

The Turing logo, featuring the word "TURING" in a bold, sans-serif font with a small orange flame-like icon above the "I".

图灵程序设计丛书

/THEORY/IN/PRACTICE

软件之道

软件开发争议问题剖析

Making Software

[美] Andy Oram Greg Wilson 编著
鲍央舟 张玳 沈欢星 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS



图灵程序设计丛书

软件之道 ——软件开发争议问题剖析

Making Software
What Really Works, and Why We Believe It

[美] Andy Oram Greg Wilson 编著
鲍央舟 张 玳 沈欢星 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc. 授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目 (C I P) 数据

软件之道：软件开发争议问题剖析 / (美) 欧莱姆 (Oram, A.), (美) 威尔逊 (Wilson, G.) 编著; 鲍央舟, 张玳, 沈欢星译. — 北京: 人民邮电出版社, 2012.3

(图灵程序设计丛书)

书名原文: Making Software

ISBN 978-7-115-27044-3

I. ①软… II. ①欧… ②威… ③鲍… ④张… ⑤沈…
… III. ①软件开发 IV. ①TP311.52

中国版本图书馆CIP数据核字(2011)第258450号

内 容 提 要

本书集合了几十位软件工程领域顶尖研究人员的实证研究, 通过呈现他们长达几年甚至几十年的研究成果, 揭示了软件开发社区普遍存在的一些确凿事实和虚构之事。书中探讨了更有效的编程语言, 对比了软件开发人员之间的效率差异, 验证了康威定理, 并反思了软件行业的最新模式。本书将帮助读者拓宽视野, 更好地选择适合的工具和技术, 并最终成为一名更加优秀的软件行业从业人员。

本书适合所有软件开发人员和研究人员阅读。

图灵程序设计丛书

软件之道：软件开发争议问题剖析

- ◆ 编著 [美] Andy Oram, Greg Wilson
译 鲍央舟 张 玳 沈欢星
责任编辑 傅志红
执行编辑 李 盼
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
- ◆ 开本: 800×1000 1/16
印张: 28.5
字数: 728千字 2012年3月第1版
印数: 1-3 500册 2012年3月北京第1次印刷
著作权合同登记号 图字: 01-2010-8061号

ISBN 978-7-115-27044-3

定价: 89.00元

读者服务热线: (010)51095186转604 印装质量热线: (010)67129223

反盗版热线: (010)67171154

版 权 声 明

© 2011 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2012. Authorized translation of the English edition, 2012 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由O'Reilly Media, Inc. 出版2011。

简体中文版由人民邮电出版社出版 2012。英文原版的翻译得到O'Reilly Media, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、在线服务、杂志、调查研究和会议等方式传播创新者的知识。自1978年开始O'Reilly一直都是发展前沿的见证者和推动者。超级极客正在开创未来，我们关注着真正重要的技术趋势，通过放大那些“微弱的信号”来刺激社会对新科技的采用。作为技术社区中活跃的参与者，O'Reilly的发展充满着对创新的倡导、创造和发扬光大。

作为出版商O'Reilly为软件开发人员带来革命性的“动物书”，创造了第一个商业网站(GNN)，组织开放源代码峰会以至于开源软件运动以此命名，通过创立Make杂志成为DIY革命的主要先锋，公司一如既往地用各种方式和渠道连接人们和他们所需要的信息。O'Reilly的会议和峰会聚集了超级极客和高瞻远瞩的商业领袖，共同描绘将开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通计算机用户。无论是通过印刷书籍、在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的信念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位少有的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

译者序

软件行业自诞生起，对软件开发中技术、流程、工具和实践的探索就从未停止过。一个个新理念像一阵阵潮流般，在支持声中诞生，在反对和质疑声中被下一个潮流所淹没。20年前的OO，12年前的UML，10年前的CMMI，昨天的Scrum，今天的看板，明天的热点又会是什么？回顾历史，我们是否在这些潮流中真正探索到了什么是“软件之道”呢？也许有。那我们是否已经找到一种普遍认同的标准软件开发方法呢？应该还没有。

有人认为，这是因为软件行业是新兴产业的缘故，不像建筑业那样有几百年的历史，所以暂时还没有一套经得起沉淀的方法论出现。也有人认为，软件行业是一个基于不断实践的行业，更像是下棋，永远无法有一种确切的方法告诉你怎么样一定能赢。无论如何，现阶段的软件开发，确实没有权威方法论的存在。这也正是本书中各种实证论文的意义：引导和帮助你分析现有的实证，并去探索和寻找切合自己的方法。

