。
- 演示测试的思维模式和技术是如何帮助软件开发人员的日常工作的，并展示如何使之成为开发者的第二特质。
- 为写出可测试的代码构建简单且足够好的知识和技术体系。我意识到这样的工作是非常难以被理解的，尤其是我还要保持这个体系的简洁，但是我仍希望建立一个足够完整的

体系，将读者从成千上万的相关书籍和在线资料中解救出来。如果你想要的话，我会提供一份“领域知识地图”。

这些就是写这本书的目的，这些想法 10 年前就有了，但是今天呢？难道这世界没有变化吗？工业界难道没有进步吗？真正令人感兴趣之处在于这本书和 10 年前一样适用。原因之一是本书的内容相对而言与技术无关，尽管仍有大量代码采用过程式编程，但面向对象编程已被广泛采用，也有一些代码采用的是函数式编程。另一个原因是本书所涵盖的这个领域，并没有像其他领域那样，取得令人激动的进步。的确，时至今日，许多开发者已经掌握了基础的测试能力，流行的架构和库中也鲜有完全不考虑可测试性的。然而我仍然认为，找到一个会用同构的 JavaScript 写前端、用在云平台上的 NoSQL 数据库做后台的开发者非常容易，而找到一个会做单元测试和重构，特别是在来自主管、同伴的进度压力下仍能坚持实践开发者测试的开发人员则难得多。

作为一个专注于软件开发、培训和指导的顾问，我有幸参与了许多软件开发团队的工作，也观察过其他团队的工作。基于这些经验，我发现无论是开发团队还是开发人员，在质量保证上都经历了非常类似的学习曲线。本书的内容组织也符合这条学习曲线，而我则尽最大努力帮助读者克服困难，尽快取得进步。

目标读者

本书不适用于初学者，本书确实介绍了许多基本概念和基础技术，但是我们假设读者已经知道怎样在自己的开发环境下完成工作、构建系统，对于持续集成及其相关工具，如静态检查、代码覆盖也不陌生。本书最适合有 3 年以上专职软件开发经验的开发者，这样，读者在阅读的时候会发现其中的一些对话很熟悉，也能够将代码示例与这些对话联系起来，这些代码示例都是来自于真实产品，而非凭想象编写。

最后希望本书的读者动手去做一做，尽管我的理想是让这些信息都唾手可得，但我仍将知识整合的工作留给读者自己完成，毕竟这不是一本烹饪手册。

关于案例

本书包含大量源代码，然而，我的目的并不是写一本编程的书，本书的重点是原理和实践。因此，这些从真实产品中提取的代码示例，采用的是各不相同的编程语言。尽管如此，我仍尽量采用各种语言通用的结构和语法习惯，因为我也不希望读者被某种特定语言或架构的某种特殊细节难住。也就是说，我尽量使这些案例有足够的普遍性，以便有一定经验的开发者都能够读懂。但是有时候这样做也很困难，一些特定的结构只有某种架构和语言能很好地实现。另一些时候，我无法判断哪种写法更普遍，这种情况下，可以在附录中找到其替代实现方案。本书中的源代码和其他相关代码，可以在本书的配套网站下载，网站地址为 http://developertesting.rocks。

如何阅读本书

本书是为一群特定的读者所写：他们是背负进度压力的开发者，他们需要为特定的问题找到

实用信息，他们不希望为此翻阅大量的文章、博客和书籍。因此，这里的潜台词就是，每个章节只需花不到一个小时就能读完，读者最好可以在工作间隙读完一章。相应地，每章节的内容各不相关，读者可以只读其中某些章节。尽管如此，仍建议先读完前 4 章，因为这几章是其他章节的基础。各章简介如下。

第 1 章：开发者测试。开发人员在编码和验证环节中，事实上已经从事了很多测试活动，不管开发人员称不称之为测试。本章还给出了开发者测试的定义。

第 2 章：测试目标、方式和角色。介绍了不同的测试方法。解释了批判型测试与支持型测试的不同。本章第二部分是关于传统测试、敏捷测试，以及各种版本的行为驱动开发。开发者测试作为一种支持型软件测试的活动，更适合敏捷开发模式。

