

程序员修炼三部曲 第二部

The Pragmatic Starter Kit-Volume II

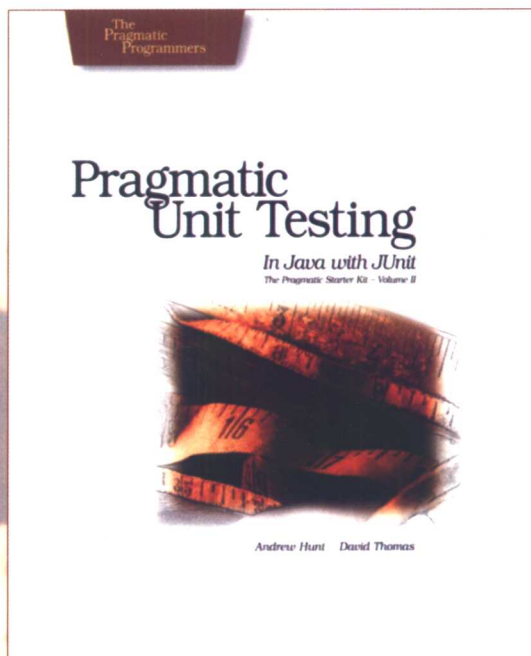
单元测试之道

Pragmatic Unit Testing

Java 版

—— 使用 JUnit

In Java with JUnit



[美] Andrew Hunt David Thomas
陈伟柱 陶文 译 著



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

程序员修炼三部曲 第二部
The Pragmatic Starter Kit-Volume II

单元测试之道 Java 版

—— 使用 JUnit ——

Pragmatic Unit Testing In Java with JUnit

[美] Andrew Hunt 著
David Thomas

陈伟柱 陶文 译

电子工业出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 简 介

《程序员修炼三部曲》是一套由四本小册子组成的丛书，旨在帮助解决程序员在日常工作中遇到的一些具体问题和需要，覆盖了对于现代软件开发非常重要的基础性知识。这套丛书不仅展现了注重实效的实际技巧、工具使用，也贯穿了作者们在其名作《程序员修炼之道：从小工到专家》中所坚持的开发哲学。而所有这些都帮助开发人员和开发团队进行正常开发，并带来高开发效率的利器。

《单元测试之道 Java 版——使用 JUnit》是本丛书的第二本（本书还有 C# 版本《单元测试之道 C# 版——使用 NUnit》），阐述使用自由公开的 JUnit 程序库以 Java 语言进行单元测试，其内容也广泛适用于其他语言和框架程序库。本书主要内容包括：如何更高效地撰写 bug 更少的代码；如何发现 bug 的藏身之处以及如何清除 bug；如何测试代码片断而不用牵连整个项目；如何利用 JUnit 简化测试代码；如何在团队中高效地进行测试，等等。

本书主要适用于具有一定编码和设计经验，但是对单元测试并不是很有经验的 Java 程序员。

Pragmatic Unit Testing in Java with JUnit, Andy Hunt, Dave Thomas, 0-9745140-2

本书中文简体版专有出版权由 The Pragmatic Programmers, LLC 授予电子工业出版社，未经许可，不得以任何方式复制或抄袭本书的任何部分。

版权贸易合同登记号：图字：01-2004-2485

图书在版编目 (CIP) 数据

单元测试之道 Java 版：使用 JUnit / (美) 托马斯 (Thomas, D.) 等著；陈伟柱，陶文译。
—北京：电子工业出版社，2005.1 (程序员修炼三部曲)

书名原文：Pragmatic Unit Testing in Java with JUnit

ISBN 7-121-00665-0

I. 单... II. ①托...②陈...③陶... III. ①软件—测试②JAVA 语言—程序设计 IV. TP311.5

中国版本图书馆 CIP 数据核字 (2004) 第 131172 号

责任编辑：周筠 陈元玉

责任校对：张兴田

印刷：北京天竺颖华印刷厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

经 销：各地新华书店

开 本：787×980 1/16 印张：11.25 字数：200 千字

印 次：2005 年 1 月第 1 次印刷

定 价：25.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：(010) 68279077。质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

读者对《实务单元测试》的评价

“本书从什么是单元测试，为什么要使用单元测试这样的问题入手，以讨论的形式作了很好的介绍。我喜欢书中遍布的轶闻趣事，它们能很好地说服你为什么要使用单元测试……。我还非常喜欢作者使用的类比，它使得代码能够融入到现实生活的场景之中。”

