● 软件工程系列从书

[美] Bertrand Meyer 著
周伯生　童长忠　马宇飞　徐红　黄征　译
董士海　周伯生　校

面向对象式软件的构造

北京航空航天大学出版社

面向对象式软件的构造

周伯生　　童长忠
马宇飞　徐　红　黄　征　　译

董士海　　周伯生　　校

北京航空航天大学出版社

新登(京)字 166 号

内 容 简 介

面向对象这个主题作为当今计算机软件界的一大热点，已受到越来越多的人们的重视。通过近二十年的发展，面向对象技术已为软件开发提供了一种新的方法学，它将大大提高软件的可重用性、可扩充性和可靠性。本书全面系统地介绍了面向对象式软件的构造方法和技术。全书共分四部分二十一章。第一部分（第一至四章）描述了软件工程中引发面向对象方法的基本问题和面向对象设计的基本概念。第二部分（第五至十六章）以 Eiffel 语言作为表述工具，详细地介绍了面向对象的技术。第三部分（第十七至二十一章）考察了面向对象概念在其它语言环境中的实现情况，这些语言包括经典语言、模块化语言和其它的面向对象语言。第四部分（附录 A 至附录 F）包含了有关 Eiffel 语言细节的几个附录。

本书可作为计算机软件专业本科高年级学生和研究生的教材，也可作为计算机软件人员、广大科技工作者的参考资料。

面向对象式软件的构造
MIANXIANG DUIXIANG SHI RUANJIAN DE GOUZAO

[美]Bertrand Meyer 著

周伯生 童长忠 马宇飞 徐 红 黄 征 译

董士海 周伯生 校

责任编辑 刘又诚

*

北京航空航天大学出版社出版

北京航空航天大学软件工程研究所微机输入排版、激光打印

北京航空航天大学出版社印刷厂印刷

*

787x1092 1/16 印张：30 字数：768 千字

1992 年 2 月第一版 1992 年 2 月第一次印刷

印数：5000 定价：18.00 元

ISBN 7-81012-362-9/TP·082

前 言

面向对象的思想最初出现于挪威奥斯陆大学和挪威计算中心共同研制的 Simula 67 语言中，其后，随着位于美国加利福尼亚的 Xerox 研究中心推出 Smalltalk-76 和-80 语言，面向对象的程序设计方法得到比较完善的实现，由此面向对象的方法得到迅猛的发展．有人把它看成是台风，有人把它看成是海啸，也有人把它看成是茶杯中的风暴——总之，一个潮浪正在扑向计算世界的海岸．

"面向对象"一词是补充、也许是替代"结构化"一词而被引入的术语．在这种情况下，不可避免的会出现不同的人赋与它不同的含义．我们认为面向对象的方法不仅不同于当今大多数人使用的软件设计方法，而且在某些内容上还与这些方法不相容，包括大多数程序设计教科书中所叙述的某些原理．我们还认为，面向对象的设计方法有明显地改进软件质量的潜力．

在众多的讨论面向对象方法的著作中，本书是被广泛引用的少量优秀著作之一．本书的作者是从软件工程学的角度来论述面向对象的设计方法．当然，面向对象的方法在人工智能、数据库和图形学等其它领域中的应用也是非常有趣的，但它们不是本书的重点．本书通过探讨面向对象的方法，从而引深出一系列软件工程的新的原理、方法和工具，而这些新的手段将可用来构造比目前质量更高的软件产品．我们希望读者通过阅读此书能了解一条新的软件设计与实现的途径，从中得到收益．

面向对象式设计是基于一种很朴素的思想．通常计算系统是在一定的对象上执行一定的行为操作；要获得灵活并且可再用的系统，更好的方法是把软件的结构建立在对象之上而不是在行为之上．

说到这里，我们还没有为读者提供实际的定义，相反，却引发了一系列的问题，例如：对象准确地说是什么？怎么发现和描述对象？程序怎样操作这些对象？对象之间可能有些什么关系？怎样探索存在于不同类型对象之间的共性？这些与传统软件工程所关心的正确性、易使用性和有效性有什么关系？等等．

要回答这些问题，需依赖于一系列新的技术，这些技术能有效地用于产生可再用、可扩充和可靠的软件，它们是：继承机制，包括单继承和多继承；动态连接和多形性；一种新的类型观点和类型检查机制；类属机制；信息隐藏；断言的使用；合约式程序设计；安全的异常处理机制．目前已存在有效的实现技术使这些想法得以实际应用．作者在本书中很好地解答了上述问题．

