TURING 图灵程序设计丛书

简约之美

软件设计之道

mplicity: The Science of Software Development

[美] *Max Kanat-Alexander* 著
余晟 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

简约之美
软件设计之道

Code Simplicity
The Science of Software Development

[美] Max Kanat-Alexander 著
余晟 译

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo
O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目（CIP）数据

简约之美 : 软件设计之道 / (美) 卡纳特-亚历山大(Kanat-Alexander,M.) 著 ; 余晟译. -- 北京 : 人民邮电出版社, 2013.1
（图灵程序设计丛书）
书名原文: Code Simplicity : The Science of Software Development
ISBN 978-7-115-30238-0

Ⅰ. ①简… Ⅱ. ①卡… ②余… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2012)第293703号

内 容 提 要

本书将软件设计作为一门严谨的科学，阐述了开发出优雅简洁的代码所应该遵循的基本原则。作者从为什么以前软件设计没有像数学等学科一样成为一门科学开始入手，道出了软件以及优秀的软件设计的终极目标，并给出了具体的指导规则。

这是一本软件思想著作，适合任何背景、使用任何语言的程序员。

图灵程序设计丛书

简约之美：软件设计之道

◆ 著 [美] Max Kanat-Alexander
译 余 晟
责任编辑 朱 巍

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京天宇星印刷厂印刷

◆ 开本：880×1230 1/32
印张：3.75
字数：94千字 2013年1月第1版
印数：1－4 000册 2013年1月北京第1次印刷

著作权合同登记号 图字：01-2012-7086号
ISBN 978-7-115-30238-0

定价：25.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

目录

第 1 章　引言 ····· 1

1.1　计算机出了什么问题？ ····· 3
1.2　程序究竟是什么？ ····· 5

第 2 章　缺失的科学 ····· 9

2.1　程序员也是设计师 ····· 12
2.2　软件设计的科学 ····· 13
2.3　为什么不存在软件设计科学 ····· 15

第 3 章　软件设计的推动力 ····· 19

第 4 章　未来 ····· 27

4.1　软件设计的方程式 ····· 29
4.1.1　价值 ····· 30
4.1.2　成本 ····· 31
4.1.3　维护 ····· 32
4.1.4　完整的方程式 ····· 33
4.1.5　化简方程式 ····· 33
4.1.6　你需要什么，不需要什么 ····· 34
4.2　设计的质量 ····· 36
4.3　不可预测的结果 ····· 37

第 5 章　变化 ····· 41

5.1　真实世界中程序的变化 ····· 43

5.2 软件设计的三大误区 46
5.2.1 编写不必要的代码 46
5.2.2 代码难以修改 48
5.2.3 过分追求通用 51
5.3 渐进式开发及设计 53
第 6 章 缺陷与设计 55
6.1 如果这不是问题 57
6.2 避免重复 59
第 7 章 简洁 61
7.1 简洁与软件设计方程式 65
7.2 简洁是相对的 65
7.3 简洁到什么程度？ 67
7.4 保持一致 69
7.5 可读性 71
7.5.1 命名 72
7.5.2 注释 73
7.6 简洁离不开设计 74
第 8 章 复杂性 77
8.1 复杂性与软件的用途 81
8.2 糟糕的技术 83
8.2.1 生存潜力 83
8.2.2 互通性 84
8.2.3 对品质的重视 84
8.2.4 其他原因 85
8.3 复杂性及错误的解决方案 85
8.4 复杂问题 86
8.5 应对复杂性 87
8.5.1 把某个部分变简单 89
8.5.2 不可解决的复杂性 90
8.6 推倒重来 90
第 9 章 测试 93
附录 A 软件设计的规则 97
附录 B 事实、规则、条例、定义 101

第1章

引言

计算机带来了社会的剧变。因为有了它，我们可以用更少的人，干更多的事情。这就是计算机的价值——它可以干很多的事情，而且速度相当快。

这挺棒的。

但是，计算机会出问题，而且总出问题。如果你家里其他东西出问题有计算机那么频繁，你多半会退货。生活在现代的大多数人，每天至少会遇到一次系统崩溃或者程序错误。

这就没那么棒了。

1.1 计算机出了什么问题？

为什么计算机这么容易出问题？如果是软件的问题，那么有而且只有一个原因：程序写得太糟糕。有些人怪罪管理，有些人怪罪客户，但是调查发现，问题的根源通常都在于编程。

那么，“程序写得太糟糕”是什么意思？这是个很模糊的说法。通常来说，程序员都是很聪明、很理智的人，他们怎么会编出“糟糕”的程序呢？

说穿了，这一切都与复杂性有关。

今天，计算机大概是我们能生产的最复杂的设备了。它每秒钟可以计算数十亿次，它内部数以亿计的电子元件必须精密协调，整台计算机才可以正常运行。

计算机上跑着的程序同样复杂。举例来说，微软的 Windows 2000 还在开发时，就可算有史以来规模最大的软件了，它包含 3000 万行代码，这大概相当于 2 亿字——是《不列颠百科全书》字数的 5 倍。