Sharee L. Johnson, 应用程序开发组的项目组长

“我希望在我早先开始实践极限编程，做测试先行的开发的时候能有这么一本书。”

Al Koscielny, 软件开发者

“我对测试并不陌生，但是我在测试的许多方面都走过弯路。在把一个新手带入单元测试世界这方面，我认为这本书做得很好；同时，它还有相当多的高级内容，用于帮助我们这些已经涉及测试并曾遇到过麻烦的人。”

Andrew Thompson, 咨询师, *Greenbrier & Russel*

“当我所在的项目需要把单元测试做得更好时（似乎总是如此），我愿意用这本书来作为主要的参考读物，并用它来启发团队。”

Bobby Woolf, I/T 咨询专家 *IBM Websphere* 软件服务部门

“我是一个相信单元测试的人，而且我希望与我一道工作的组员能够忠实地使用书中推荐的技术。我认为书中有大量优秀的实践性信息，而它们正是任何软件工程师应该融入日常工作的金玉良言。”

James J. O'Connor III, 首席系统设计师

译序

随着敏捷开发方法的风行，单元测试的思想也在遍地开花。越来越多的开发人员和公司开始学习和接受这种思想，并且在日常的开发工作中进行一些尝试。但是在尝试中，人们却遇到了不少问题和障碍。比如，如何将单元测试的思想传播给小组中的每一位成员，并让他们心悦诚服地接受；如何坚持这种边写边测的做法并最终养成一种习惯；如何进行高效的单元测试，而不是把时间无意义地浪费在对 getter、setter 的测试之上。

在 UT 中，Dave 和 Andy 为我们揭开了这层层面纱，游刃有余地为我们展现单元测试的世界。尤其难能可贵的是，作者并没有采用列举教条的方式来说教，而是以一种与朋友对话谈心的方式来说服你，进而从良师益友的角度来教导你，最后再很有导师风范地给你指明一些设计层次上的问题；最终让你充满自信，做到对代码胸有丘壑。我相信，本书将不仅仅是学习单元测试的 starter kit，还将会成为你工作中常伴身边的良师益友。

对于每个在单元测试上探索和工作的读者，本书希望能够帮助你们少跌几次跤，少碰几次壁，尽快踏上单元测试的康庄大道。这也是作者和我们的愿望。

本书由我和陶文两人合译。陶文的文笔技术都很不错，与他合作是一段美妙的经历。本书编辑方舟、陈元玉付出了辛勤的工作，为中文版增色不少。力求体现原书的思想精髓，使作者的睿智之言不因语言转化而略失光彩，这是我们最大的心愿。同时，也希望读者阅读过程中能够提出宝贵意见，帮助我们不断提高和改进译本。

在此，我要感谢周筠老师、陈英老师、方舟、陈元玉和我的爱人！

关于程序员修炼三部曲

在本系列的第一本书《The Pragmatic Programmer (程序员修炼之道：从小工到专家)》中，对现代软件开发的许多实质性话题作了概要性的介绍，并且获得了广泛的好评。自从该书 1999 年第一次印刷出版之后，就有许多读者向我们询问是否能再出一些后续书籍，或者该书的续篇。我们将会考虑此事；但首先，我们觉得有必要先提供几本基础方面的系列书籍。

在《程序员修炼之道：从小工到专家》出版之后的这些年来，我们发现：那些刚开始从事软件开发的读者，非常希望能够在软件开发的基本细节方面得到适当的指引，这样有助于他们早点养成良好的开发习惯；另一方面，那些经验丰富的读者，虽然已经能够完全理解书中的大多数内容，但是他们仍然希望在说服和指导开发小组中其它组员时，能得到一些帮助。现在，我们可以很高兴地告诉这些读者，我们已经有了几本能够给他们带来真正帮助的书籍了。

《Pragmatic Starter Kit (程序员修炼三部曲)》是一套由三本小册子组成的丛书，覆盖了对于现代软件开发非常重要的基础性知识。就内容而言，这三本小册子展现了实际操作、工具使用和开发哲学，而所有这些都是帮助开发小组正常开发、不断进步、带来高开发效率的利器。了解并掌握了这些知识之后，你和你的小组成员将能够很容易地养成好的开发习惯，并且对于你们所开发的项目而言，等于在外面织上了一层安全的网，它能让你感到安全和舒适。