本书系统地介绍了面向对象式软件的构造方法和技术．第一篇（第一至四章）描述了软件工程中引发面向对象方法的基本问题和面向对象式设计的基本概念．第二篇（第五至十六章）以 Eiffel 语言作为表述工具详细地介绍了面向对象的技术．第三篇（第十七至二十一章）考察了面向对象概念在其它语言环境中的实现情况，这些语言包括经典的非面向对象式语言，如 Fortran、Pascal 和 C；模块化但不是真正的面向对象式语言，如 Ada 和 Modula-2；和除 Eiffel 以外的其它的面向对象语言，如 Simula 67 和 Smalltalk．第三

篇还简短地讨论了并行性和持续性等当前面向对象方法所面临的一些问题。第四篇（附录A至附录F）包含了有关Eiffel语言细节的几个附录。

Eiffel语言是本书的一个重要部分，本书对Eiffel语言的描述占用了一定的篇幅。作者通过Eiffel语言这一表述工具，具体深入地讨论了面向对象的软件设计问题，这些讨论是有其典型性和普遍意义的。我们估计，即便是那些对面向对象设计感兴趣但又无法接触Eiffel环境的读者，本书的90%的内容仍然是有用的。余下的10%主要是与附录和各章结束部分的语法注释有关的内容。第三篇解释了这些概念可以怎样地被转换到其它语言中。

在第二篇的某些章中还包含了"讨论"这样的小节，在这些小节中描述了Eiffel语言的设计过程中遇到的一些设计上的问题和它们是怎样被解决的。作者，同时又作为Eiffel语言的设计者，在此向读者披露了一些极其有用的信息。这部分内容值得深入细读。

在本书中虽然使用了一种程序设计语言作为表述工具，但读者不要误认为面向对象技术只覆盖软件的实现阶段。相反，本书大部分内容是关于**设计**问题的。软件设计有时被错误地认为是与实现完全相分离的一个活动。但实际上，设计包含了与编程同样的智能机理和同样的智力挑战，只不过是在更高的抽象层次上而已。如果把设计与实现集成到同一概念结构中，那么采用这样的方法我们必然会获得更大的收益。Eiffel正是以此为其目的，例如延迟类、信息隐藏和直接使用断言等语言特性充分表现这一点。本书的一些章节（尤其是第三、四、七、九、十二和十四章）专门讨论了高层设计问题。

在本书的翻译工作中，周伯生翻译第一至四章，童长忠翻译第五至九章，徐红翻译第十至十五章，马宇飞翻译第十六章至二十一章，黄江翻译附录A至F。董士海、周伯生审校了全书。由于时间仓促、水平有限，难免有不当之处，欢迎广大读者批评指正。

最后，我们想借此机会感谢罗燕京、刘又诚、张金荣和其他一些同志对本书的出版给与的大力支持和帮助。为使本书能尽早奉献给广大读者，许多同志为此付出了辛勤的劳动，在此谨向他们致以深切的感谢！

译者　1992年1月

于北京航空航天大学软件工程研究所

目　　录

第一篇

基本问题和基本原理

第一章　软件质量的各个方面

软件工程的主要目的是帮助人们生产高质量的软件．本书介绍一系列可以大幅度改善软件产品质量的软件工程技术．

在讨论这些技术之前，首先必须明确它们的目的．我们应知道，软件质量并不单单是一种思想，最好把它看成"**要由各方面来确定**"这样一种观念，其中每一方面要用一系列因素来描述．本章分析其中的某些因素，指明最需要改善的方面，指出今后解决问题的方向．

1.1 外部因素和内部因素

人们都希望自己的程序快速、可靠、易用、可读、模块化、结构化等等，但这些修饰词所描述的是两类不同类型的质量．

一方面，人们考虑在一个软件产品中是否具有如速度或易用性等质量时，它可能要由这个产品的用户来检测．这类质量可以称之为**外部质量因素**(external quality factors)．这里"用户"这个术语应该采取足够广泛的观点，它不仅应该包括真正直接使用产品的最终用户(例如航空订票系统的订票员)，而且应该包括购买、投资开发或改进软件的客户(例如负责采购航空订票系统的总经理)．因此，能比较容易适应规格说明的修改的质量因素(**可扩充性**)就属于外部因素．

软件产品的其他质量因素，如模块化、可读性等，都是**内部质量因素**(internal quality factors)，只有计算机专业人员才能理解．

显然，只有外部因素才是人们关心的终极目标．例如，当人们使用电子制表软件或核反应工厂控制软件系统时，如果该软件已经成功运行很长时间，或者一个错误输入造成该核反应工厂停工，人们很少关心该软件的源程序是否可读．然而，内部因素是保证外部因素能否满意的关键．因此，要使用户满意那些可见质量，软件的设计人员与实现人员必须应用各种内部技术，以保证获得满意的可见质量．

本书将讨论一系列获得很好的内部质量的当代先进技术．当然，我们不应忘记总的目标，内部技术本身并不是终极目标，它仅仅是获得软件外部质量的一种方法．下面讨论这些质量因素中最重要的几个方面．

1.2 外部质量因素

1.2.1 正确性(correctness)

