第 12 章

Software Engineering: A Practitioner's Approach, Eighth Edition

设计概念

要点浏览

概念：几乎每位工程师都希望做设计工作。设计体现了创造性——利益相关者的需求、业务要求和技术考虑都集中地体现在产品或系统的形成中。设计创建了软件的表示或模型，但与需求模型（注重描述所需的数据、功能和行为）不同，设计模型提供了软件体系结构、数据结构、接口和构件的细节，而这些都是实现系统所必需的。

人员：软件工程师负责处理每一项设计任务。

重要性：设计是让软件工程师为将要构建的系统或产品建立模型。在生成代码、进行测试以及大量最终用户使用之前，要对模型的质量进行评估，并进行改进。软件质量是在设计中建立的。

步骤：设计可以采用很多不同的方式描述软件。首先，必须表示系统或产品的体系结构；其次，为各类接口建模，这些接口在软件和最终用户、软件和其他系统与设备以及软件和自身组成的构件之间起到连接作用；最后，设计构成系统的软件构件。每个视图表现了不同的设计活动，但所有的视图都要遵循一组指导软件设计工作的基本设计概念。

工作产品：在软件设计过程中，包含体系结构、接口、构件级和部署表示的设计模型是主要的工作产品。

质量保证措施：软件团队从以下各方面来评估设计模型：确定设计模型是否存在错误、不一致或遗漏；是否存在更好的可选方案；设计模型是否可以在已经设定的约束、时间进度和成本内实现。

软件设计包括一系列原理、概念和实践，可以指导高质量的系统或产品开发。设计原理建立了指导设计工作的最重要原则。在运用设计实践的技术和方法之前，必须先理解设计概念，设计实践本身会产生软件的各种表示，以指导随后的构建活动。

设计是软件工程是否成功的关键。在 20 世纪 90 年代早期，Lotus 1-2-3 的创始人 Mitch Kapor 在《Dr. Dobbs》杂志上发表了如下“软件设计宣言”：

什么是设计？设计是你身处两个世界——技术世界和人类的目标世界，而你尝试将这两个世界结合在一起……

罗马建筑批评家 Vitruvius 提出了这样一个观念：设计良好的建筑应该展示出坚固、适用和令人愉悦的特点。对好的软件来说也是如此。坚固：程序应该不含任何妨碍其功能的缺陷。适用：程序应该符合开发的目标。愉悦：使用程序的体验应是愉快的。本章，我们开始介绍软件设计理论。

关键概念

抽象
体系结构
方面
内聚
数据设计
设计过程
功能独立
良好的设计
信息隐蔽
模块化
面向对象的设计
模式
质量属性
质量指导原则

设计的目标是创作出坚固、适用和令人愉悦的模型或表示。为此，设计师必须先实现多样化，然后再进行聚合。Belady[Bel81] 指出，“多样化是指要获取所有方案和设计的原始资料，包括目录、教科书和头脑中的构件、构件方案和知识。”在各种信息汇聚在一起之后，应从满足需求工程和分析模型（第 8 ～ 11 章）所定义的需求中挑选适合的元素。此时，考虑并取舍候选方案，然后进行聚合，使之成为“构件的某种特定的配置，从而创建最终的产品”[Bel81]。

关键概念
重构
关注点分离
软件设计
逐步求精

多样化和聚合需要直觉和判断力，其质量取决于构建类似实体的经验、一系列指导模型演化方式的原则和启发、一系列质量评价的标准以及导出最终设计表示的迭代过程。

引述 软件工程最常见的奇迹是从分析到设计以及从设计到代码的转换。
Richard Due'

新的方法不断出现，分析过程逐步优化，人们对设计的理解也日渐广泛，随之而来的是软件设计的发展和变更[⊖]。即使是今天，大多数软件设计方法都缺少那些更经典的工程设计学科所具有的深度、灵活性和定量性。然而，软件设计的方法是存在的，设计质量的标准是可以获得的，设计表示法也是能够应用的。在本章中，我们将探讨可以应用于所有软件设计的基本概念和原则、设计模型的元素以及模式对设计过程的影响。在第 12 ～ 18 章中，我们将介绍应用于体系结构、接口和构件级设计的多种软件设计方法，也会介绍基于模式和面向 Web 的设计方法。

12.1　软件工程中的设计

软件设计在软件工程过程中属于核心技术，并且它的应用与所使用的软件过程模型无关。一旦对软件需求进行分析和建模，软件设计就开始了。软件设计是建模活动的最后一个软件工程活动，接着便要进入构建阶段（编码和测试）。

225

需求模型的每个元素（第 9 ～ 11 章）都提供了创建四种设计模型所必需的信息，这四种设计模型是完整的设计规格说明所必需的。软件设计过程中的信息流如图 12-1 所示。由基于场景的元素、基于类的元素和行为元素所表示的需求模型是设计任务的输入。使用后续章节所讨论的设计表示法和设计方法，将得到数据或类的设计、体系结构设计、接口设计和构件设计。

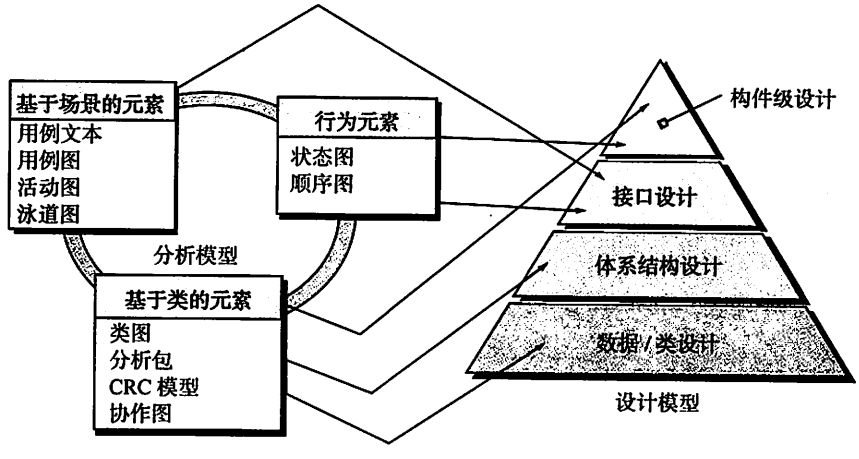

图 12-1　从需求模型到设计模型的转换

⊖ 对软件设计原理感兴趣的读者可能会对 Philippe Kruchen 关于“后现代”设计的有趣讨论感兴趣 [Kru05]。

数据设计或类设计将类模型（第 10 章）转化为设计类的实现以及软件实现所要求的数据结构。CRC 图中定义的对象和关系，以及类属性和其他表示法描述的详细数据内容为数据设计活动提供了基础。在软件体系结构设计中也可能会进行部分类的设计，更详细的类设计则将在设计每个软件构件时进行。

体系结构设计定义了软件的主要结构化元素之间的关系、可满足系统需求的体系结构风格和模式（第 13 章）以及影响体系结构实现方式的约束[Sha96]。体系结构设计表示可以从需求模型导出，该设计表示基于的是计算机系统的框架。

接口设计描述了软件和协作系统之间、软件和使用人员之间是如何通信的。接口意味着信息流（如数据和控制）和特定的行为类型。因此，使用场景和行为模型为接口设计提供了大量的信息。

构件级设计将软件体系结构的结构化元素变换为对软件构件的过程性描述。从基于类的模型和行为模型中获得的信息是构件设计的基础。

设计过程中所做出的决策将最终影响软件构建的成功与否，更重要的是，会影响软件维护的难易程度。但是，设计为什么如此重要呢？

软件设计的重要性可以用一个词来表达——质量。在软件工程中，设计是质量形成的地方，设计提供了可以用于质量评估的软件表示，设计是将利益相关者的需求准确地转化为最终软件产品或系统的唯一方法。软件设计是所有软件工程活动和随后的软件支持活动的基础。没有设计，将会存在构建不稳定系统的风险，这样的系统稍做改动就无法运行，而且难以测试，直到软件过程的后期才能评估其质量，而那时时间已经不够并且已经花费了大量经费。

建议　软件设计应该总是先考虑数据，数据是所有其他设计元素的基础。基础奠定之后，才能进行体系结构设计。再之后，才能进行其他设计任务。

226

引述　有两种构建软件设计的方式：一种是使其尽可能简单以致明显没有不足；另一种是使其尽可能复杂以致没有明显的不足。第一种方式更为困难。

C. A. R Hoare

SafeHome　设计与编码

[场景] Jamie 的房间，团队成员准备将需求转化为设计。

[人物] Jamie、Vinod 和 Ed，SafeHome 软件工程团队所有成员。

[对话]

