深入 C++ 系列

Multi-Paradigm Design for C++

C++ 多范型设计

[美] James O. Coplien 著
鄢爱兰 周辉 等 译

Addison Wesley

中国电力出版社
www.infopower.com.cn

深入 C++ 系列

Multi-Paradigm Design for C++

C++ 多范型设计

[美] James O. Coplien 著
鄢爱兰 周辉 等 译

中国电力出版社

Multi-Paradigm Design for C++ (ISBN 0-201-82467-1)

James O. Coplien

Authorized translation from the English language edition, entitled Multi-Paradigm Design for C++, published by Addison Wesley Longman Inc., Copyright©1999

All rights reserved.

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

CHINESE SIMPLIFIED language edition published by China Electric Power Press Copyright©2003

本书由美国培生集团授权出版。

北京市版权局著作权合同登记号 图字：01-2002-4850 号

图书在版编目（CIP）数据

C++多范型设计 /（美）考帕里安编；鄢爱兰等译. 北京：中国电力出版社，2003

（深入 C++系列）

ISBN 7-5083-1824-2

Ⅰ.C... Ⅱ.①考...②鄢... Ⅲ.C 语言－程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 108179 号

丛 书 名：深入 C++系列

书　　名：C++多范型设计

编　　著：（美）James O. Coplien

翻　　译：鄢爱兰　周辉 等

责任编辑：邬学三　曾妍

出版发行：中国电力出版社

地址：北京市三里河路6号　　邮政编码：100044

电话：（010）88515918　　传　　真：（010）88518169

印　　刷：汇鑫印务有限公司

开　　本：787×1092　1/16　　印　　张：13.5　　字　　数：252 千字

书　　号：ISBN 7-5083-1824-2

版　　次：2004 年 2 月北京第 1 版　　2004 年 2 月第 1 次印刷

定　　价：26.00 元

版权所有　翻印必究

译者序

C++是一种支持多范型的编程语言。C++之父 Bjarne Stroustrup 认为，“多范型”可以有一种比较朴实的说法——“多种编程风格”。多范型设计的两个主要原则是共同性和差异性。本书正是根据共同性和差异性的基础来确定“范型”概念的。多范型程序设计是综合应用多种程序设计风格的程序设计方法，是产生架构的活动的集合，其目标是找到能够最自然地表达应用领域结构的方案领域结构。多范型设计是一种技巧，它并不完全是一种严格的规范。

本书详细地介绍了从“应用领域”到“方案领域”的 C++设计实现方法，以及开发者在设计思考和设计实践过程中需要用到的记法、图表和设计模型。在读完这本书以后，读者将会了解如何根据应用领域的共同性和差异性分析来确定 C++方案领域的结构，也就是如何选择适用于应用领域同时又为 C++所支持的范型来形成方案领域的结构。

全书涵盖了这样一些重要的概念或方法，需要读者重点把握：软件族、共同性、积极或消极差异性、领域分析、领域划分、领域词汇表、面向对象的分析、复用、迭代、绑定、依赖关系图、范型混合，以及模式等。

本书第 1 章分析多范型设计的必要性，第 2 章和第 3 章分别介绍共同性和差异性分析。第 4 章解释如何使用领域分析来找到应用领域中的抽象。第 5 章说明怎样将领域工程的原则用作对象范型的抽象技术的基础。第 6 章应用“分析”来描述“方案领域”的特征，并且将 C++结构放到形成共同性和差异性基础的正式框架中。第 7 章综合考虑前几章的内容，介绍了结构复杂性不同的设计问题的分类，以及可作为基于领域分析和多范型设计技术基础的高级活动集合。第 8 章研究结构复杂的设计，并介绍如何组合各种范型。第 9 章补充说明了流行的设计模式与领域工程之间的关系，提供了对模式、领域工程以及二者关系的新的认识。

阅读本书需要读者一定程度地掌握了 C++编程语言，并具有一定的面向对象编程的经验。书中大量引用了相关著作，读者朋友可根据需要参阅参考文献中相关书目。

经过两个月的艰苦努力，本书的译稿终于完成并交付出版。译者由衷地希望读者朋友们能从这本译作中获取新的知识，提高自己的编程能力。本书由鄢爱兰、周辉主译，李明对全书进行了审校。欧阳宇、谢君英、谢俊、盛海燕、谢小花、李常青为本书的翻译做了很多工作。另外，吴晓晶、林知原也为本书的翻译工作提供了很大帮助，盛海燕和谢小花还完成了本书的初排和代码的录入工作，在此对他们表示衷心的感谢。由于译者水平和时间所限，译文中难免出现错误和不当之处，敬请广大读者批评指正。

