信息与软件工程学院
School of Information and Software Engineering

高级软件工程

第二章 软件工程师的专业技能

姓名 | 许毅

xuyi0421@uestc.edu.cn

2024/2/21

第二章目录

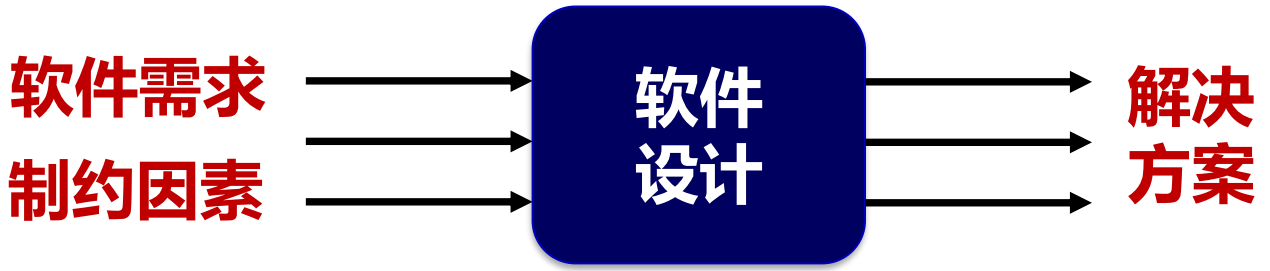
2.2.1

CATALOGUE

软件设计的概念

何为软件设计?

- **软件设计**
  - ✓针对**软件需求**，综合考虑各种**制约因素**，探究软件实现的**解决方案**
- **设计前提：软件需求**
  - ✓定义了要做什么样的软件
- **设计考虑：制约因素**
  - ✓**资源**：时间、人力、财力、开发辅助工具
  - ✓**技术**：技术平台，如DBMS还是文件系统

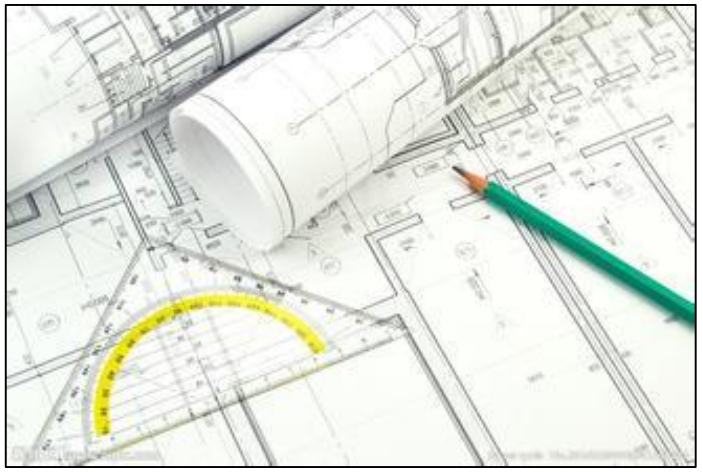


软件设计是要给出软件需求的实现解决方案

何为软件系统的解决方案?

- 描述了如何来构造和实现软件系统
  - ✓模块的组织
  - ✓模块的功能和接口
  - ✓模块间的交互
  - ✓模块内部的算法
  - ✓人机交互的界面和方式
  - ✓数据结构设计
  - ✓数据库的设计和组织 ……

- 不同设计内容
- 不同设计层次
- 不同设计视角



- 软件系统的解决方案类似于软件实现图纸
- 从实现的角度，软件设计方案应该是什么样的?

从需求到设计和编码

- **需求 → 设计**
  - ✓回答**如何做（How）→ 设计图纸**
  - ✓根据需求来进行设计，确保设计的**质量**
- **设计 → 实现**
  - ✓基于设计来指导**施工和实现**
  - ✓设计的好坏直接决定了最终产品的好坏！