Jamie：大家都知道，Doug（团队管理者）沉迷于设计。老实说，我真正喜欢的是编码。如果给我 C++ 或者 Java，我会非常高兴。

Ed：不，你喜欢设计。

Jamie：你没听我说吗？编码才是我喜欢的。

Vinod：我想 Ed 的意思是你并不是真的喜欢编码，而是喜欢设计，并喜欢用代码表达设计。代码是你用来表示设计的语言。

Jamie：那有什么问题吗？

Vinod：抽象层。

Jamie：嗯？

Ed：程序设计语言有利于表示诸如数据结构和算法的细节，但不利于表示体系结构或者构件之间的协作……就是这个意思。

Vinod：一个糟糕的体系结构甚至能够摧毁最好的代码。

Jamie（思考片刻）：那么，你们的意思是我不能用代码表示体系结构……这不是事实。

Vinod：你肯定能在代码中隐含体系结构，但在大部分程序设计语言中，通过检查

代码而快速看到体系结构的全貌是相当困难的。

227 **Ed**：那正是我们在开始编码之前需要的。

Jamie：我同意，也许设计和编码不同，但我仍然更喜欢编码。

12.2 设计过程

软件设计是一个迭代的过程，通过这个过程，需求被变换为用于构建软件的“蓝图”。刚开始，蓝图描述了软件的整体视图，也就是说，设计是在高抽象层次上的表达——在该层次上可以直接跟踪到特定的系统目标以及更详细的数据、功能和行为需求。随着设计迭代的开始，后续的细化导致更低抽象层次的设计表示。这些表示仍然能够跟踪到需求，但是连接更加错综复杂了。

12.2.1 软件质量指导原则和属性

在整个设计过程中，我们使用第 20 章中讨论的一系列技术评审来评估设计演化的质量。McGlaughlin[McG91] 提出了可以指导良好设计演化的三个特征：

- 设计应当实现所有包含在需求模型中的明确需求，而且必须满足利益相关者期望的所有隐含需求。
- 对于那些编码者和测试者以及随后的软件维护者而言，设计应当是可读的、可理解的指南。
- 设计应当提供软件的全貌，从实现的角度对数据域、功能域和行为域进行说明。

引述 编写一段能工作的灵巧的代码是一回事，而设计能支持某个长久业务的东西则完全是另一回事。

C. Ferguson

以上每一个特征实际上都是设计过程的目标，但是如何达到这些目标呢？

质量指导原则。为了评估某个设计表示的质量，软件团队中的成员必须建立良好设计的技术标准。在 12.3 节中，我们将讨论设计概念，这些概念也可以作为软件质量的标准。现在，考虑下面的指导原则：

1. 设计应展现出这样一种体系结构：（1）已经使用可识别的体系结构风格或模式创建；（2）由能够展现出良好设计特征的构件构成（将在本章后面讨论）；（3）能够以演化的方式[⊖]实现，从而便于实施与

228 测试。

2. 设计应该模块化，也就是说，应将软件逻辑地划分为元素或子系统。
3. 设计应该包含数据、体系结构、接口和构件的清晰表示。
4. 设计应导出数据结构，这些数据结构适用于要实现的类，并从可识别的数据模式提取。
5. 设计应导出显示独立功能特征的构件。
6. 设计应导出接口，这些接口降低了构件之间以及构件与外部环境之间连接的复杂性。
7. 设计的导出应采用可重复的方法进行，这些方法由软件需求分析过程中获取的信息而产生。

提问 良好设计的特征是什么？

引述 设计不仅仅是它看上去以及感觉上如何，还要看它是如何运作的。

Steve Jobs

⊖ 对于较小的系统，设计有时也可以线性地进行开发。

8. 应使用能够有效传达其意义的表示法来表达设计。

满足这些设计原则并非依靠偶然性，而是通过应用基本的设计原则、系统化的方法学和严格的评审来得到保证。

信息栏 评估设计质量——技术评审

设计之所以重要，在于它允许一个软件团队在软件实现前评估软件的质量[⊖]——此时修正错误、遗漏或不一致都不困难且代价不高。但是我们如何在设计过程中评估质量呢？此时，不可能去测试软件，因为还没有可执行的软件，那怎么办？

在设计阶段，可以通过开展一系列的技术评审（Technical Review，TR）来评估质量。技术评审将在第20章[⊜]中进行详细讨论，这里先概述一下该技术。技术评审是由软件团队成员召开的会议。通常，根据将要评审的设计信息的范围，选择2～4人参与。每人扮演一个角色：评审组长策划会议、拟定议程并主持会议；记录员做笔记以保证没有遗漏；制作人是指其工作产品（例如某个软件构件的设计）被评审的人。在会议之前，评审小组的每个成员都会收到一份设计工作产品的拷贝并要求阅读，寻找错误、遗漏或含糊不清的地方。目的是在会议开始时注意到工作产品中的所有问题，以便能够在开始实现该产品之前修正这些问题。技术评审通常持续60～90分钟。评审结束时，评审小组要确认在设计工作产品能够被认可为最终设计模型的一部分之前，是否还需要进一步的行动。

229

质量属性。Hewlett-Packard[Gra87]制订了一系列的软件质量属性，并取其首字母组合为FURPS，其中各字母分别代表功能性（functionality）、易用性（usability）、可靠性（reliability）、性能（performance）及可支持性（supportability）。FURPS质量属性体现了所有软件设计的目标：

- 功能性通过评估程序的特征集和能力、所提交功能的通用性以及整个系统的安全性来评估。
- 易用性通过考虑人员因素（第6～15章）、整体美感、一致性和文档来评估。
- 可靠性通过测量故障的频率和严重性、输出结果的精确性、平均故障时间（Mean-Time-To-Failure，MTTF）、故障恢复能力和程序的可预见性来评估。
- 性能通过考虑处理速度、响应时间、资源消耗、吞吐量和效率来度量。
- 可支持性综合了可扩展性、可适应性和可用性。这三个属性体现了一个更通用的术语：可维护性。此外，还包括可测试性、兼容性、可配置性（组织和控制软件配置元素的能力，第29章）、系统安装的简易性和问题定位的容易性。

引述 质量不是那些被束之高阁的观赏品，也不是像圣诞树上的闪亮金箔那样的装饰品。

Robert Pirsig

建议 软件设计师往往注重于问题的解决。不要忘记的是：FURPS属性总是问题的一部分，因此必须要考虑到这些属性。

⊖ 第30章讨论的质量因素可以帮助评审小组评估质量。

⊜ 设计时，可以提前看一下第20章。技术评审是设计过程中的关键部分，也是达到设计质量的重要方法。

在进行软件设计时，并不是每个软件质量属性都具有相同的权重。有的应用问题可能强调功能性，特别突出安全性；有的应用问题可能要求性能，特别突出处理速度；还有的可能关注可靠性。抛开权重不谈，重要的是：必须在设计开始时就考虑这些质量属性，而不是在设计完成后和构建已经开始时才考虑。

12.2.2　软件设计的演化

软件设计的演化是一个持续的过程，它已经经历了 60 多年的发展。早期的设计工作注重模块化程序开发的标准 [Den73] 和以自顶向下的“结构化”方式对软件结构进行求精的方法（[Wir71]，[Dah72]，[Mil72]）。较新的设计方法（如 [Jac92]、[Gam95]）提出了进行设计导出的面向对象方法。近年来，软件体系结构 [Kru06] 和可用于实施软件体系结构及较低级别设计抽象（如 [Hol06]、[Sha05]）的设计模式已经成为软件设计的重点。面向方面的方法（如 [Cla95]、[Jac04]）、模型驱动开发 [Sch06] 以及测试驱动开发 [Ast04] 日益受到重视，这些方法强调在设计中实现更有效的模块化和体系结构的技术。

引述　设计师知道当设计中不再需要减去任何东西，而并不是不再需要增加任何东西的时候，他的设计就很完美了。

Antoine de St-Exupery

230

许多设计方法刚刚被提出，它们从工作中产生，并正被运用于整个行业。正如第 9 ～ 11 章提出的分析方法，每一种软件设计方法引入了独特的启发式和表示法，同时也引入了某种标定软件质量特征的狭隘观点。不过，这些方法都有一些共同的特征：（1）将需求模型转化为设计表示的方法；（2）表示功能性构件及它们之间接口的表示法；（3）细化和分割的启发式方法；（4）质量评估的指导原则。