程序的复杂性问题可能更加麻烦，因为程序里没有摸得着的东西。程序出问题的时候，你也找不到什么实实在在的东西，打开瞧瞧里面发生了什么。程序完全是抽象的，非常难处理。其实，常见的计算机程

序就已经足够复杂，没有人可以从头到尾理解所有代码是如何工作的。程序越大，越是如此。

这样说来，编程就成了把复杂问题化解为简单问题的劳动。否则，一旦程序达到某种复杂程度，就没有人可以理解了。程序中复杂的部分必须以某种简单方式组织起来，这样，不需要神那样强大的思维，普通程序员也可以开发出来。

这就是编程所要用到的艺术和才能——化繁为简。

“差程序员”是不会化繁未简的。他们总以为用编程语言（这东西本来就够复杂了）写出“能跑通”的程序，就已经化解了复杂性，而没有考虑降低其他程序员需要面对的复杂性。

大概就是这么一回事。

设想一下，为了把钉子钉到地板上，工程师发明了一台机器，上面有皮带轮，有绳子，还有磁铁。你多半会觉得这很荒唐。

再设想，有人告诉你说：“我需要一段代码，它能用在任何程序的任何地方，能够通过任何介质，实现两台计算机之间的通信。”这个问题无疑很难化繁为简。所以，有些程序员（大多数程序员）在这种情况下给出的解法，就像是一台装备了皮带轮、绳子、磁铁的机器，其他人当然很难看得懂。并不是这些程序员缺少理性，他们的脑子也没有进水。他们面对的问题确实困难，设定的期限也很紧，他们能做的就是这些。在这些程序员看来，写出来的东西是能用的，它符合要求。这就是他们的老板需要的，看来也应该是客户需要的。

不过，这些程序员毕竟没做到化繁为简。完工之后，他们把结果交给其他程序员，其他程序员又会在这之上继续增添复杂性，完成自己的工作。程序员对化解复杂性考虑得越少，程序就越难懂。

于是程序变得无比复杂，最终没办法找出其中的各种问题。喷气式飞机差不多也有这么复杂，但它们的造价是几百万甚至几十亿美元，而且仔细排查过错误。大多数软件的售价只有 50 ~ 100 美元。价钱这

么低，没有人有足够的时间和资源，在几乎无限复杂的系统里找到所有的问题。

所以，“好程序员”应当竭尽全力，把程序写得让其他程序员容易理解。因为他写的东西都很好懂，所以要找出 bug 是相当容易的。

这个关于简单性的想法有时被误解为，程序不应当包含太多代码，或者是不应当使用先进技术。这么想是不对的。有时候，大量的代码也可以带来简单，只不过增加了阅读和编写的工作量而已，这是完全正常的。你只要保证，那些大段的代码提供了化解复杂性所必须的简短注释，就足够了。同样，通常来说，更先进的技术只会让事情更简单，只是一开始你得学习，所以整个过程可能没那么简单。

有些人相信，把程序写得简单所花的时间，要比写“能用就好”的程序更多。其实，花更多的时间把程序写简单，相比一开始随意拼凑些代码再花大量的时间去理解，要快得多。这个问题说起来轻巧，显得轻描淡写，其实软件开发的历史教训很多，诸多事例已经反复证明了这一点。许多大型程序的开发之所以会停滞数年，就是因为一开始没有做好，结果必须等上这么长的时间，才能给之前开发出来的怪物加上新功能。

正因为如此，计算机经常出问题——因为大多数程序都有这个问题，许多程序员在写程序时并没有化解复杂性。是的，这么做很难。但是如果程序员做不到这一点，设计出的系统过于复杂、经常出问题，用户在使用的过程中就会经受无穷无尽的折磨。这么一比，把程序写简单所费的工夫实在算不了什么。

1.2 程序究竟是什么？

大多数人说的“计算机程序”，其实有完全不同的定义：

(1) 给计算机的一系列指令
(2) 计算机依据指令进行的操作

第一种定义是程序员写程序时所用的。第二种定义是使用程序的普通用户所用的。程序员命令计算机：在屏幕上显示一头猪。这就是第一种定义，它包含若干指令。计算机接收到指令之后，会控制电信号，在屏幕上显示一头猪。这是后一种定义，即计算机执行的操作。程序员和用户都会说自己在和“计算机程序”打交道，但是他们的用法是很不一样的。程序员面对的是字母和符号，用户看到的是最终结果——计算机执行的操作。

所以，计算机程序其实是这两者的混合体：程序员的指令、计算机执行的操作。编写指令的最终结果就是让计算机执行那些操作——如果不需要执行操作，就没必要去写代码了。这就好像在生活里，你列了一张购物单（相当于指令），告诉自己该买哪些东西。如果你只是列了单子，但没去商店，单子就没有任何意义。指令必须得到实际的结果。

