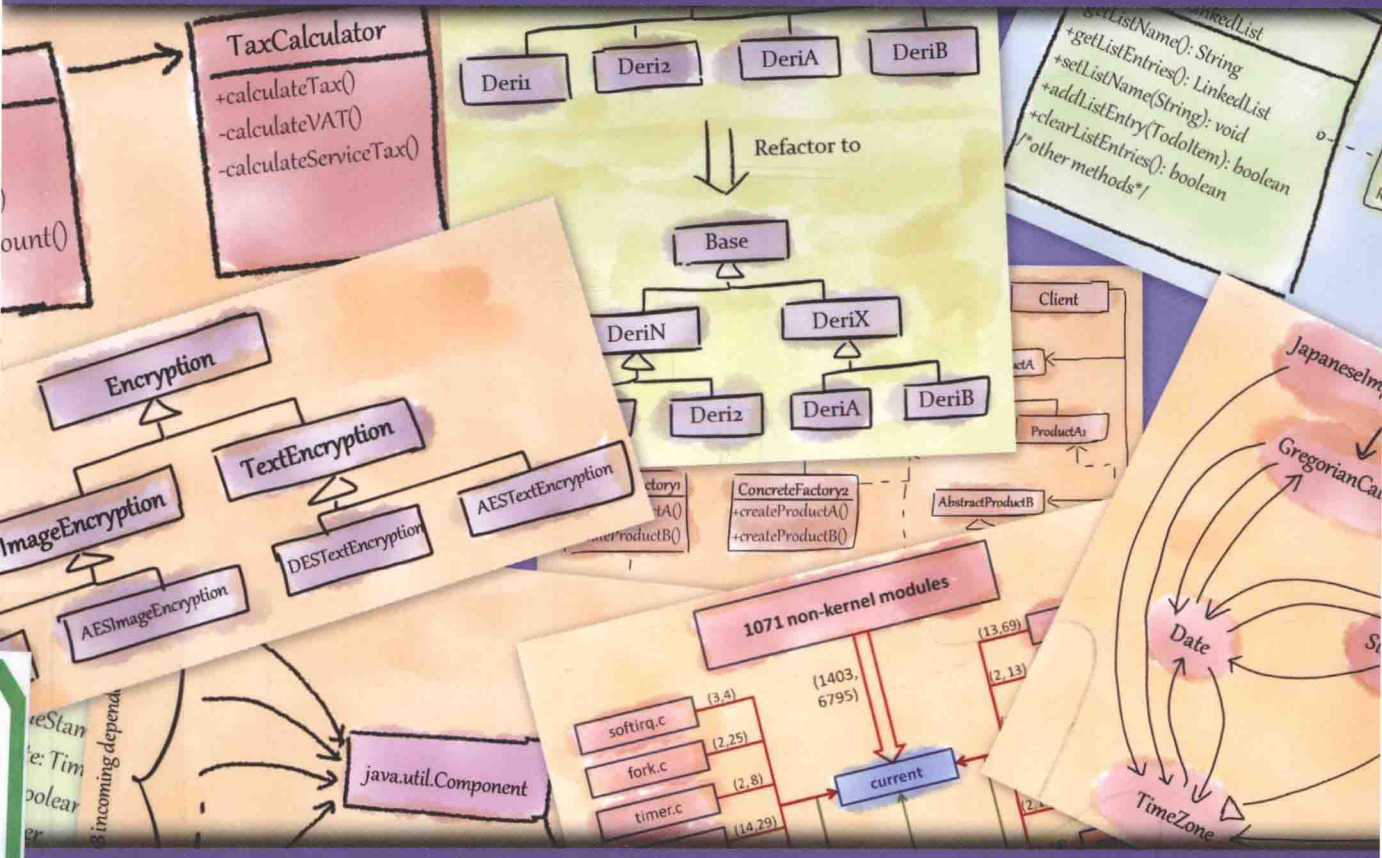
TURING 图灵程序设计丛书

ELSEVIER 爱思唯尔

软件设计重构

【印度】Girish Suryanarayana Ganesh Samarthyam Tushar Sharma 著
袁国忠 译