正如书中所说，那些我们所熟知的技术、流程、工具、实践所产生的作用是因人而异的。有人说在开发之前要尽可能全面地做架构设计，也有人说随着敏捷开发的浪潮，我们更鼓励边开发边设计，而不是提前把架构设计好；有人说测试驱动开发会有效地提升质量，也有人说测试驱动开发的性价比太低；有人说好的程序员的生产力是差的程序员的10倍甚至100倍，也有人说根本没有有效的方法能找到这样的程序员。对于这些众说纷纭的热点话题，本书提供了相应的证据供大家参考。但是更重要的是，本书告诉我们，每个人都需要从自己的实际角度出发，判断证据的可信度和适用度。因为软件开发的复杂性决定了，对别人适用的证据并不一定对你适用。

在翻译这本书的半年中，我也不断对过去一些实践和想法做出了总结。不得不承认，之前对业界的一些热点话题有过盲目听信或者盲目反对。书中对我触动最深的是第2章中的最后一小节，其中说到，作为一个软件工程师，每个人都不应该放弃判断现有证据的资格和能力。分析现有证据中的数据，并结合自己的实际情况做出相应的判断，这是每个专业工作者的责任。盲目地跟风或者盲目地批判都不是专业的表现。

于此同时，本书也对业界的研究者提出了更高的要求。在还不算成熟的软件行业中，我们应该如何向更成熟的行业（如医学）学习，从而把已有的研究有效地聚集起来，形成对整个行业都有用的数据库。在此基础之上，从业者们可以更好地找到适应特定环境的证据，也能更容易地做出判断。

如果你曾对软件行业中纷纷扰扰的说法存有疑惑，如果你正在摸索属于自己的“软件之道”，如果你想寻求为整个行业做出贡献的方法，那这本书应该能起到传道授业解惑的作用。

本书的翻译团队包括鲍央舟、沈欢星和张玳，三者均为软件行业从业人员。在半年中抽取了大量空余时间投入本书的翻译和审校工作。在翻译的过程中，遇到了众多数学、统计学、心理学的专业术语，在一些朋友的帮助下，才得以顺利完成。在这里想感谢所有帮助过我们的朋友，也感谢我的公司Out Softing对我工作的支持。如有疏漏和不足之处，希望广大读者批评指正。

鲍央舟

前 言

MMR疫苗会引发孤独症吗？电视里的暴力镜头会使孩子们更暴力吗？某些编程语言比其他一些更好吗？人们每天都会争论这些问题。要认真地回答前两个问题必须依靠科学方法：小心地收集证据，公平地评估效果。然而，迄今为止，很少有人试图用这样的技巧来回答第三个问题。当说到计算机相关工作的时候，边喝啤酒边说出来的有关华沙创业公司的轶事，通常就是大部分程序员所期望的所有“证据”了。

这种情况正在改变，部分归功于本书撰稿人的工作。本书作者和他们的同事从不同领域获取数据，诸如数据挖掘、认知心理学以及社会学……他们正在创造一种软件工程的循证方法。通过从无数初始材料中搜集证据并分析结果，他们正在为一些软件工程的恼人问题带来新的光明。大部分程序员在他们的第一份工作中会如何出错？测试驱动开发会产生更好的代码吗？结对编程或代码审查又如何？可能在发布之前预测一段代码中缺陷的大概数目吗？如果能的话，怎么做？

这本书中的论文会提供一些问题的解答，并解释为什么其他问题仍然没有答案。同样重要的是，它们会告诉你如何用定量和定性的方法自己找到并评估证据。每个程序员都是独特的，没有任何两个程序是完全相同的，但如果你仔细、耐心、开明的话，就能说服他们说出秘密。

我们希望本书中的问题和解答能改变你对软件开发的想法。我们也希望这些论文能说服你，在下次有人声称C和Java中这样放置大括号比那样放置更好的时候，你会说“请举证”。和《代码之美》^①一样（Andy和Greg与O'Reilly的上次合作），作者的所有版税会捐赠于有关机构。

本书的结构

本书中的每一章节都来自不同的撰稿人或团队。顺序并不重要，但是我们把一些关于研究、有效性和其他具有高层意义的章节放在开头。我们觉得在读过第一部分后，读者会掌握一定背景知识，从而能更好地理解第二部分的内容。

