软件工程

第11章 设计概念

徐本柱 软件学院

2018-09

主要内容

- 软件工程中的设计
- 设计过程
- 设计概念
- 设计模型

要点浏览

- 设计创建了软件的表示或模型
- 设计模型与需求模型不同点
  - 需求模型注重描述所需的数据、功能和行为
  - 提供软件体系结构、数据结构、接口和构件的细节
- 设计为将要构建的系统或产品建立模型
  - 在生成代码、进行测试以及最终用户使用之前
  - 对模型的质量进行评估，并进行改进
- 软件质量在设计中建立

设计的步骤

- 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
    - 在软件和最终用户、软件和其他系统及设备
    - 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。

工作产品及质量保证

- 在软件设计过程中，主要的工作产品包含
  - 体系结构、接口、构件和部署表示的设计模型
- 从以下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
  - 是否存在更好的方案可供选择
  - 是否可在设定的约束、时间进度和成本内实现

软件设计

- 软件设计包括一套原理、概念和实践
  - → 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重要原则
- 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
- 设计良好的软件应该展示出：
  - 坚固性：程序应该不含任何妨碍其功能的　缺陷
  - 适用性：程序应该符合开发的目标
  - 愉悦性：使用程序的体验应是愉快的

11.1 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
  - 与所使用的软件过程模型无关
- 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）
- 需求模型的每个元素
  - 都提供了创建4种设计模型所必需的信息，
  - 这4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11-1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**分析模型**是设计任务的输入。
  - 使用相应的设计表示和设计方法，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

4种设计模型

- 数据/类设计——
  - 分析类需求模型转化为设计类实现模型及数据结构
- 体系结构设计——
  - 描述软件主要构造元素之间的关系
- 接口设计——
  - 描述软件元素、硬件元素和终端用户间如何通信
- 构件级设计——
  - 将软件体系结构的构造元素变换为对软件构件的过程性描述

需求模型→设计模型

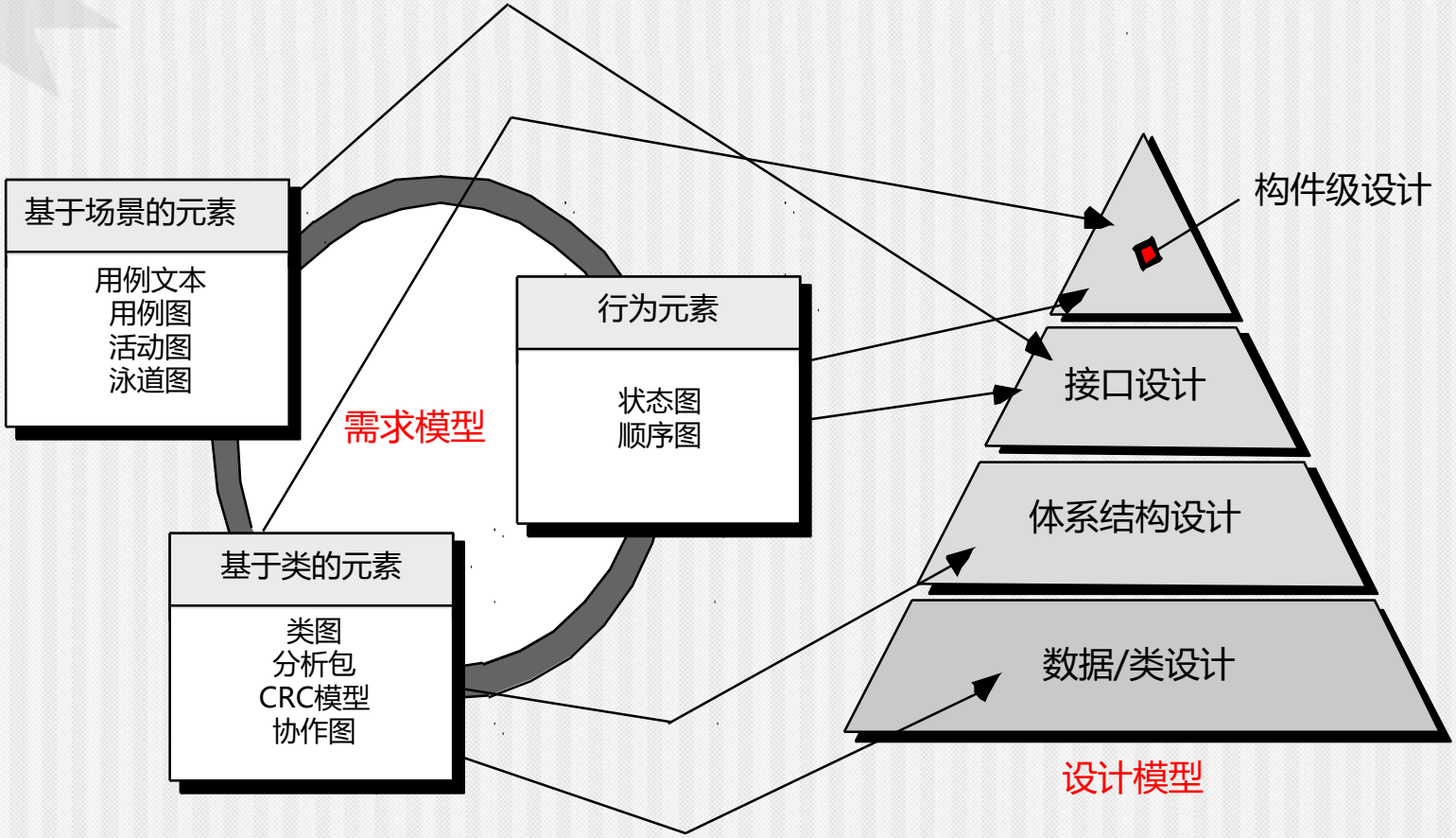


图11-1 从需求模型到设计模型的转换

软件设计的重要性

- 用一个词来表达——质量
- 设计是软件工程中质量形成的地方
  - 提供了可以用于质量评估的软件表示
  - 将用户需求准确转化为软件产品或系统的唯一方法
  - 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，
    - 会存在构建不稳定系统的风险，
    - 稍做改动就无法运行，
    - 而且难以测试，
    - 直到软件工程过程的后期才能评估其质量。

11.2 设计的过程

- 软件设计是一个迭代的过程
  - 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，
    - 设计是在高抽象层次上的表达——
    - 可直接跟踪到特定的系统目标和
    - 更详细的数据、功能和行为需求
  - 随着设计迭代的开始，
    - 后续的精化导致更低抽象层次的设计表示
    - 仍然能够跟踪到需求，但是连接更加错综复杂

11.2.1 软件质量指导原则和属性

- 指导评价良好设计演化的3个特征：
  - 设计必须实现所有需求模型中的明确需求，必须满足客户期望的所有隐含需求
  - 对于程序员、测试人员和维护人员而言，设计必须是可读的、可理解的指南
  - 设计必须提供软件的全貌，从实现的角度说明数据域、功能域和行为域

质量指导原则（1）

- 1 设计应展示出这样一种结构：
  - (1)用可识别的体系结构风格或模式创建；
  - (2)由展示出良好设计特征的构件构成；
  - (3)能够以演化的方式实现，便于实现和测试。
- 2 设计应该模块化；
  - 即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
  - 和构件的清楚表示。

质量指导原则（2）

- 4 设计应导出数据结构，
  - 适用于要实现的类，并从可识别的数据模式中提取。
- 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，
  - 降低了构件之间以及与外部环境连接的复杂性。
- 7 设计的导出应根据软件需求分析过程中获取的信息采用可重复使用的方法进行。
- 8 应使用能够有效传达其意义的表示法来表达

质量属性

- HP制定了一系列软件质量属性，称为FURPS，
  - 分别代表功能性、易用性、可靠性、性能、可支持性。
  - FURPS质量属性体现了所有软件设计的目标。
- 功能性：
  - 评估程序的特征集和能力、所提交功能的通用性、整个系统的安全性。
- 易用性：
  - 通过考虑人员因素、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、故障恢复能力和程序的可预见性来评估。
- 性能：
  - 度量处理速度、响应时间、资源消耗、吞吐量和效率。
- 可支持性：
  - 综合了程序的可扩展性、适应性和耐用性3方面的能力，
  - 以及可测试性、兼容性、可配置性、系统安装和问题定位的简易性。

11.2.2 软件设计的演化

一个持续的过程

- 自顶向下求精的方法
- 结构化程序设计
- 数据流（数据结构）驱动设计方法
- OO设计方法
- 设计模式
- 面向方面的方法
- 模型驱动开发
- 测试驱动开发

共同特征

- 1 需求模型转换为设计表示的方法
- 2 功能性构件及之间接口的表示法
- 3 细化和分割的启发式方法
- 4 质量评估的指导原则

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。

11.3 设计概念

- 软件工程的历史中产生了一系列基本的软件设计概念。
- 每一种概念的关注程度不断变化，但都经历了时间的考验。
- 每一种概念都为软件设计者提供了应用更加复杂设计方法的基础。
- 基础的软件设计概念为“使程序正确”提供了必要的框架。