定义：正确性是指软件产品准确执行软件需求规格说明中所规定的任务的能力．

很清楚，正确性是首要的质量因素．如果一个软件不能准确执行指定给它的任务，那么其他的一切都是不能说明问题的．只要采用完全形式化的方法表达系统需求，那么满足

这些需求就是正确的．然而，这个目标说起来容易，做起来却并不简单．

1.2.2 鲁棒性(robustness)

定义：鲁棒性是指在异常条件下软件系统仍能运行的能力．

鲁棒性讨论的是在异常条件下软件的行为．它与正确性不同，正确性讨论的是在需求规格说明中明确陈述的软件行为．

鲁棒性在本质上是比正确性更为模糊的观念，它讨论的并不是在规格说明中明确陈述的问题，而是在某种情况下可能发生的问题．我们不能像讨论正确性时那样去说，在这种情况下系统应该执行什么任务，因为按照定义我们并不知道这些任务是什么；如果这些任务规定得比较准确，那么其异常情况将成为规格说明的一部分，从而将纳入正确性的范围．但是事实上往往是在规格说明中并不明确讨论这些问题．鲁棒性需求的作用在于保证：如果这种情况一旦发生，它应使系统的执行终止，或进入"降级运行(graceful degradation)"模式，而不致使系统发生灾难性事件．

"可靠性(reliability)"这个术语有时用来代替鲁棒性，但它包含了更为广泛的概念，最好把它理解为是正确性与鲁棒性两者之和．

1.2.3 可扩充性(extendibility)

定义：可扩充性是指软件产品适应需求规格说明变化的难易程度．

在原理上，软件要完成的是"软任务"，取出一个程序元素并进行修改应该是很容易做到的事．然而要知道，可扩充性问题是一个与规模有关的问题，对小型程序来说，修改通常并不是一个严重问题，但对大型程序来说，修改确实并不容易．程序的规模愈大，适应修改的任务也就愈难．通常的情况是：一个大型软件系统是一个宏伟但又很脆弱的结构，只要抽出其中的一块砖头，都会使这座巨大的建筑物倒塌．

虽然大多数改善可扩充性的技术可能是用小型示例或在引论性课程中介绍的，但这样做的实质仅仅是为了把问题陈述得更加清楚．

对改善可扩充性来说，主要应遵循两条原理：

- **设计简单**：简单的系统结构总是比复杂的系统结构更容易适应修改．
- **控制分散**：在软件系统结构中的模块愈是自主的，一个简单的修改只影响一个模块或只影响很少几个模块的可能性也就愈大，也就愈不致引起整个系统都需要修改那种连锁反应．

简单和分散也确实是我们下面要讨论的主要论题中的两个．

1.2.4 可再用性(reusability)

定义：可再用性是指软件产品可被全部或部分地再用到新的应用中去的能力．

人们观察到软件系统的很多元素遵循共同的模式，因而应该有可能揭露其共同性，以避免对以前曾经遇到的问题重新求解，这就是可再用性的客观需求．

可再用性的重要性是显而易见的．要特别注意到，可再用性对软件质量的所有其他方面都有影响．例如为了解决一个具有可再用性的问题，本质上就意味着所要编写的软件较少，如果整个软件的总费用容许不变，则就可以把更多的人工用来改善正确性与鲁棒性等

其他因素.

第三章整个一章是专门用来讨论可再用性问题的，第三章的讨论将导致人们去开发面向对象式设计技术.

1.2.5 兼容性(compatibility)

定义：兼容性是指软件产品与其他软件组合成一个整体的难易程度.

兼容性是重要的，因为软件产品不是在真空中开发的，它们需要与其他软件相互作用，但是一个软件产品常常作出与客观世界相矛盾的假设，因此很难与其他软件相互作用，这种情况是屡见不鲜的，不少操作系统所支持的文件格式常常不能兼容，这就是一个典型示例．一个程序要能直接利用另一个程序的结果作为它的输入，两者的文件格式必须是兼容的.

兼容性的关键在于设计的同构，还在于要符合程序内部通信的标准约定．这个问题的解决办法可以有以下几种:

- **标准化的文件格式**．如Unix系统，那里每一个正文文件都看成是一个字符串.
- **标准化的数据结构**．如Lisp系统，那里所有的数据和程序都是用二叉树(在Lisp中称之为列表)来表达的.
- **标准化的用户界面**．如Smalltalk系统，那里所有的工具与用户的通信都采用一种统一的基于窗口、图像、图形等的范式.

对软件所操作的所有重要实体定义标准化的存取协议，是更通用的解决兼容性问题的方法．这就是抽象数据类型(见第四章)和面向对象式方法的内在思想.

1.2.6 其他质量因素

面向对象式设计技术对改善以上所讨论的质量因素最为有用，是今后讨论的重点，但是我们不应忽视软件质量的另外几个因素.