第一本小册子《Pragmatic Version Control (版本控制之道)》讲述如何使用版本控制给整个项目打基础。打个比方来说，一个没有使用版本控制的项目，就像一个没有 **UNDO** 按钮的文字处理器：你输入的字符越多，错误所造成的伤害也就越大，因为没有撤销选择的机制。《Pragmatic Version Control (版本控制之道)》这本书将会告诉你：如何有效地使用版本控制系统，从中获取最大的好处和安全性，而不必拘泥于极端死板或者冗长可怕的过程。

《Pragmatic Unit Testing (单元测试之道)》是这个系列的第二本。单元测试是一项很重要的技术,它能够在程序员编写代码的同时,提供及时真实的反馈。遗憾的是,许多开发者对单元测试都不十分理解,没有意识到它可以令开发者的工作变得更加轻松。

第三本书《Pragmatic Automation (项目自动化之道)》¹囊括了在代码构建、测试和发布过程的自动化方面需要的一些非常重要的实践和技术。很少有项目会因为时间过多而失败,往往都是由于时间不足所造成;因此,《Pragmatic Automation (项目自动化之道)》主要是告诉你如何让计算机来完成更多的重复性工作,从而程序员能更加专注于比较有趣也比较困难的工作。

就书的风格而言,这几本书和我们的第一本书一样都是很通俗的。主要是帮助解决和满足程序员在日常工作中遇到的一些具体问题和需要。然而,这几本书并不是那种只给出一般问题、泛泛而谈的肤浅之作,它会让你充分理解这些知识,这样即使面对新出现的问题(本书可能没有明确提到的问题),你也能够根据这些知识找出自己的解决方法。

需要本书和其它书籍的更新信息,以及一些针对开发者和项目经理的相关资源,请访问我们的网站:

<http://www.pragmaticprogrammer.com>

感谢你的阅读,请记住要让阅读充满乐趣!

¹ 2004年7月出版(编者注:中译本《项目自动化之道》预定2005年初出版)。

前 言

欢迎来到单元测试的世界。我们希望这本书能成为你和你的团队非常有价值的资源。你可以告诉我们它给你带来的帮助,或者我们需要做哪些改进,也可以通过访问我们网站²中为《Pragmatic Unit Testing (单元测试之道)》专门开设的网页,并点击“FeedBack”来给我们提供反馈。

反馈能使这些书变得更加优秀,从而使开发者本身甚至整个项目都变得更加优秀。注重实效的 (Pragmatic) 程序员几乎都是使用这种来自现实的反馈 (如单元测试) 来优化代码,调整设计。

是什么让我们对单元测试如此重视呢?我们都明白,对一个程序员而言,单元测试是非常重要的,因为它提供了你所需要的反馈。试想,如果没有单元测试的话,你可能就像在一个 yellow legal pad³上写程序,只能做最好的祈祷来盼望它能够正常运行。

这当然不是个可行的做法。

对此,本书可以给你提供帮助,它主要针对的读者是那些具有一定编码和设计经验,但是对单元测试并不是很有经验的 Java 程序员。

虽然我们是使用 Java 来编写书中的代码,使用的是 JUnit 框架;但是就单元测试中的这些概念而言,无论你用的是什么语言,比如 C++、Fortran、Ruby、Smalltalk 或者 VisualBasic 都是一样的;而且迄今为止,有超过 60 种不同的语言具有自己的、类似于 JUnit 的测试框架,这些框架都可以从网上免费下载⁴。

² <http://www.pragmaticprogrammer.com/sk/ut/>。

³ 译注: A pad of ruled, usually yellow writing paper that measures 8¹/₂ by 14 inches。

⁴ <http://www.xprogramming.com/software.htm>。

对于已经用过单元测试的一些高级程序员，我们希望本书同样可以给你带来一些惊喜。你们可以跳过介绍 JUnit 用法的那些基础性章节，把精力放到后面的一些章节，内容包括如何来考虑怎样测试，测试是如何影响设计的，以及你可能会遇到的某些能影响整个开发小组的问题。