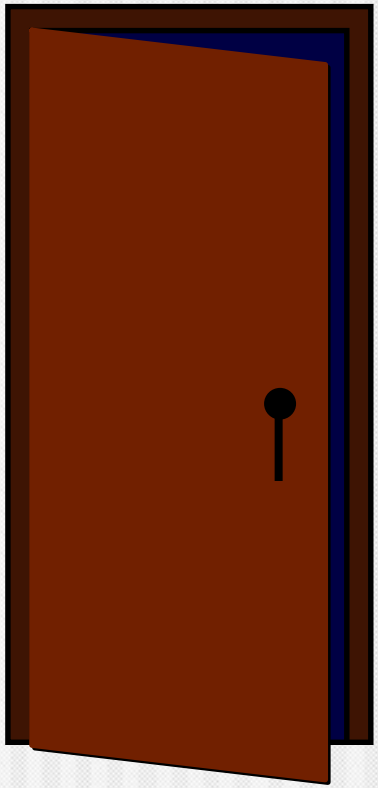
基本概念

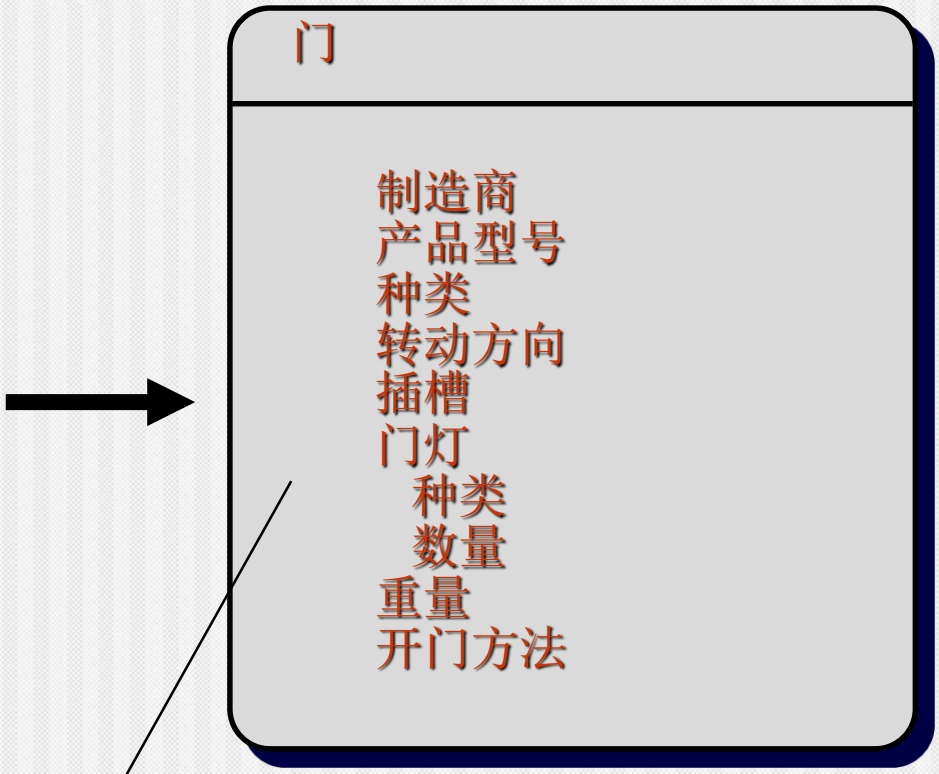
- 抽象——数据，过程，控制
- 体系结构——软件的整体结构
- 模式——传递已验证设计方案的精髓
- 关注点分离——任何复杂问题在被分解为若干块后将更易处理
- 模块化——数据和功能的分割
- （信息）隐蔽——控制接口
- 功能独立——单一功能和低耦合
- 求精——细化所有抽象的细节
- 方面——理解全局需求如何影响设计的机制
- 重构——简化设计的重组技术
- 面向对象的设计概念——附录 II
- 设计类——提供设计细节以实现分析类

11.3.1 抽象

- 某一问题的模块化解决方案，可有许多抽象级。
  - 在最高的抽象级上，使用问题所处领域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（领域术语和实现术语同时使用）
  - 在最低的抽象层次上以一种能直接实现的方式陈述
- 开发不同层次的抽象→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
  - 过程抽象暗示了这些功能，隐藏了具体的细节
  - 数据抽象是描述数据对象的冠名数据集合

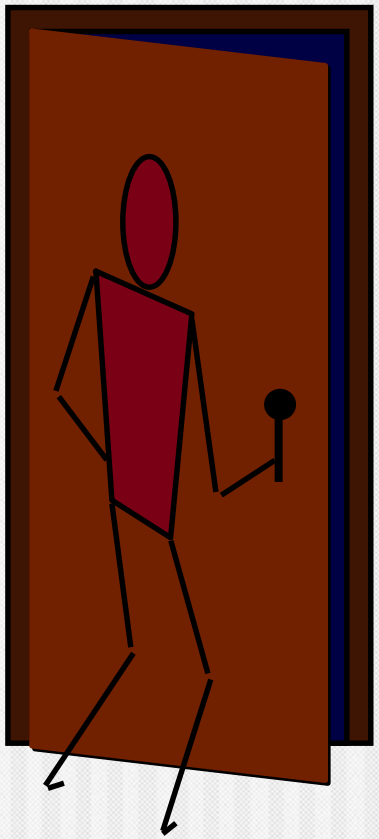
数据抽象

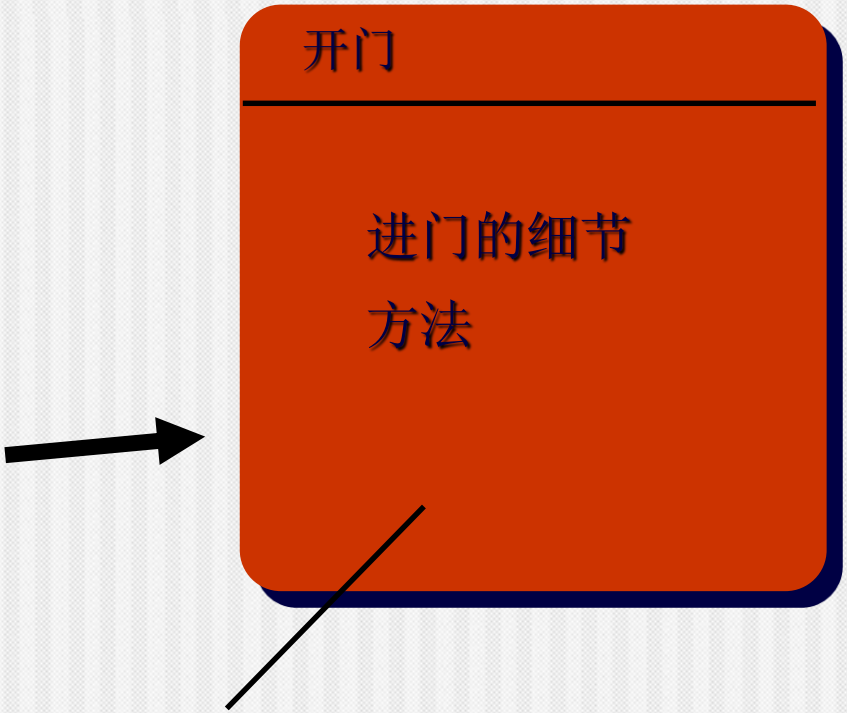




与数据结构类似过程进行抽象

过程抽象





运用与进入有关的事物的“知识”完成抽象

11.3.2 体系结构

- 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上完整性的方式”
  - 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
  - 概括为主要系统元素及其交互方式的表示
- 软件设计的目标之一是
  - 导出系统的体系结构示意图，
  - 该示意图作为一个框架，将指导更详细的设计活动。
- 体系结构模式能够复用设计层的概念。

体系结构属性

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。例如对象封装了数据和过程，过程操纵数据并通过方法调用进行交互

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经常遇到的重复性模式。本质上，设计应当能够重用体系结构构件。

软件体系结构表示

- 结构模型
  - 将体系结构表示为程序构件的一个有组织的集合
- 框架模型
  - 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作为外部事件的函数将如何变化。
- 过程模型
  - 注重系统必须提供的业务设计或技术流程设计。
- 功能模型
  - 可以用来表示系统的功能层次结构

11.3.3 模式

- 设计模式→特定场景中解决设计问题的设计结构。
- 每个设计模式有一个描述，能够确定：
  - (1)模式是否适合当前的工作；
  - (2)模式是否能够复用；
  - (3)模式是否能够用于指导开发一个类似但是功能或结构不同的模式。

11.3.4 关注点分离

- 关注点是一个设计概念，表明
  - 复杂问题分解为可以独立解决和(或)优化的若干块，
  - 该复杂问题能够更容易地被处理。
- 关注点是
  - 一个特征或行为，
  - 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，使得解决一个问题需要付出更少的工作量和时间