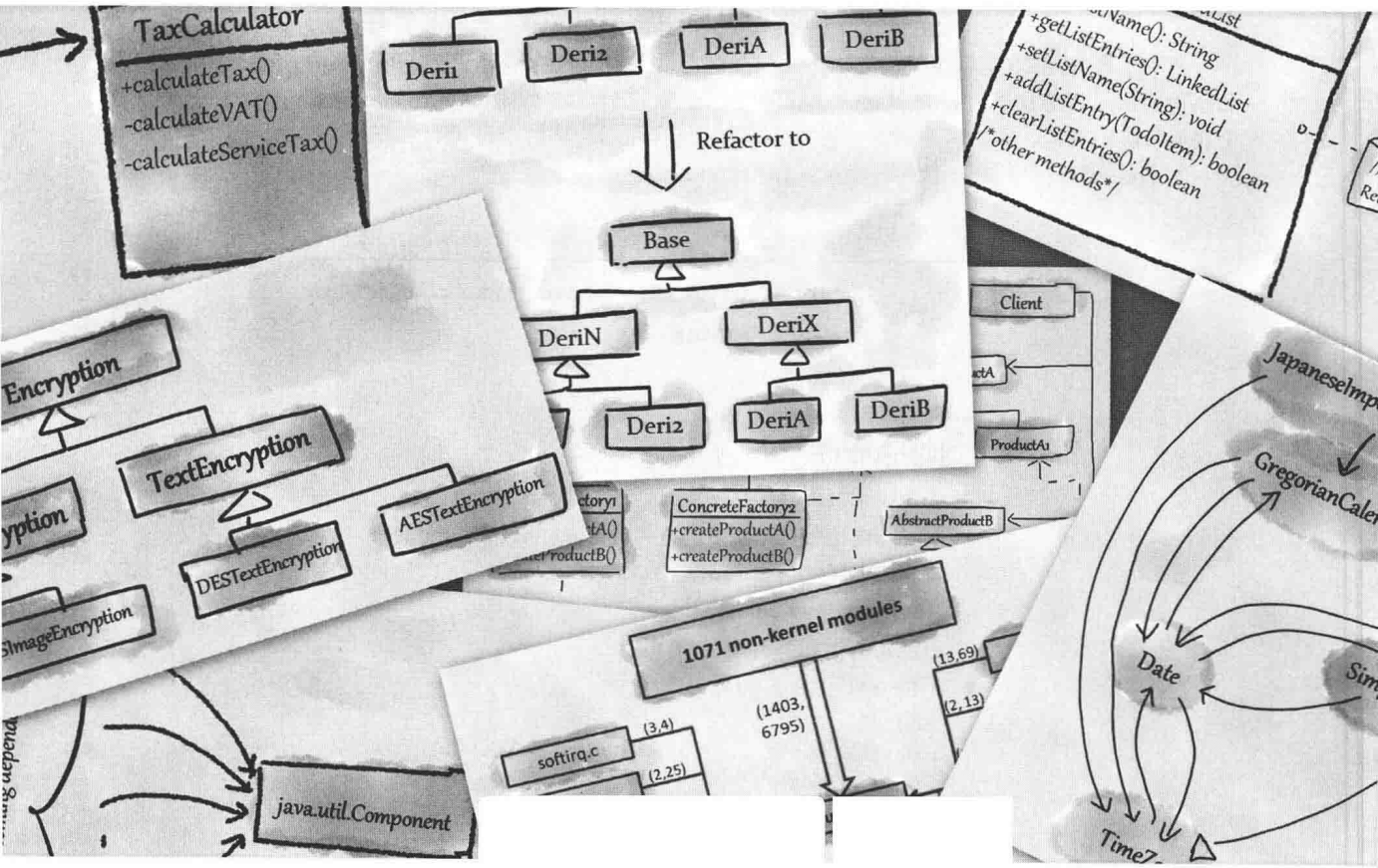
- Martin Fowler经典著作《重构》的极佳补充
- 通过发现和消除设计中的坏味来改善软件质量
- IBM软件工程首席科学家Grady Booch、Synctique公司联合创始人Stéphane Ducasse作序推荐