第一部分包含以下内容：

第1章 探寻有力的证据（Tim Menzies和Forrest Shull著）

第2章 可信度，为什么我坚决要求确信的证据（Lutz Prechelt和Marian Petre著）

^① 该书由机械工业出版社于2009年出版。——编者注

- 第3章 我们能从系统性评审中学到什么 (Barbara Kitchenham著)
- 第4章 用定性研究方法理解软件工程学 (Andrew Ko 著)
- 第5章 在实践中学习成长: 软件工程实验室中的质量改进范式 (Victor R. Basili 著)
- 第6章 性格、智力和专业技能对软件开发的影响 (Jo E. Hannay著)
- 第7章 为什么学编程这么难 (Mark Guzdial著)
- 第8章 超越代码行: 我们还需要其他的复杂度指标吗 (Israel Herraiz和Ahmed E. Hassan著)
- 第二部分包含以下内容:
- 第9章 自动故障预报系统实例一则 (Elaine J. Weyuker和Thomas J. Ostrand著)
- 第10章 架构设计的程度和时机 (Barry Boehm著)
- 第11章 康威推论 (Christian Bird著)
- 第12章 测试驱动开发的效果如何 (Burak Turhan、Lucas Layman、Madeline Diep、Hakan Erdogmus和Forrest Shull著)
- 第13章 为何计算机科学领域的女性不多 (Michele A. Whitecraft和Wendy M. Williams著)
- 第14章 两个关于编程语言的比较 (Lutz Prechelt著)
- 第15章 质量之战: 开源软件对战专有软件 (Diomidis Spinellis著)
- 第16章 码语者 (Robert DeLine著)
- 第17章 结对编程 (Laurie Williams著)
- 第18章 现代化代码审查 (Jason Cohen著)
- 第19章 公共办公室还是私人办公室 (Jorge Aranda著)
- 第20章 识别及管理全球性软件开发中的依赖关系 (Marcelo Cataldo著)
- 第21章 模块化的效果如何 (Neil Thomas和Gail Murphy著)
- 第22章 设计模式的证据 (Walter Tichy著)
- 第23章 循证故障预测 (Nachiappan Nagappan和Thomas Ball著)
- 第24章 采集缺陷报告的艺术 (Rahul Premraj和Thomas Zimmermann著)
- 第25章 软件的缺陷都从哪儿来 (Dewayne Perry著)
- 第26章 新手专家: 软件行业的应届毕业生们 (Andrew Begel和Beth Simon著)
- 第27章 挖掘你自己的证据 (Kim Sebastian Herzig和Andreas Zeller著)
- 第28章 正当使用“复制-粘贴”大法 (Michael Godfrey和Cory Kapser著)
- 第29章 你的API有多好用 (Steven Clarke著)
- 第30章 “10倍”意味着什么? 编程生产力的差距测量 (Steve McConnell著)

本书的排版规范

本书使用的排版规范如下所示。

- 楷体

用于表示新的术语。

- 等宽字体

表示程序片段，也在段落中表示程序中使用的变量、函数名、命令行实用工具、环境变量、语句和关键字等元素。

- 等宽加粗

这种字体表示用户需要手动输入的命令或者相应的文本。

- 等宽斜体

用户需要根据自己所提供的值或由上下文所确定的值进行更改的部分。

Safari®在线图书



Safari在线图书是应需而变的数字图书馆。它能够让你非常轻松地搜索7500多种技术性和创新性参考书以及视频，以便快速地找到需要的答案。

订阅后就可以访问在线图书馆内的所有页面和视频。可以在手机或其他移动设备上阅读。还能在新书上市之前抢先阅读，也能够看到还在创作中的书稿并向作者反馈意见。复制粘贴代码示例、放入收藏夹、下载部分章节、标记关键点、做笔记甚至打印页面等有用的功能可以节省大量时间。这本书也在其中。

欲访问本书英文版的电子版，或者由O'Reilly或其他出版社出版的相关图书，请到<http://my.safaribook-sonline.com>免费注册。

使用代码范例

让本书助你一臂之力。也许你要在自己的程序或文档中用到本书中的代码。除非大段大段地使用，否则不必与我们联系取得授权。例如，无需请求许可，就可以用本书中的几段代码写成一个程序。但是销售或者发布O'Reilly图书中代码的光盘则必须事先获得授权。引用书中的代码来回答问题也无需授权。将大段的示例代码整合到你自己的产品文档中则必须经过许可。