译　者

2003 年 7 月

试读结束，需要全本PDF请购买 www.ertongbook.com

前　言

我很少像这次一样，花如此多的精力来为一本书命名。随着对原稿的修改，书名逐渐从强调一组基本元素（领域、工程、多范型、分析、设计、编程，以及 C++）变为总是强调一个或两个概念。出版社担心，使用大家所不熟悉的术语［“领域工程”（domain engineering）］难以迎合目标市场。审阅者 Tim Budd 担心他使用的“多范型”（multi-paradigm）会与本书中所使用的“领域工程”相混淆。而我则关心类似“分析”（analysis）这样的术语，因为我希望本书能对普通程序员（本书就是要介绍他们想要了解的问题）有所帮助。Tim Budd 婉转地提出，我们的要求分歧大到足以容纳“多范型”定义的宽泛区间。我坚持使用一个强调程序员的作用、而不是强调方法论者作用的书名。结果我们愉快地达成了一致，也就是形成了现在的书名——“C++多范型设计”。

我向来不考虑包含“模式”（pattern）、“对象”（object）、CORBA、“组件”（component）和 Java 类似措辞的书名。多范型试图比任何技术和方法挖掘得更深，以阐明软件的抽象和设计的基本问题。什么是多范型？分析、设计和实现之间是什么关系？这些问题涉及到支撑编程的基本范型的抽象基础。

一个最基本的问题是——什么是抽象？抽象是软件设计的关键工具之一。对于管理计算机系统无限的、不断增长的复杂性来说，抽象是必需的。此问题的共同答案通常都与“对象”有些关系，因此这也反映了过去十年或二十年形成的、用于支持面向对象技术的文献和工具的总体。但这种回答忽略了程序员经常使用的“共同设计结构”，它们并非是面向对象的：模板、重载函数系列、模块、通用函数及其他一些内容。C++中这种用法非常普遍，尽管它并不是惟一的。

有些抽象原则是与所有的这些技术所共同拥有的。每项技术都是依据它们共同的属性（包括各个实体互不相同的规律性）进行分组抽象的不同方法。对有些技术来说，共同性捕获了系统经常出现的外部属性，这些属性与系统的领域有密切的关系。对其他一些技术来说，共同性有助于使分析在递归解决方案中为领域所揭示的隐式结构规则化。

多范型设计兼具上述两个方面的特性。例如，多范型设计技术（被称为面向对象的设计）将对象分组为“类”，它描述了这些对象的共同结构和行为的特性。这种技术将类分组为层次或图表，它们反映了结构和行为中的共同性，同时允许结构和实现某个给定行为的算法中有规则的差异存在。模板可以用共性和差异的不同特性进行描述。共同性和差异性提供了宽泛而又简单的抽象模型，比对象和类都要宽泛，足以处理大部分的设计和编程技术。

共同性和差异性对软件设计模型来说并非是什么新鲜的事物。Parna 关于软件族的概念[Parnas 1976]至少在二十年以前就产生了。族是共同性相关的软件元素的集合，族的单个成员依据各自的变化而不同。从软件族中形成的设计思想，经常可以在被广泛接受的编程语言中获得体现。这里有一些很好的例子：模块、类和对象，以及通用结构。Lai 和 Weiss 在专用语言环境方面的努力将这一思想发挥到了极限[Weiss 1999]。注重软件族的发现的所谓分析活动，与注重如何表达这些抽象的所谓编码活动，总是紧密融合在一起的。多范型设计显然认可了语言、设计、领域结构，及它们表达共性和差异的方式之间的这种紧密联系。

我们在“领域分析”（它是另一个拥有很长历史的领域[Neighbors 1980]）的活动中找到软件族。软件复用是领域分析的最初目标，此目标与软件族正好相符。多范型设计显然注重对于复用比较重要的问题。为帮助设计者思考能对预期市场的范围自适应的软件，多范型设计将共同性（假设它不变）与差异性（假设它确实发生变化）显式地分离开来。我们要努力进行领域分析，而不止于一般意义上的分析。我们要设计抽象族，而不是设计抽象。如果进展顺利，这种设计方法将使系统长期获得更简单的维护（假设我们对差异性预测得很好）和更有弹性的架构（每次进行修改时不必“挖出”系统的“根基”）。当然，多范型开发只是帮助支持复用的技术目标的工具。只有在组织问题、市场问题和软件经济学的更大型的环境中，才能产生高效的复用。

