全模型软件系统设计

马莘权 著

吉林大学出版社
JILIN UNIVERSITY PRESS

全模型软件系统设计

马莘权 著

吉林大学出版社
JILIN UNIVERSITY PRESS

图书在版编目（CIP）数据

全模型软件系统设计/马莘权著. —长春：吉林大学出版社，2013.6
ISBN 978-7-5677-0011-6

Ⅰ.①全… Ⅱ.①马… Ⅲ.①软件设计 Ⅳ.①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 103198 号

书　名：全模型软件系统设计
作　者：马莘权 著

责任编辑、责任校对：李卓彦　　封面设计：中尚图
吉林大学出版社出版、发行　　北京紫瑞利印刷有限公司　印刷
开本：710×1000 毫米　1/16　　2013 年 6 月　第 1 版
印张：16　　字数：200 千字　　2013 年 6 月　第 1 次印刷
ISBN 978-7-5677-0011-6　　定价：36.00 元

社址：长春市明德路 501 号　邮编：130021
发行部电话：0431-89580026
网址：http//www.jlup.com.cn
E-mail：jlup@mail.jlu.edu.cn

读者指南

本书根据系统建模时所使用的方法和工具的不同，将内容分为两部分——人机界面设计和非人机界面设计。

第一章到第四章介绍的是非人机界面（通常我们称之为系统“后台”）设计；第五章讲的是人机界面的设计；第六章我们从一个更高的角度，来重新认识产品演化背后的客观规律。

非人机界面设计部分的内容，又分为基础知识部分和建模部分。第一章到第三章介绍的，是我们介绍第四章内容前需要大家掌握的基础知识。第一章介绍的是建立初始模型的方法。第二章介绍的是模型推理的方法。第三章我们通过对比目前常用的系统设计工具，来帮助大家认识模型设计的优势所在。第四章我们才真正开始全模型软件系统设计的介绍。基础知识的掌握情况，影响到大家对关键章节——第四章模型推理过程的理解。所以，笔者在这里建议大家还是按照章节顺序依次进行阅读。

第五章关于人机界面部分设计，主要是介绍 UI 设计的原则和方法。

第六章的内容虽然与第五章关联，但它是从一个新的高度和角度，重新审视了产品功能定位、演化背后的客观规律。希望第六章的内容，能够帮助大家扩展视野，打破思维定式，设计出更加丰富、生动、完美的产品。

本书的附录 1，为使用 UML 建模的设计师简单介绍了色彩的基本知识，以帮助大家理解平面设计师的语言。附录 2 提供了一个连续的 UML 模型推导示例。示例虽然只有短短十个视图，但足以展示 UML 模型的推理方法和过程。附录 3 提供了组合关系模型的示例。在讲解类优化的章节中，我们会讲到组合关系模型。

对于关注 UML 建模方法的读者来说，阅读第一章到第四章即可。对于项目级或产品级的设计师来说，笔者建议大家通读全文。在阅读第四章时，建议大家在读懂每一步推导的原因后，再继续阅读后面的内容。模型推导需要严格遵守因果关系，只要有一步没有理解，整个推理的链条就断了。其实模型推导并不难学，只是在阅读时大家需要严谨和耐心而已。在阅读时，如果遇到困难，可以根据上下文谈论到的知识要点回头阅读相关章节。

最后，衷心地希望本书内容能够给大家有所帮助。

前　言

本书是一本介绍在软件系统开发中如何使用模型的书。书中提出了一种新的项目运作方式。使用这种项目运作方式，我们可以解决高风险、高成本和高复杂度的软件产品的开发问题。通过大量使用模型测试来代替传统的软件测试，这种开发方式把系统的设计过程，从传统的“设计—实施—测试—优化设计”的循环，缩减为“设计—测试—优化设计”的循环，从整体上优化了软件系统开发过程中的信息、资源的布局，缩短了新产品研发周期，提升了研发效率，降低了项目风险和投资。

在这种开发方式中，整个软件项目的绝大部分设计、测试工作是通过模型来完成的。因此，笔者将这种开发方式称之为“全模型软件系统设计”开发方式。全模型软件系统设计使用的是 UML、思维导图等模型和配套的建模方法。在设计过程中，模型的演化完全基于面向对象思想的运用和因果关系的推导。通过对模型的推导，我们可以递归的完成软件系统的设计、测试。这种方法解除了传统软件项目运作中，软件设计必须编码后才能测试的结构性限制。从根本上解决了传统项目运作中，项目组无法进行有效的质量控制、风险控制等问题。在开发高成本、高风险和复杂的软件系统时，使用全模型软件系统设计可以大大降低项目风险和成本，并从整体上缩短项目周期。