我们非常希望你能标明出处，但并不强求。出处一般含有书名、作者、出版商和ISBN，例如：“*Making Software :What Really Works, and Why We Believe it* by Andy Oram and Greg Wilson. Copyright 2011 O'Reilly, Media, Inc., 978-0-596-80832-7”。

如果有关于使用代码的未尽事宜，可以随时与我们联系：permissions@oreilly.com。

我们的联系方式

请把对本书的评论和问题发给出版社。

美国：

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国:

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室 (100035)

奥莱利技术咨询(北京)有限公司

O'Reilly 的每一本书都有专属网页, 你可以在那儿找到关于本书的相关信息, 包括勘误表、示例代码以及其他的信息。本书的网站地址是:

<http://www.oreilly.com/catalog/9780596808327/>

中文书:

<http://www.oreilly.com.cn/book.php?bn=index.php?func=book&isbn=9787115270443>

对于本书的评论和技术性的问题, 请发送电子邮件到:

bookquestions@oreilly.com

关于本书的更多信息、会议、资源中心和网络, 请访问以下网站:

<http://www.oreilly.com>

<http://www.oreilly.com.cn>

目 录

第一部分 搜寻和使用证据的一般原则

第 1 章 探寻有力的证据	2
1.1 起步阶段	2
1.2 当今证据的状态	3
1.2.1 精确性研究的挑战	3
1.2.2 统计强度的挑战	3
1.2.3 结果可复制性的挑战	4
1.3 我们可以相信的改变	5
1.4 背景的影响	7
1.5 展望未来	7
1.6 参考文献	9
第 2 章 可信度, 为什么我坚决要求 确信的证据	12
2.1 软件工程中的证据是如何发现的	12
2.2 可信度和适用性	13
2.2.1 适用性, 为什么使你信服的 证据不能使我信服	14
2.2.2 定性证据对战定量证据: 错误的二分法	15
2.3 整合证据	16
2.4 证据的类型以及它们的优缺点	17
2.4.1 对照实验和准实验	18
2.4.2 问卷调查	19
2.4.3 经验汇报和案例研究	20
2.4.4 其他方法	20
2.4.5 报告中的可信度 (或缺乏可信 度) 的标识	21
2.5 社会、文化、软件工程和你的	23
2.6 致谢	24
2.7 参考文献	24

第 3 章 我们能从系统性评审中学到 什么	25
3.1 系统性评审总览	26
3.2 系统性评审的长处和短处	27
3.2.1 系统性评审的流程	28
3.2.2 开展一项评审所牵连的问题	30
3.3 软件工程中的系统性评审	31
3.3.1 成本估算研究	32
3.3.2 敏捷方法	33
3.3.3 检验方法	35
3.4 结论	35
3.5 参考文献	36
第 4 章 用定性研究方法来理解软件 工程学	40
4.1 何为定性研究方法	41
4.2 如何解读定性研究	42
4.3 在工作中运用定性研究方法	44
4.4 推广应用定性研究的结果	45
4.5 定性研究方法是系统的研究方法	46
4.6 参考文献	46
第 5 章 在实践中学习成长: 软件工程 实验室中的质量改进范式	47
5.1 软件工程研究独有的困难之处	47
5.2 实证研究的现实之路	48
5.3 NASA 软件工程实验室: 一个充满 活力的实证研究测试平台	48
5.4 质量改进范式	49
5.4.1 表征	51
5.4.2 设立目标	51
5.4.3 选择流程	51