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

软件设计重构

【印度】Girish Suryanarayana Ganesh Samarthyam Tushar Sharma 著
袁国忠 译



人民邮电出版社
北 京

图书在版编目（CIP）数据

软件设计重构 /（印）吉里什·苏尔亚那拉亚那（Girish Suryanarayana），（印）加内什·撒马尔蒂亚姆（Ganesh Samarthyam），（印）图沙尔·夏尔马（Tushar Sharma）著；袁国忠译. -- 北京：人民邮电出版社，2016.8
（图灵程序设计丛书）
ISBN 978-7-115-43124-0

Ⅰ. ①软… Ⅱ. ①吉… ②加… ③图… ④袁… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2016)第174638号

内 容 提 要

本书主要介绍25个软件架构坏味，它们在确定设计问题时的作用以及可用的重构方法，并结合图表和示例给出了详尽说明，旨在引领读者掌握代码易读、易修改的关键，让代码具备重构能力。另外，本书将何时应该重构、重构时遇到的一些常见问题穿插在了示例讲解中。

本书适合软件架构师、软件开发工程师和项目经理。

◆ 著　[印度] Girish Suryanarayana　Ganesh Samarthyam　Tushar Sharma
译　袁国忠
责任编辑　岳新欣
执行编辑　杨　琳　魏书莉
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
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：14
字数：331千字　2016年8月第1版
印数：1-3 500册　2016年8月河北第1次印刷
著作权合同登记号　图字：01-2015-2832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

序　一

Anique Hommels 在其精彩著作 *Unbuilding Cities: Obduracy in Urban Sociotechnical Change* 中，讨论了城市改造中的技术、社会、经济和政治问题。在城市不断扩张和发展的过程中，有些部分会消失，有些部分会繁荣兴旺；为了满足市民的需求，每座城市都需要改造。有时候，需要有意地改造城市（第二次世界大战后，欧洲的很多城市满目疮痍，必须重建）；大多数时候，城市是在逐步地改造（如伦敦金丝雀码头的改造），但有时有形和无形的债务非常巨大，会让曾经盛极一时的城市陷入绝望的境地（如当今的底特律）。

软件密集型系统亦是如此。它们不断地发展、演变，时而凋谢，时而繁荣。要理解系统承受的作用力，关键是明白技术债务的概念，因为它常常能够解释系统承压的位置、方式和原因。在城市中，基础设施的修缮工作常常被推迟，改造通常是循序渐进而非大刀阔斧地进行。软件密集型系统亦是如此。系统的复杂性变幻莫测，改进工作被推迟，循序渐进的修改不充分，这些都让用户深受困扰；而负责改善系统的开发人员深受无法编写高品质代码之苦，因为他们始终在追赶用户需求变化的步伐。

令人欣慰的是，本书重点介绍了技术债务和一种切实可行的技术债务管理方式——重构。

在城市的街区，只要有一丝丝煤气泄漏，就可循着气味找到泄漏的根源。软件密集型系统亦是如此，但其中的坏味更加稀薄，任何感官都感觉不到，除非有敏锐的洞察力。我认为在阅读本书的过程中，你会遇到从未发现过的有趣而复杂的软件坏味。实际上，你手捧的是一部名副其实的坏味实战指南，犹如非常真切的 *Birds of the West Indies*，只是它探讨的是软件而非麻雀。

抽象、封装、模块化和层次结构都是软件工程最佳实践的基本要素，本书对它们也做了阐述。总之，你将发现这是一部实用的著作，读起来令人愉快、引人入胜。

Grady Booch

IBM 研究院院士兼软件工程首席科学家

序　二

在开发大型面向对象应用程序的过程中，我和合著者（O. Nierstrasz 和 S. Demeyer）开始思考面向对象再造模式。我们试图记录在设计层面处理这种系统的良好方法，并略微涉猎流程方面。经过三年的深入研究，我们撰写了 *Object-Oriented Reengineering Patterns* 一书（http://scg.unibe.ch/download/oorp/）。在此期间，我们还从事独立于语言的重构研究，因此我有机会审阅了 Martin Fowler 的著作《重构：改善既有代码的设计》。在我看来，这些书籍都具有里程碑意义，为软件工程科学奠定了基础。