问题分解

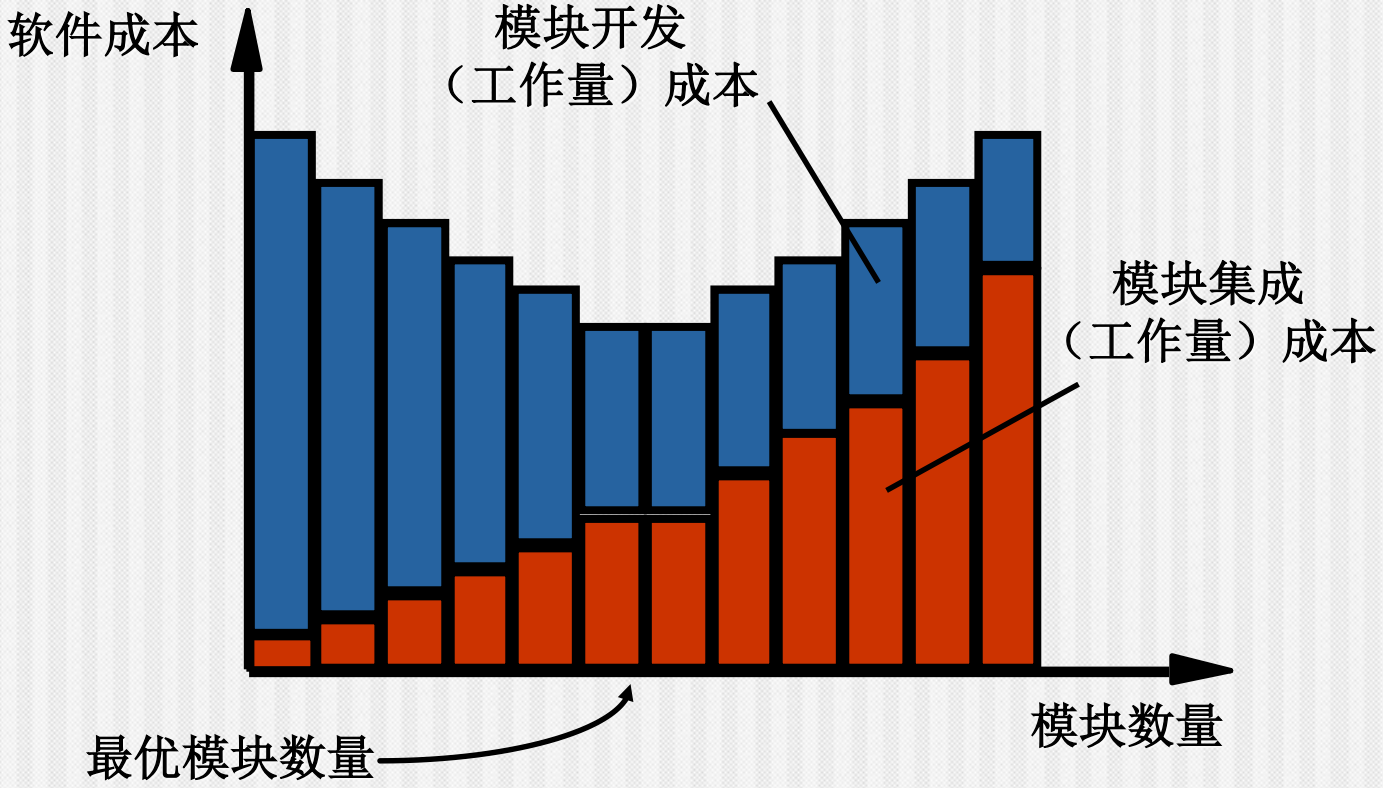
- 两个问题一起考虑
  - 总认知复杂度高于每个问题各自的认知复杂度之和。
  - “分而治之”的策略：
    - 将一个复杂问题分解成可以管理的若干块
    - 更容易解决问题。

11.3.5 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
  ——模块，
  把这些构件集成到一起可以满足问题的需求。
- 软件工程师难以掌握单块软件。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。

模块化：权衡

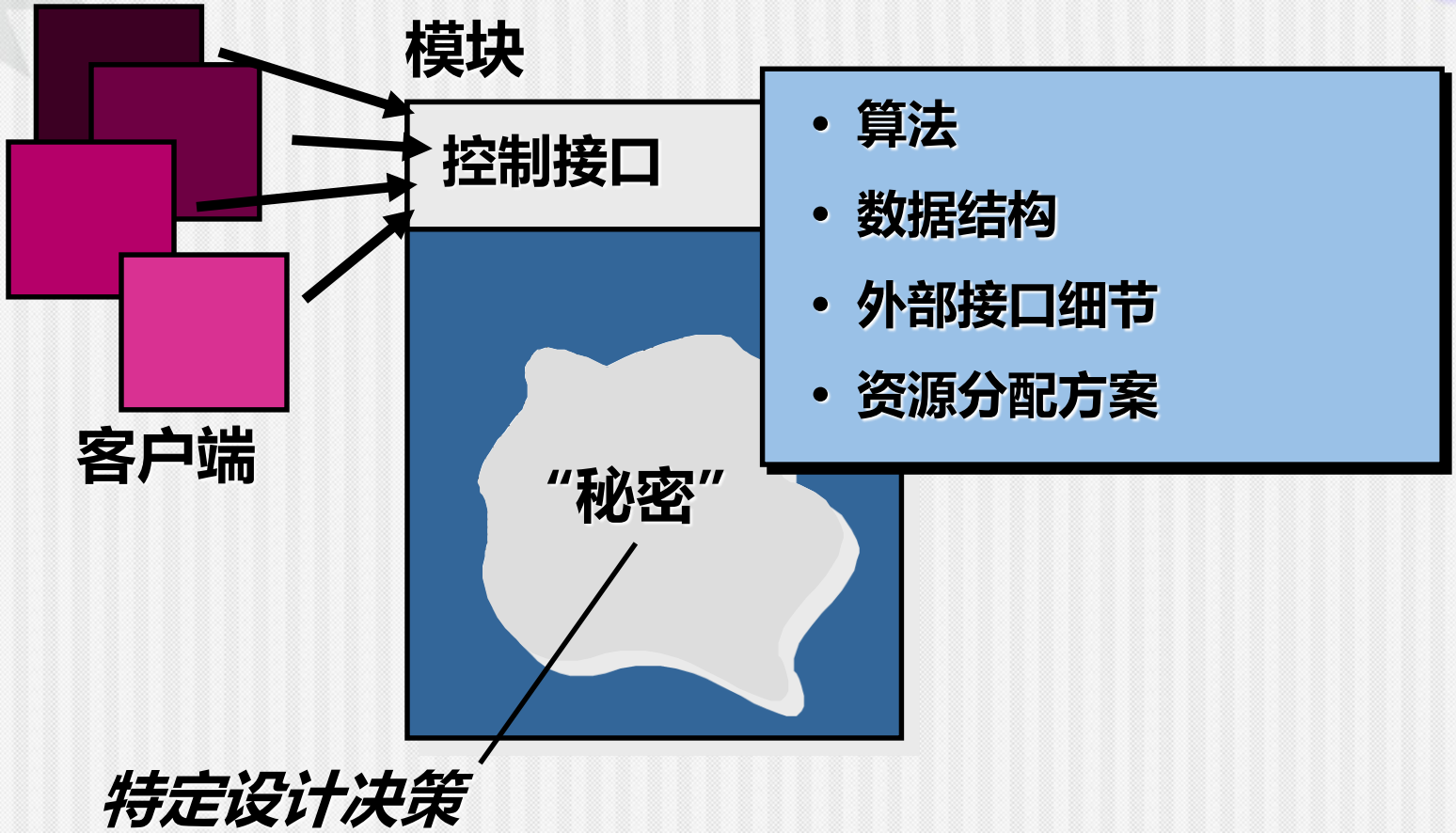
对于特定的软件设计来说，多少数量的模块是“合适的”？



模块化的优点

- 做模块化设计(以及由其产生的程序)
  - 可以使开发工作更容易规划;
  - 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;
  - 可以开展长期维护而没有严重的副作用。

11.3.6 信息隐蔽



信息隐蔽

- 模块化一个基本问题：
  - 将如何分解软件解决方案以获得最好的模块集合？
- 信息隐蔽原则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。
  - 即模块应该被特别说明并设计，使信息（算法和数据）包含在模块内，其他模块无需访问这些信息。

为什么隐蔽信息？

- 降低“副作用”的可能性；
- 减少局部设计决定对全局的影响；
- 突出控制接口处通信；
- 阻止全局数据使用；
- 促进封装——高质量设计的属性之一；
- 形成高质量软件；