而且，要记住本书只是一个开始。它可能是你读过的第一本讲述单元测试的书，但是我们不希望它是最后一本。

■ 何处获取源码

在本书的许多地方，你都会发现 Java 代码的例子；其中某些是完整的程序，而另外一些则是一些程序的片断。如果你希望运行这些例子代码，或者查看完整的源码（而不是书中所给出的代码片断），你可以留意一下源码所在页的边缘：我们将会给出包含该源码片断的完整源码所在的文件名。

某些源码片断将会随着讨论的深入不断增加，因此在某些主目录和子目录中，你会发现一些源码文件的名字是相同的，只是子目录所在的版本有更新而已（如 rev1，rev2 等等）。

本书中所有代码可以在本书专题网站上获取，地址为：

<http://www.pragmaticprogrammer.com/starter kit/ut/index.html>

■ 排版标记的规范

黑体	表明这里的名词是正要被定义的名词，或者来自于其它语言的名词。
定宽字体	表明这里是方法名称，文件名称，类的名称，或者其它各种常量字符串。
xxx xx xx;	表明这里是源码中不重要的部分，在书中的代码片断中，它们被省略了。



曲线箭头标记表明这些内容是比较高级的，如果你第一次没有看懂的话，可以跳过。



“开发者 Joe”，他是我们的卡通朋友，在此他会提出一个相关的问题，你或许会发现这个问题非常有用。



Stop 按钮表明：读到这个地方的时候，你应该停下来，开始思考所提出的问题，或者在计算机上亲手做一些实验；得到结果之后，才继续阅读。

致谢

我们要特别感谢本书下面的许多参与者：**Mitch Amiano, Nascif Abousalh-Neto, Andrew C. Oliver, Jared Richardson** 以及 **Bobby Woolf**，他们帮助我们输入，给了我们许多宝贵的建议和开发过程中发生的小故事。

我们还要感谢我们的审阅者。在指正我们的错误、疏忽和偶尔不太规范的语言习惯方面，他们花了很多时间和精力；这其中包括：**Will Gwaltney, Sharee L. Johnson, Eric Kalendra, Al Koscielny, James J. O'Connor III, Mike Stok, Drew Thompson** 以及 **Eric Vought**。

非常感谢所有对本书做出贡献的人们，感谢你们的工作和支持。

Andy Hunt and Dave Thomas

September, 2003

`pragprog@pragmaticprogrammer.com`

目 录

关于程序员修炼三部曲.....	xi
前言.....	xiii
第 1 章 序言.....	1
1.1 自信地编码.....	2
1.2 什么是单元测试.....	3
1.3 为什么要使用单元测试.....	4
1.4 我需要做什么呢.....	5
1.5 如何进行单元测试.....	7
1.6 不写测试的借口.....	7
1.7 本书概要.....	12
第 2 章 你的首个单元测试.....	13
2.1 计划你的测试.....	14
2.2 测试一个简单的方法.....	15
2.3 更多的测试.....	20
第 3 章 使用 JUnit 编写测试.....	21
3.1 构建单元测试.....	21
3.2 JUnit 的各种断言.....	22
3.3 JUnit 框架.....	26
3.4 JUnit 测试的组成.....	27
3.5 自定义 JUnit 断言.....	32
3.6 JUnit 和异常.....	33
3.7 关于命名的更多说明.....	35
3.8 JUnit 测试骨架.....	35

第4章 测试哪些内容: Right-BICEP	37
4.1 结果是否正确.....	38
4.2 边界条件.....	41
4.3 检查反向关联.....	42
4.4 使用其他手段来实现交叉检查.....	42
4.5 强制产生错误条件.....	43
4.6 性能特性.....	44
第5章 CORRECT 边界条件	47
5.1 一致性.....	48
5.2 有序性.....	49
5.3 区间性.....	51
5.4 引用/耦合性.....	54
5.5 存在性.....	55
5.6 基数性.....	56
5.7 时间性.....	58
5.8 自己动手尝试.....	60
第6章 使用 Mock 对象	65
6.1 简单的替换.....	66
6.2 Mock 对象.....	67
6.3 测试 Servlet.....	71
6.4 Easy Mock 对象.....	74
第7章 好的测试所具有的品质	79
7.1 自动化.....	80
7.2 彻底的.....	81
7.3 可重复.....	83
7.4 独立的.....	83
7.5 专业的.....	84
7.6 对测试进行测试.....	86
第8章 在项目中进行测试	89
8.1 把测试代码放到哪.....	89
8.2 测试的礼貌.....	93
8.3 测试的频率.....	94

