总结

Summary of Design Principles 设计原则摘要

Here are the most important software design principles discussed in this book:

> 这是本书中讨论的最重要的软件设计原则：

1. Complexity is incremental: you have to sweat the small stuff (see p. 11).
2. Working code isn’t enough (see p. 14).
3. Make continual small investments to improve system design (see p. 15).
4. Modules should be deep (see p. 22)
5. Interfaces should be designed to make the most common usage as simple as possible (see p. 27).
6. It’s more important for a module to have a simple interface than a simple implementation (see pp. 55, 71).
7. General-purpose modules are deeper (see p. 39).
8. Separate general-purpose and special-purpose code (see p. 62).
9. Different layers should have different abstractions (see p. 45).
10. Pull complexity downward (see p. 55).
11. Define errors (and special cases) out of existence (see p. 79).
12. Design it twice (see p. 91).
13. Comments should describe things that are not obvious from the code (see p. 101).
14. Software should be designed for ease of reading, not ease of writing (see p. 149).
15. The increments of software development should be abstractions, not features (see p. 154).

> 1. 复杂性是逐步增加的：您必须流汗一些小东西（请参阅第 11 页）。
> 2. 工作代码还不够（请参阅第 14 页）。
> 3. 持续进行少量投资以改善系统设计（请参阅第 15 页）。
> 4. 模块应较深（请参见第 22 页）
> 5. 接口的设计应尽可能简化最常见的用法（请参阅第 27 页）。
> 6. 一个模块具有一个简单的接口比一个简单的实现更重要（请参阅第 55、71 页）。
> 7. 通用模块更深入（请参阅第 39 页）。
> 8. 通用和专用代码分开（请参见第 62 页）。
> 9. 不同的层应具有不同的抽象（请参见第 45 页）。
> 10. 降低复杂度（请参阅第 55 页）。
> 11. 定义不存在的错误（和特殊情况）（请参阅第 79 页）。
> 12. 设计两次（请参阅第 91 页）。
> 13. 注释应描述代码中不明显的内容（请参见第 101 页）。
> 14. 软件的设计应易于阅读而不是易于编写（请参见第 149 页）。
> 15. 软件开发的增量应该是抽象而不是功能（请参见第 154 页）。

Summary of Red Flags 红旗摘要

Here are a few of of the most important red flags discussed in this book. The presence of any of these symptoms in a system suggests that there is a problem with the system’s design:

> 这是本书中讨论的一些最重要的危险信号。系统中任何这些症状的存在表明系统的设计存在问题：

- Shallow Module: the interface for a class or method isn’t much simpler than its implementation (see pp. 25, 110).
- Information Leakage: a design decision is reflected in multiple modules (see p. 31).
- Temporal Decomposition: the code structure is based on the order in which operations are executed, not on information hiding (see p. 32).
- Overexposure: An API forces callers to be aware of rarely used features in order to use commonly used features (see p. 36).
- Pass-Through Method: a method does almost nothing except pass its arguments to another method with a similar signature (see p. 46).
- Repetition: a nontrivial piece of code is repeated over and over (see p. 62).
- Special-General Mixture: special-purpose code is not cleanly separated from general purpose code (see p. 65).
- Conjoined Methods: two methods have so many dependencies that its hard to understand the implementation of one without understanding the implementation of the other (see p. 72).
- Comment Repeats Code: all of the information in a comment is immediately obvious from the code next to the comment (see p. 104).
- Implementation Documentation Contaminates Interface: an interface comment describes implementation details not needed by users of the thing being documented (see p. 114).
- Vague Name: the name of a variable or method is so imprecise that it doesn’t convey much useful

information (see p. 123).
- Hard to Pick Name: it is difficult to come up with a precise and intuitive name for an entity (see p. 125).
- Hard to Describe: in order to be complete, the documentation for a variable or method must be long. (see p. 131).
- Nonobvious Code: the behavior or meaning of a piece of code cannot be understood easily. (see p. 148).

> - 浅模块：类或方法的接口并不比其实现简单得多（请参见第 25、110 页）。
> - 信息泄漏：设计决策反映在多个模块中（请参阅第 31 页）。
> - 时间分解：代码结构基于执行操作的顺序，而不是信息隐藏（请参见第 32 页）。
> - 过度暴露：API 强制调用者注意很少使用的功能，以便使用常用功能（请参阅第 36 页）。
> - Pass-Through Method：一种方法几乎不执行任何操作，只是将其参数传递给具有相似签名的另一种方法（请参见第 46 页）。
> - 重复：一遍又一遍的重复代码（请参见第 62 页）。
> - 特殊通用混合物：特殊用途代码未与通用代码完全分开（请参阅第 65 页）。
> - 联合方法：两种方法之间的依赖性很大，以至于很难理解一种方法的实现而又不理解另一种方法的实现（请参阅第 72 页）。
> - 注释重复代码：注释旁边的代码会立即显示注释中的所有信息（请参阅第 104 页）。
> - 实施文档污染了界面：界面注释描述了所记录事物的用户不需要的实施细节（请参阅第 114 页）。
> - 含糊不清的名称：变量或方法的名称过于精确，以至于它不能传达很多有用的信息（请参阅第 123 页）。
> - 难以选择的名称：很难为实体提供准确而直观的名称（请参见第 125 页）。
> - 难以描述：为了完整起见，变量或方法的文档必须很长。（请参阅第 131 页）。
> - 非显而易见的代码：一段代码的行为或含义不容易理解。（请参阅第 148 页）。

About the Author 关于作者

John Ousterhout is the Bosack Lerner Professor of Computer Science at Stanford University. He is the creator of the Tcl scripting language and is also well known for his work in distributed operating systems and storage systems. Ousterhout received a BS degree in Physics from Yale University and a PhD in Computer Science from Carnegie Mellon University. He is a member of the National Academy of Engineering and has received numerous awards, including the ACM Software System Award, the ACM Grace Murray Hopper Award, the National Science Foundation Presidential Young Investigator Award, and the U.C. Berkeley Distinguished Teaching Award.

> John Ousterhout 是斯坦福大学的 Bosack Lerner 计算机科学教授。他是 Tcl 脚本语言的创建者，并且以在分布式操作系统和存储系统中的工作而闻名。Ousterhout 在耶鲁大学获得了物理学学士学位，并在卡内基梅隆大学获得了计算机科学博士学位。他是美国国家工程院院士，并获得了无数奖项，包括 ACM 软件系统奖，ACM Grace Murray Hopper 奖，美国国家科学基金会总统年轻研究者奖和 UC Berkeley 杰出教学奖。