提问　所有设计方法的共同设计特征是什么？

无论使用哪种设计方法，都应该将一套基本概念运用到数据设计、体系结构设计、接口设计和构件级设计，这些基本概念将在后面几节中介绍。

任务集　通用设计任务集

1. 检查信息域模型，并为数据对象及其属性设计合适的数据结构。
2. 使用分析模型选择一种适用于软件的体系结构风格（模式）。
3. 将分析模型分割为若干设计子系统，并在体系结构内分配这些子系统：
   - 确保每个子系统是功能内聚的。
   - 设计子系统接口。
   - 为每个子系统分配分析类或功能。
4. 创建一系列的设计类或构件：
   - 将分析类描述转化为设计类。
   - 根据设计标准检查每个设计类，考虑继承问题。
   - 定义与每个设计类相关的方法和消息。
   - 评估设计类或子系统，并为这些类或子系统选择设计模式。
   - 评审设计类，并在需要时进行修改。
5. 设计外部系统或设备所需要的所有接口。
6. 设计用户接口：
   - 评审任务分析的结果。
   - 基于用户场景对活动序列进行详细说明。
   - 创建接口的行为模型。
   - 定义接口对象和控制机制。
   - 评审接口设计，并根据需要进行修改。
7. 进行构件级设计：
   - 在相对较低的抽象层次上详细描述

所有算法。
- 细化每个构件的接口。
- 定义构件级的数据结构。
- 评审每个构件并修正所有已发现的错误。

8. 开发部署模型。

12.3 设计概念

在软件工程的历史上，产生了一系列基本的软件设计概念。尽管多年来人们对于这些概念的关注程度不断变化，但它们都经历了时间的考验。每一种概念都为软件设计者应用更加复杂的设计方法提供了基础。每种方法都有助于：定义一套将软件分割为独立构件的标准，从软件的概念表示中分离出数据结构的细节，为定义软件设计的技术质量建立统一标准。 231

M. A. Jackson[Jac75] 曾经说过："软件工程师的智慧开始于认识到'使程序工作'和'使程序正确'之间的差异。"在后面几节中，将对基本的软件设计概念进行介绍，这些概念为"使程序正确"提供了必要的框架。

12.3.1 抽象

引述 抽象是人类处理复杂问题的基本方法之一。
Grady Booch

当考虑某一问题的模块化解决方案时，可以给出许多抽象级。在最高的抽象级上，使用问题所处环境的语言以概括性的术语描述解决方案。在较低的抽象级上，将提供更详细的解决方案说明。当力图陈述一种解决方案时，面向问题的术语和面向实现的术语会同时使用。最后，在最低的抽象级上，以一种能直接实现的方式陈述解决方案。

建议 作为设计师，致力于得到解决现有问题的过程抽象和数据抽象，但如果他们能在整个问题域中起作用，那就更好了。

在开发不同层次的抽象时，软件设计师力图创建过程抽象和数据抽象。*过程抽象*是指具有明确和有限功能的指令序列。"过程抽象"这一命名暗示了这些功能，但隐藏了具体的细节。过程抽象的例子如开门。开隐含了一长串的过程性步骤（例如，走到门前，伸出手并抓住把手，转动把手并拉门，从移动门走开等）。⊖

*数据抽象*是描述数据对象的具名数据集合。在过程抽象开的场景下，我们可以定义一个名为 door 的数据抽象。同任何数据对象一样，door 的数据抽象将包含一组描述门的属性（例如，门的类型、转动方向、开门方法、重量和尺寸）。因此，过程抽象开将利用数据抽象 door 的属性中所包含的信息。

12.3.2 体系结构

网络资源 关于软件体系结构深入的讨论可以在 www.sei.cmu.edu/ata/ata_init.html 找到。 232

*软件体系结构*意指"软件的整体结构和这种结构为系统提供概念完整性的方式"[Sha95a]。从最简单的形式来看，体系结构是程序构件（模块）的结构或组织、这些构件交互的方式以及这些构件所用数据的结构。然而在更广泛的意义上，构件可以概括为主要的系统元素及其交互方式的表示。

软件设计的目标之一是导出系统体系结构示意图，该示意图作为一个

⊖ 然而，需要注意的是：只要过程抽象隐含的功能相同，一组操作就可以被另一组操作代替。因此，如果门是自动的并连接到传感器上，那么实现"开"所需的步骤将会完全不同。

框架，将指导更详细的设计活动。一系列的体系结构模式使软件工程师能够重用设计层概念。

引述 软件体系结构是在质量、进度和成本方面具有最高投资回报的工作产品。
Len Bass et al.

Shaw 和 Garlan[Sha95a] 描述了一组属性，这组属性应该作为体系结构设计的一部分进行描述。*结构特性*定义了“系统的构件（如模块、对象、过滤器）、构件被封装的方式以及构件之间相互作用的方式”。*外部功能特性*指出“设计体系结构如何满足需求，这些需求包括性能需求、能力需求、可靠性需求、安全性需求、可适应性需求以及其他系统特征需求”。*相关系统族*“抽取出相似系统设计中常遇到的重复性模式”。

一旦给出了这些特性的规格说明，就可以用一种或多种不同的模型来表示体系结构设计[Gar95]。*结构模型*将体系结构表示为程序构件的有组织的集合。*框架模型*可以通过确定相似应用中遇到的可复用体系结构设计框架（模式）来提高设计抽象的级别。*动态模型*强调程序体系结构的行为方面，指明结构或系统配置如何随着外部事件的变化而产生变化。*过程模型*强调系统必须提供的业务或技术流程的设计。最后，*功能模型*可用于表示系统的功能层次结构。

为了表示以上描述的模型，人们已经开发了许多不同的体系结构描述语言（Architectural Description Language，ADL）[Sha95b]。尽管提出了许多不同的 ADL，但大多数 ADL 都提供描述系统构件和构件之间相互联系方式的机制。

引述 每个模式都描述了一个在我们所处环境内反复发生的问题，然后描述该问题的核心解决方案，你可以几百万次地重复使用该解决方案，而根本不需要用同样的方式重复工作两次。
Christopher Alexander

需要注意的是，关于体系结构在设计中的地位还存在一些争议。一些研究者主张将软件体系结构的设计从设计中分离出来，并在需求工程活动和更传统的设计活动之间进行。另外一些研究者认为体系结构设计是设计过程不可分割的一部分。第 13 章将讨论软件体系结构特征描述方式及软件体系结构在设计中的作用。

12.3.3 模式

Brad Appleton 以如下方式定义*设计模式*：“模式是具名的洞察力财宝，对于竞争事件中某确定环境下重复出现的问题，它承载了已证实的解决方
233 案的精髓”[App00]。换句话说，设计模式描述了解决某个特定设计问题的设计结构，该设计问题处在一个特定环境中，该环境会影响到模式的应用和使用方式。

每种设计模式的目的都是提供一种描述，以使设计人员可以决定：（1）模式是否适用于当前的工作；（2）模式是否能够复用（因此节约设计时间）；（3）模式是否能够用于指导开发一个相似的但功能或结构不同的模式。设计模式将在第 16 章进行详细讨论。

12.3.4 关注点分离

*关注点分离*是一个设计概念[Dij82]，它表明任何复杂问题如果被分解为可以独立解决或优化的若干块，该复杂问题便能够更容易地得到处理。*关注点*是一个特征或一个行为，被指定为软件需求模型的一部分。将关注点分割为更小的关注点（由此产生更多可管理的块），便可用更少的工作量和时间解决一个问题。

另一个结果是：两个问题被结合到一起的认知复杂度经常高于每个问题各自的认知复杂度之和。这就引出了“分而治之”的策略——把一个复杂问题分解为若干可管理的块来求解

时将会更容易。这对于软件的模块化具有重要的意义。

关注点分离在其他相关设计概念中也有体现：模块化、方面、功能独立、求精。每个概念都会在后面的小节中讨论。

12.3.5 模块化

模块化是关注点分离最常见的表现。软件被划分为独立命名的、可处理的构件，有时被称为模块，把这些构件集成到一起可以满足问题的需求。

有人提出“模块化是软件的单一属性，它使程序能被智能化地管理”[Mye78]。软件工程师难以掌握单块软件（即由一个单独模块构成的大程序）。对于单块大型程序，其控制路径的数量、引用的跨度、变量的数量和整体的复杂度使得理解这样的软件几乎是不可能的。绝大多数情况下，为了理解更容易，都应当将设计划分成许多模块，这样做的结果是降低构建软件所需的成本。

回顾关于关注点分离的讨论，可以得出结论：如果无限制地划分软件，那么开发软件所需的工作量将会小到忽略不计！不幸的是，其他因素开始起作用，导致这个结论是不成立的（十分遗憾）。如图 12-2 所示，开发单个软件模块的工作量（成本）的确随着模块数的增加而下降。给定同样的需求，更多的模块意味着每个模块的规模更小。然而，随着模块数量的增加，集成模块的工作量（成本）也在增加。这些特性形成了图示中的总体成本或工作量曲线。事实上，的确存在一个模块数量 M，这个数量可以带来最小的开发成本。但是，我们缺乏成熟的技术来精确地预测 M。