但是，列购物单和写程序有显著的区别。如果购物单列得很乱，只不过会降低买东西的速度。但是如果程序写得很乱，实现最终的目标就显得尤其困难。为什么呢？因为购物单是简单短小的，用完就可以扔掉。而程序是很复杂很庞大的，你可能还需要维护很多年。所以，同样是没有秩序，购物单只会给你造成一点儿小麻烦，程序却可以给你增添无尽的烦恼。

而且，除软件开发之外，没有任何领域的指令和结果联系得这么紧密。在其他领域，人们先编写指令，然后交给其他人，指令通常要等很长的时间才会执行。比如设计房子，建筑师首先给出指令——也就是蓝图。这份蓝图经很多人的手，过了很长的时间，才能建起真正的房子。所以，房子是大家所有人解读建筑师指令的结果。相反，写程序时，在我们和计算机之间没有任何人。我们让计算机干什么，就会得到怎样的结果；计算机绝对服从命令。结果的质量完全取决于机器的质量、我们想法的质量，代码的质量。

在这三个因素当中，代码的质量是如今软件工程需要面对的最重要问题。根据这一点，以及上面提到的其他原因，本书的大部分内容都在论述如何提高代码质量。会有一些地方涉及机器的质量和想法的质量，

但是大多数内容都在论述怎样改善你交给机器的指令的结构和质量。

虽然花了这么多的时间来谈代码，但也很容易忘记，改善代码质量的一切努力都是为了获得更好的结果。本书绝不姑息任何差劲的结果——我们学习提高代码质量的全部原因都在于，要想改进结果，提高代码质量是最重要的问题。

所以，我们最需要掌握的，就是提高代码质量的科学方法。

第2章

缺失的科学

本书大部分篇幅讲解的是“软件设计”，你以前可能听过这个词，甚至读过这方面的书。不过，现在请抛弃原有的观点，从全新的、准确的定义开始，重新认识“软件设计”。

软件是什么，我们都知道，因此真正要定义的就是“设计”了。

动词

(1) 为创造性活动制订计划。例句：工程师本月会设计一座桥梁，下个月建造它。

名词

(1) 为某种创造性活动而制订，但尚未实施的计划。例句：工程师已经确定了桥梁的设计，下个月建造它。
(2) 业已存在的创造物所遵循的计划。例句：那座桥的设计相当不错。

谈到软件设计，下列这些定义都适用。

- 我们“设计软件”（动词“设计”）时，是在进行计划活动。设计软件时要关心的事情有很多，包括代码的结构、所用的技术等，还要制订许多技术决策。通常，我们只是在脑子里做这些决定，有时候会把它写下来或画出来。
- 上一步的结果是“软件的设计”（第一种意义的名词“设计”），也就是计划。同样，它可能是落实下来的文档，也可能是我们脑中的若干决定。
- 已经存在的程序同样有“设计”（第二种意义的名词“设计”），也就是它的结构，或者它所遵循的计划。有些程序可能根本没有清楚的结构，所以是“无设计”的，也就是说开发它的程序员没有任何明确的计划。在“无设计”和“完整的设计”之间，存在着广阔的灰色地带，比如“部分的设计”、“在某段程序中存在的若干矛盾的设计”、“接近完成的设计”等等。一些刻意而为的糟糕设计甚至比无设计还要差劲。比如你遇到的那些刻意制造纠结或复杂的程序，就属于这类刻意而为的糟糕设计。

软件设计的科学就是为软件做计划、制定决策的科学，它帮助大家做出这类决定：

- 程序的代码应当采用什么结构？
- 是程序的速度重要，还是代码容易阅读重要？
- 为满足需求，应该选择哪种编程语言？

软件设计与下列问题无关：

- 公司的结构应该是怎样的？
- 什么时候召开团队会议？
- 程序员的工作时间应该如何安排？
- 程序员的绩效如何考核？

这些决策与软件本身无关，只与组织有关。显然，保证这类决策的合理性也是重要的——许多软件项目之所以失败，就是管理失当。但是这不是本书的主题，本书关注的是，如何为你的软件制订合理的技术决策。

> *软件系统中任何与架构有关的技术决策，以及在开发系统中所做的技术决策，都可以归到“软件设计”的范畴里。*

2.1 程序员也是设计师

在软件项目中，每个程序员的工作都与设计有关。首席程序员负责设计程序的总体架构；高级程序员负责大的模块；普通程序员则设计自己的那一小块，甚至只是某个文件的一部分。但是，即便仅仅是写一行代码，也包含设计的因素。

哪怕是单干，也离不开设计。有时候你在敲键盘之前，就做了决定，这就是在做设计。有的夜晚，你躺在床上，还在思考要怎样编程。

每个写代码的人都是设计师，团队里的每个人都有责任保证自己的代码有着良好的设计。任何软件项目里，任何写代码的人，在任何层面上，都不能忽略软件设计。