效率(efficiency)是指对硬件资源(如处理器、内存和外存、通信装置等)利用的合理程度．虽然早期在讨论程序设计时曾过份强调了软件的时空效率，但对空间和时间等可用资源的合理使用，任何时候都应该是对任何软件产品的主要需求之一.

可移植性(portability)是指把软件产品转移到其他硬件环境和软件环境的难易程度.

可验证性(verifiability)是指在确认和运行阶段，编制验收规程(尤其是测试数据)、故障检测与差错追踪规程的难易程度.

完整性(integraty)是指软件系统的各个部件(程序、数据和文档)免受非授权人员存取和修改的保护能力.

易使用性(ease of use)是指对软件系统进行学习、操作、准备输入数据、解释结果以及根据出错信息进行恢复等工作的难易程度.

1.2.7 各种因素的权衡

在评论软件质量的外部因素时，已经看到需求的各项条款本身可能并不是完全兼容的，例如，如果不借助于各种各样的保护设施，则很难做到完善的完整性；如果保护设施太多，则将不可避免地伤害易使用性．又如，效率的优化既要求完全适应特定的硬件和软

件环境，又要求完全适应特定的规格说明，前者将使可移植性变坏，后者将使可扩充性和可再用性变差。

在很多情况下，可能会发现问题的解要满足明显矛盾的不同因素，此时必须综合考虑、恰当权衡。在这类情况下，把准则表达得很清楚是十分重要的。

1.3 软件维护问题

在讨论软件及其质量时，通常考虑开发阶段的问题。然而实际上应该在更广泛的范围内考虑问题。在程序设计过程中，通常并不涉及维护问题。但是根据众所接受的估计，维护阶段的花费常常要占软件总花费的 70%。如果在讨论软件质量时不考虑维护问题，显然这样的讨论是不会令人满意的。

对软件来说，"维护"意味着什么？骤然看来，这个术语是不恰当的。因为软件产品在重复使用时，不会发生磨损，因此并不像汽车或电视机那样需要"维护"。事实上，这个词是软件人员用来概括某些是绚丽多彩而另一些也并非是绚丽多彩的活动的用语。其中绚丽多彩的活动是指对软件的修改，因为要反映客观世界的变化，对计算机系统的需求也应随之改变，从而必须对系统进行修改。其中不那么绚丽多彩的工作是指对软件的后期排错，把在开发阶段没有发现的差错剔除。

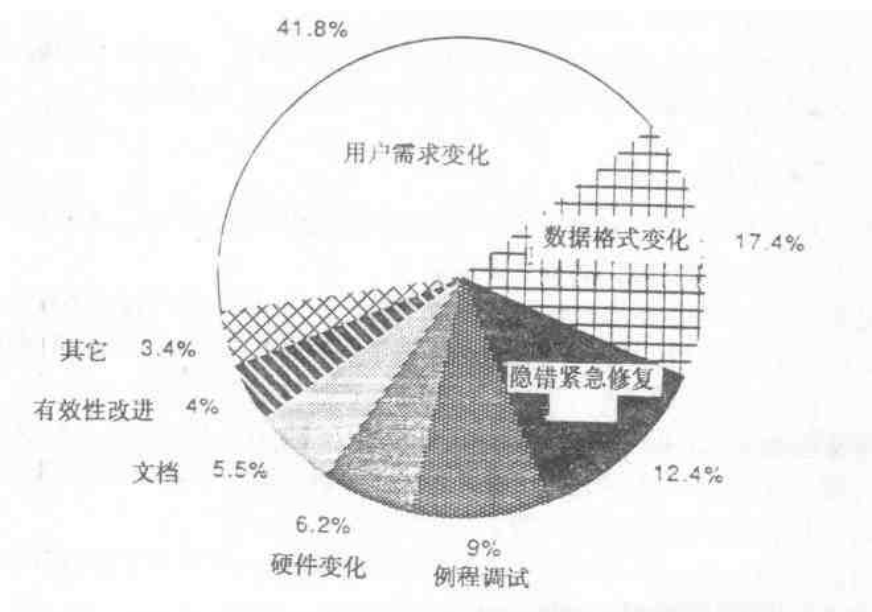

图 1.1 软件维护费用分布图(参阅[Lientz 1979])

图 1.1 可用来表明维护这个术语的全部含义，它是根据 487 个不同场所安装的各类软件的各种维护花费的统计数据绘制出来的。该研究结果表明，2／5 以上的维护活动是用户所要求的扩充与修改活动，这些活动是不可避免的。这就是我们所说的绚丽多彩的软件维护活动。然而这种比例似乎表明，通常实现的软件缺乏良好的可扩充性，系统的修改比想像的要难得多。本书所研究的各种技术所能带来的最大好处之一，就是可以使软件更易