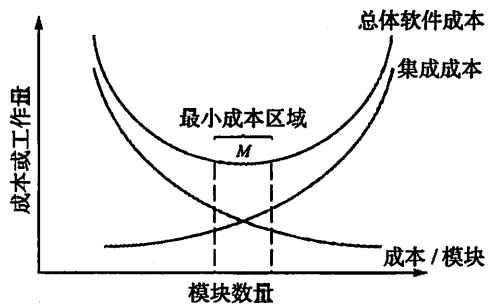

图 12-2 模块化和软件成本

234

在考虑模块化的时候，图 12-2 所示的曲线确实提供了有益的指导。在进行模块化的时候，应注意保持在 M 附近，避免不足的模块化或过度的模块化问题。但是如何知道 M 的附近在哪里呢？怎样将软件划分成模块呢？回答这些问题需要理解本章后面提出的其他设计概念。

提问 对于一个给定的系统，合适的模块数量是多少？

模块化设计（以及由其产生的程序）使开发工作更易于规划；可以定义和交付软件增量；更容易实施变更；能够更有效地开展测试和调试；可以进行长期维护而没有严重的副作用。

12.3.6 信息隐蔽

模块化概念面临的一个基本问题是：“应当如何分解一个软件解决方案以获得最好的模块集合？”信息隐蔽原则 [Par72] 建议模块应该“具有的特征是：每个模块对其他所有模块都隐蔽自己的设计决策”。换句话说，模块应该被特别说明并设计，使信息（算法和数据）都包含在模块内，其他模块无需对这些信息进行访问。

隐蔽的含义是，通过定义一系列独立的模块得到有效的模块化，独立模块之间只交流实现软件功能所必需的信息。抽象有助于定义构成软件的过程（或信息）实体。隐蔽定义并加强了对模块内过程细节的访问约束以及对模块所使用的任何局部数据结构的访问约束 [Ros75]。

关键点 信息隐蔽的目的是将数据结构和处理过程的细节隐藏在模块接口之后。用户不需要了解模块内部的具体细节。

235

将信息隐蔽作为系统模块化的一个设计标准，在测试过程以及随后的

软件维护过程中需要进行修改时，将使我们受益匪浅。由于大多数数据和过程对软件的其他部分是隐蔽的，因此，在修改过程中不小心引入的错误就不太可能传播到软件的其他地方。

12.3.7　功能独立

功能独立的概念是关注点分离、模块化、抽象和信息隐蔽概念的直接产物。在关于软件设计的一篇里程碑性的文章中，Wirth[Wir71] 和 Parnas[Par72] 间接提到增强模块独立性的细化技术。Stevens、Myers 和 Constantine[Ste74] 等人在其后又巩固了这一概念。

通过开发具有“专一”功能和“避免”与其他模块过多交互的模块，可以实现功能独立。换句话说，软件设计时应使每个模块仅涉及需求的某个特定子集，并且当从程序结构的其他部分观察时，每个模块只有一个简单的接口。

人们会提出一个很合理的问题：独立性为什么如此重要？具有有效模块化（也就是独立模块）的软件更容易开发，这是因为功能被分隔而且接口被简化（考虑由一个团队进行开发时的结果）。独立模块更容易维护（和测试），因为修改设计或修改代码所引起的副作用被限制，减少了错误扩散，而且模块复用也成为可能。概括地说，功能独立是良好设计的关键，而设计又是软件质量的关键。

提问　为什么我们总是努力构造独立模块？

独立性可以通过两条定性的标准进行评估：内聚性和耦合性。内聚性显示了某个模块相关功能的强度；耦合性显示了模块间的相互依赖性。

内聚性是 12.3.6 节说明的信息隐蔽概念的自然扩展。一个内聚的模块执行一个独立的任务，与程序的其他部分构件只需要很少的交互。简单地说，一个内聚的模块应该只完成一件事情（理想情况下）。即使我们总是争取高内聚性（即专一性），一个软件构件执行多项功能也经常是必要的和可取的。然而，为了实现良好的设计，应该避免“分裂型”构件（执行多个无关功能的构件）。

关键点　内聚性是一个模块对于一件事情侧重程度的定性指标。

关键点　耦合性是一个模块和其他模块及外部世界连接程度的定性指标。

耦合性表明软件结构中多个模块之间的相互连接。耦合性依赖于模块
236 之间的接口复杂性、引用或进入模块所在的点以及什么数据通过接口进行传递。在软件设计中，应当尽力得到最低可能的耦合。模块间简单的连接性使得软件易于理解并减少“涟漪效果”（ripple effect）的倾向 [Ste74]。当在某个地方发生错误并传播到整个系统时，就会引起“涟漪效果”。

12.3.8　求精

*逐步求精*是一种自顶向下的设计策略，最初由 Niklaus Wirth[Wir71] 提出。通过连续细化过程细节层进行应用开发，通过逐步分解功能的宏观陈述（过程抽象）进行层次开发，直至最终到达程序设计语言的语句这一级。

建议　有一种趋势是直接进行全面详细的设计，而跳过细化步骤，这将导致错误和遗漏，并使得设计更难于评审。因此，一定要实施逐步求精。

求精实际上是一个细化的过程。该过程从高抽象级上定义的功能陈述（或信息描述）开始。也就是说，该陈述概念性地描述了功能或信息，但是没有提供有关功能内部的工作或信息内部的结构。可以在原始陈述上进行细化，随着每次细化的持续进行，将提供越来越多的细节。

抽象和细化是互补的概念。抽象能够明确说明内部过程和数据，但对“外部使用者”隐藏了低层细节；细化有助于在设计过程中揭示低层细节。这两个概念均有助于设计人员在设计演化中构建出完整的设计模型。

12.3.9 方面

当我们开始进行需求分析时，一组“关注点”就出现了。这些关注点“包括需求、用例、特征、数据结构、服务质量问题、变量、知识产权边界、协作、模式以及合同”[AOS07]。理想情况下，可以按某种方式组织需求模型，该方式允许分离每个关注点（需求），使得我们能够独立考虑每个关注点（需求）。然而实际上，某些关注点跨越了整个系统，从而很难进行分割。

引述 时常浏览一下封面，确保它不是一本关于软件设计的书，这样你会更容易读懂书中的“魔法”原理。
Bruce Tognazzini

开始进行设计时，需求被细化为模块设计表示。考虑两个需求，A 和 B。“如果已经选择了一种软件分解（细化），在这种分解中，如果不考虑需求 A 的话，需求 B 就不能得到满足”[Ros04]，那么需求 A *横切*需求 B。

关键点 横切关注点是系统的某个特征，它适用于许多不同的需求。

例如，考虑 www.safehomeassured.com 网站应用中的两个需求。用第 9 章中讨论的用例 ACS-DCV 描述需求 A，设计求精将集中于那些能够使注册用户通过放置在空间中的摄像机访问视频的模块。需求 B 是一个通用的安全需求，*要求注册用户在使用* www.safehomeassured.com *之前必须先进行验证*，该需求用于 SafeHome 注册用户可使用的所有功能中。当设计求精开始的时候，A* 是需求 A 的一个设计表示，B* 是需求 B 的一个设计表示。因此，A* 和 B* 是关注点的表示，且 B* *横切* A*。

237

*方面*是一个横切关注点的表示，因此，*需求注册用户在使用* www.safehomeassured.com *之前必须先进行验证*的设计表示 B* 是 SafeHome 网站应用的一个方面。标识方面很重要，以便于在开始求精和模块化的时候，设计能够很好地适应这些方面。在理想情况下，一个方面作为一个独立的模块（构件）进行实施，而不是作为“分散的”或者和许多构件“纠缠的”软件片断进行实施 [Ban06]。为了做到这一点，设计体系结构应当支持定义一个方面，该方面即一个模块，该模块能够使该关注点经过它横切的所有其他关注点而得到实施。

12.3.10 重构

很多敏捷方法（第 5 章）都建议一种重要的设计活动——*重构*，重构是一种重新组织的技术，可以简化构件的设计（或代码）而无需改变其功能或行为。Fowler[Fow00] 这样定义重构：“重构是使用这样一种方式改变软件系统的过程：不改变代码（设计）的外部行为而是改进其内部结构。”

网络资源 重构的优秀资源可以在网站 www.refactoring.com 找到。

在重构软件时，检查现有设计的冗余性、没有使用的设计元素、低效的或不必要的算法、拙劣的或不恰当的数据结构以及其他设计不足，修改这些不足以获得更好的设计。例如，第一次设计迭代可能得到一个构件，表现出很低的内聚性（即执行三个功能但是相互之间仅有有限的联系）。在深思熟虑之后，设计人员可能决定将原构件重新分解为三个独立的构件，其中每个构件都表现出更高的内聚性。结果则是，软件更易于集成、测试与维护。