担任 JOT 副主编期间，我读到了本书作者的一篇文章，文章中介绍了本书中展示的研究成果。我当时立即表示支持，并迫不及待地询问作者是否打算写一本书，要是确有这样的打算，能否让我担任审稿人。这是为什么呢？因为那时市面上还没有讨论坏味的优秀图书，而我们每天都面临着代码坏味。本书是一部优秀作品，具有划时代意义，值得每位专业人员阅读。书中介绍了与实际代码打交道的专家才具有的高深知识，这些知识是专家们多年实践经验的结晶。在开发 Pharo 运行环境和库以及 Moose 软件和数据分析平台时，我没有一天不面临设计方面的挑战。这是因为真正的设计绝不简单，即便对拥有丰富实践经验的人来说也极具挑战性。这就是有良好的抽象可用至关重要的原因所在。在记录这种经验和反思方面，模式和坏味无疑提供了有效的途径。

我喜欢阅读优秀图书并将它们推荐给学生、同事和朋友。我喜欢让人感到充实并反思实际工作的图书。这样的图书如瑰宝般少之又少，而本书堪称其中之一。

对我的著作 *Object-Oriented Reengineering Patterns* 和 Fowler 的著作《重构》来说，本书是个极佳的补充，是我渴望编写却可能写不出来的。我深信你将从中学到很多并会享受阅读的过程。感谢作者 Girish、Ganesh 和 Tushar 花时间编写了本书。我知道创作过程烦琐而漫长，但这是人类文化和知识的重要组成部分。还要感谢他们邀请我为本书作序。

Stéphane Ducasse
软件分析和软件再工程专家，Pharo、Moose 平台主要开发者，
Synctique 公司联合创始人

前　言

程序的复杂度将随程序的演化而递增，除非采取措施来保持或降低复杂度。

——雷曼的复杂度递增定律[1]

关于本书

改变很难却又不可避免，生活如此，软件亦如此。为满足用户不断增长的需求，软件必须不断演化。鉴于软件的无形特征，要管理这种不断的变化很难。其典型的结果是软件质量低下[①]和巨额技术债务。

本书介绍如何通过发现和消除设计中的坏味来改善软件质量和减少技术债务。医疗领域有种说法："好医生熟悉药品，而伟大的医生熟悉病症。"套用这种说法，"好的设计人员熟悉设计解决方案，而伟大的设计人员明白设计中的问题（坏味）、导致问题的原因，以及如何应用成熟合理的设计原则来解决问题"。本书基于此理念，旨在引领你成为更优秀的设计人员，能够识别和理解设计中存在的"病症"，并能采取妥善的"治疗"措施，进而改善软件质量，保障技术债务可控。

本书主要内容

本书介绍了 25 种结构型设计坏味及其有助于管理技术债务的重构建议。我们认为遵循软件设计原则是开发高品质软件的关键。鉴于此，我们根据 4 个基本设计原则来组织本书介绍的坏味。根据坏味违反的具体原则对其进行命名。介绍每种坏味时，我们都指出了它违反的原则，讨论了一些可能导致该坏味的因素，并列出了该坏味可能影响的重要质量指标，让读者知道设计欠下的技术债务。

描述每种坏味时，我们还提供了真实的示例以及基于行业项目经验的趣闻轶事；对于每个示例，都提供了消除其坏味的可能的重构解决方案。我们认为，要判断坏味的影响，必须考虑设计的具体情况。因此，我们也介绍了由于条件限制（如语言或平台的局限性）或出于大局考

① Capers Jones[3] 发现，糟糕的软件质量在美国每年的花费超过 1500 亿美元，在全球每年的花费超过 5000 亿美元。

虑而有意识地引入坏味的情形。

根据粒度的不同，坏味可分为架构级坏味和代码级坏味；根据性质的不同，坏味又可分为结构型坏味和行为型坏味。本书并不会介绍各种粒度和性质的坏味，而是主要探讨结构型设计级坏味。另外，本书只讨论有文献记载且常出现在实际项目中的坏味。