信息隐蔽的定义和优点

- 定义一系列独立的模块→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实体。
  - 隐蔽定义并加强了模块内过程细节的访问约束
  - 和模块所使用的局部数据结构的访问约束。
- 作为模块化系统的一个设计标准，
  - 在测试和随后的软件维护过程中在需要修改时，将提供最大的益处。
  - 大多数数据和过程对其他部分是隐蔽的，在修改过程中不小心引入的错误不会传播到软件其他地方。

11.3.7 功能独立

- 是模块化、抽象概念和信息隐蔽的直接产物。
- 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。
- 即软件设计时每个模块
  - 仅涉及需要的某个特定子集，
  - 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。

功能独立的重要性

- 软件更容易开发——功能被分隔且接口被简化。
- 更容易维护和测试
  - 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。
- 是优秀设计的关键，设计又是软件质量的关键。
- 两条定性的标准评估：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。

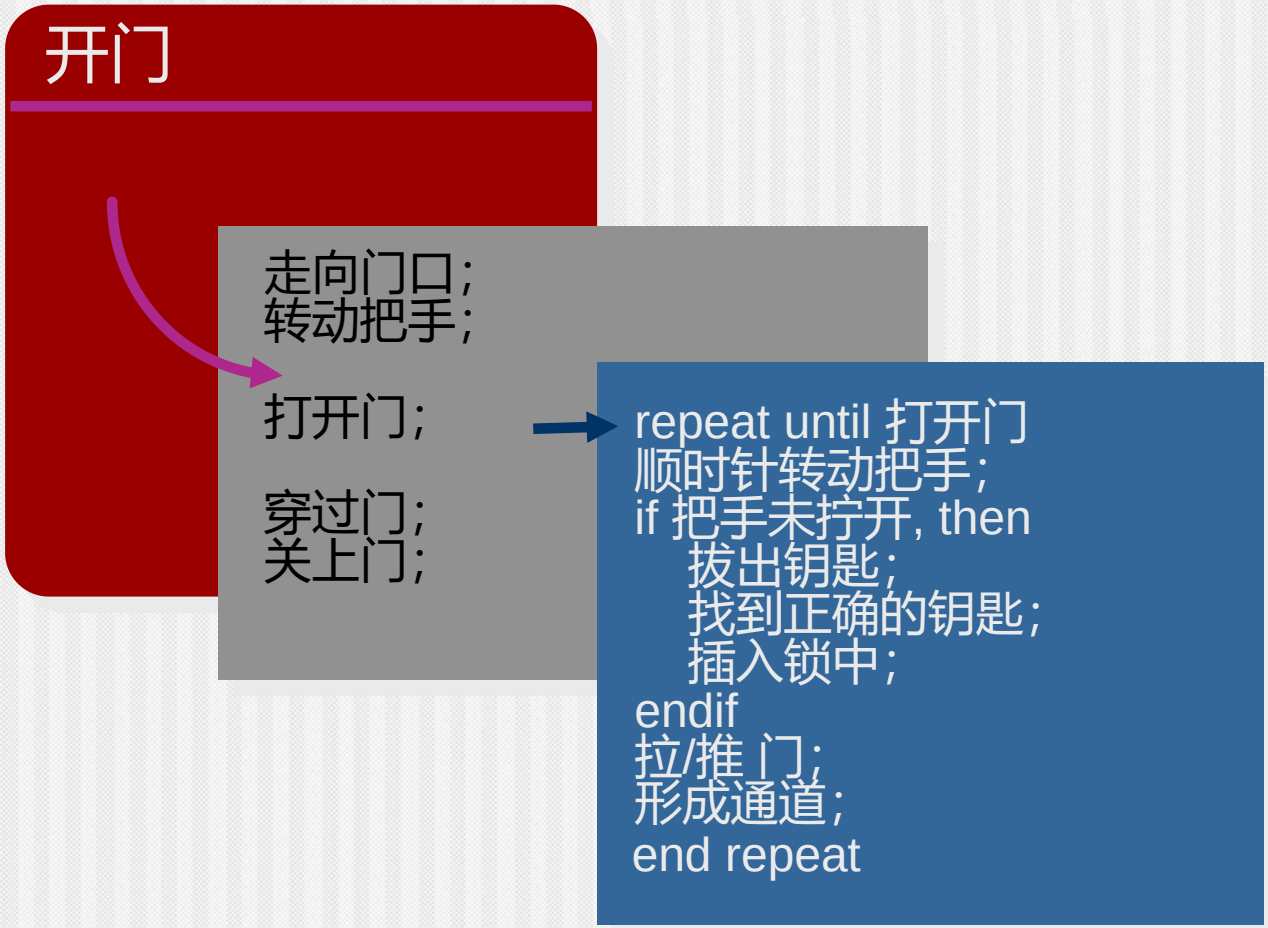
内聚性和耦合性

- 内聚性是信息隐蔽概念的自然扩展。
  - 内聚的模块执行独立的任务，很少与其他部分交互。
  - 简单地说，一个内聚的模块应该只完成一件事情。
- 耦合性表明多个模块之间的相互连接。
  - 依赖于模块之间接口的复杂性、引用或进入模块所在的点以及什么数据通过接口传递。在软件设计中，我们将尽力得到尽可能低的耦合。
- 高内聚、低耦合

11.3.8 求精

- 逐步求精是一种自顶向下的设计策略。
  - 通过连续精化过程细节层次来实现程序的开发
  - 通过逐步分解功能的宏观陈述→程序设计语言语句
- 求精实际上是一个细化的过程。
- 抽象和精化是互补的概念。
  - 抽象——明确说明过程和数据而同时忽略低层细节；
  - 精化有助于设计人员在设计过程中揭示低层的细节。

示例


开门
走向门口;
转动把手;
打开门;
穿过门;
关上门;
repeat until 打开门
顺时针转动把手;
if 把手未拧开, then
拔出钥匙;
找到正确的钥匙;
插入锁中;
endif
拉/推 门;
形成通道;
end repeat

11.3.9 方面

- 考虑两个需求，A和B。
  - “如果已经选择了一种软件分解[精化]，
  - 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”，
  - 那么需求A横切需求B
- 方面是一个横切关注点的表示

方面——举例

- 考虑**SafeHomeAssured.com**网站应用中的两个需求。依照用例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相机访问视频的模块。需求B是一个通用的安全需求，要求注册用户在使用**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*。
- 方面是一个横切关注点的表示，因此，需求“注册用户在使用**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。

11.3.10 重构

- 重构是一种重新组织的技术，可以简化构件的设计而无需改变其功能或行为。
- 不改变代码的外部行为而是改进其内部结构
- 当重构软件时，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
  - 以及其他不足，并通过修改获得更好的设计

11.3.11面向对象的设计概念

- 设计类：实体类、边界类、控制类
- 继承——超类的属性的任何改变都可以立即被所有子类继承
- 消息——刺激接收对象产生某种行为
- 多态——这种特性可显著减少扩展已存在的设计所需工作量

11.3.12 设计类

- 在设计模式演化时，定义一组设计类：
  - (1)提供设计细节对分析类求精，促成类的实现；
  - (2)实现支持业务解决方案的软件的基础设施。
- 分析类使用业务域的专门用语描述对象；
- 设计类更多地表现技术细节，作为实现的指导。
- 5种不同设计类，分别表示设计体系结构不同层次

设计类

- 分析类在设计中会被精化为实体类
- 边界类 在设计中被用于创建用户在实用软件时所观看并交互的接口
  - 边界类在设计中用于管理实体对象代表用户的方式
- 控制类 用于管理
  - 实体对象的创建或更新；
  - 从实体对象获得信息后的边界对象实例化；
  - 各对象间的复杂交流；
  - 对象间或用户与应用间数据交流的确定性

设计类的4个特征（1）

- 完整性与充分性：
  - 完整地封装所有可合理预见的类中属性和方法。
  - 充分性确保设计类只包含那些“对实现这个类的目的足够”的方法，不多也不少。
- 原始性：
  - 和某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，类就不应该再提供另外一种完成同一事情的方法。

设计类的4个特征（2）

- 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。
- 低耦合性：
  - 在设计模型内，设计类之间相互协作是必然的。
  - 协作应该保持在一个可以接受的最小范围内。
  - Demeter定律，建议一个方法应该只向周围类中的方法发送消息。（不要和陌生人说话）

SAFEHOME实例[26]