网络资源 大量重构模式可以在网站 http://c2.com/cgi/wiki?RefactoringPatterns 找到。

尽管重构的目的是以某种方式修改代码，而并不改变它的外部行为，但意外的副作用可能发生，并且也确实会发生。为此，可以使用重构工具 [Soa10] 自动分析变更，并“生成可用于检测行为变更的测试套件”。

12.3.11　面向对象的设计概念

面向对象（Object-Oriented, OO）范型广泛应用于现代软件工程。附录2是为那些不熟悉面向对象概念（如类、对象、继承、消息和多态等）的读者提供的。

238

SafeHome 设计概念

[场景] Vinod的房间，设计建模开始。

[人物] Vinod、Jamie和Ed，SafeHome软件工程团队成员；还有Shakira，团队的新成员。

[对话]

（这四个团队成员上午都参加了一位本地计算机科学教授举行的名为“应用基本的设计概念”的研讨会，他们刚从会上回来。）

Vinod：你们从研讨会学到什么没有？

Ed：大部分的东西我都已经知道，但我想重温一遍总不是什么坏事。

Jamie：我在计算机科学专业学习时，从没有真正理解信息隐蔽为什么像他们说得那么重要。

Vinod：因为……底线……这是减少错误在程序内扩散的一种技术。实际上，功能独立做的也是同样的事。

Shakira：我不是计算机科学专业的，因此教授提到的很多东西对我而言都是新的。我能生成好的代码而且速度快，我不明白这个东西为什么这么重要。

Jamie：我了解你的工作，Shak，你要知道，其实你是在自然地做这些事情……这就是为什么你的设计和编码很有效。

Shakira（微笑）：是的，我通常的确是尽量将代码分割，让分割后的代码关注于一件事，保持接口简单而且有约束，在任何可能的时候重用代码……就是这样。

Ed：模块化、功能独立、隐蔽、模式……现在明白了。

Jamie：我至今还记得我上的第一节编程课……他们教我们用迭代方式细化代码。

Vinod：设计可以采用同样的方式，你知道的。

Jamie：我以前从未听说过的概念是“方面”和“重构”。

Shakira：我记得她说那是用在极限编程中的。

Ed：是的。其实它和细化并没有太大不同，它只是在设计或代码完成后进行。我认为，这是软件开发过程中的一种优化。

Jamie：让我们回到SafeHome设计。我觉得在我们开发SafeHome的设计模型时，应该将这些概念用在评审检查单上。

Vinod：我同意。但重要的是，我们都要能够在设计时想一想这些概念。

12.3.12　设计类

提问　设计者要创建哪些类型的类？

分析模型定义了一组分析类（第10章），每一个分析类都描述问题域中的某些元素，这些元素关注用户可见的问题方面。分析类的抽象级相对较高。

当设计模型发生演化时，必须定义一组设计类，它们可以：（1）通过提供设计细节对分析类进行求精，而这些设计细节将促成类的实现；（2）实现支持业务解决方案的软件基础设施。以下给出了五种不同类型的设计类[Amb01]，每一种都表示了设计

体系结构的一个不同层次：（1）用户接口类，定义人机交互（Human-Computer Interaction, HCI）所必需的所有抽象，并且经常在隐喻的环境中实施 HCI；（2）业务域类，识别实现某 239
些业务域元素所必需的属性和服务（方法），通过一个或更多的分析类进行定义；（3）过程类，实现完整的管理业务域类所必需的低层业务抽象；（4）持久类，用于表示将在软件执行之外持续存在的数据存储（例如，数据库）；（5）系统类，实现软件管理和控制功能，使得系统能够运行，并在其计算环境内与外界通信。

随着体系结构的形成，每个分析类（第 10 章）转化为设计表示，抽象级就降低了。也就是说，分析类使用业务域的专门用语描述数据对象（以及数据对象所用的相关服务）。设计类更多地表现技术细节，用于指导实现。

Arlow 和 Neustadt[Arl02] 给出建议：应当对每个设计类进行评审，以确保设计类是“组织良好的”（well-formed）。他们为组织良好的设计类定义了四个特征。

完整性与充分性。设计类应该完整地封装所有可以合理预见的（根据对类名的理解）存在于类中的属性和方法。例如，为视频编辑软件定义的 Scene 类，只有包含与创建视频场景相关的所有合理的属性和方法时，它才是完整的。充分性确保设计类只包含那些“对实现该类的目的是足够”的方法，不多也不少。

提问 什么才是“组织良好的”设计类？

原始性。和一个设计类相关的方法应该关注于实现类的某一个服务。一旦服务已经被某个方法实现，类就不应该再提供完成同一事情的另外一种方法。例如，视频编辑软件的 VideoClip 类，可能用属性 start-point 和 end-point 指定剪辑的起点和终点（注意，加载到系统的原始视频可能比要用的部分长）。方法 setStartPoint() 和 setEndPoint() 为剪辑提供了设置起点和终点的唯一手段。

高内聚性。一个内聚的设计类具有小的、集中的职责集合，并且专注于使用属性和方法来实现那些职责。例如，视频编辑软件的 VideoClip 类可能包含一组用于编辑视频剪辑的方法。只要每个方法只关注于和视频剪辑相关的属性，内聚性就得以维持。

低耦合性。在设计模型内，设计类之间相互协作是必然的。但是，协作应该保持在一个可以接受的最小范围内。如果设计模型高度耦合（每一个设计类都和其他所有的设计类有协作关系），那么系统就难以实现、测试，并且维护也很费力。通常，一个子系统内的设计类 240
对其他子系统中的类应仅有有限的了解。该限制被称作 Demeter 定律 [Lie03]，该定律建议一个方法应该只向周边类中的方法发送消息。⊖

SafeHome 将分析类细化为设计类

[场景] Ed 的房间，开始进行设计建模。

[人物] Vinod 和 Ed，SafeHome 软件工程团队成员。

[对话]

（Ed 正进行 FloorPlan 类的设计工作（参考 10.3 节的讨论以及图 10-2），并进行设计模型的细化。）

Ed：你还记得 FloorPlan 类吗？这个类用作监视和住宅管理功能的一部分。

Vinod（点头）：是的，我好像想起来我们把它用作住宅管理 CRC 讨论的一部分。

Ed：确实如此。不管怎样，我们要对设计

⊖ Demeter 定律的一种非正式表述是：“每个单元应该只和它的朋友谈话，不要和陌生人谈话。”

进行细化，希望显示出我们将如何真正地实现 FloorPlan 类。我的想法是把它实现为一组链表（一种特定的数据结构）。像这样……我必须细化分析类 FloorPlan（图 10-2），实际上，是它的一种简化。

Vinod：分析类只显示问题域中的东西，也就是说，在电脑屏幕上实际显示的、最终用户可见的那些东西，对吗？

Ed：是的。但对于 FloorPlan 设计类来说，我已经开始添加一些实现中特有的东西。需要说明的是 FloorPlan 是段（Segment 类）的聚集，Segment 类由墙段、窗户、门等的列表组成。Camera 类和 FloorPlan 类协作，这很显然，因为在平面图中可以有很多摄像机。

