前言

80 多年来，人们一直在为电子计算机编写程序，但令人惊讶的是，关于如何设计这些程序或什么是好的程序的讨论却很少。关于软件开发过程（如敏捷开发）和开发工具（如调试器、版本控制系统和测试覆盖工具）已经有了相当多的讨论。针对编程技术的分析也已经相当广泛，如面向对象编程和函数式编程，以及设计模式和算法。所有这些讨论都是有价值的，但是软件设计的核心问题在很大程度上仍然没有触及。David Parnas 的经典论文“关于将系统分解成模块的标准”发表于 1971 年，但是在随后的 45 年里，软件设计的技术水平并没有超过这篇论文。

计算机科学中最基本的问题是 *问题分解*：如何将一个复杂的问题分解为可以独立解决的部分。问题分解是程序员每天都要面对的首要设计任务，但是，除了这本书里描述的工作之外，我还没有在任何一所大学里找到一门以问题分解为中心的课程。我们讲授 `for` 循环和面向对象的程序设计，而不是软件设计。

此外，不同的程序员在产出质量和效率上存在巨大差异，但是我们几乎没有尝试去了解什么能使最好的程序员变得更好，也没有在我们的课堂上讲授这些技能。我曾与几位我认为是优秀的程序员的人进行过交谈，但是他们中的大多数人都难以阐明赋予他们优势的特定技术。许多人认为软件设计技能是天生的天赋，无法讲授。但是，有相当多的科学证据表明，许多领域的杰出表现更多地与高质量的实践有关，而不是与先天能力有关（例如，参阅 Geoff Colvin 的《*哪来的天才？练习中的平凡与伟大*》）。

多年来，这些问题使我感到困惑和沮丧。我想知道软件设计是否是可以被讲授的，并且我假设设计技巧是区分优秀程序员和普通程序员的原因。我最终决定，回答这些问题的唯一方法是尝试讲授软件设计课程。结果就是斯坦福大学的 CS 190 课程。在这个课程中，我提出了一套软件设计原则。然后，学生将通过一系列项目来学习和实践这些原则。该课程的授课方式类似于传统的英语写作课。在英语课堂上，学生使用迭代的过程，在过程中编写草稿、获取反馈、然后重写以进行改进。在 CS 190 中，学生从头开始开发大量软件。然后，我们将进行大量的代码审查以识别设计问题，然后学生修改其项目以解决这些问题。这使学生可以了解如何通过应用设计原则来改进其代码。

到现在，我已经教过几次该软件设计课程，并且本书是基于该课程中出现的设计原则编写的。这些原则立足于比较高的层级，类似于哲学概念（比如“通过定义来规避错误”），因此学生很难以抽象的方式理解这些思想。通过编写代码、犯错误、然后查看他们的错误以及后续的修正与这些原则之间的关系，学生将学得最好。

在这一点上，您可能会想知道：是什么让我认为我知道有关软件设计的所有答案？老实说，我并不知道。当我学会编程时，也没有关于软件设计的课程，而且从来没有导师来教我设计原则。在我学习编程时，代码审查还不存在。我对软件设计的想法来自于编写和阅读代码的个人经验。在我的职业生涯中，我已经用多种语言编写了大约 25 万行代码。我曾在一些团队中工作过，这些团队从零开始创建了三个操作系统、多个文件和存储系统、基础架构工具（例如调试器、构建系统和图像界面工具包）、一种脚本语言以及用于文本、图形、演示文稿和集成电路的交互式编辑器。一路上，我亲身经历了大型系统的问题，并尝试了各种设计技术。另外，我阅读了很多其他人编写的代码，这使我接触到了很多方法，无论是好是坏。

从所有这些经验中，我尝试提取通用线索，包括需要避免的错误和使用的技巧。本书反映了我的经验：这里描述的每个问题都是我亲身经历的，每种建议的技术都是我在自己的编码中成功使用过的。

我不希望这本书成为软件设计的定论。我敢肯定，我错过了一些有价值的技术，从长远来看，我的一些建议可能会变成坏主意。但是，我希望本书能开启有关软件设计的对话。将本书中的想法与您自己的经验进行比较，以确定此处介绍的方法是否确实降低了您的软件复杂性。这是一本呈现我个人观点的书，所以有些读者会不同意我的一些建议。如果您不同意，请尝试理解原因在哪。我有兴趣了解对您有用的东西、不起作用的东西以及您可能对软件设计有的任何其他想法。我希望随后的对话将增进我们对软件设计的集体理解。

与我交流这本书的最好方法是将电子邮件发送到以下地址：

software-design-book@googlegroups.com

我有兴趣听到有关本书的反馈，例如缺陷或改进建议，以及与软件设计相关的通用思想和经验。我对可以在本书未来版本中使用的好的示例特别感兴趣。最好的示例能说明重要的设计原则，并且足够简单，可以在一两个段落中进行解释。如果您想在电子邮件地址上看到其他人在说什么并参与讨论，可以加入 Google Group `software-design-book`。

如果出于某种原因 Google Group `software-design-book` 将来会消失，请在互联网上搜索我的主页，它将包含有关如何与这本书进行交流的更新说明。请不要将与图书相关的电子邮件发送到我的个人电子邮件地址。

我建议您使用本书建议时持保留态度。总体目标是降低复杂性，这比您在此处阅读的任何特定原则或想法更为重要。如果您尝试从本书中获得一个想法并发现它实际上并没有降低复杂性，那么您就不必继续使用它（但是，请让我知道您的经历，不管方法有效还是无效，我都想获得相关的反馈意见）。

许多人提出了批评或建议，以提高本书的质量。以下人员对本书的各种草稿提供了有用的意见：Abutalib Aghayev，Jeff Dean，Will Duquette，Sanjay Ghemawat，John Hartman，Brian Kernighan，James Koppel，Amy Ousterhout，Kay Ousterhout，Rob Pike，Partha Ranganathan，Daniel Rey，Keith Schwartz 和 Alex Snaps。Christos Kozyrakis 为类和接口建议了术语“深”和“浅”，代替了之前有点模糊的术语“厚”和“薄”。我很感激 CS 190 中的学生，阅读他们的代码并与他们讨论的过程有助于明确我对设计的想法。