TURING
图灵原创

大话重构

范 钢 ◎著

明白真正的**专业级**软件开发是如何进行的
明白真正的**重构具体**是一步步怎么做的

高效可行的重构七步。面对实际重构，不会卡壳。
超越代码级重构，渗透系统与设计的各个层面。

第一次真正理解那些最熟悉的陌生技术，
惊讶于它们各就各位榫卯成强韧的整体。

人民邮电出版社
POSTS & TELECOM PRESS

原创经典

TURING
图灵原创

大话重构

范钢 ◎著

人民邮电出版社
北京

图书在版编目（CIP）数据

大话重构 / 范钢著. -- 北京 : 人民邮电出版社, 2014.5
（图灵原创）
ISBN 978-7-115-34885-2

Ⅰ. ①大… Ⅱ. ①范… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2014)第041929号

内 容 提 要

本书的价值在于两点：

一、让你明白真正的专业级软件开发是如何进行的；

二、让你明白真正的重构具体是一步步怎么做的。

本书运用大量源于实践的示例，从编码、设计、组织、架构、测试、评估、应对需求变更等方面，深入而多角度地讲述了我们应该如何重构，建设性地提出了高效可行的重构七步。

读完本书，实践重构不再卡壳，需求变更不再纠结。全面领悟重构之美，遗留系统不再是梦魇，自动化测试原来可以这样做。

本书帮助程序员告别劣质代码步入精妙设计，让遗留系统的维护者逐步改善原有设计，指导重构实践者走出困惑步步坚定。同时，也为管理者加强软件质量的管理与监督，提供了好的方法与思路。

◆ 著　　范　钢
策划编辑　陈　冰
责任编辑　朱　巍
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
三河市海波印务有限公司印刷
◆ 开本：800×1000　1/16
印张：16.75
字数：314千字　　2014年5月第1版
印数：1－5 000册　　2014年5月河北第1次印刷

定价：45.00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：京崇工商广字第0021号

编辑的话

当你面对一本书，你最想知道的应该是这本书究竟可以给你带来什么。

对于这本书，它最大的价值在于两点：

一、让你明白真正的专业级软件开发是如何进行的。

打个比方，你是摄影新手，拽得很，但无论你怎么摆弄你的数码单反，就是照不出专业摄影师似乎轻轻松松就搞出来的东西。他们使用的东西，你没见过，瞧不出是干嘛用的，自然也不知道为什么要用它。他们所做的至关重要的措施，你也不知道有什么意义。所有这些你看不明白的过程，造就了最终一目了然的差距。

你在软件开发上投入了不少时间，攻克过不少难关，解决了一堆问题，但做出来的东西、写出来的代码、设计出的架构还是透着或淡或浓的业余味。原因在于有些重要和基本的东西你还不知道。在得到专业之前，你必须见多识广。

本书带你领教立竿见影的专业级软件开发的过程。你听过或没听过的那些术语和概念，多少明白或完全不明白的技术和方法，知道却没用过或完全不知道的工具和软件，这些之前各玩各的的独立散碎，在这本书中被榫卯成一个强韧的整体。你会明了它们中每一个的作用，应被安插到的位置，并见识它们各就各位时所发挥出的能量。头脑从未有过的清醒，你理解了以前所不理解的。

二、让你明白真正的重构具体是一步步怎么做的。

重构的书，市面上有一些了。但这些书瞄准的均是代码级的重构。把这行代码改成这样会好些，把这段程序改成这样会更合适些。遇到这样的代码，你要这么改；面对那样的情况，你得那样处理。一切都没有问题，直到你面对实际中的重构。

当你在实际中真正要去做一次重构时，你发现你面对的不是一行或一段代码，而是整个软件。你看到这里有好些不乖的代码，那里也有数百头粗野的程序。所有这些显然都需要重构，但应该先改哪处呢，要从哪儿开始呢？在看了那么多重构书后，你发现你竟然在重构的第一步就卡壳了。因为没有书告诉你第一步该做什么。

本书明确地告诉你第一步要做什么，那就是分解大函数，这是软件退化的重灾区，也是重构过程的不二起点。本书不仅告诉你第一步该做什么，更将看似漫无头绪繁复冗长的软件重构清楚明白地划分成七个步骤，告诉你整个重构七步中的每一步都该做些什么，并详细且