本书在阐述软件业务流程建模时，使用了 ISO 标准的 UML 建模语言。因此，本书也是一本介绍如何使用 UML 来完成软件设计的书。书中介绍的建模方法，涵盖从需求分析开始到整个系统设计完毕为止的全部项目活动。与市面上常见的 UML 相关的书籍不同，本书主要是介绍建模方法和思想，对模型符号没有做过多的解说。因此，本书适合那些已经知道 UML 符

号意义，并希望在此基础上进一步得到提升，进一步将 UML 建模用于实际项目的读者学习和阅读。同时本书也适合那些希望了解、学习系统整体设计的架构师、系统设计师学习和阅读。希望本书介绍的方法、思想对大家有所帮助。

本书的内容分为两部分：人机界面部分的设计和非人机界面部分的设计。人机界面部分的设计主要介绍的是如何完成人与机器之间交互的设计；非人机界面部分的设计则是介绍如何设计系统的后台程序。之所以这样划分，不是因为项目运作中可以这样划分，而是因为人机界面设计与非人机界面设计使用的方法、工具不同。为了方便大家理清模型演化的原理和方法，我们才如此划分。大家可以在实践中将这两部分的设计融合在一起，来加深对本书内容的理解。

本书中的前面章节有一些相关理论的阐述。这些理论能帮助大家理解模型推导的每一步背后暗含的科学依据。为了便于大家理解本书的内容，书中每个理论知识、推导规则等都伴有示例，所以本书的内容并非像专业技术论文那样晦涩难懂。

全模型软件系统设计方法是在我多年的项目实践中总结出来的。实践对于掌握这种设计方法尤为重要。在实践中发现问题，再在重复阅读本书中找到答案，这样大家才能充分理解和掌握书中介绍的知识和方法。理论加实践，这是笔者给大家的建议。

全模型软件系统设计方法并非已经完美。至少我们用来建模的 UML 语言本身就不能描述模型的性能参数。因此，本人欢迎大家来信对本书的内容进行批评指正。无论从书中采用的示例、参考，还是笔者遗漏的已知应用，或应该包含的其他方面的知识，甚至是对全模型软件系统设计的改进等等。大家可以通过电子邮件联系笔者，笔者的电子信箱是：iorishinier@163.com。

目录

第1章 面向对象的思想可用于建立初始模型

面向对象思想是当今最流行、最普及的编程思想。我们就从这个最熟悉但又是最容易被误解的“面向对象”开始，介绍本书的内容。

1.1 面向对象思想的科学之处

面向对象思想，虽然是现代软件设计的基础思想，但是“面向对象”四个字并没有解释清楚这一思想的本质和核心内容。本人曾简单地认为用类封装的代码，就是面向对象的编程，用支持类封装的编程语言设计的软件就是面向对象的设计，但最终我发现它们是不同的。其实“面向对象”思想的确简单，而且也并非是程序员的专利。很多不懂编程的人也在使用着这一思想，只是他们并不知道这就是“面向对象”而已。

我们以图 1.1 - 1 为例来说明这个问题。见过汽车的人都知道，图 1.1 - 1的设计是有问题的——图中的汽车缺了轮子。不过，能说出图 1.1 - 1中设计缺陷的人，并非都能解释为什么这辆汽车缺少轮子就是设计缺陷。如果我们非要他们给个解释的话，最可能得到的回答就是“因为现实中的汽车是有轮子的”——这其实就是运用面向对象思想得出的结果。

这种以现实事物为对照标准的论证法，正是面向对象思想的核心。

图 1.1 -1

对图 1.1 -1 中的设计缺陷，刚才的这种似是而非的回答其实并不荒谬。它虽然“突兀”地将现实事物作为判断对错的标杆，但其背后是有科学依据的。我们以 1769 年至 1891 年汽车结构的变化发展过程，来揭示这种把现实事物作为“正确的模型”的论证法背后隐藏着的科学规律。

从 1769 年到 1891 年，汽车的结构经历了如下变化：
1769 年第一辆蒸汽机驱动的汽车问世；
1803 年蒸汽机驱动的汽车的引擎被换成了新型高压蒸汽机；
1838 年第一台内燃机点火装置问世；
1842 年汽车轮胎被更换成为实心的硬橡胶轮胎；
1858 年第一只用陶瓷绝缘制成的电点火火花塞问世；
1859 年铅酸蓄电池问世；
1860 年第一部用电火花点燃煤气的煤气机问世；
1862 年第一部二冲程内燃机问世；
1867 年第一台往复活塞式四冲程煤气发动机问世；
1876 年单缸卧式、压缩比为 2.5 的 3 千瓦煤气机问世；
1886 年三轮汽车架构被申请专利，第一辆四轮汽车问世；
1888 年充气轮胎问世；
1889 年齿轮变速器和差速装置问世；
1891 年汽车正式使用发动机前置、后轮驱动的结构形式。