我们使用共同性和差异性这些基础来确定“范型”（paradigm）的概念。尽管范型广泛使用在当代的软件设计中，但它只是根据共性和差异属性组织系统抽象的一种方法。对象范型根据基于结构中共同性的抽象，以及系统和算法中的行为和差异来组织系统。模板范型是基于族成员的结构共同性的，它将变化因素显式地转化成模板参数。重载函数形成这样的族：族成员共享相同的名字和语义，族的每个成员依据各自的形参类型相区别。

C++是一种支持多范型（类、重载函数、模板、模块、普通的程序性编程等）的编程语言。这正如 C++的创造者 Bjarne Stroustrup 期望的那样。多数程序员使用了对象以外的 C++特性（尽管有些人过度滥用这些特性，而另有一些人则在应当使用其他语言特性提供的更自然的设计表达时，对设计强制套用了面向对象的模式）。John Barton 和 Lee

Nackman[Barton 1994]的强大的模板代码或许是对多范型设计的不错尝试。

尽管 Stroustrup 指明了 C++是一种多范型语言，但 C++中并没有创建一种与丰富的 C++特征相配的设计方法的认真尝试。C++提供了多范型编程的一个特别丰富而生动的例子，其他的编程语言也同样获得了多范型开发的机会。当前的设计风格与当前实践中反映出来的 C++特性的预期用法之间存在一个鸿沟。本书使用简单的记法和词汇表来帮助开发者将多种范型有启发性地组合在一起，为这个鸿沟架起一座桥梁。

1995 年 9 月，我在 DePaul 大学演讲期间，系主任 Helmut Epp 先生建议本书使用“meta-design”这个术语，因为本书首先关心的是确定适合领域（软件正是为该领域而开发）的设计技术。这种看法对本书所采用的方法很有用，实际上，它描述了多数开发者是如何开始设计的。开发者首先必须决定使用什么范型，然后可以使用范型的规则和工具划分适合它们使用的系统。领域不仅属于系统构架者和设计者，它还属于普通的程序员。

决定使用什么范型是一回事，使用工具表达给定范型的抽象是另外一回事。我们可以使用共性和差异的原则来分析应用领域，以将其分成多个子领域，每个子领域适合用某种具体的范型进行设计。开发阶段进行的这种划分通常称为“分析”。但是，因为它试图创建实现技术能够表达出的抽象，所以将它当作设计的初始阶段更好。并非所有的实现工具（编程语言和其他工具，比如应用程序生成器）都可以表达全部的范型。因此，进行领域分析，（而不仅仅是应用领域的分析，还包括解决领域的分析）是很重要的。多范型设计将领域分析变成一种很明确的活动。解决领域的分析是多范型设计的“meta-design”本质的另一个方面。

本书有诸多特点。本书不是一种综合的设计方法、软件开发的生命周期模型、或是准备进行设计的方法。多数优秀的新设计都与以前的设计有所不同。我们很少会面临一个全新的或惟一的软件问题。对新系统中的每个模块都使用本书的记法和技术既不合适，也浪费时间。但在新问题出现的时候，我们应当准备好去面对它们，发现其中的结构，并将对它的理解运用到设计和实现中。而且，多范型设计的记法和技术为文件设计（可以为面向对象技术增加其他范型）提供了统一的方式。

多范型设计是一种技巧，它既不完全是一门艺术，也不完全是一种严格的规范。本书介绍了支持开发者思考过程的记法、图表和设计模型。由于有了所有的这些记法，我们总想尝试使用它们。多范型设计是产生架构的活动的集合，架构与各个部分之间的关系有关，但架构也与效用和审美有关——没有使用通常方法的优秀软件的属性。要获得良好的体验，有一个关键的地方，我并不期望多范型技术能够推动自动设计和编码。良好的体验部分来自经验，部分来自于良好的洞察力。因此，这也不是一本入门书。读者

应当具有一年或两年在大型系统中使用C++进行面向对象（至少）编程的经验。

把这些技术与常识一起使用，将很好地补充人们的判断力和经验。如果读者发现应用这些技术得到了一个既不喜欢又无法理解的设计，那么请不要采用这个设计。多范型设计技术只是工具，而不是指令。但所有的读者都应当通过本书认识到这样一点：对象范型或其他范型只是一组有用的范型，设计所表达的结构必须比任何单个范型所能表达的结构更宽泛。