8.4 测试与遗留代码	95
8.5 测试与评审	98
第9章 设计话题	101
9.1 面向测试的设计	101
9.2 为测试而重构	103
9.3 测试类的不变性	114
9.4 测试驱动的设计	117
9.5 测试无效的参数	119
附录 A Gotchas	121
A.1 只要代码能工作就可以	121
A.2 “冒烟”测试	121
A.3 “请让我的机器来运行”	122
A.4 浮点数问题	122
A.5 测试耗费的时间太多了	123
A.6 测试总是失败	123
A.7 在某些机器上测试失败	124
A.8 我的 main 没有被运行	125
附录 B 安装 JUnit	127
B.1 命令行安装	128
B.2 管用吗	129
附录 C JUnit 测试骨架	131
C.1 辅助类	133
C.2 基本模板	133
附录 D 资源	137
D.1 网络资源	137
D.2 参考书目	139
附录 E 注重实效的单元测试：总结	141
附录 F 习题答案	143
索引	155

第 1 章

序言

Introduction

在一个软件项目中，我们可以做许多各式各样的测试，而且这些测试也是必须的。其中的某些测试需要用户的大量参与；而某些则需要有专门的质量保证小组来进行；或者需要其他的一些昂贵资源。

然而，这些测试并不是我们所要谈论的主题。

我们谈的是**单元测试**：它是项目成功、个人成功的一个不可或缺的部分，但对它，人们却又存在各种各样的误解。单元测试其实是相对廉价而简单的技术，但它能让你更高效地写出质量更好的代码。

说到测试，大凡组织和个人都会满怀雄心壮志，但是往往只是在项目快要结束的时候才想起测试。而那时的进度压力一定非常紧迫，所以结果往往只是浅尝辄止或者干脆就不测了。

许多程序员觉得测试只是一件麻烦事：一种自找的烦恼，唯一的“效果”就是让他们没法专注于手上的正经事——`cutting code`（堆砌代码）。

每个人都同意，是的，该做更多的测试。这种人人同意的事情还多着呢，是的，该多吃蔬菜，该戒烟，该多休息，该多锻炼……这并不意味着我们中的所有人都会这么去做，不是吗？

但是单元测试却远远不仅仅是上面这些——也许你会认为单元测试是花菜那一类的，而我要说它更像一勺美味的调料，它让每份菜肴品尝起来都

更加可口。单元测试的设计目的并不是为了获得一些更好的整体质量。也就是说，它并不是一个针对最终用户、项目经理和开发组长的工具；而是由程序员自己来完成，并且最终受益的也是程序员自己。我们是为了自身的利益去使用单元测试，从而让我们的工作变得更加轻松。

简而言之，在有些时候，使用单元测试本身就能决定你的项目的成败。接下来，让我们来看看下面这个小故事。

1.1 自信地编码

有一次——或许就是上个礼拜二——有两个开发者：**Pat** 和 **Dale**。他们面临着相同的最后期限，而这一天也越来越近了。**Pat** 每天都在着急地编写代码，写完一个类又写一个类，写完一个函数又接着写另一个函数，还经常不得不停下来做一些调整，使得代码能够通过编译。

Pat 一直保持着这种工作方式，直到最后期限的前一天。而这时已经是演示所有代码的时候了。**Pat** 运行了最上层的程序，但是一点输出也没有，什么都没有。这时只好用调试器来单步跟踪了。“Hmm，决不可能是这样的”，**Pat** 想，“此时这个变量绝对不是 0 啊”。于是，**Pat** 只能回过头来看代码，尝试着跟踪一下这个难以琢磨的程序的调用流程。

时间已经越来越晚了，**Pat** 找到并且纠正了这个 bug；但在这个过程中，**Pat** 又找到了其他好几个 bug；如此几次过后，bug 还是存在。而程序输出那边，仍然没有结果。这时，**Pat** 已经筋疲力尽了，完全搞不清楚为什么会这样，认为这种（没有输出的）行为是毫无道理的。