5.4.4 执行流程	53	8.3.1 源代码行数 (SLOC)	96
5.4.5 分析	53	8.3.2 代码行数 (LOC)	96
5.4.6 封装	53	8.3.3 C 函数的数量	96
5.5 结论	55	8.3.4 McCabe 圈复杂度	96
5.6 参考文献	55	8.3.5 Halstead 软件科学指标	97
第 6 章 性格、智力和专业技能对 软件开发的影响	57	8.4 统计分析	98
6.1 如何辨别优秀的程序员	58	8.4.1 总体分析	98
6.1.1 个体差异: 固定的还是可 塑造的	58	8.4.2 头文件和非头文件之间的 区别	99
6.1.2 个性	59	8.4.3 干扰效应: 文件大小对相关 性的影响	100
6.1.3 智力	63	8.5 关于统计学方法的一些说明	103
6.1.4 编程任务	65	8.6 还需要其他的复杂度指标吗	103
6.1.5 编程表现	65	8.7 参考文献	104
6.1.6 专业技能	66	第二部分 软件工程的特有话题	
6.1.7 软件工作量估算	68	第 9 章 自动故障预报系统实例一则	106
6.2 环境因素还是个人因素	68	9.1 故障的分布	106
6.2.1 软件工程中应该提高技能还是 提高安全保障	69	9.2 故障高发文件的特征	109
6.2.2 合作	69	9.3 预测模型概览	109
6.2.3 再谈个性	72	9.4 预测模型的复验和变体	110
6.2.4 从更广的角度看待智力	72	9.4.1 开发人员的角色	111
6.3 结束语	74	9.4.2 用其他类型的模型来预测 故障	113
6.4 参考文献	75	9.5 工具的设计	115
第 7 章 为什么学编程这么难	81	9.6 一些忠告	115
7.1 学生学习编程有困难吗	82	9.7 参考文献	117
7.1.1 2001 年 McCracken 工作小组	82	第 10 章 架构设计的程度和时机	119
7.1.2 Lister 工作小组	83	10.1 修正缺陷的成本是否会随着项目 的进行而增加	119
7.2 人们对编程的本能理解是什么	83	10.2 架构设计应该做到什么程度	120
7.3 通过可视化编程来优化工具	85	10.3 架构设计的成本-修复数据给予 我们的启示	123
7.4 融入语境后的改变	86	10.3.1 关于 COCOMO II 架构 设计和风险解决系数的 基础知识	123
7.5 总结: 一个新兴的领域	88		
7.6 参考文献	89		
第 8 章 超越代码行: 我们还需要 其他的复杂度指标吗	92		
8.1 对软件的调查	92		
8.2 计算源代码的指标	93		
8.3 指标计算案例	94		

10.3.2 Ada COCOMO 及 COCOMO II 中的架构设计以及风险应对 系数.....	125	13.2 值得在意吗.....	168
10.3.3 用于改善系统设计的投入 的 ROI.....	130	13.2.1 扭转这种趋势, 我们可以 做些什么.....	170
10.4 那么到底架构要做到什么程度 才够.....	132	13.2.2 跨国数据的意义.....	171
10.5 架构设计是否必须提前做好.....	135	13.3 结论.....	172
10.6 总结.....	135	13.4 参考文献.....	172
10.7 参考文献.....	136	第 14 章 两个关于编程语言的比较.....	175
第 11 章 康威推论.....	138	14.1 一个搜索算法决定了一种语言的 胜出.....	175
11.1 康威定律.....	138	14.1.1 编程任务: 电话编码.....	176
11.2 协调工作、和谐度和效率.....	140	14.1.2 比较执行速度.....	176
11.3 微软公司的组织复杂度.....	143	14.1.3 内存使用情况的比较.....	178
11.4 开源软件集市上的小教堂.....	148	14.1.4 比较效率和代码长度.....	178
11.5 总结.....	152	14.1.5 比较可靠性.....	180
11.6 参考文献.....	152	14.1.6 比较程序结构.....	180
第 12 章 测试驱动开发的效果如何.....	153	14.1.7 我可以相信吗.....	181
12.1 TDD 药丸是什么.....	153	14.2 Plat_Forms: 网络开发技术和文化.....	182
12.2 TDD 临床试验概要.....	154	14.2.1 开发任务: 人以类聚.....	182
12.3 TDD 的效力.....	156	14.2.2 下注吧.....	183
12.3.1 内部质量.....	156	14.2.3 比较工作效率.....	184
12.3.2 外部质量.....	157	14.2.4 比较软件工件的大小.....	185
12.3.3 生产力.....	157	14.2.5 比较可修改性.....	186
12.3.4 测试质量.....	158	14.2.6 比较稳健性和安全性.....	187
12.4 在试验中强制 TDD 的正确剂量.....	158	14.2.7 嘿, “插入你自己的话题” 如何.....	189
12.5 警告和副作用.....	159	14.3 那又怎样.....	189
12.6 结论.....	160	14.4 参考文献.....	189
12.7 致谢.....	160	第 15 章 质量之战: 开源软件对战 专有软件.....	191
12.8 参考文献.....	160	15.1 以往的冲突.....	192
第 13 章 为何计算机科学领域的 女性不多.....	163	15.2 战场.....	192
13.1 为什么女性很少.....	163	15.3 开战.....	195
13.1.1 能力缺陷, 个人喜好以及 文化偏见.....	164	15.3.1 文件组织.....	196
13.1.2 偏见、成见和男性计算机 科学文化.....	166	15.3.2 代码结构.....	200
		15.3.3 代码风格.....	204
		15.3.4 预处理.....	209
		15.3.5 数据组织.....	211
		15.4 成果和结论.....	213