Vinod：咳，让我们看看新的 FloorPlan 设计类图。

（Ed 向 Vinod 展示图 12-3。）

Vinod：好的，我看出来你想做什么了。这样你能够很容易地修改平面图，因为新的东西可以在列表（聚集）中添加或删除，而不会有任何问题。

Ed（点头）：是的，我认为这样是可以的。

Vinod：我也赞同。

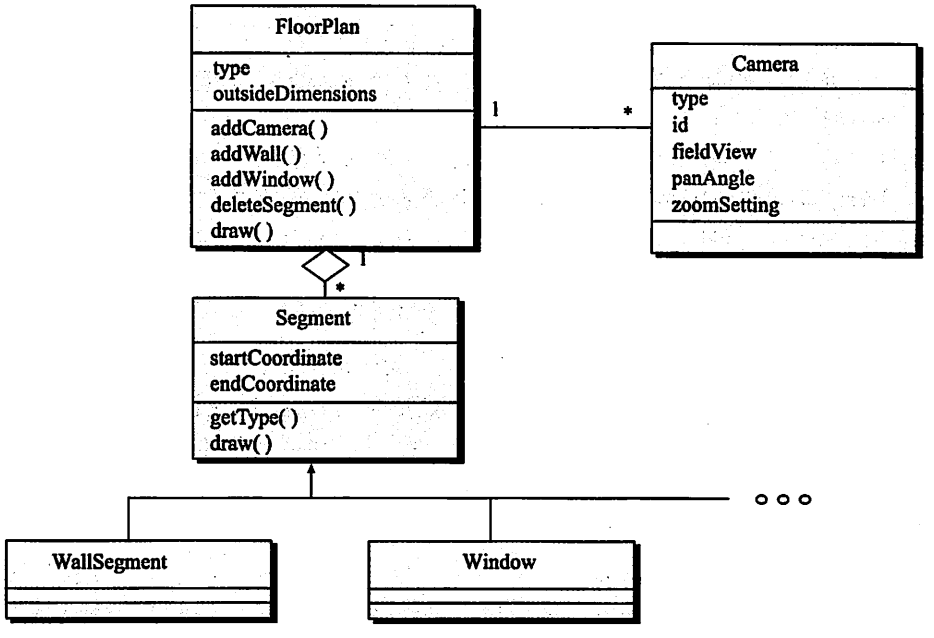

图 12-3　FloorPlan 的设计类和类的复合聚集

12.3.13　依赖倒置

许多旧一些的软件体系结构是层次结构。在体系结构的顶层，“控制”构件依赖于较低层的“工作者”构件，以完成各种内聚任务。比如，考虑一个具有三个组件的简单程序，该程序的目的是读取键盘的按键并将结果输出到打印机。控制模块 C 需要协调另外两个模块——按键读取模块 R 和写入打印机模块 W。

由于控制模块 C 高度依赖于 R 和 W，因此程序设计是耦合的。为了消除这种依赖，“工作者”模块 R 和 W 应当通过抽象由控制模块 C 调用。在面向对象的软件工程中，抽象被作为抽象类 R* 和 W* 以获得实现。然

提问　依赖倒置的原则是什么？

后，这些抽象类可以用来调用执行读写功能的工作者类。因此，一个 copy 类 C 调用抽象类 R* 和 W*，抽象类指向合适的工作者类（例如，R* 类可能指向一个环境中 keyboard 类的 read() 操作和另一个环境中 sensor 类的 read() 操作）。这种方法降低了耦合性，并提高了设计的可测性。

前段中讨论的例子可以与依赖倒置原则一概而论 [Obj10]，该原则这样描述：*高层模块（类）不应当（直接）依赖于低层模块，两者都应当依赖于抽象。抽象不应当依赖于细节，细节应当依赖于抽象。*

12.3.14 测试设计

引述 测试要快，失败要快，调整要快。
Tom Peters

到底应当先开始软件设计还是测试用例设计，这是一个争论不休的鸡与蛋的问题。Rebecca Wirfs-Brock[Wir09] 写道：

测试驱动开发的倡导者们在编写任何其他代码之前先编写测试代码。他们谨记 Peter 的信条——测试要快，失败要快，调整要快。他们以一种简短而快速的方式实施周期演变，编写测试代码——测试未通过——编写足够的代码以通过测试——测试通过，在这一过程中，测试指导他们的设计。

242

但如果先开始进行设计，那么设计（以及编码）必须带有一些接缝。所谓接缝，即详细设计中的一些位置，在这些位置可以“插入一些测试代码，这些代码可用以探测运行中软件的状态”，以及“将待测试的代码从它的产生环境中分离出来，以便在受控的测试环境中执行这些代码”[Wir09]。

有时候，接缝也被称为“测试沟”，它们必须被有意识地在构件级进行设计。为了实现这一点，设计者必须考虑将用于演练构件的测试。正如 Wirf-Brock 所述：“简言之，需要提供适当的测试启示——以一种方式将设计分解为若干因素，测试代码可以询问并控制运行中的系统。”

12.4 设计模型

可以从两个不同的维度观察设计模型，如图 12-4 所示。*过程维度表示设计模型的演化，设计任务作为软件过程的一部分被执行。抽象维度表示详细级别，分析模型的每个元素转化为一个等价的设计，然后迭代求精。*参考图 12-4，虚线表示分析模型和设计模型之间的边界。在某些情况下，分析模型和设计模型之间可能存在明显的差异；而有些情况下，分析模型慢慢地融入设计模型而没有明显的差异。

243

关键点 设计模型有4个主要元素：数据、体系结构、构件和接口。

设计模型的元素使用了很多 UML 图[⊖]，有些 UML 图在分析模型中也会用到。差别在于这些图被求精和细化为设计的一部分，并且提供了更多具体的实施细节，突出了体系结构的结构和风格、体系结构中的构件、构件之间以及构件和外界之间的接口。

然而，要注意到，沿横轴表示的模型元素并不总是顺序开发的。大多数情况下，初步的体系结构设计是基础，随后是接口设计和构件级设计（通常是并行进行）。通常，直到设计全部完成后才开始部署模型的工作。

⊖ 附录 1 提供了基本 UML 概念和符号的使用手册。

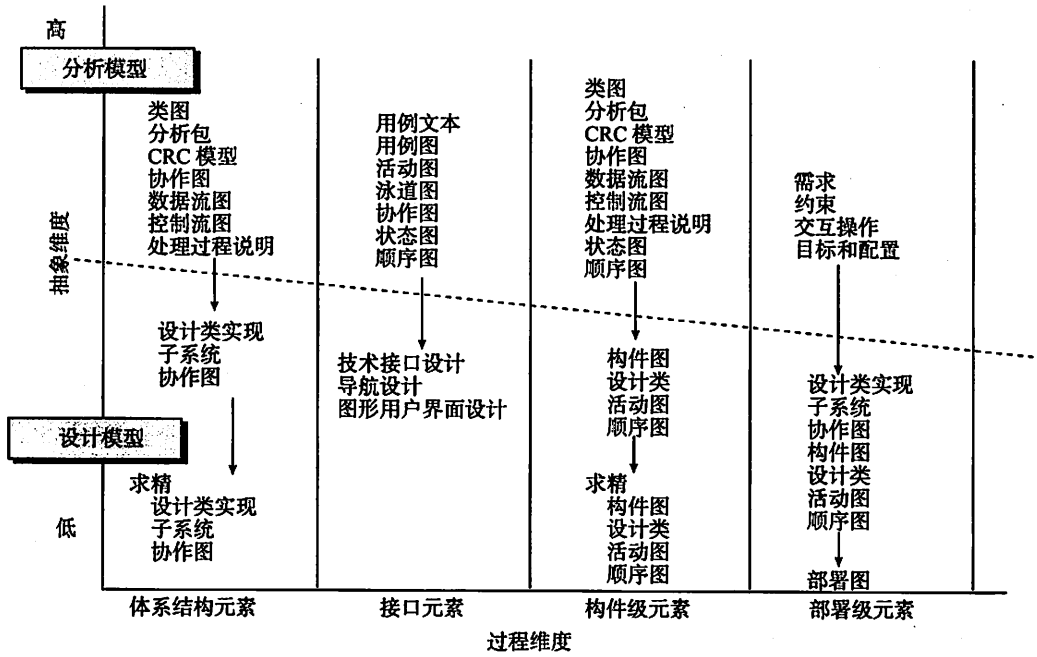

图 12-4　设计模型的维度

在设计过程中的任何地方都可以应用设计模式（第 16 章）。这些模式能够使设计人员将设计知识应用到他人已经遇到并解决了的特定领域问题中。

引述　提出“设计是否是必要的”或“能否负担得起”这样的问题非常离题，因为设计是不可避免的。不是好的设计就是坏的设计，根本不可能不要设计。

Douglas Martin

12.4.1　数据设计元素

和其他软件工程活动一样，数据设计（有时也称为*数据体系结构*）创建了在高抽象级上（以客户或用户的数据观点）表示的数据模型和信息模型。之后，数据模型被逐步求精为特定实现的表示，亦即计算机系统能够处理的表示。在很多软件应用中，数据体系结构对于必须处理该数据的软件的体系结构将产生深远的影响。

数据结构通常是软件设计的重要部分。在程序构件级，数据结构设计以及处理这些数据的相关算法对于创建高质量的应用程序是至关重要的。在应用级，从数据模型（源自于需求工程）到数据库的转变是实现系统业务目标的关键。在业务级，收集存储在不同数据库中的信息并重新组织为“数据仓库”要使用数据挖掘或知识发现技术，这些技术将会影响到业务本身的成功。在每一种情况下，数据设计都发挥了重要作用。第 13 章将更详细地讨论数据设计。

关键点　在体系结构（应用）级，数据设计关注文件或数据库；在构件级，数据设计考虑实现局部数据对象所需的数据结构。

12.4.2　体系结构设计元素

软件的*体系结构*设计等效于房屋的平面图。平面图描绘了房间的整体布局，包括各房间的尺寸、形状、相互之间的联系，能够进出房间的门窗。平面图为我们提供了房屋的整体视图；而体系结构设计元素为我们提供了软件的整体视图。