目标读者

软件架构师和设计人员。如果你是软件架构师或设计人员，你将是本书的最大受益者。本书将从多个方面给予你帮助。作为架构师和设计人员，你的主要职责是负责软件设计及其质量，因此比其他任何人都更想知道软件的质量需求。本书介绍的设计坏味和重构建议知识无疑将提供这方面的帮助——你可直接使用这些知识来改善设计质量。本书明确阐述了坏味对重要质量指标的影响，让你能够更深入地认识到，即便是微不足道的设计决策也可能给软件质量带来重大影响。通过真实的趣闻轶事和案例研究，你将认识到为避免坏味的出现，需要注意哪些因素。

软件开发人员。我们观察到，开发人员喜欢走捷径，这些捷径看似可以帮助完成工作，却会降低设计质量。我们还注意到，如果设计方案未对某些方面做出明确的规范，那么开发人员就会在编码期间做出一些重要的设计决策。在这种情况下，如果没有正确地应用设计原则或根本不应用设计原则，坏味就会出现。我们深信，通过阅读本书开发人员将认识到，即便是看似无关紧要的设计决策或捷径，也可能严重影响软件质量。有了这种认识，开发人员将能够发现并解决设计和代码中的问题，成为更加优秀的设计人员。

项目经理。项目经理总是担心项目不能按时完成或超支。当前，越来越多的人认识到，导致这些问题的主要原因是技术债务，因此很多项目经理都在不断寻找减少技术债务的解决方案。这样的项目经理将受益于本书。通过阅读本书，他们将对设计中可能出现的问题有更深入的认识，进而更加重视重构。

学生。本书内容与讨论软件设计的计算机科学或软件工程课程紧密相关。软件设计课程的重点之一是指导高品质软件建模和设计的基本原则。一种学习这些原则的有效方法是，先研究错误应用设计原则的后果（即坏味），再学习如何正确地应用它们（即重构）。因此我们认为，阅读本书有助于学生明白遵循良好设计原则和实践的重要性，进而能够在实际项目中提供高品质设计。

阅读前提

最好你能具备基本的面向对象编程和设计知识，并至少熟悉一种面向对象语言，如 C++、Java 或 C#；另外也最好熟悉面向对象概念，如类、抽象类、接口（它们统称为抽象或类型）、继承、委托、组合和多态。

本书结构

本书按照逻辑划分为如下三部分。

- 第 1 章和第 2 章介绍相关的背景知识。第 1 章介绍技术债务的概念、引发技术债务的因素以及技术债务对软件项目的影响；第 2 章介绍设计坏味，描述本书给设计坏味分类和命名时采用的基于原则的分类方案。
- 第 3~6 章介绍 25 种设计坏味。这 4 章组成了一个坏味目录，它们分别对应坏味违反的 4 种基本原则：抽象、封装、模块化和层次结构。介绍每种设计坏味时，都将提供说明性示例，阐述坏味对重要质量指标的影响，讨论消除示例中坏味的可能的重构解决方案。
- 第 7 章和第 8 章反思我们所做的工作并介绍如何偿还项目的技术债务。第 7 章以坏味目录中的一些坏味为例，阐述了坏味与设计的其他部分之间的相互作用。第 8 章就如何在实际项目中重构坏味以管理技术债务，提供了一些实用指南和技巧。

附录列出了本书提及的设计原则以及可帮助检测和消除设计坏味的工具，概述了本书使用的类似于 UML 的表示法，列出了有助于拓展软件设计知识的推荐读物。

考虑到 Java 是当前使用最广泛的面向对象语言，我们提供的都是 Java 编码示例。然而，重要的经验教训都是基于设计原则的，因此对 C++ 和 C# 等其他面向对象语言来说也适用。另外，本书使用了类似于 UML 的简单类图，这在附录 C 进行了解释。

本书讨论的很多示例都摘自 JDK 7.0。我们对 JDK 中的坏味进行了解读，这种解读基于通过分析源代码和注释获得的有限信息。JDK 中出现这些坏味，也许有很充分的理由。

在本书中，我们分享了大量的趣闻轶事和案例研究，它们来自于：

- 在线坏味论坛（Smells Forum）参与者分享的经历，我们建立这个论坛旨在从社区收集坏味故事；
- 会议、讲座、培训、社区论坛的参与者与我们分享的事件和故事；
- 图书、期刊、杂志和其他在线出版物；
- 本书作者担任设计评估和重构独立顾问的经历。