通俗易懂地讲解了具体你该如何去做。

本书所讲解的重构远远超越了代码级，充分渗透到软件系统与设计的各个层面，涵盖从代码、函数、类与对象，直至设计模式、分层架构、领域模型、软件测试的整个过程。

在阅读本书的过程中，你经常会为读到精彩之处而喜悦（头脑豁然开朗），而读完本书后，你会成竹在胸。以往在精神和肉体上折磨你很久的客户需求变更，或是因前期考虑不周而引起的设计变化，都不会再让你感到纠结，因为你可以通过重构润物细无声地容纳这些变化，而且你清楚如何去做。

再来说说这本书的一些趣事。本书的作者范钢对软件开发有着无比的热忱，做事极为认真，这份书稿是我收到过的交稿中错别字最少的一部，在进行文稿加工时，常常连续多页都没有一处加工痕迹。本书语言通俗易懂，逻辑清晰，读起来很轻松。唯一的缺点是个别之处有些啰嗦，这些啰嗦之处已被我剔除，以获得更好的阅读体验。

做为本书的策划编辑，我是带着喜悦的心情读完本书的，相信你也会的。

陈冰

2014年2月6日

前　言

我常常感到幸运，我们现在所处的是一个令人振奋的时代，我们进入了软件工业时代。在这个时代里，我们进行软件开发已经不再是一个一个的小作坊，我们在进行着集团化的大规模开发。我们开发的软件不再是为某个车间、某个工序设计的辅助工具，它从某个单位走向整个集团，走向整个行业，甚至整个社会，发挥着越来越重要的作用。一套软件所起到的作用与影响有多大，已经远远超越了所有人的想象，成为一个地区、一个社会，乃至整个国家不可或缺的组成部分。慢慢地，人们已经难以想象没有某某软件或系统的生活和工作会是怎样的。这就是软件工业时代的重要特征。

然而，在这个令人振奋的软件工业时代，处于时代中心的各大软件企业却令人沮丧。在软件规模越来越庞大，软件结构越来越复杂的同时，却是软件质量越来越低下，软件维护变得越来越困难，以至于每个小小的变更都变得需要伤筋动骨。研发人员为此手足无措，测试人员成为唯一的救星，每个小小的变更都需要付出巨大代价进行测试。软件企业在这样一种恶性循环中苦苦支撑。毫无疑问，这也成为这个令人振奋的时代的另一个特征。

是的，面对软件工业时代我们并没有做好准备。过去，一套软件的生命周期不过 2~3 年时间，随着软件需求的变化，我们总是选择将软件推倒了重新开发，但是现在这样的情况在发生着改变。随着软件规模的扩大，软件数据的积累，软件影响力的提升，我们，以及我们的客户，都真切地感受到，要推倒一套软件重新开发，将变得越来越困难且不切实际。这样的结果就是，我们的软件将不停地修改、维护、再修改、再维护……直到永远。这是一件多么痛苦的事情！

一套软件，当它第一次被开发出来的时候，一切都十分清晰：清晰的业务需求、清晰的设计思路、清晰的程序代码。但经历了几次需求变更与维护以后，一切就变得不那么清晰了。业务需求文档变得模糊不清，设计思路已经跟不上变更的脚步，程序代码则随着业务逻辑的复杂而臃肿不堪。程序员开始读不懂代码，软件开发工作变得不再是一种乐趣。

随着时间的推移，软件经过数年、数十次的变更与维护，情况变得越来越糟。最初的程序员已经不愿再看到自己的代码而选择离去。他的继任者们变得更加无所适从，由于看不懂程序，代码的每一次修改如同在走钢丝。测试人员变成了唯一的希望，开发人员的每一次修改都意味着测试人员需要把所有程序测试一遍。继任者们开始质问最初的设计者们，程序是

怎么设计的。如果此时恰巧又有什么新技术出现，就会更显得原有系统的破旧与不堪。

相信这就是软件工业时代的所有企业都不得不面对的尴尬境地。难道真的是我们最初的设计错了吗？是的，我们都这样质问过我们自己，因此我们开始尝试在软件设计之初投入更多的精力。我们开始投入更多的时间作需求调研，考虑更多可能的需求变化，做更多的接口，实现更加灵活但复杂的设计。然后呢，解决了我们的问题了吗？显然是没有。需求并没有像我们想象的那样发生变更：我们之前认为可能发生的变更并没有发生，使我们为之做出的设计变成了摆设；我们之前没有考虑到的变更却发生了，让我们猝不及防，软件质量开始下降，我们被打回了原形。难道真的是无药可解了吗？在我看来，如果我们没有看明白软件开发的规律与特点，那么我们永远找不到那味向往已久的解药。现在，让我们真正静下心来分析分析软件开发的规律与特点。