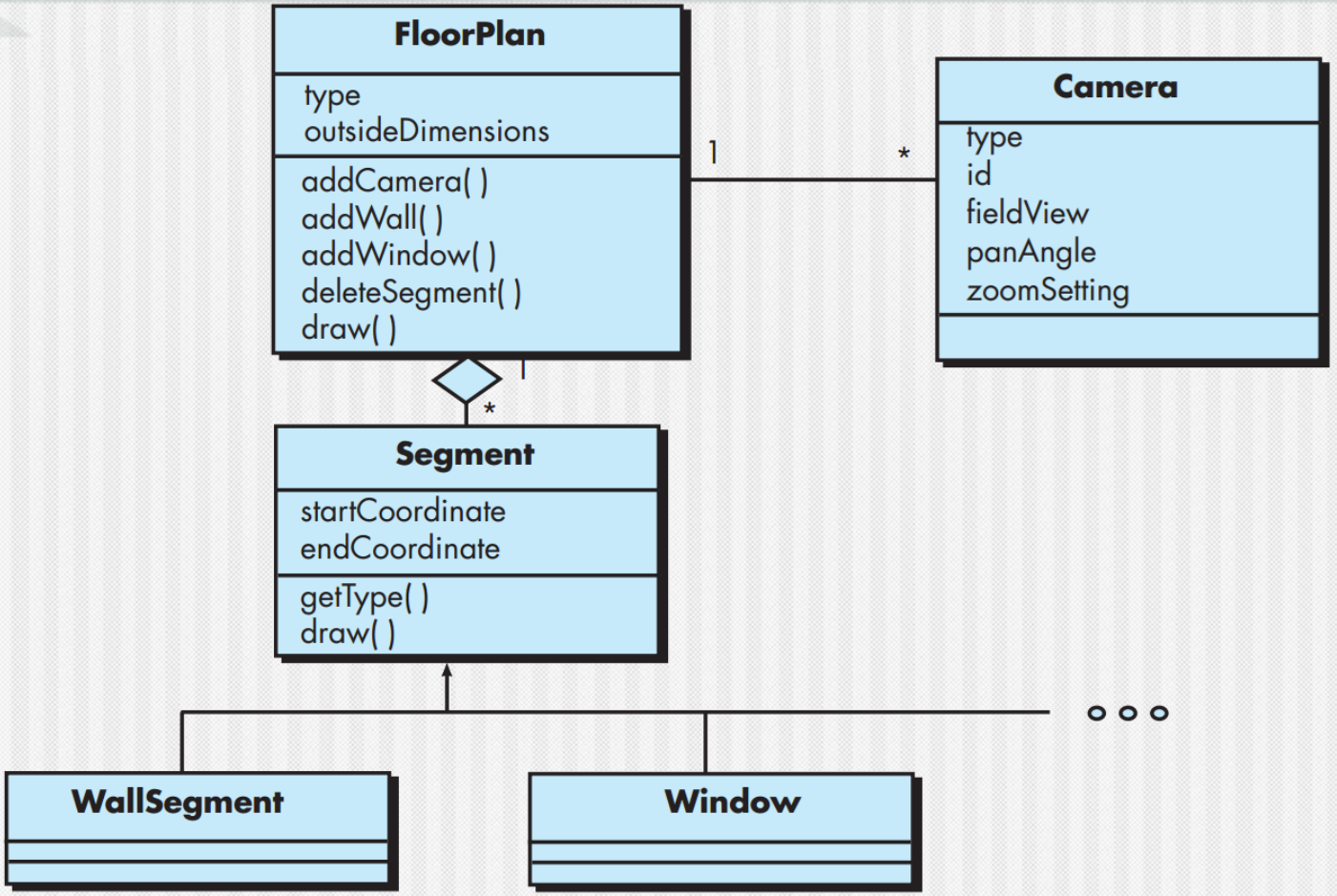
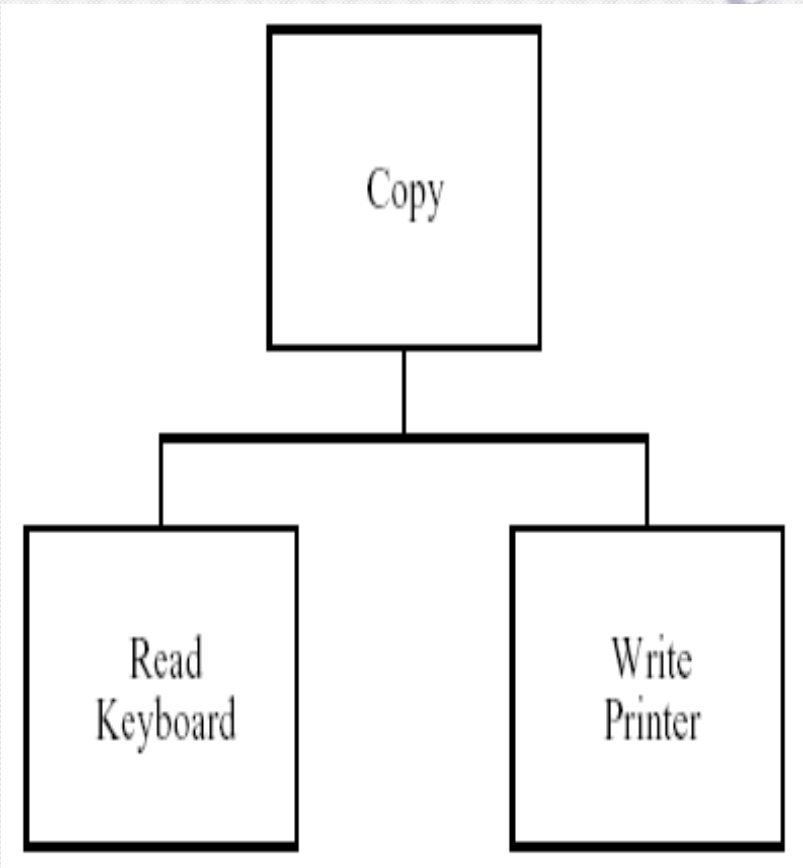


图11-3 （对比图9-2）

11.3.13 依赖倒置

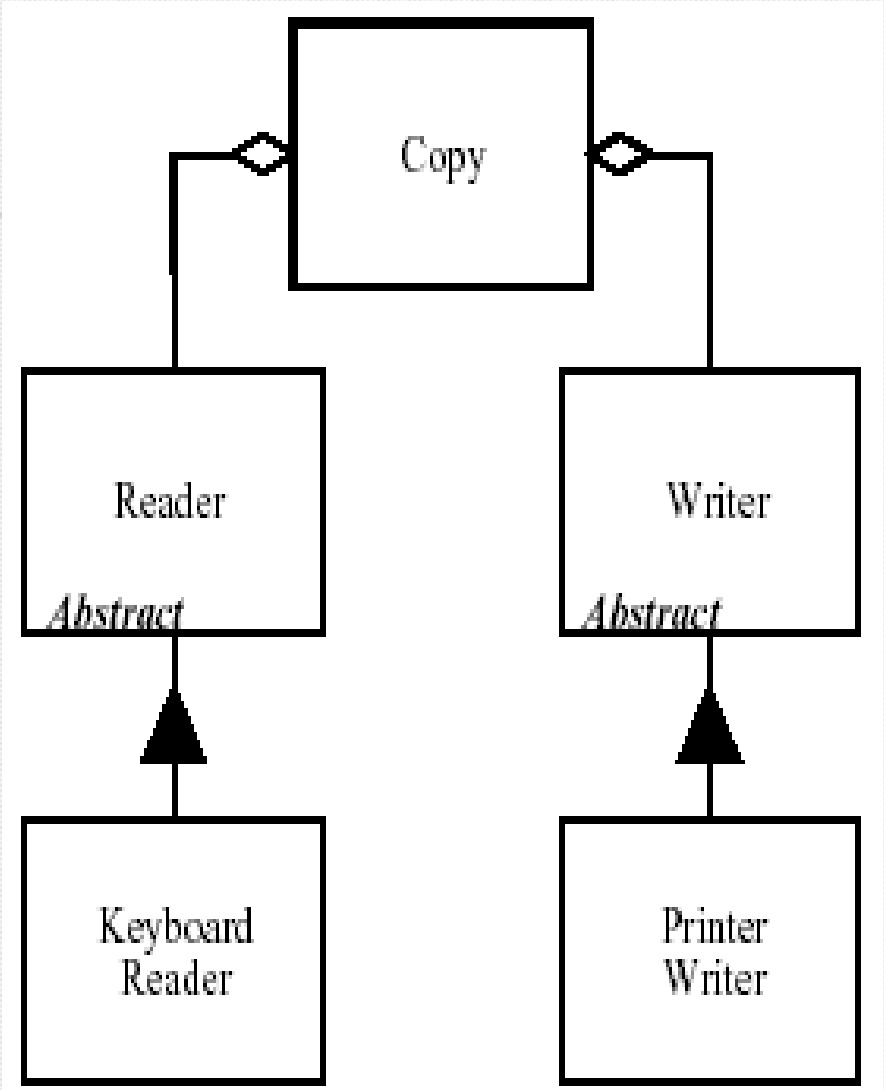
- 从一个例子说起：
  - 一个简单的程序，其任务就是实现将键盘输入的字符拷贝到打印机上