15.5 致谢	215	18.3 团队影响	247
15.6 参考文献	215	18.3.1 是否有必要开会	247
第 16 章 码语者	219	18.3.2 虚假缺陷	247
16.1 程序员的一天	219	18.3.3 外部审查真的需要吗	248
16.1.1 日记研究	220	18.4 结论	249
16.1.2 观察研究	220	18.5 参考文献	249
16.1.3 程序员们是不是在挣表 现	220	第 19 章 公共办公室还是私人办公室	251
16.2 说这么多有什么意义	221	19.1 私人办公室	251
16.2.1 问问题	221	19.2 公共办公室	253
16.2.2 探寻设计理念	223	19.3 工作模式	255
16.2.3 工作的中断和多任务	223	19.4 最后的忠告	257
16.2.4 程序员都在问什么问题	224	19.5 参考文献	257
16.2.5 使用敏捷方法是不是更利 于沟通	227	第 20 章 识别及管理全球性软件开发 中的依赖关系	258
16.3 如何看待沟通	228	20.1 为什么协调工作对于 GSD 来说是 挑战	258
16.4 参考文献	229	20.2 依赖关系及其社会/技术二重性	259
第 17 章 结对编程	230	20.2.1 技术方面	261
17.1 结对编程的历史	230	20.2.2 社会/组织结构方面	263
17.2 产业环境中的结对编程	232	20.2.3 社会—技术方面	266
17.2.1 结对编程的行业实践	232	20.3 从研究到实践	267
17.2.2 业内使用结对编程的效果	233	20.3.1 充分使用软件储存库中的 数据	267
17.3 教育环境中的结对编程	234	20.3.2 团队领导和管理者在依赖 关系管理中的角色	268
17.3.1 教学中特有的实践	234	20.3.3 开发人员、工作项目和分 布式开发	269
17.3.2 教学中使用结对编程的 效果	235	20.4 未来的方向	269
17.4 分布式结对编程	235	20.4.1 适合 GSD 的软件架构	269
17.5 面对的挑战	236	20.4.2 协作软件工程工具	270
17.6 经验教训	237	20.4.3 标准化和灵活度的平衡	271
17.7 致谢	237	20.5 参考文献	271
17.8 参考文献	237	第 21 章 模块化的效果如何	274
第 18 章 现代化代码审查	243	21.1 所分析的软件系统	275
18.1 常识	243	21.2 如何定义“修改”	276
18.2 程序员独立进行小量代码审查	243	21.3 如何定义“模块”	280
18.2.1 防止注意力疲劳	244	21.4 研究结果	281
18.2.2 切忌速度过快	244	21.4.1 修改的范围	281
18.2.3 切忌数量过大	245		
18.2.4 上下文的重要性	246		