软件，特别是管理软件，其实质是对真实世界的模拟。我们通过对真实世界的模拟，实现计算机的信息化管理，来提高我们的生产效率。然而，真实的世界复杂而多变的，我们认识世界却是一个由简单到复杂循序渐进的过程，这是一个我们无法改变的客观规律。因此，毫无疑问，遵循着这样一个客观规律，我们的软件开发过程必然也是一个由简单到复杂循序渐进的过程。

最初，我们开发的是一个对真实世界最简单、最主要、最核心部分的模拟。因为简单，我们的思路变得清晰而明了。但是，我们的软件不能永远只是模拟那些最简单、最主要、最核心的部分。我们的客户在使用软件的过程中，如果遇到那些不那么简单、不那么主要、不那么核心的情况时，我们的软件就无法处理了，这是客户所不能接受的。因此，当软件的第一个版本交付客户以后，客户的需求就开始变更。

客户的需求永远不会脱离真实世界，也就是说，真实世界不存在的事物、现象、关系永远都不可能出现在软件需求中。但是，真实世界的事物、规则与联系并不是那么地简单与清晰的。随着我们的软件对它模拟得越来越细致，程序的业务逻辑开始变得不再清晰而易于理解，这就是软件质量下降最关键的内因。

任何一个软件的设计，总是与软件的复杂度有密切的关系。举例来说吧，客户资料是许多系统都必须要记录的重要信息。起初，我们程序简单，客户资料只记录了一些简单的信息，如客户名称、地址、电话等等，但随着程序复杂度的增加，客户资料开始变得复杂。比如，起初“地址”字段就仅仅需要一个字符串就可以了，但随着需求的变更，它开始有了省份、城市、地区、街道等信息。随后还会有邮政编码、所属社区、派出所等信息。起初增加一个两个字段时我们还可以在“客户信息表”里凑合一下，但后来我们必须要及时调整我们的设

计，将地址提取出来单独形成一个“地址信息表”。如果不及时予以调整，“客户信息表”将越来越臃肿，由 10 来个字段，变成 50 个、80 个、上 100 个……

信息表尚且如此，业务操作更是如此。起初的业务操作是如此地简单而明了，以至于我们不需要花费太多的类就可以将它们描述清楚。比如开票操作，最初的需求就是将已开具的票据信息读取出来，保存，并统计出本月开票量及金额。这样一个简单操作，设计成一个简单的“开票业务类”合情合理。但随后的业务逻辑变得越来越复杂，我们要检查客户是否存在、开票人是否有权限、票据是否还有库存，等等。起初的开票方式只有一种，但随着非正常开票的加入，开票方式不再单一，而统计方式也随之变化……随着业务的不断增加，软件代码的规模也在发生着质的变化。如果这时我们不及时调整我们的设计，而是将所有的程序都硬塞进“开票业务类”，那么程序质量必然会退化。“开票业务类”由原有的数十行，激增到数百行，甚至上千行。这时的代码将难于阅读，维护它将变成一种痛苦，毫无乐趣可言。

面对这样的状况，我们应当怎样走出困境呢？毫无疑问，就是重构。开票前的校验真的属于“开票业务类”吗？它们是否应当被提取出来，解耦成一个一个的校验类。正常开票与非正常开票真的应该写在一起吗？是否我们应当把“开票业务类”抽象成接口，以及正常开票与非正常开票的实现类。这就是我给大家的良方：当软件因为需求变更而开始渐渐退化时，运用软件重构改善我们的结构，使之重新适应软件需求的变化。

范钢

2014 年元旦

目　录

第一部分　基础篇

第二部分　实践篇

第三部分 进阶篇

第一部分　基础篇

第1章　重构：改变既有代码的一剂良药

前面我们提到了，面对软件工业时代的到来，我们的软件企业陷入了一种更深的迷茫之中，一种“后有追兵，前有悬崖，进退两难”的境地。

后有追兵：面对维护了数十年之久的大型遗留系统，我们到底改还是不改？不改，面对越来越多的需求变更，我们维护的成本越来越高，变更变得越来越困难；面对不断涌现的新技术，我们的系统显得越来越丑陋与落后；面对越来越多的竞争者，我们面临着被市场淘汰的风险。