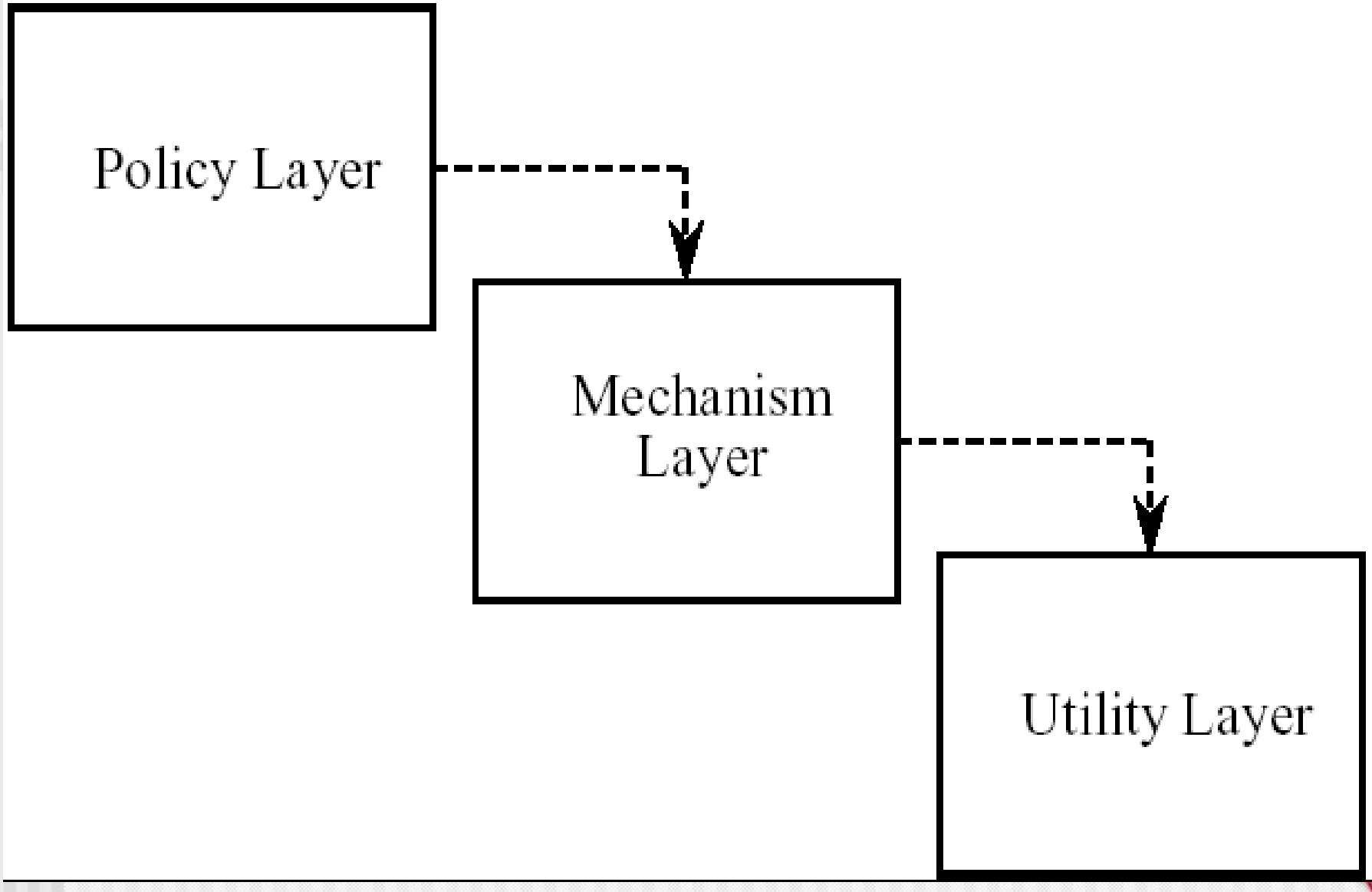
```
void Copy()
{
  int c;
  while ((c = ReadKeyboard() !=
     EOF)
     WritePrinter（c）;
}
```



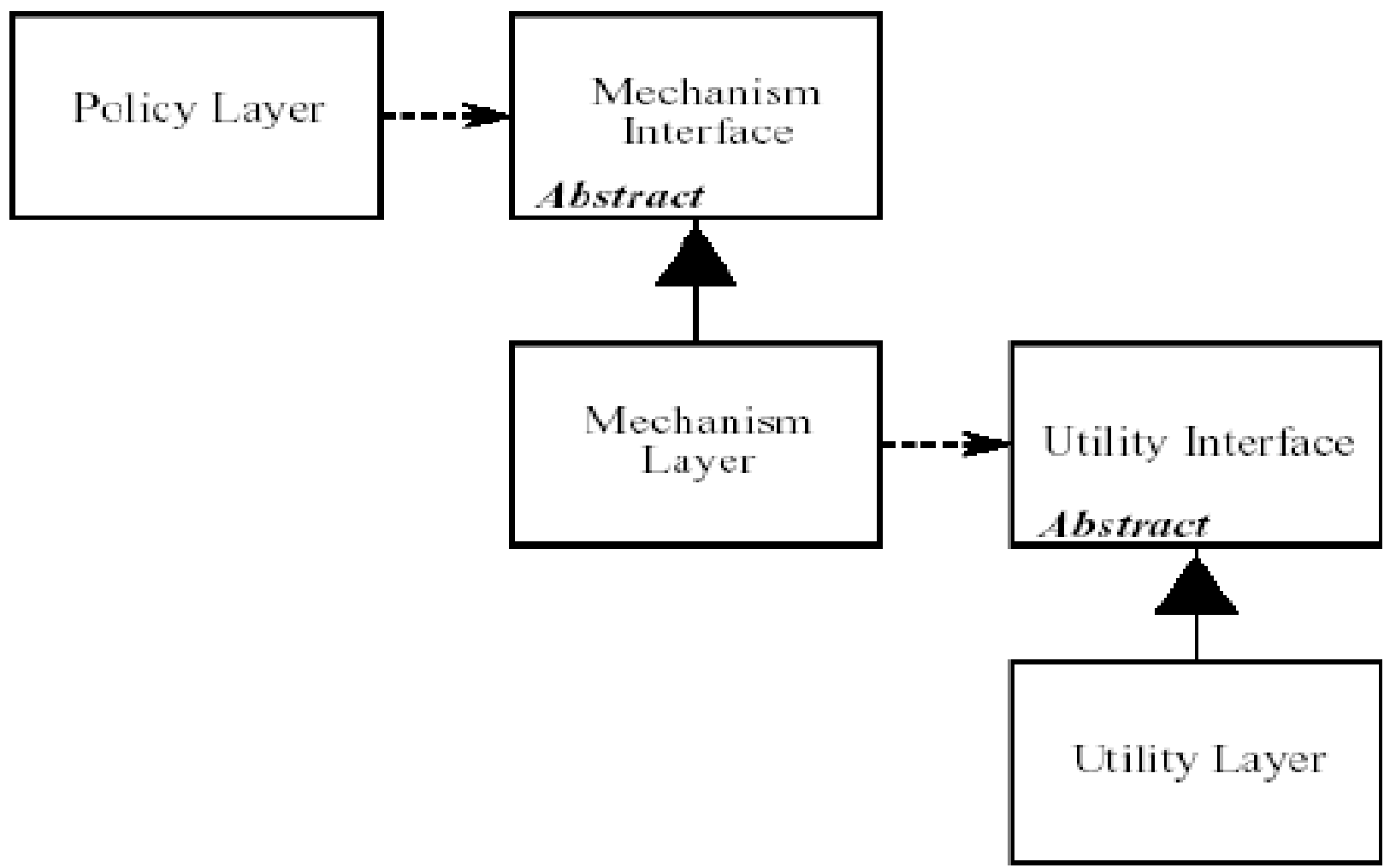
- 依赖倒置
  - 问题是包含高层策略的模块依赖于它所控制的低层模块

```
class Reader {
public:
    virtual int Read（） = 0;
};
class Writer {
public:
    virtual void Write（char） = 0;
};
void Copy(Reader& r, Writer& w)
{
    int c;
    while((c=r.Read() != EOF)
        w.Write（c）;
}
```



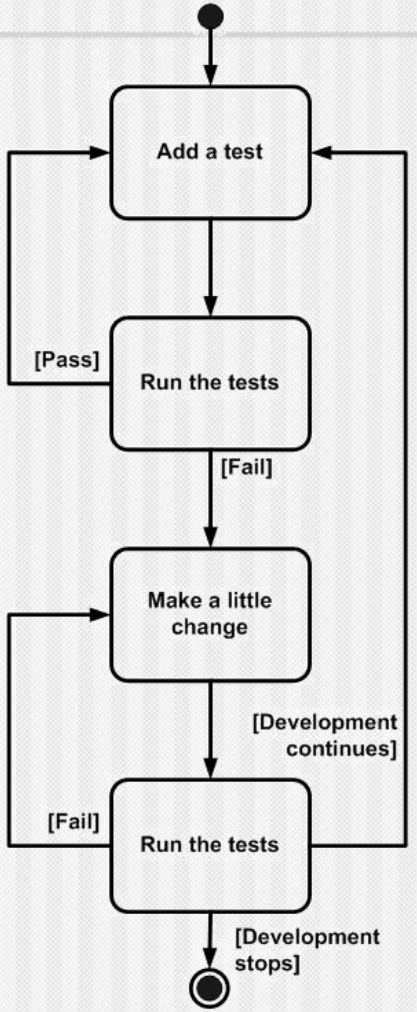

Policy Layer
Mechanism Layer
Utility Layer

- 下图显示了一个更为合适的模型



11.3.14 测试设计（测试驱动开发）

- 测试驱动开发是一种先开发测试的技术
- 是开发技术，而不是测试
- 是先文档再代码的过程，反映开发思路
- 每一行产品代码落笔之前，先写一个测试来给这行代码找一个存在的理由
- **TDD=TFD+重构**