244

体系结构模型 [Sha96] 从以下三个来源导出：（1）关于将要构建的软件的应用域信息；（2）特定的需求模型元素，如数据流图或分析类、现有问题中它们的关系和协作；（3）可获得的体系结构风格（第 13 章）和模式（第 16 章）。

引述 你可以在绘图桌上使用橡皮或在建筑工地现场使用大铁锤。
Frank Lloyd Wright

体系结构设计元素通常被描述为一组相互联系的子系统，且常常从需求模型中的分析包中派生出来。每个子系统有其自己的体系结构（如图形用户界面可能根据之前存在的用户接口体系结构进行了结构化）。体系结构模型特定元素的导出技术将在第 13 章中介绍。

12.4.3 接口设计元素

引述 与良好的设计相比，公众更熟悉拙劣的设计。实际上，公众更习惯拙劣的设计，因为生活就是如此。新的有危险，而旧的更让人安心。
Paul Rand

软件的接口设计相当于一组房屋的门、窗和外部设施的详细绘图（以及规格说明）。门、窗、外部设施的详细图纸（以及规格说明）作为平面图的一部分，大体上告诉我们：事件和信息如何流入和流出住宅以及如何在平面图的房间内流动。软件接口设计元素描述了信息如何流入和流出系统，以及被定义为体系结构一部分的构件之间是如何通信的。

接口设计有三个重要的元素：（1）用户界面（User Interface，UI）；（2）和其他系统、设备、网络、信息生成者或使用者的外部接口；（3）各种设计构件之间的内部接口。这些接口设计元素能够使软件进行外部通信，还能使软件体系结构中的构件之间进行内部通信和协作。

关键点 接口设计元素有三部分：用户接口、系统和外部应用的接口、应用系统内部构件之间的接口。

UI 设计（越来越多地被称作可用性设计）是软件工程中的主要活动，这会在第 15 章中详细地考虑。可用性设计包含美学元素（例如，布局、颜色、图形、交互机制）、人机工程元素（例如，信息布局、隐喻、UI 导航）和技术元素（例如，UI 模式、可复用构件）。通常，UI 是整个应用体系结构内独一无二的子系统。

外部接口设计需要发送和接收信息实体的确定信息。在所有情况下，这些信息都要在需求工程（第 8 章）过程中进行收集，并且在接口[⊖]设计开始时进行校验。外部接口设计应包括错误检查和适当的安全特征检查。

245

引述 现在的每一样东西以后都会消失。稍微放松一下，再返回到你的工作中，你的判断将更可靠。因为离开一定距离，工作看起来更小，更容易远眺，更容易发现和谐和比例的缺失。
Leonardo DaVinci

内部接口设计和构件级设计（第 14 章）紧密相关。分析类的设计实现呈现了所有需要的操作和消息传递模式，使得不同类的操作之间能够进行通信和协作。每个消息的设计必须提供必不可少的信息传递以及所请求操作的特定功能需求。

在有些情况下，接口建模的方式和类所用的方式几乎一样。在 UML 中，接口如下定义 [OMG03a]：“接口是类、构件或其他分类符（包括子系统）的外部可见的（公共的）操作说明，而没有内部结构的规格说明。”更简单地说，接口是一组描述类的部分行为的操作，并提供了这些操作的访问方法。

例如，SafeHome 安全功能使用控制面板，控制面板允许户主控制安全功能的某些方面。在系统的高级版本中，控制面板的功能可能会通过移

⊖ 接口特征可能随时间变化。因此，设计者应当确保接口的规格说明是准确且完整的。

动平台（例如智能手机或平板电脑）实现。

ControlPanel 类（图 12-5）提供了和键盘相关的行为，因此必须实现操作 readKeyStroke() 和 decodeKey()。如果这些操作提供给其他类（在此例中是 Tablet 和 SmartPhone），定义如图 12-5 所示的接口是非常有用的。名为 KeyPad 的接口表示为 <<interface>> 构造型（stereotype），或用一个带有标识且用一条线和类相连的小圆圈表示，定义接口时并没有实现键盘行为所必需的属性和操作集合。

网络资源　有关 UI 设计非常有用的信息可以在 www.useit.com 找到。

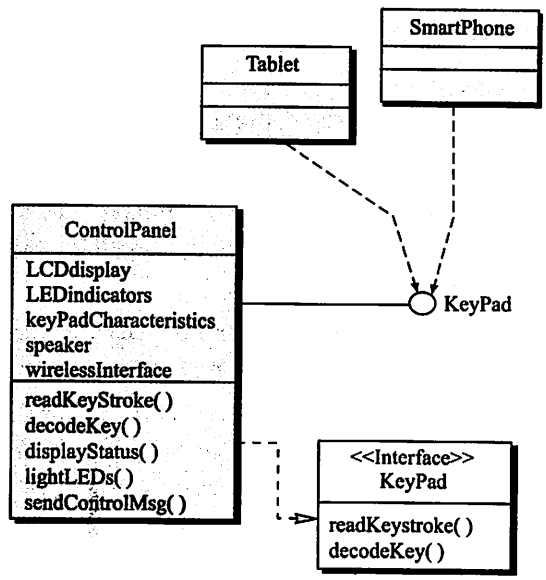

图 12-5　ControlPanel 的接口表示

引述　在设计傻瓜级的东西时，人们常犯的一个错误是低估傻瓜的聪明。
Douglas Adams

246　带有三角箭头的虚线（图 12-5）表示 ControlPanel 类提供了 KeyPad 操作以作为其行为的一部分。在 UML 中，这被称为实现。也就是说，ControlPanel 行为的一部分将通过实现 KeyPad 操作来实现。这些操作将被提供给那些访问该接口的其他类。

12.4.4　构件级设计元素

软件的构件级设计相当于一个房屋中每个房间的一组详图（以及规格说明）。这些图描绘了每个房间内的布线和管道、电器插座和墙上开关、水龙头、水池、淋浴、浴盆、下水道、壁橱和储藏室的位置，以及房间相关的任何其他细节。

软件的构件级设计完整地描述了每个软件构件的内部细节。为此，构件级设计为所有局部数据对象定义数据结构，为所有在构件内发生的处理定义算法细节，并定义允许访问所有构件操作（行为）的接口。

在面向对象的软件工程中，使用 UML 图表现的一个构件如图 12-6 所示。图中表示的构件名为 SensorManagement（SafeHome 安全功能的一部分）。虚线箭头连接了构件和名为 Sensor 的类。SensorManagement 构件完成所有和 SafeHome 传感器相关的功能，包括监控和配置传感器。第 14 章将进一步讨论构件图。

引述　细节并不仅仅是细节，细节构成设计。
Charles Eames

构件的设计细节可以在很多不同的抽象级进行建模。UML 活动图可用来表示处理逻辑，构件详细的过程流可以使用伪代码表示（类似编程语言的表示方法，在第 14 章讨论），也可以使用一些图形（例如流程图或盒图）来表示。算法结构遵守结构化编程的规则（即一套约束程序构造）。基于待处理数据对象的特性所选择的数据结构，通常会使用伪代码或程序语言进行建模，以便将之用于实施。

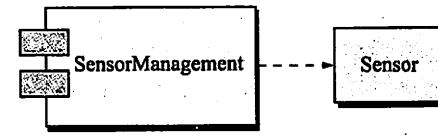

图 12-6　UML 构件图

247

12.4.5　部署级设计元素

部署级设计元素指明软件功能和子系统将如何在支持软件的物理计算环境内进行分布。例如，SafeHome 产品元素被配置在三种主要的计算环境内运行——基于住宅的 PC、SafeHome 控制面板和位于 CPI 公司的服务器（提供基于 Internet 的系统访问）。此外，移动平台也可以提供有限的功能。

在设计过程中，开发的 UML 部署图以及随后的细化如图 12-7 所示。图中显示了 3 种计算环境（实际上，还可能包括传感器、摄像机和移动平台传送的功能）。图中标识出了每个计算元素中还有子系统（功能）。例如，个人计算机中有完成安全、监视、住宅管理和通信功能的子系统。此外，还设计了一个外部访问子系统，以管理外界资源对 SafeHome 系统的访问。每个子系统需要进行细化，用以说明该子系统所实现的构件。

关键点　部署图刚开始使用描述符形式，粗略描述部署环境。后来使用实例形式，明确描述配置的元素。

如图 12-7 所示，图中使用了描述符形式，这意味着部署图表明了计算环境，但并没有明确地说明配置细节。例如，“个人计算机”并没有进一步地明确它是一台 Mac PC、一台基于 Windows 的 PC、Linux 系统还是带有相关操作系统的移动平台。在设计的后续阶段或构建开始时，需要用实例形式重新为部署图提供这些细节，明确每个实例的部署（专用的称为硬件配置）。