前有悬崖：原本运行得好好的软件系统，凑合一下还可以运行几年。一不小心改出问题了，企业立马就歇菜儿了，面对大量的用户投诉，企业四处救火，竞争对手趁火打劫，这是任何软件企业都不能承受的巨大风险。

难道真的“鱼与熊掌不能兼得”吗？真的没有一种方法，能够既保证我们的系统可以技术改造，又能有效地避免改造过程的风险吗？有，那就是系统重构。

1.1　什么是系统重构

提到重构，许多人都讳莫如深、敬而远之。那么什么是系统重构呢？大家可能有很多不同的看法：

1. 系统重构是那些系统架构师、技术大牛玩的高端玩意儿，咱普通屌丝不懂，跟咱没啥关系。
2. 系统重构就是改代码，大改特改那种，整个重来一遍那种，这个比较邪恶，比较容易改出事儿，还是不要轻易尝试为妙。
3. 我知道系统重构，也知道它能改善遗留系统，但我还是不敢轻易尝试，因为改出问题来怎么办，还是算了吧。

然而我认为，现在我们对系统重构有太多的误解，以至于我们还不怎么了解它，就已经将它拒之门外。什么是系统重构？它是一套严谨而安全的过程方法，它通过一系列行之有效

的方法与措施，保证软件在优化的同时，不会引入新的 BUG，保证软件改造的质量。这一点在我后面一步一步的拆解中，你可以慢慢体会到。

我们先看看系统重构的概念。系统重构，就是在不改变软件的外部行为的基础上，改变软件内部的结构，使其更加易于阅读、易于维护和易于变更[①]。

系统重构中一个非常关键的前提就是“不改变软件的外部行为”，这个前提非常重要，它保证了我们在改造原有系统的同时，不会为原系统带来新的 BUG，以确保改造的安全。这里，什么是“为原系统带来新的 BUG”？我们必须为其做出一个严格的定义，那就是“改变了软件原有的外部行为”。也许你对此有些不太赞同，改变了软件原有的外部行为，怎么就能武断地认为，是为原系统带来了新 BUG 呢？为此我们来举个例吧。

假如一个系统的报表查询功能，原来在表格里的返回结果中，日期是这样表示的：“2013-2-18”。经过系统改造以后变成这样了：“2013-2-18 00:00:00”。这是 BUG 吗？作为开发人员你可能认为这算什么 BUG，但作为客户那就是 BUG，因为它让表格变得难看，使用不再方便了。系统重构，对于客户来说应当是完全透明的。我们为之做了很多工作，而他们应当完全感觉不到它的存在。如果我们的重构做到了这一点，那么我们的重构就必然是安全的、可靠的、没有风险的。

更广泛一些来说，如果我们打开软件内部，保证系统中的每个接口与改造前是等价的，也就是说，其输入输出在改造前后都是一致的。当我们的每个改造都是这样进行的，则必然不会为系统带来新的 BUG。这就是我们进行改造的保险索，它也是我现在所说的重构与以往那种拿着代码一阵瞎改的根本区别。

总而言之，系统重构不是那种冒着极大风险进行的代码修改，而是必须保证修改前后输入输出的一致，这就是我们说的“不改变外部行为”。为此，贯穿整个重构过程的是不断地测试。起初这种测试是手工测试，随后逐渐转变为自动化测试。每修改一点点就进行一个测试，再修改一点点。测试，就是系统重构的保险索。

1.2　在保险索上走钢丝

当我们开始系统重构的时候，不是着手去修改代码，而是首先建立测试机制。不论什么程序，只要是被我们修改了，理论上就可能引入 BUG，因此我们就必须要进行测试。既然

① Refactoring: a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. ——引自*Refactoring: Improving the Design of Existing Code*一书

是测试就必须要有一个正确与否的评判标准。以往的测试，其评判的标准就是是否满足业务需求。因此，测试人员往往总是拿着需求文档测试系统。

与以往的代码修改不同，重构没有引入任何新的需求，系统原来什么功能，重构以后还是这些功能。因此，重构的测试标准就只有一个，就是与之前的功能完全保持一致，仅此而已。

然而在许多经典的重构书籍中，大师们总是建议我们首先建立自动化测试机制，这已经被我在无数次实践中证明是不现实的。要知道我们现在重构的是一个遗留系统，它往往设计混乱，接口不清晰，程序相互依赖。所有这些都使得最初的自动化测试变得非常困难而不切实际。