21.4.2 需要参考的模块	283	24.3.2 报告者眼中的缺陷报告 内容	326
21.4.3 自发式的模块化	284	24.4 来自不一致信息的证据	327
21.5 有效性的问题	286	24.5 缺陷报告的问题	329
21.6 总结	287	24.6 重复缺陷报告的价值	330
21.7 参考文献	287	24.7 并非所有的缺陷都被修复了	332
第 22 章 设计模式的证据	289	24.8 结论	333
22.1 设计模式的例子	290	24.9 致谢	334
22.2 为什么认为设计模式可行	292	24.10 参考文献	334
22.3 第一个实验：关于设计模式文档的 测试	293	第 25 章 软件的缺陷都从哪儿来	335
22.3.1 实验的设计	293	25.1 研究软件的缺陷	335
22.3.2 研究结果	295	25.2 本次研究的环境和背景	336
22.4 第二个实验：基于设计模式的解决 方案和简单解决方案的对比	297	25.3 第一阶段：总体调查	337
22.5 第三个试验：设计模式之于团队 沟通	300	25.3.1 调查问卷	337
22.6 经验教训	302	25.3.2 数据的总结	339
22.7 总结	304	25.3.3 第一部分的研究总结	342
22.8 致谢	304	25.4 第二阶段：设计/代码编写类故障 调查	342
22.9 参考文献	305	25.4.1 调查问卷	342
第 23 章 循证故障预测	306	25.4.2 统计分析	345
23.1 简介	306	25.4.3 界面故障与实现故障	358
23.2 代码覆盖率	308	25.5 研究结果可靠吗	360
23.3 代码变动	308	25.5.1 我们调查的对象是否正确	360
23.4 代码复杂度	311	25.5.2 我们的方法是否正确	361
23.5 代码依赖	312	25.5.3 我们能利用这些结果做什么	362
23.6 人与组织度量	312	25.6 我们明白了什么	362
23.7 预测缺陷的综合方法	315	25.7 致谢	364
23.8 结论	317	25.8 参考文献	364
23.9 致谢	319	第 26 章 新手专家：软件行业的应届 毕业生们	367
23.10 参考文献	319	26.1 研究方法	368
第 24 章 采集缺陷报告的艺术	322	26.1.1 研究对象	369
24.1 缺陷报告的优劣之分	322	26.1.2 任务分析	370
24.2 优秀缺陷报告需要具备的要素	323	26.1.3 任务案例	370
24.3 调查结果	325	26.1.4 做回顾的方法	371
24.3.1 开发人员眼中的缺陷报告 内容	325	26.1.5 有效性问题	371
		26.2 软件开发任务	372
		26.3 新手开发人员的优点和缺点	374

26.3.1	优点分析	375	28.3.1	分叉	400
26.3.2	缺点分析	375	28.3.2	模板	401
26.4	回顾	376	28.3.3	定制	402
26.4.1	管理层的介入	377	28.4	我们的研究	403
26.4.2	毅力、疑惑和新人特质	377	28.5	总结	405
26.4.3	大型的软件团队环境	378	28.6	参考文献	406
26.5	妨碍学习的误解	378	第 29 章	你的 API 有多好用	407
26.6	教育方法的反思	379	29.1	为什么研究 API 的易用性很重要	407
26.6.1	结对编程	380	29.2	研究 API 易用性的首次尝试	409
26.6.2	合理的边际参与	380	29.2.1	研究的设计	410
26.6.3	导师制	380	29.2.2	第一次研究的结论摘要	411
26.7	改变的意义	381	29.3	如果一开始你没有成功	412
26.7.1	新人培训	381	29.3.1	第二次研究的设计	412
26.7.2	学校教育	382	29.3.2	第二次研究的结论摘要	412
26.8	参考文献	383	29.3.3	认知维度	414
第 27 章	挖掘你自己的证据	385	29.4	使用不同的工作风格	418
27.1	对什么进行数据挖掘	385	29.5	结论	421
27.2	设计你的研究	386	29.6	参考文献	422
27.3	数据挖掘入门	387	第 30 章	“10 倍”意味着什么？编程 生产力的差距测量	423
27.3.1	第一步：确定要用哪些 数据	387	30.1	软件开发中的个人效率的变化	423
27.3.2	第二步：获取数据	388	30.1.1	巨大的差距带来的负面 影响	424
27.3.3	第三步：数据转换 (可选)	389	30.1.2	什么造就了真正的“10 倍 程序员”	424
27.3.4	第四步：提取数据	389	30.2	测量程序员的个人生产力的问题	424
27.3.5	第五步：解析 bug 报告	390	30.2.1	生产力=每月产出的代码 行数吗	424
27.3.6	第六步：关联数据	390	30.2.2	生产力=功能点吗	425
27.3.7	第六步：找出漏掉的关联	391	30.2.3	复杂度呢	425
27.3.8	第七步：将 bug 对应到文 件	391	30.2.4	到底有没有办法可以测量 个人生产力	425
27.4	下面怎么办	392	30.3	软件开发中的团队生产力差距	426
27.5	致谢	394	30.4	参考文献	427
27.6	参考文献	394	撰稿人		429
第 28 章	正当使用“复制-粘贴”大法	396			
28.1	代码克隆的示例	396			
28.2	寻找软件中的克隆代码	398			
28.3	对代码克隆行为的调查	399			