工匠师傅的“测试驱动开发”

工匠一:先拉上一根水平线，砌每一块砖时，都与这根水平线进行比较，使得每一块砖都保持水平。

工匠二:先将一排砖都砌完，然后拉上一根水平线，看看哪些砖有问题，再进行调整。

TDD的目标

- **Clean Code That Works**
- Works : 让代码奏效，把事情做对
- Clean Code: 让代码洁净，把事情做好

TDD的节奏

- Refactoring

编写TestCase --> 实现TestCase --> 重构
（不可运行） （可运行） （重构）

11.4 设计模型

- 设计模型可以从两不同的维度观察，
- 过程维度表示设计模型的演化，设计工作作为软件过程的一部分被执行；
- 抽象维度表示过程的详细级别，分析模型的每个元素转化为一个等价的设计，然后迭代精化。

设计模型的维度

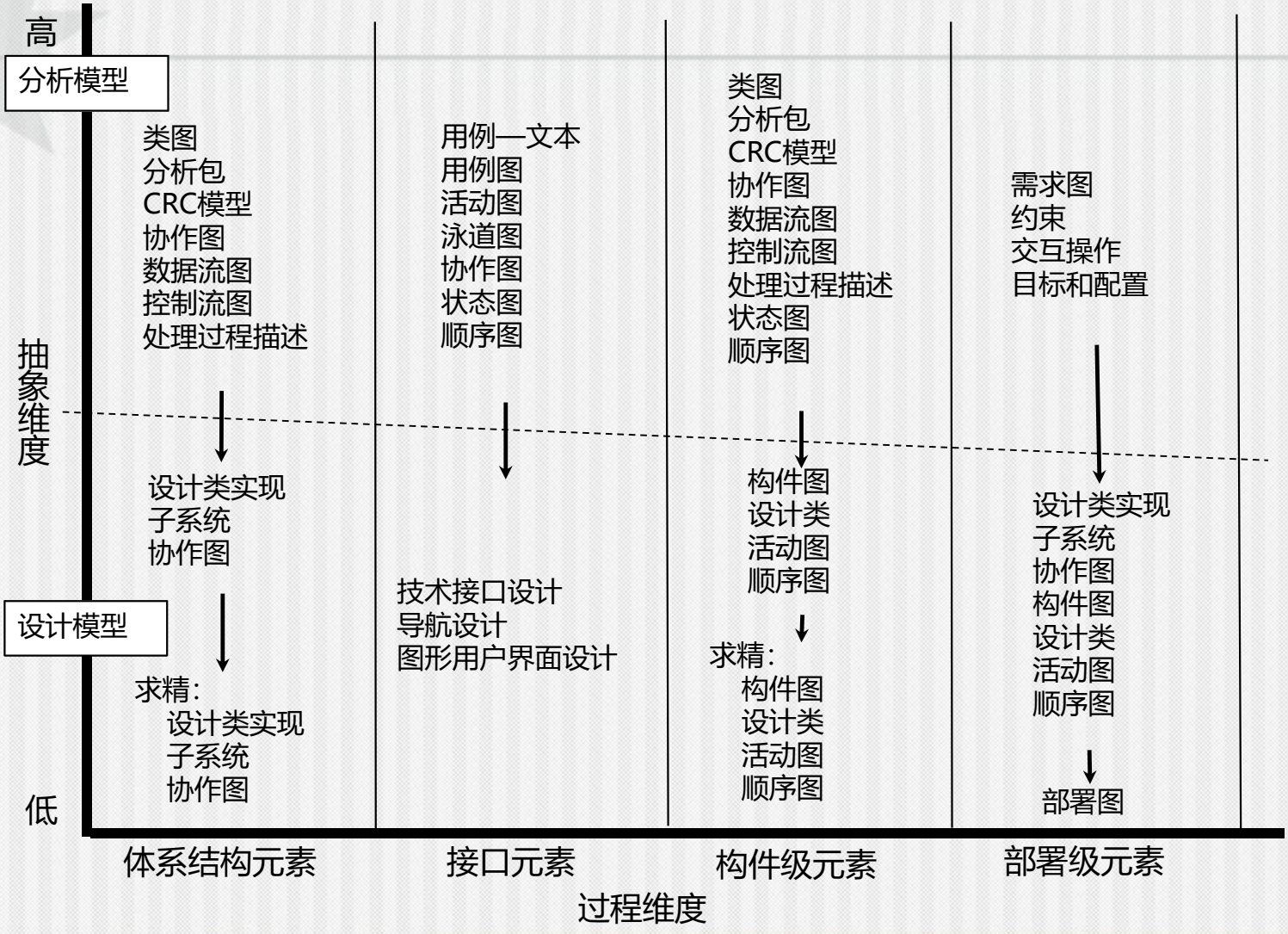


图11-4 设计模型的维度

11.4.1 数据设计元素

- 数据设计创建在高抽象级上（以客户/用户的数据观点）表示的数据模型和/或信息模型。
- 数据模型被精化为越来越和实现相关的特定表示，即基于计算机的系统能够处理的表示。
- 数据结构通常是软件设计的重要部分。
  - 在程序构件级，数据结构设计以及相关的处理这些数据的算法对于创建高质量的应用程序是至关重要的。
  - 在应用程序级，数据模型到数据库的转变是实现系统业务目标的关键。
  - 在业务级，收集存储在不同的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，这些技术影响业务本身的成功。

11.4.2 体系结构设计元素

- 软件的体系结构等效于房屋的平面图。
  - 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。
  - 平面图提供房屋的整体视图；
  - 而体系结构设计元素提供了软件的整体视图。
- 体系结构模型从以下三个来源获得：
  - (1)关于将要构建的软件的应用域信息；
  - (2)特定的分析模型元素，如数据流图或分析类、现有问题中它们的关系和协作；
  - (3)体系结构模式和风格的可获得性。

11.4.3 接口设计元素

- 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。
  - 绘图描绘了门窗的尺寸和形状、门窗的工作方式、设施连接入室的方式和在平面图上的室内布置。
  - 图纸可以告诉我们门铃在哪、是否使用内部通信以通知有客来访以及如何安装保安系统。
  - 告诉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。
- 软件接口设计元素告诉
  - 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。

3个重要的元素

- 接口设计有3个重要的元素：
  - (1)用户界面(UI)；
  - (2)和其他系统、设备、网络或其他的信息生产者或使用者的外部接口；
  - (3)各种设计构件之间的内部接口。
- 接口设计元素
  - 允许软件和外部通信，
  - 并使得构件之间能够内部通信和协作。

用户界面设计

- UI设计是软件工程的主要活动。
- UI活动包含
  - 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系统。

外部接口和内部接口

- 外部接口设计
  - 需要关于发送和接收信息的实体的确定信息。
  - 信息应在需求工程中收集，
  - 一旦开始进行接口设计，还要检验这些信息。
  - 外部接口设计应包括错误检查和适当的安全特征。
- 内部接口设计
  - 和构件级的设计紧密相关。
  - 分析类的设计实现了所有操作和消息传递模式
  - 不同类的操作之间能够进行通信和协作。

接口设计元素

- UML定义接口如下：
  - “接口是类、构件或其他分类（包括子系统）的外部可见的操作说明，而没有内部结构的规格说明。”
- 接口是
  - 一组描述类的部分行为的操作，
  - 并提供了那些操作的访问方法

SAFEHOME实例[27]

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无线PDA或移动电话实现。

ControlPanel类（图8-5）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WirelessPDA和MobilePhone），定义如图8-5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。

在控制面板的右边带有三角箭头的虚线（图8-5）表示ControlPanel类提供了KeyPad操作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。

SAFEHOME实例[28]