而于此同时，**Dale** 并没像 **Pat** 那么快地写代码。**Dale** 在写一个函数的时候，会附带写一个简短的测试程序来测试这个函数。这里没有什么特殊的地方，只是添加了一个简单的测试，来判断函数的功能是否和程序员期望的一致。显然，考虑如何写，然后把测试写出来，是需要占用一定时间的；但是 **Dale** 在未对刚写的函数做出确认之前，是不会接着写新代码的。也就是说，只有等到已知函数都得到确认之后，**Dale** 才会继续编写下一个函数，然后调用前面的函数等等。

在整个过程中，Dale 几乎不使用调试器；而且对 Pat 的模样也有些困惑不解：只见他头埋在两手之间，嘀咕着各种难听的话语，咒骂着计算机，充血的眼球同时盯着好几个调试窗口。

最后期限终于到了，Pat 未能完成任务。而 Dale 的代码被集成到整个系统中，并且能够很好地运行。之后，在 Dale 的模块中，出现了一个小问题；但是 Dale 很快就发现了问题所在，在几分钟之内就把问题解决了。

现在，是该总结一下上面这个小故事的时候了：Dale 和 Pat 的年纪相当，编码能力相当，智力也差不多。唯一的区别就是 Dale 非常相信单元测试；对于每个新写的函数，在其他代码使用这个函数并对它形成依赖之前，都要先做单元测试。

而 Pat 则没有这么做，他总是“知道”代码的行为应该和所期望的完全一样，并且等到所有代码都差不多写完的时候，才想起来运行一下代码。然而到了这个时候，要想定位 bug，或者，甚至是确定哪些代码的行为是正确的，哪些代码的行为是错误的，都为时已晚了。

1.2 什么是单元测试

单元测试是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件（或者场景）下某个特定函数的行为。例如，你可能把一个很大的值放入一个有序 list 中去，然后确认该值出现在 list 的尾部。或者，你可能会从字符串中删除匹配某种模式的字符，然后确认字符串确实不再包含这些字符了。

执行单元测试，是为了证明某段代码的行为确实和开发者所期望的一致。

对于客户或最终使用者而言，这种测试必要吗，它与验收测试有关吗？这个问题仍然很难回答。事实上，我们在此并不关心整个产品的确认、验证和正确性等等；甚至此时，我们都不去关心性能方面的问题。我们所要做的一切就是要证明代码的行为和我们的期望一致。因此，我们所要测试的是规模很小的、非常独立的功能片断。通过对所有单独部分的行为建立起信心，确信它们都和我们的期望一致；然后，我们才能开始组装和测试整个系统。

毕竟，要是我们对手上正在写的代码的行为是否和我们的期望一致都没把握，那么其他形式的测试也都只能是浪费时间而已。在单元测试之后，你还需要其他形式的测试，有可能是更正规的测试，那一切就都要看环境的需要来决定。总之，做测试如同做善事，总是要从家¹开始。

1.3 为什么要使用单元测试

单元测试不但会使你的工作完成得更轻松，而且会令你的设计变得更好，甚至大大减少你花在调试上面的时间。

在我们上面的小故事里面，Pat 因为假设底层的代码是正确无误的而卷入麻烦之中，先是在高层代码中使用了底层代码；然后这些高层代码又被更高层的代码所使用，如此往复。在对这些代码的行为没有任何信心的前提下，Pat 等于是在假设上面用竖立卡片堆砌了一间房子——只要将下面卡片轻轻移动，整间房子就会轰然倒塌。

当基本的底层代码不再可靠时，那么必需的改动就无法只局限在底层。虽然你可以修正底层的问题，但是这些对底层代码的修改必然会影响到高层代码，于是高层代码也连带地需要修改；以此递推，就很可能动到更高层的代码。于是，一个对底层代码的修正，可能会导致对几乎所有代码的一连串改动，从而使修改越来越多，也越来越复杂。于是，整间由卡片堆成的房子就由此倒塌，从而使整个项目也以失败告终。

Pat 总是说：“这怎么可能呢？”或者“我实在想不明白为什么会这样”。如果你发现自己有时候也会有这种想法，那么这通常是你对自己的代码还缺乏足够信心的表现——你并不能确认哪些是工作正常的而哪些不是。

为了获得 Dale 所具有的那种对代码的信心，你需要“询问”代码究竟做了什么，并检查所产生的结果是否确实和你所期望的一致。

这个简单的想法描述了单元测试的核心内涵：这个简单有效的技术就是为了令代码变得更加完美。

¹ 译注：begins at home，这里的家指的是代码最基本的正确性。