需要强调的是，我们无意通过这些趣闻轶事、案例研究和示例来指责任何软件企业及其设计中的坏味，而只是想让软件工程师了解软件设计中可能存在的问题。鉴于此，在本书的趣闻轶事和案例研究中，我们对一些细节做了适当的修改，以免透露企业名称、项目名和产品名等私密信息。

你不必从头到尾逐页阅读本书，可查看目录并直接跳到感兴趣的章节。

获取更多信息

要获悉本书的更详细信息，请访问 Elsevier Store 网站（http://www.store.elsevier.com/product.jsp?isbn=9780128013977）；要获取更多的信息、补充材料和资源，请访问 http://www.designsmells.com；如果你有任何建议和反馈，请通过 designsmells@gmail.com 与我们联系。

为何编写本书

软件设计天生就是一项复杂的工作，要求软件工程师对设计原则了如指掌，并具备丰富的经验和技能。你必须细致地考虑和分析需求，确保对它们有深入的认识。然而，当今的软件工程师必须在很短的时间内打造出非常复杂的软件，还要面对不断变化的需求。在这种情况下，要确保设计和整个软件的质量无疑是巨大的挑战。

鉴于此，我们决定承担起帮助软件工程师改善设计和软件质量的任务。我们最初的努力方向是开发一种方法，用于评估行业软件的设计质量（并以论文[4]的方式发表了研究成果）。我们调查了大量软件工程师，试图了解他们在项目设计期间面临的挑战。调查表明，虽然很多软件工程师掌握了重要设计原则的理论知识，但不清楚如何实际应用这些原则，结果导致其设计中出现了坏味。

我们的调查还揭示了一个重要的洞见：可利用设计坏味来帮助理解软件工程师在应用设计原则时所犯的错误。鉴于此，我们开始了一次“长途旅行”，旨在研究并了解设计中出现的各种坏味。在此期间，我们发现了散落在各种资料（包括图书、文章、论文和工具）中的坏味，并将它们记录下来。最终收集的坏味多达 530 种！

我们原本希望这种研究可帮助我们更深入地了解坏味。结果也确实如此，但此时我们面临着一项艰巨的任务——搞懂收集到的大量坏味。鉴于收集的坏味如此之多，我们决定只考虑那些实际项目中常见的结构型设计坏味。接下来的任务是，以有意义的方式对这些坏味进行分类。我们尝试了多种分类方案，但都不满意。在尝试过程中，我们逐渐意识到，要对这些坏味进行组织，以便与架构师、设计人员和开发人员分享时能够给他们提供帮助，分类方案必须与基本

设计原则相关联。这让我们获得了如下洞见：

从违反了哪些底层设计原则的角度来看待每种坏味时，我们对该坏味将有更深入的认识；更重要的是，这也自然而然地指出了消除该坏味的潜在重构方法。

基于这个颇具启发性的洞见，我们采纳了 Booch 提出的基本设计原则——抽象、封装、模块化和层次结构，将它们作为分类框架的基础。我们根据坏味主要违反了哪个设计原则对其进行分类、合并，并创建了一个包含 27 种坏味的目录。我们通过论文 [5] 发布了这项初始成果①，当一些论文审阅人建议我们将这项成果扩展为一部著作时，我们受到很大的鼓舞。在此期间，我们还开始向公司提供设计坏味方面的培训，并注意到软件从业人员认为我们的坏味目录确实很有用。受到论文审阅人和软件从业人员正面反馈的鼓舞，我们决定将这项初始成果扩展为一部著作。

本书旨在提供一个框架，帮助你明白违反设计原则是如何导致坏味的，以及如何重构坏味以管理技术债务。根据我们的经验，要成为更优秀的设计人员，关键是利用坏味来明白如何在实际工作中有效地应用设计原则。这是我们想通过本书表达的中心思想。

在研究设计坏味和编写本书的整个过程中，我们始终乐在其中，但愿你也享受阅读本书的过程！

① 这篇论文一共总结了 31 种坏味。——编者注

致　谢

在本书的写作和出版过程中，很多人伸出了援手，这里要真诚地感谢他们的帮助和支持。

首先，要感谢给予我们编写本书的灵感的人，其中包括 Grady Booch、David Parnas、Martin Fowler、Bob Martin、Stéphane Ducasse、Erich Gamma、Ward Cunningham 和 Kent Beck 等软件设计和重构领域的开路先锋。通过阅读他们的书籍、文章和博客，我们对软件设计概念有了更深入的认识。