本书的结构安排

为建立新的概念，并增加读者对领域工程和多范型技术的理解，每一章都建立在其前面章节的基础之上。多数读者都应按顺序阅读各章，然后返回具体章节作为参考。

第1章到第7章是基础章节，为领域工程打下了基础。

- 第1章介绍词汇表，分析多范型设计的必要性，并为领域工程打下高级的基础。
- 第2章和第3章分别介绍共同性和差异性分析。这两个概念在应用中是一起使用的，但为了介绍方便并强调其语言的精妙，本书中将二者分开讲述。第3章介绍了积极和消极差异性的主要概念。
- 第4章解释如何使用领域分析来找到应用领域中的抽象，这需要建立在前面章节所介绍的方法的基础上。
- 第5章说明领域工程的原则是怎样被用作对象范型的抽象技术的基础的。
- 第6章是重要的一章，因为本章以不同寻常的方式应用“分析”来描述“方案领域”的特征，并将C++结构放到形成共同性和差异性基础的正式框架中。
- 第7章将前面所有章节连接成思考设计的一个内聚框架中。它介绍了拥有不同的结构复杂性的设计问题的分类。同时，介绍了用以指导好的设计、并可作为基于领域分析和多范型设计技术基础的高级活动集合。本章论及这样一种简单的情况：每个领域都可用单个范型，并在很大程度上独立于其他领域进行开发。
- 第8章更进一步，研究了结构复杂的设计，简单的分而治之的方法对此并不奏效。本章为这种递归的、结构复杂的设计提供了多个示例，并对“打破递归”进行了启发性的介绍。
- 第9章补充说明了流行的设计模式与领域工程之间的关系。这种对比将会引起很多读者的兴趣，本章还提供了对模式、领域工程以及二者关系的新的认识。尽管本章并不是理解领域工程的核心内容，但它对理解用对象和其他范型（使用或者不使用领域工程）将多种模式结合在一起的当代设计实践非常重要。

本书中的领域名使用这样的字体：**Text Buffer**。模式名称也使用了这样的字体：**Template Method**。成员函数、类名和代码示例使用这样的字体：`Code Example`。

本书中的分类图表遵循统一建模语言（UML）符号。

致谢

非常感谢所有这些通过对话、评论和反馈提高了原稿质量的朋友。Van Den Broecke、Frank Buschmann、Paul Chisholm、Russell Corfman、David Cuka、Cay Horstman、Andrew Klein、Andrew Koenig、Stan Lippman、Tom Lyons、Lee Nackman、Brett Schuchert、Larry Schutte、Herb Sutter、Steve Vinoski 及 David Weiss 都在不同层次（从高层次的总体设计到最小的 C++细节）为本书提供了评论意见。非常感谢他们的帮助。我还要感谢孜孜不倦、充满耐心的编辑 Debbie Lafferty，她还参与了我的第一本书的编辑工作。与她一起工作真是一种乐趣。还要特别感谢我的制作编辑（production editor）Jacquelyn Young；感谢本书的审稿 Laura Michaels；感谢本书的排版 Kim Arney。与 Lalita Jagadeesan 的探讨激发我创建了多个有用的例子。还要特别感谢采编 Tom Stone，感谢他早期给我的建议，以及很早就在 Addison-Wesley 中给予本书的热情支持。特别感谢 Andrew Klein 在 UML 图表上给予的帮助。最后，要特别感谢贝尔实验室的管理层和我的同事们，尤其是 David Weiss，感谢他们让我分享他们的著作，以及他们所提供的支持和鼓励。

目　录

试读结束，需要全本PDF请购买 www.ertongbook.com

第 1 章
简介：多范型的必要性

本章介绍软件开发中的多范型，并将领域工程和多范型设计与面向对象设计相关的前沿技术和新兴主题结合起来。本章还介绍了以下重要词汇：软件族（software family）、共同性（commonality）和差异性（variability）。

1.1 领域工程和多范型

当代的软件似乎将“设计”与“对象”同等看待。对象范型提供了一种新的强大工具，使我们获得许多应用领域的共同抽象。同时，面向对象的设计已经成为一种得到普遍应用的工具。为实现这些设计，C++在本书编写之际仍然是首选的语言。然而，C++并不是完全面向对象的，并且多数实现使用了 C++的非面向对象特性。这表明，多数的设计实际上有一个非面向对象的重要的构件。假定大多数的项目使用面向对象的设计方法，那么这些非面向对象的结构从何而来？