248

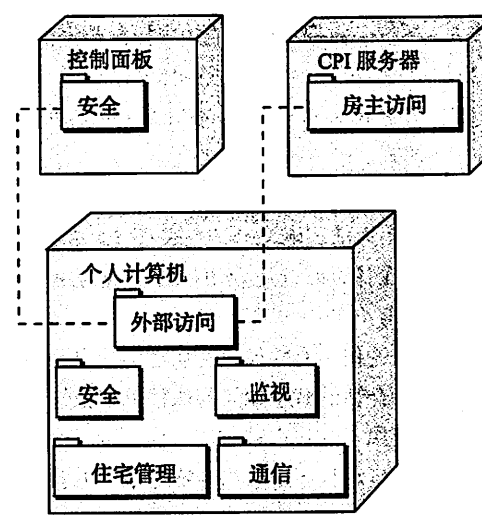

图 12-7　UML 部署图

12.5　小结

当第一次需求工程得到结论时，软件设计便开始了。软件设计的目的是应用一系列原则、概念和实践，以引导高质量的系统或产品开发。设计的目标是创建软件模型，该模型将正确地实现所有的客户需求，并为软件用户带来愉悦的感受。软件设计人员必须从大量可供选择的设计中筛选并确定一个解决方案，该方案最能满足项目利益相关者的需要。

设计过程从软件的“宏观”视图向微观视图转移，后者定义了实现系统所必需的细节。设计过程开始时关注于体系结构，然后定义子系统、建立子系统之间的通信机制、识别构件、制定每个构件的详细说明，另外还要设计外部接口、内部接口和用户接口。

设计概念在软件工程刚开始的 60 年内不断发展。这些概念描述了计算机软件的属性

（而并不应考虑所选择的软件工程过程）、描述所使用的设计方法或所使用的编程语言。实质上，设计概念强调了：（1）抽象的必要性，它提供了一种创造可重用软件构件的方法；（2）体系结构的重要性，它使得人们能够更好地理解系统整体结构；（3）基于模式的工程的有益性，它将已证明的能力用于软件设计；（4）关注点分离和有效模块化的价值，它们使得软件更易理解、更易测试以及更易维护；（5）信息隐蔽的直接作用，当错误发生时，它能够减少负面影响的传播；（6）功能独立的影响，它是构建有效模块的标准；（7）求精作为一种设计方法的作用；（8）横切系统需求方面的考虑；（9）重构的应用，目的是优化已导出的设计；（10）面向对象的类和与类相关特征的重要性；（11）使用抽象降低构件之间耦合的需要；（12）测试设计的重要性。

设计模型包含四种不同的元素。随着每个元素的开发，逐渐形成更全面的设计视图。体
249 系结构元素使用各种信息以获得软件、软件子系统和构件的完整的结构表示，这些信息来自于应用域、需求模型以及模式和风格的分类。接口设计元素为外部和内部的接口以及用户接口建模。构件级元素定义体系结构中的每一个模块（构件）。最后，部署级设计元素划分体系结构、构件和容纳软件的物理配置的接口。

习题与思考题

12.1　当你“编写”程序时是否会设计软件？软件设计和编码有什么不同？

12.2　如果软件设计不是程序（它肯定不是），那么它是什么？

12.3　如何评估软件设计的质量？

12.4　查看设计任务集，在任务集中的什么地方对质量进行评估？评估是如何完成的？如何达到12.2.1节中讨论的质量属性？

12.5　举三个数据抽象和能用来控制数据的过程抽象的例子。

12.6　用你自己的话描述软件体系结构。

12.7　为你每天都能遇到的东西（例如家用电器、汽车、设备）推荐一个设计模式，简要地描述该模式。

12.8　用你自己的语言描述关注点分离。分而治之的方法有时候不合适吗？这种情况对模块化的观点有多大的影响？

12.9　应在什么时候把模块设计实现为单块集成软件？如何实现？性能是实现单块集成软件的唯一理由吗？

12.10　讨论作为有效模块化属性的信息隐蔽概念和模块独立性概念之间的联系。

12.11　耦合性的概念如何与软件可移植性相关联？举例支持你的论述。

12.12　应用“逐步求精方法”为下列一个或多个程序开发三种不同级别的过程抽象：（1）开发一个支票打印程序，给出金额总数，并按支票的常规要求给出大写金额数；（2）为某个超越方程迭代求解；（3）为操作系统开发一个简单的任务调度算法。

12.13　考虑需要实现汽车导航功能（GPS）、手持通信设备的软件。描述两个或三个要表示的横切关注点。讨论如何将其中一个关注点作为方面来表示。

12.14　“重构”意味着迭代地修改整个设计吗？如果不是，它意味着什么？

12.15　用你自己的语言描述什么是依赖倒置？

12.16　测试设计为什么很重要？

250 12.17　简要描述设计模型的四个元素。

扩展阅读与信息资源

Donald Norman 编写了三本书：《Emotional Design: We love(or hate) EveryDay Things》（Basic Books，2005）；《The Design of Everyday Things》（Doubleday，1990）以及《The Psychology of Everyday Things》（HarperCollins，1988）。这三本书已经成为设计学中的经典著作，任何人想设计人类使用的任何东西，都“必须”阅读这些著作。Adams 的著作（《Conceptual Blockbusting》，4th ed.，Addison-Wesley，2001）对那些希望拓宽思路的设计人员极为重要。最后，Polya 在其编写的一本经典著作（《How to Solve It》，2nd ed.，Princeton University Press，1988）中提供了通用的问题解决流程，在软件设计人员面对复杂问题时能够提供帮助。

Hanington 和 Martin（《Universal Methods of Design: 100 ways to Research Complex Problems，Develop Innovative Ideas，and Design Effective Solutions》，Rockport，2012 和《Universal Principles of Design: 125 Ways to Enhance Usability，Influence Perception，Increase Appeal，Make Better Design Decisions，and Teach through Design》，2nd ed.，Rockport，2010）讨论了通常的设计原则。

遵循同样的传统，Winograd 等人（《Bringing Design to Software》，Addison-Wesley，1996）讨论了成功与不成功的软件设计及其理由。Wixon 和 Ramsey 编写了一本令人着迷的书（《Field Methods Casebook for Software Design》，Wiley，1996），书中建议使用领域搜索方法（和人类学家所使用的那些方法非常类似）理解最终用户是如何工作的，然后设计满足用户需要的软件。Holtzblatt（《Rapid Contextual Design: A How-to Guide to Key Techniques for User-Center Design》，Morgan Kaufman，2004）以及 Beyer 和 Holtzblatt（《Contextual Design: A Customer-Centered Approach to Systems Designs》，Academic Press，1997）提供了软件设计的另一种视图，即将客户 / 用户集成到软件设计流程的各个方面。Bain（《Emergent Design》，Addison-Wesley，2008）将模式、重构和测试驱动的开发方法结合到有效的设计方法中。

Otero（《Software Engineering Design:Theory and Practice》，Auerbach，2012）、Venit 和 Drake（《Prelude to Programming: Concepts and Design》，5th ed.，Addison-Wesley，2010）、Fox（《Introduction to software Engineering Design》，Addison-Wesley，2006）以及 Zhu（《Software Design Methodology》，Butterworth-Heinemann，2005）介绍了软件工程中设计的综合性处理。McConnell（《Code Complete》，2nd ed.，Microsoft Press，2004）对高质量计算机软件的实践方面进行了精彩的论述。Robertson（《Simple Program Design》，5th ed.，Course Technology，2006）介绍性地论述了软件设计，这对初学者很有帮助。Budgen（《Software Design》，2nd ed.，Addison-Wesley，2004）介绍了大量流行的软件设计方法，并对它们进行了对比。Fowler 和他的同事（《Refactoring: Improving the Design of Existing Code》，Addison-Wesley，1999）讨论了软件设计增量优化的技术。Rosenberg 和 Stevens（《Use Case Driven Object Modeling with UML》，Apress，2007）讨论了以用例为基础的面向对象的设计开发。

从 Free-man 和 Wasserman（《Software Design Techniques》，4th ed.，IEEE，1983）编辑的文集中，可以一览软件设计的精彩发展历史。这本教程中转载了许多经典的论文，这些论文已经奠定了软件设计当前发展趋势的基础。Card 和 Glass（《Measuring Software Design Quality》，Prentice-Hall，1990）从技术和管理两个角度介绍并思考了软件质量的度量。

网上有关于软件设计的大量信息资源。在 SEPA 网站 www.mhhe.com/ pressman 上可以找到与软件设计以及设计工程相关的最新参考文献。