其次，要感谢在本书编写过程中所有帮助过我们的人。这里要深深地感谢与我们分享经历和难忘故事的人，这包括我们开展的培训课程的参与者、我们建立的坏味论坛的参与者，以及在各种会议和活动中同我们交流过的人。

还要感谢本书的技术审阅人 Grady Booch、Stéphane Ducasse 和 Michael Feathers，他们批判性的分析和一丝不苟的审阅极大地改善了本书的质量。感谢 Venkat Subramaniam 提供宝贵的审阅评论和建议。另外，感谢我们的朋友 Vishal Biyani、Nandini Rajagopalan 和 Sricharan Pamudurthi，抽出宝贵的时间阅读书稿，并提供深思熟虑的反馈意见。

我们非常感激 Grady Booch 和 Stéphane Duccase 对本书的价值深信不疑并欣然作序。

如果没有 Morgan Kaufmann/Elsevier 整个团队的大力帮助和支持，本书就不可能付梓。特别感谢 Todd Green 对我们想法的信任，并在整个成书过程中始终如一地给予大力支持。还要真诚地感谢 Lindsay Lawrence 在整个出版过程中提供支持。另外，感谢 Punithavathy Govindaradjane 及其团队在印制过程中提供帮助。最后，感谢设计师 Mark Rogers 出色的设计工作。

最后，感谢在本书写作期间给予我们帮助的人。这里要特别感谢 K. Ravikanth 和 PVR Murthy，他们颇具见地的讨论让本书的质量更上一层楼。Girish 和 Tushar 要感谢印度西门子研究和技术中心的 Leny Thangiah、Rohit Karanth、Mukul Saxena 和 Ramesh Vishwanathan，以及印度西门子技术和服务有限公司的 Raghu Nambiar 和 Gerd Hoefner，感谢他们始终如一的支持和鼓励。Ganesh 要感谢印度 ZineMind 技术有限公司的 Ajith Narayan 和 Hari Krishnan 的支持。

多少个周末和假日，我们都在为编写本书而奋战，没能陪孩子玩耍或外出购物。感谢家人在本书漫长的编写过程中始终如一的关爱、耐心和支持。

版权声明

Refactoring for Software Design Smells: Managing Technical Debt, 1st Edition
Girish Suryanarayana, Ganesh Samarthyam, Tushar Sharma
ISBN: 978-0-12-801397-7.

目　录

第 1 章

技术债务

要开启设计坏味重构之旅，必须回答的第一个也是最基本的问题是：何为设计坏味以及为何必须重构设计以消除坏味？

在其著作《人月神话》[6] 中，Fred Brooks 描绘了软件的固有特征（即复杂性、一致性、可修改性和不可见性）导致软件设计是个“本质性”难题。为破解这个难题，必须遵循良好的设计实践。一种这样的实践是，软件设计人员必须找出并消除设计决策可能导致的设计坏味，这正是本书要探讨的主题。

那么，何为设计坏味呢？

> 设计坏味是特定的设计结构，它们违反了基本设计原则，给设计质量带来负面影响。

换句话说，设计坏味表明设计结构中存在潜在的问题。医疗领域为理解坏味提供了很好的类比：可将病人的症状比作“坏味”，将疾病比作“设计问题”。

还可将这种类比延伸到诊断过程。例如，医生分析症状，确定引发症状的疾病，再提出治疗方案。类似地，设计人员必须分析设计中发现的坏味，判断导致坏味的问题，再确定如何重构以解决问题。

介绍了设计坏味后，让我们再说说必须重构[①]设计以消除坏味的原因。

问题的答案在于*技术债务*——一个在最近几年被软件开发界广泛关注的术语。软件开发人员必须对技术债务有大致的认识，这样才能明白其日常设计决策的长远影响。鉴于此，本章余下的篇幅将专注于讨论技术债务。

① 本书中，我们使用重构（refactoring）这个词来表示“行为保留式程序转换”（behavior preserving program transformations）[13]。