这些问题涉及到软件设计的核心。设计的首要目标是满足业务需求，这种需求通常由用户期望来支配。设计的成功与否要根据对所建系统要服务的业务或领域的充分了解来判定。然而设计还有其他目标。比如，设计的实现应当可以理解，并易于建立（按照爱因斯坦的解释，应当是尽可能易于建立，而不是更容易建立）。设计还要尽量建立可以随时间演变、能适应新市场和新应用的系统。

实际上，所有这些设计目标是同一个基本原理的不同结果。许多目标都与业务的结构和业务结构如何演变有关。我们对业务的了解增加了满足客户需求的可能性。这种了

解也是系统结构以及有助于对设计进行表达和构型的词汇表的基础。通过了解业务的广泛、稳定的方面，我们可以设计出能良好演进的结构。

领域工程（domain engineering）是一种软件设计规程，注重对业务（一个领域）的抽象，目的是实现设计和工件（artifact）的复用（reuse）。复用是一个成功的设计的良好体现之一，并且，良好的设计技术会带来良好的复用，也会随时间的推移而带来可扩展性和可维护性。

多范型设计包含对象范型的许多目标，但这些目标的作用范围不同。对象范型也注重可扩展性和重用性。实现这一目标的方法是：将设计的共同和稳定的部分与不同和不确定的部分分离开来。经过简化后我们发现，稳定不变的行为和数据都包含在基类（尤其是抽象基类）中，而变量都包含在派生类中。当设计能够按照行为和数据结构这条线分成共同（稳定）部分与差异（不确定）部分时，上面的这种方法将非常适用。但是还有其他许多有用的方法，可以区分哪些是共同部分，哪些是有差异的部分。例如，我们可能要在一个相同的数据结构中选择 `short` 或 `long`，此时 C++模板非常适用——没有涉及到面向对象的问题。函数重载、函数模板及 C++语言的其他特性表现出其他种类的共同性和变化，它们比对象范型更加宽广。领域工程覆盖了所有这些需要考虑的事项。

设计是解决问题过程中的结构方面，包括对问题的抽象、划分及系统建模等活动，从而使设计者能够理解所要解决的问题。通过使用业务经验法则、耦合原理、内聚和子类型化以及对对象的非典型、非正式的任务分配，面向对象的设计能帮助我们“发现对象”。在一个非完全面向对象的领域中，我们究竟该如何建立设计抽象和实现模块呢？首先，很重要的一点是要认识到我们要处理的不再是单范型，而是多范型，每个范型都有各自抽象、划分和建模的规则。其次，很有用的一点是要知道大部分的软件范型——当然是指 C++所支持的这些软件范型——的特性都可以用更加通用的设计（涉及共同性和差异性）来描述。我们称之为“领域分析”（domain analysis）。领域分析建立了业务的一种模型（多种模型之一）。我们可以用这种模型建立解决方案的结构。设计的第一个要点是了解该模型构件的共同性和差异性。

领域分析揭示了抽象和工件的分组（二者通过它们的共同性——或许是通过它们差异性的相似本质——结合在一起）。这些分组被称为“族”（family）。如果说面向对象的设计是一个“发现对象”的过程，那么领域分析则是“发现族”的过程。注意，面向对象的设计是发现族的一种特殊情况：类为对象族，类层次为类族。因为它们有共同的地方，所以将它们组合在一起。除了通过这种密切关系，我们还可以通过对象范型使用“准则”（criteria）来发现其他重要的非面向对象的族——这就是多范型设计的重要性所在。

设计的第二个焦点是如何将这些共同性和差异性匹配到实现的技术结构（比如类、

函数、模板、类层次及数据结构）中。这称为“应用工程”（application engineering），也是多数人一听到“设计”这个词就会联想到的内容。如果我们理解了具体的范型所表示的共同性和差异性的种类，那么应用工程就可以使用领域分析的结果，为系统的架构和实现选择适当的范型。

本书展示了能够支持此类设计过程的形式记法、概念和简单表示法的一个框架。书中广泛介绍了基于领域分析的设计，以及该领域设计的一个实现。我将这种方法称为“多范型设计”（multi-paradigm design），它是针对领域工程（建立在C++编程语言所支持的一个小而丰富的范型集合之上）的具体方法。此方法可以推广至其他的编程语言（尤其是具有丰富类型的编程语言）及其他实现技术。这里强调C++是因为对“焦点”（focus）和“实用性”（pragmatism）的兴趣。