为什么汽车轮胎不是实心的？历史上确实是有实心轮胎。不过大家可以想象其笨重且舒适度太差，所以在1888年换成了充气轮胎。为什么汽车的发动机位于汽车前面而不是后面？1891年之前确实是有发动机放在后面的汽车。这种汽车乘客坐在前面，发动机在后面。在撞击试验时，乘客的座位被障碍物和安装在汽车尾部的发动机前后夹击，非常容易导致乘客死亡。所以1891年汽车的标准结构才将发动机前置，将乘客的座位设计到发动机的后面。这样，在撞击障碍物时，发动机还能作为缓冲装置消耗掉一部分撞击力，保护乘客的安全。为什么汽车必须要有变速齿轮和差速装置？因为缺少这个东西，汽车在转弯时很容易冲出路基。

如果继续追问汽车的结构设计为何要如此这般，我们都能得到相应科学合理的解释。这就是现实中的事物背后的科学性——事物能以这样的结构和流程存在于现实生活中，背后就必然有相应的科学和客观规律在支撑。只有那些通过了历史反反复复的检验的事物，才能真正地延存至今——就像生物的进化一样。所以，人们直接把这些现实中已经存在的事物，作为评判对错的标杆的做法并不荒谬，相反这种做法还有其科学性和合理性。

同样的道理，如果我们需要设计一个新系统，我们也可以将现有的事物作为系统设计的原型或标杆。我们为什么不模仿现实世界中的系统，来设计我们想要的系统呢？既然现实世界的背后有其科学合理的解释，那么我们直接从模仿现实世界得来的模型，也应该是科学合理的。我们完全没必要一切都从零开始，让我们的设计到处都是挑战和风险。

正是因为明白这一点，我们的前辈们——那些“面向对象”思想的倡导者，才提出从现实生活中借鉴已有的模型的“面向对象”思想。他们提倡设计师们设计的软件，应该基于这些在现实中已经被证明成功的结构、流程之上，以降低软件设计的风险和难度。

总结上面的内容，我们可以得出这样的结论：

“面向对象”就是在系统设计时，模仿现实中已经存在的结构和流程，来设计软件系统的结构和流程的一种思维方法。面向对象思想最关键的地方，是借鉴现实世界中的事物、关系及流程。

面向对象思想并不关注事物背后的科学道理。它只关注事物的外在表现。我们以下面这个问题为例，进一步解释“面向对象”思想的这一特征。

假如生产一种产品需要两道工序。A 厂生产时，第一道工序需要花费 1 小时，第二道工序需要花费 2 小时。而 B 厂生产时，第一道工序需要花费 1.5 小时，第二道工序需要花费 1.5 小时。如果 A、B 两厂同时生产，每周工作 40 小时的产量是多大？

数学家们在解决这道问题时，通常都是这样一种思维步骤：

1. A 厂生产一件产品需要 1 + 2 = 3 小时，B 厂生产一件产品需要 1.5 + 1.5 = 3 小时，A、B 两厂生产一件产品的耗时一样；

2. A 厂生产时第一道工序快，第二道工序慢；而 B 厂恰好相反。因此可以尝试让 A 厂做第一道工序，而 B 厂做第二道工序；

3. 如果让 A 厂做第一道工序，B 厂做第二道工序，那么 B 厂开工会晚，A 厂停工会早，会有时间浪费……

数学家们只关注事物的本质，他们研究的是现象背后的逻辑关系。他们通过这些背后的逻辑关系来分析问题。

有面向对象思想的程序员则是这样想的：

1. 产品生产需要两道工序，因此我需要一个整型变量 m 来记录第一道工序完成的产品数，另一个变量 n 来记录第二道工序完成的产品数；

2. 无论 A 厂还是 B 厂在刚开始生产或完成手头工序后，都检查 m 的值。如果 m 大于 0，则让 m 减 1，自己进入第二道工序的生产；如果 m 值为 0，则开始进入第一道工序的生产。