第 3 章：测试术语。本章可以视为一个大的词汇表。本章解释了测试领域使用的术语，给出了一些常用模型，例如测试级别和测试类型矩阵、敏捷测试四象限。所有的术语都是以开发者的视角来解释的，对于有歧义或有不同解释的术语则给出了其各种解释。

第 4 章：开发者眼中的可测试性。为什么开发者需要关注可测试性？本章给出了具备良好可测试性的软件实例，并明确了可测试性带来的好处。可测试性质量属性被分解为可观察性、可控制性、小规模，并对它们做了进一步说明。

第 5 章：契约式编程。本章介绍了在软件开发中采用契约式编程的好处，无论在编码前是否先写出了用例，这种好处都是存在的。这种技术在被调用代码和调用代码之间约定了正式的责任和义务，这对于写出可测试性好的软件是很重要的。本章还介绍了断言的概念，在所有的测试框架中，断言都是核心概念。

第 6 章：可测试性的驱动者。一些代码结构对可测试性就有很大的影响，因此，能够发现并辨别出这些结构就显得很重要。本章介绍了显性和隐性输入/输出、状态、时序耦合和域—值比率。

第 7 章：单元测试。本章首先介绍了 xUnit 测试框架的基础知识。然后切入到单元测试更深入的话题，例如构建用例并为其命名、正确使用断言、基于约束的断言，以及单元测试的其他技术。

第 8 章：基于规格说明的测试。这是测试的常规内容。本章从开发者的视角解释了基本的测试技术。了解这些，才能够回答“我需要写多少测试用例？”。

第 9 章：依赖关系。类、组件、层之间的依赖关系，都会以不同的方式影响到可测试性。本章讲解了每种依赖关系及其影响，以及如何处理相应的问题。

第 10 章：数据驱动和组合测试。本章介绍了如何利用参数化用例的方法和思想，处理那些看起来类似的测试用例。本章还介绍了比参数化用例更进一步的技术——生成式测试。最后，本章还介绍了一些测试工程师用来解决组合测试中用例数量暴增问题的相关技术。

第 11 章：准单元测试。本书所指的单元测试，不包含那些运行起来几乎和单元测试一样快，但并不叫“单元测试”的测试方法。为了显性地区别两者，将后者称为“快速媒介测试”。这类测试最典型的是构建轻量级的服务器，例如 servlet（小服务程序）容器、邮件服务器、内存数据库。本章就是介绍这些测试方法。

第 12 章：测试替身。本章介绍典型的测试替身，例如桩对象（stub）、模拟对象（mock）、伪对象（fake）、哑对象（dummy）。本章的重点是理解测试替身的概念，无需理解任何一种 mocking 或其他技术的框架。本章还介绍了基于状态的测试与基于交互的测试之间的区别。

第 13 章：模拟框架。本章具有很好的可操作性，文中应用模拟对象框架 Moq、Mockito，以及

测试替身工具 Spock，为不同的场景、满足不同的需要而创建测试替身——主要是桩对象和模拟对象。本章还介绍了应用模拟框架时的陷阱与反模式。

第 14 章：测试驱动开发——经典风格。本章基于一个完整的案例介绍了经典的测试驱动开发（TDD）。这个案例中展示了该技术的各种细节，如用例的编写顺序、测试执行的策略等。

第 15 章：测试驱动开发——Mockist 风格。TDD 不止一种途径，本章就介绍了另一种 TDD 风格。与在一个具体的类或者组件的实现过程中应用 TDD 相比，测试驱动系统设计更为重要，本章用一个实例证明了这一点。

第 16 章：复制。本章解释了为什么代码复制不利于可测试性，但是有时候为了独立性和开发效率的考量，复制又不可避免。本章介绍了两类复制形式：机械复制和知识复制。

第 17 章：使用测试代码。本章包含了关于写测试代码前需要做的准备，以及何时删除测试代码的一些建议。

第 18 章：超越单元测试。单元测试是开发者测试的基础，但这仅仅是测试工作的一小部分。时至今日，软件系统日趋复杂，因而需要不同抽象层次、不同粒度的测试，集成测试、系统测试，以及端到端测试应运而生。本章通过一系列案例介绍这些测试活动及其特征。

第 19 章：测试思路与启发式。这是最后一章，在附录之前，将本书中的测试思想和启发式进行汇总。