总之，领域分析是区分软件族的一套方法，而应用工程是实现和管理软件族的一套方法。领域分析和应用工程结合起来形成了称之为“领域工程”（domain engineering）的规范。多范型设计是领域工程的一种形式，它同时为应用领域和方案领域提供领域分析。在多范型设计中，应用工程建立在已有的工具和范型之上。

多范型设计和领域工程重新回到抽象的第一个原则和拓展对象以外的解空间的设计上来。这种拓展对设计、架构和实现很有意义。下一节按照共同性和变化的原则，建立了关于这些基本概念的词汇表。掌握了这些基本概念，我们就能更直观地理解大部分多范型设计的原则。

1.2 设计、分析、领域、族：术语定义

按照软件工程的惯例，“设计”（design）是指在分析和实现之间所进行的活动。“分析”（analysis）是指得出用户需求和用户期望的活动。在被软件采用之前，这些术语包含更广阔的含义。多范型设计正是迎合了这些术语更广阔的含义。本节介绍多范型设计的关键字和领域工程的词汇表。

在领域工程中，分析不仅是指得出给定客户需求的应用分析。我们还必须确定工具箱中需要哪些工具，必须根据我们的安排考虑工具和技术的局限性和优点，必须使用它们让设计者和用户都能获得最大利益。例如，设计者何时需要考虑是使用 **Unix** 库还是 **MFC**（Microsoft Foundation Classe）？这看起来像是一个实现决策，然而它通常是市场需求中的一个因素。很显然，这是影响最终产品的架构和功能性的一个决策。此分析必须在我们将其结果转化成一个实现之前进行。分析当然应该先于任何传统设计定义中的“设计”（即从需求中得出实现结构的过程）。但是，正如设计可以在实现的后期继续进

行一样，分析也可以在设计活动的同时继续进行。正在进行的开发可以洞察工具、方法和范型的选择适当与否。

1.2.1 分析

在多范型设计中，我们要讨论关于“应用分析”和“方案分析”的内容。应用分析是问题空间的“传统”分析。方案分析借用了应用分析的抽象工具和形式化方法，并将它们应用到“方案”空间中。同时理解“开发等式”（development equation）的两端有助于我们对适当的工作选用适当的工具，从而提高成功的可能性，甚至在应用分析阶段，设计者就需要确定工程工具箱的内容。

通常，术语“分析”表示“理解问题”的意思。术语“面向对象的分析”渐渐形成通用方法，这表明对象范型的设计原则为整理问题提供了有效的工具。不幸的是，分析期间使用的任何设计范型都会影响实现中使用的范型，即使实际上其他某个范型可以提供更好的划分。解决这个两难问题的方法是，在分析中避免任何的设计偏见。然而，这又导致分析不能直接预见使用可用工具的简单实现。通过考虑分析期间的所有实现范型，多范型设计解决了这个两难的问题。更确切地讲，该分析考虑了抽象和划分（这两者是大部分软件设计范型的基础）的基本原则。我们并不是在整个分析期间都进行划分，那样需要一个提供了划分规则的范型。分析收集这些信息来支持对一个和多个范型的选择。

1.2.2 设计

“设计”是为给定问题提供解决方案结构的活动。设计以一个问题开始，以一种解决方案结束。这里，“问题”表示当前状态与期望状态之间的一种“失配”（mismatch），这是一个足够广泛的定义，包括了 bug 修复和（系统）增强。设计必须与实效相权衡——由业务和易控制性等现实考虑所强加给设计的约束。与建立设计以及与某些机械工程规范类似，软件中的解决方案——设计的成果——包括“架构”（architecture）和实现。

我们仍然将设计看作这样一种活动：它带着我们从需求陈述阶段进入到实现阶段，但我们扩大了它的作用范围。由于分析需要同时仔细审查应用和解决方案，所以从某种程度上讲，分析也是一种设计活动。因为在多范型设计看来，实现工具就好比编程语言的表达结构，所以它涉及许多实现的问题。设计和分析都无法摆脱这种模型。这种看法使从应用结构到解决方案结构的过渡更加平滑，从而有助于避免结构化技术中数据流图之后发生的“相移”（phase shift）；同时避免了将解决方案结构与问题结构匹配的错误预期。这是假定分析类直接映射到 C++类的早期面向对象方法的通病。

试读结束，需要全本PDF请购买 www.ertongbook.com