后来，这个项目重新启动，几乎从头开始编写整个系统，Kent Beck 受邀做了顾问。他做了几件迥异以往的事，其中最重要的一件就是坚持以持续不断的重构行为来整理代码。这个团队效能的提升，以及重构在其中扮演的角色，启发了我撰写本书的第 1 版，如此一来我就能够把 Kent 和其他一些人已经学会的“以重构方式改进软件质量”的知识，传播给所有读者。

自本书第 1 版问世至今，读者的反馈甚佳，重构的理念已经被广泛接纳，成为编程的词汇表中不可或缺的部分。然而，对于一本与编程相关的书而言，18 年已经太漫长，因此我感到，是时候回头重新修订这本书了。我几乎重写了全书的每一页，但从其内涵而言，整本书又几乎没有改变。重构的精髓仍然一如既往，大部分关键的重构手法也大体不变。我希望这次修订能帮助更多的读者学会如何有效地进行重构。

什么是重构

所谓重构（refactoring）是这样一个过程：在不改变代码外在行为的前提下，对代码做出修改，以改进程序的内部结构。重构是一种经千锤百炼形成的有条不紊的程序整理方法，可以最大限度地减小整理过程中引入错误的概率。本质上说，重构就是在代码写好之后改进它的设计。

“在代码写好之后改进它的设计”这种说法有点儿奇怪。在软件开发的大部分历史时期，大部分人相信应该先设计而后编码：首先得有一个良好的设计，然后才能开始编码。但是，随着时间流逝，人们不断修改代码，于是根据原先设计所得的系统，整体结构逐渐衰弱。代码质量慢慢沉沦，编码工作从严谨的工程堕落为胡砍乱劈的随性行为。

“重构”正好与此相反。哪怕手上有一个糟糕的设计，甚至是一堆混乱的代码，我们也可以借由重构将它加工成设计良好的代码。重构的每个步骤都很简单，甚至显得有些过于简单：只需要把某个字段从一个类移到另一个类，把某些代码从一个函数拉出来构成另一个函数，或是在继承体系中把某些代码推上推下就行了。但是，聚沙成塔，这些小小的修改累积起来就可以根本改善设计质量。这和一般常见的“软件会慢慢腐烂”的观点恰恰相反。

有了重构以后，工作的平衡点开始发生变化。我发现设计不是在一开始完成的，而是在整个开发过程中逐渐浮现出来。在系统构筑过程中，我学会了如何不断改进设计。这个“构筑－设计”的反复互动，可以让一个程序在开发过程中持续保有良好的设计。