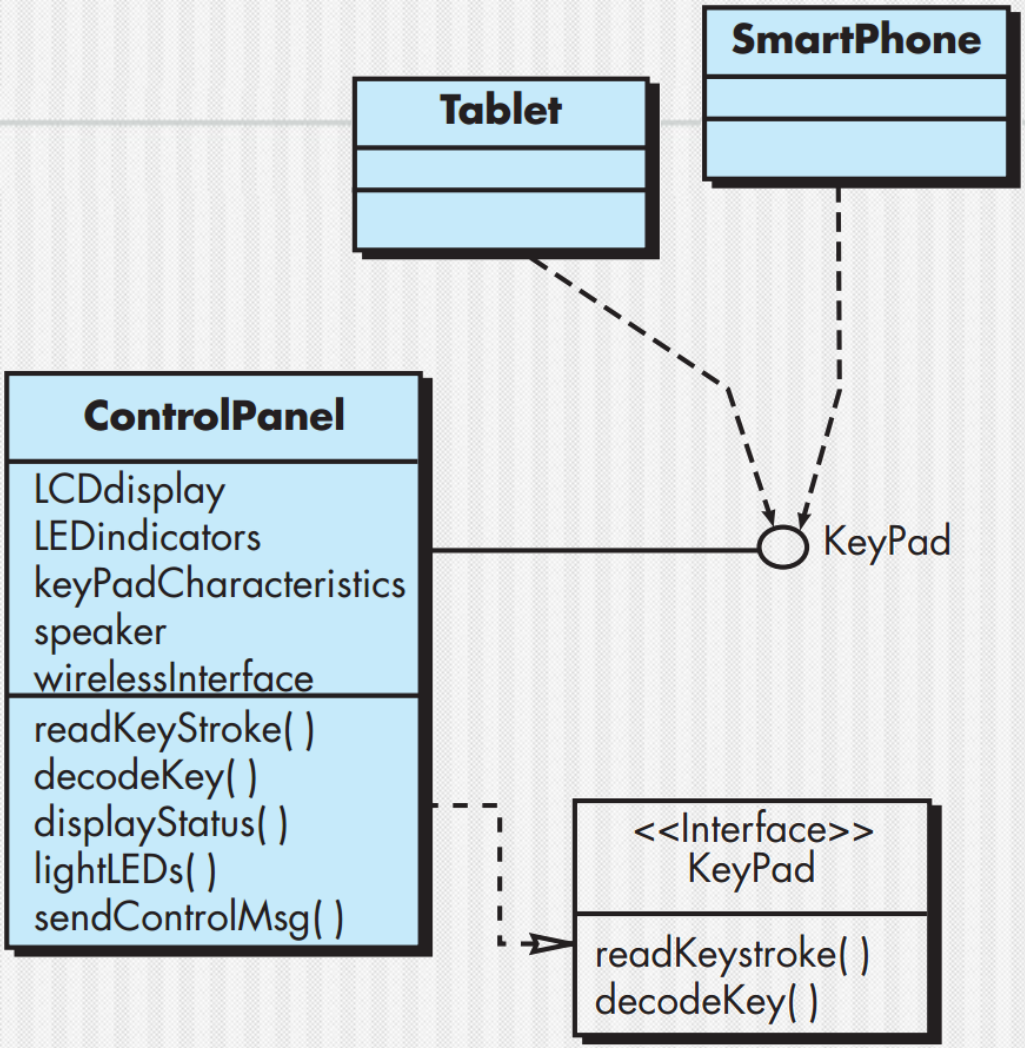


图11-5 ControlPanel的接口表示

11.4.4 构件级设计元素

- 软件的构件级设计相当于
  - 房屋中每个房间的一组详细绘图（以及规格说明）。
  - 描绘了每个房间内的布线和管道、电器插座和开关、水龙头、水池、浴室、浴盆、下水道、壁橱和储藏室的位置，还说明了所使用的地板、装饰以及和房间相关的任何细节。
- 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。
  - 构件级设计为所有本地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，
  - 并定义允许访问构件操作（行为）的接口。

SAFEHOME实例[30]

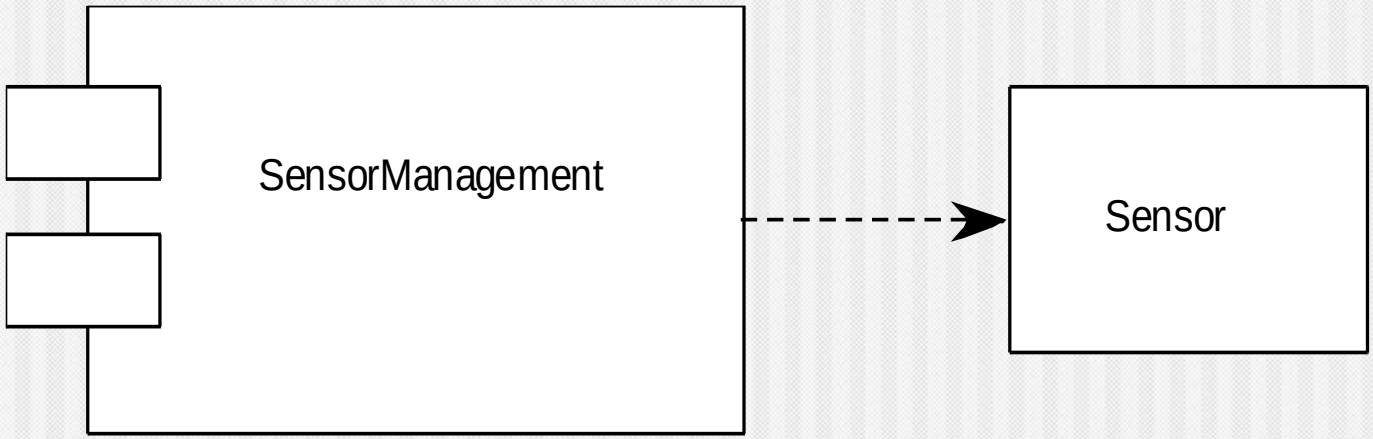


图11-6 SensorManagement的UML构件图

7.4.5 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分布。
  - 例如SafeHome产品元素被配置为在三个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。
- 在设计过程中，开发的UML部署图以及随后的精化如图

SAFEHOME实例[31]

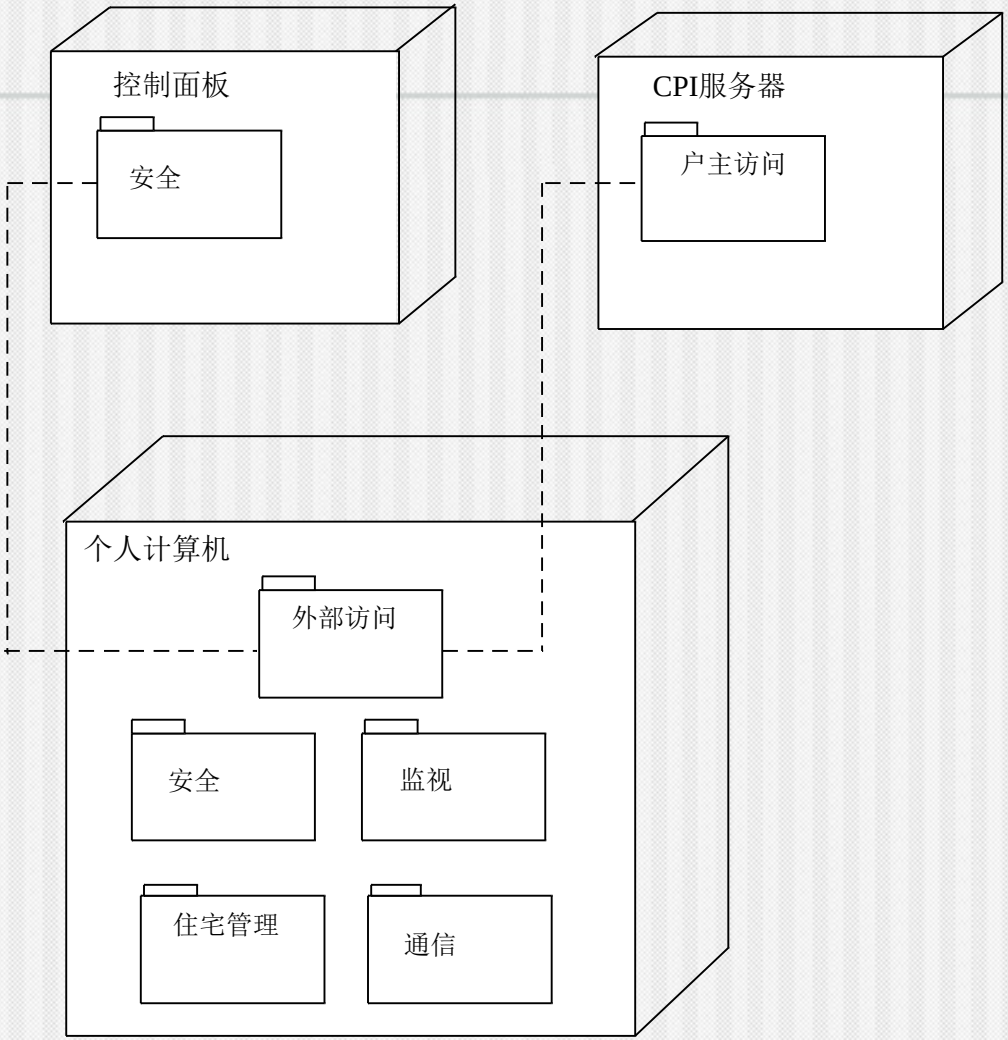


图11-7 SafeHome的UML部署图

谢谢！