3. 无论 A 厂还是 B 厂，在完成第二道工序后，让 n 加 1；

4. A 厂进入第一道工序后，1 秒后完成此工序，让 m 加 1；B 厂则是 1.5 秒后让 m 加 1。

5. A 厂进入第二道工序后，2 秒后完成此工序，让 n 加 1；B 厂则是 1.5 秒后让 n 加 1。

6. 程序运行 40 秒后输出 n 的值。

有面向对象思想的程序员并没有研究现象背后的逻辑关系，他做的仅仅是对相关场景的模仿——只要“依葫芦画瓢”就是对的。本人觉得，把“面向对象”改为“模仿事物”在表达上会更准确些。

1.2 “面向对象”并不等价于“类”

面向对象思想在软件业已经流行了很多年。现在随便一个程序员都能用支持面向对象的编程语言如 C + + 或 Java，轻松编写出一个关于类的代码。不过，“面向对象”就等于“类”吗？我们看看某程序员编写的这个“汽车”类，如图 1.2 - 1 所示。

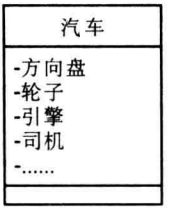

图 1.2 - 1

图 1.2 - 1 是我们使用 UML 类模型描述的一个类设计。在图 1.2 - 1 中，“汽车”类有“方向盘”、“轮子”、“引擎”、“司机”等等成员变量。如果“类”等于“面向对象”的话，那图 1.2 - 1 中“汽车”类里出现的“司机”是在表述车祸？或者说，汽车厂商在出售汽车时，如果汽车没有配司机，那汽车就算是缺少零部件，可以退货了？

显然这些问题都很可笑。我们都知道，在现实生活中，汽车没有司机仍然叫汽车，司机没有汽车仍然叫司机。车市上卖的汽车都是不配司机的。而且，只有在分配工作或驾驶时，汽车与司机才存在联系——谁负责开哪一辆汽车。在这些场景下，“汽车”和“司机”都是最小颗粒度的客

观对象，不应该再细分。图 1.2 - 1 中的类对“汽车”进行了进一步的细分，显然这个程序员想关注的是汽车内部。既然他关注的是汽车内部，就不应该把汽车外部的“司机”牵连进来。

由此我们可以看出，虽然程序中我们可以这样定义“汽车”类，但这样的设计却是违背面向对象思想的。类虽然源自于面向对象思想，但像图 1.2 - 1 这样的“汽车”类，它与客观世界中的“汽车”和“司机”都是不同的。在现实世界中，无论“汽车”与“司机”在什么场景下，它们都应该是如图 1.2 - 2 所示的这种关系——“汽车”和“司机”是两个不同的对象。像图 1.2 - 1 这样的设计，本人把它称为“面向直觉”的设计。它忽略了客观世界中“司机”与“汽车”之间存在“联系”的对应场景。也就是说，图 1.2 - 1 并没有从客观世界中模仿，而只是设计者“觉得应该是这个样子”而已。

我们从图 1.2 - 1 中的问题可以看出，面向对象的设计是要基于客观世界的场景的。这个场景明确的定义了客观事物之间的关系、类识别的颗粒度等等信息。我们基于面向对象思想，模仿的都是这个场景中的事物和关系。当然，不同的场景下我们看到的事物的颗粒度可能不同。但不同场景之间事物的特征，是不会相互矛盾的，就像图 1.2 - 2 中“汽车”与“司机”不可能是整体与部分的关系那样。因为“现实世界就是这个样子的”。

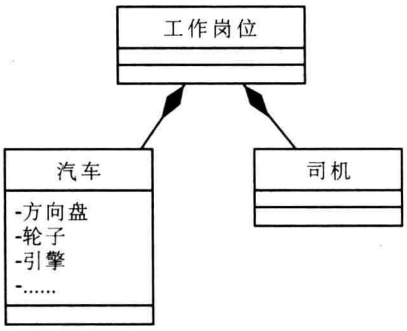

图 1.2 - 2

第2章 通过多视角对比可改造模型

虽然我们可以根据面向对象的思想，为要设计的系统建立初始模型。但要完成系统的设计，单单只会建立初始模型是不够的。我们还需要能指导我们进行模型测试和改造的科学方法。本章节，我们就来介绍这种能够测试和改造模型的方法。

2.1 模型完善的原理

本人的损友迈克曾经跟我开过这样一个玩笑：

迈克：哥们儿，我给你介绍一个女朋友，如何？

本人：好啊。

迈克：她有长长的头发大大的眼睛。

本人：哇！太好了！

迈克：给你她的照片。（给本人出示了她的玉照）

本人：……你欠揍！

我们把迈克所说的内容和我的想象，对照着放在表2.1－1中：