因此，从一开始就实现自动化测试是不切实际的，我们所能采用的还是手工测试。在重构之前首先将系统启动起来，执行相应的功能，得到各自相应的输出。然后开始重构，每次重构对代码的修改量不要太大，花费的时间不要太长。因为，修改量太大，花费时间太长，一旦测试不通过，很难定位错误的原因。在这种情况下，我们只能还原代码，将此次修改的工作完全作废。如果此次修改已经花费了你数天甚至数月的时间，这样的损失就实在太大了。

每做一次重构，修改一点儿代码，然后提交，对系统进行一次测试。如果系统依然保持与以往一样的功能，重构成功，继续进行下一次重构。如果不是，你不得不还原回来重新重构。频繁地测试着实让你挺烦，但却有效减少了重构失败带给你的损失。一个折中的办法就是，平时频繁测试的时候，测试项目少一些，只测试主要项目，定期再进行全面的测试。录制 QTP[①]脚本也是一个不错的方式，它虽然有诸多问题，但却可以在系统重构初期有效地建立自动化测试，系统级别的自动化测试。随着系统重构的不断深入，我们的程序开始在改善，耦合变得越来越松散，程序变得越来越内聚，接口变得越来越清晰。这时候，当自动化测试条件成熟时，我们才可以逐渐开始自动化测试，而这种自动化测试才是建立在代码级别的真正的自动化测试。

一旦某个修改测试不通过，则还原回来。这种一次一小步的修改模式，我们形象地称之为“小步快跑”。在测试集成工具的不断监督下一点一点地修改程序，是系统重构异于以往的另外一个特点。通过小步快跑可以使我们在重构的过程中，以最快的速度发现修改的问题，将因修改错误带来的损失减到最小，毕竟是人都不可能避免犯错。

① QTP，Quicktest Professional的简称，是一种自动测试工具。使用QTP的目的是想用它来执行重复的手动测试，主要是用于回归测试和测试同一软件的新版本。

1.3　大布局与小步快跑

以往我们在重新设计一个系统时，总是喜欢大布局。全面地整理系统需求，全面地分析系统功能，再全面地设计系统、开发、测试。这样一个过程往往会持续数月，花费大量的工作量。但是，不到最后设计出来，谁都不知道会不会存在问题。这就是“大布局”的弊病。

正因为如此，软件大师在讲述系统重构时总是强调，系统重构应当避免大设计，而尽量采用一个一个连续不断的小设计。这就是我们所说的“小步快跑”的设计模式。

小步快跑体现出了敏捷软件开发的特点——简单与快速反馈。不要想得太多了，活在今天的格子里，你永远不可能预知今后会发生什么。所以，做今天的设计，解决今天的问题，完成今天的重构，让明天见鬼去吧。要知道，简单对于我们是多么地重要。当我们的大脑开始思考各种复杂的问题时，就开始充血，然后就是梦游，最后的结果就是顾此失彼。既然如此，我们为何不选择一种更加简单的生活方式呢？

非常遗憾的是，很多时候我们不能选择这种简单的生活方式，因为我们害怕明天，害怕明天会出现那些我们处理不了的业务场景，因此我们开始过度设计。我不是批判我们不应当有预见，预见性设计与过度设计往往就是一线之隔。有限的预见是可以的，但不要想得过于遥远。当明天真的需求变更了，我们现在的设计不能满足了，怎么办呢？没什么大不了的，因为我们有重构。通过安全而平稳的重构方法先重构我们的系统，使之可以应付那个需求，然后再添加代码，实现新需求。这个过程被称为“两顶帽子”：一顶是只重构而不新增功能，一顶是增加新的功能实现新需求。正因为如此，我们在设计时思考当下就可以了。

另外一个问题就是及时反馈，落实到此地就是及时测试。只有测试通过了，此次重构才算成功，我们才能继续往下重构，否则我们必须还原。从这里我们不难看出，重构的周期是多么地重要。在我以往的重构工作中，一次重构的周期也就在 10 分钟到 1 小时。重构的周期越长，说明你考虑的问题越复杂，最终出错的概率也就越大。所以，我们一定要习惯“小步快跑”的工作方式，让自己只活在当下。

说了那么多重构，相信一定有一个疑问开始在你脑中萦绕。既然系统重构对于客户来说是透明的，客户完全感觉不到它的存在，毫无疑问，它对于客户来说就是毫无价值的。这下疑问就来了：既然重构对于客户来说毫无价值，我们做它还有什么意义呢？要说明白这个问题，我们需要首先谈一谈软件修改的四种动机。