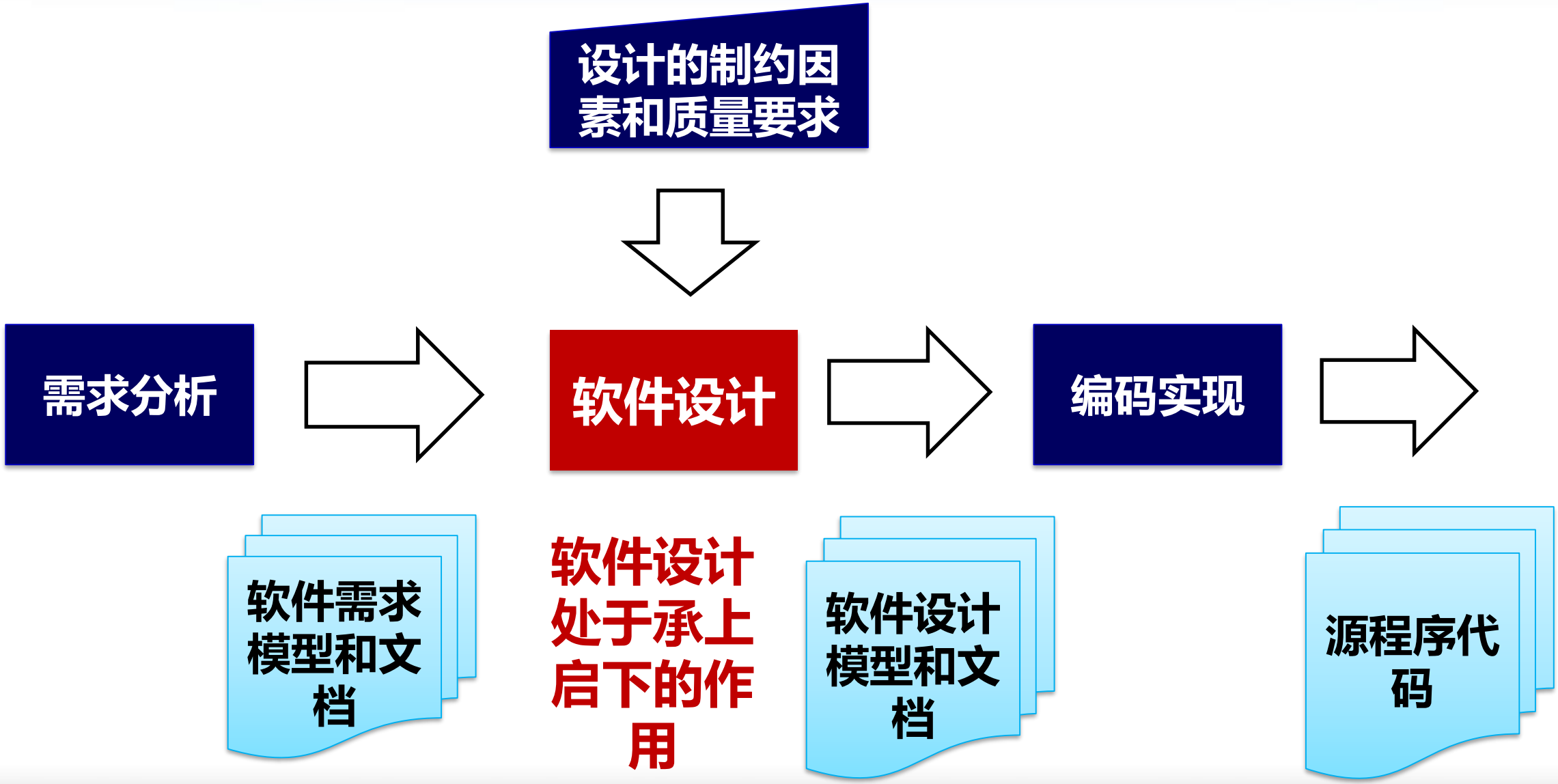
软件设计关注于软件需求的实现问题

软件需求

软件设计

编码实现

需求分析、软件设计、软件实现间的关系



软件设计是需求分析和软件实现间的桥梁

软件需求

要做什么，明确问题和目标

软件设计

如何做，绘制图纸

软件代码

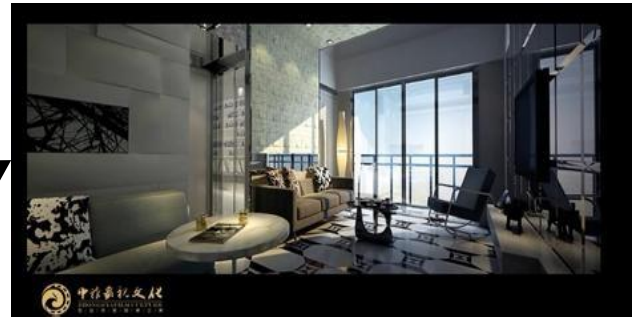
做出来，开展施工

设计的多样性和差异性：质量

- ✓ 如何区分设计的差异性?
- ✓ 如何评价设计的优劣?
- ✓ 除了满足需求之外，设计还需要注意哪些要素?

用户需求

软件设计



设计结果1

设计结果2

设计结果n

- ✓ 软件设计是一个创作的过程
- ✓ 一个软件需求会有多种软件设计方案

软件设计的质量要求

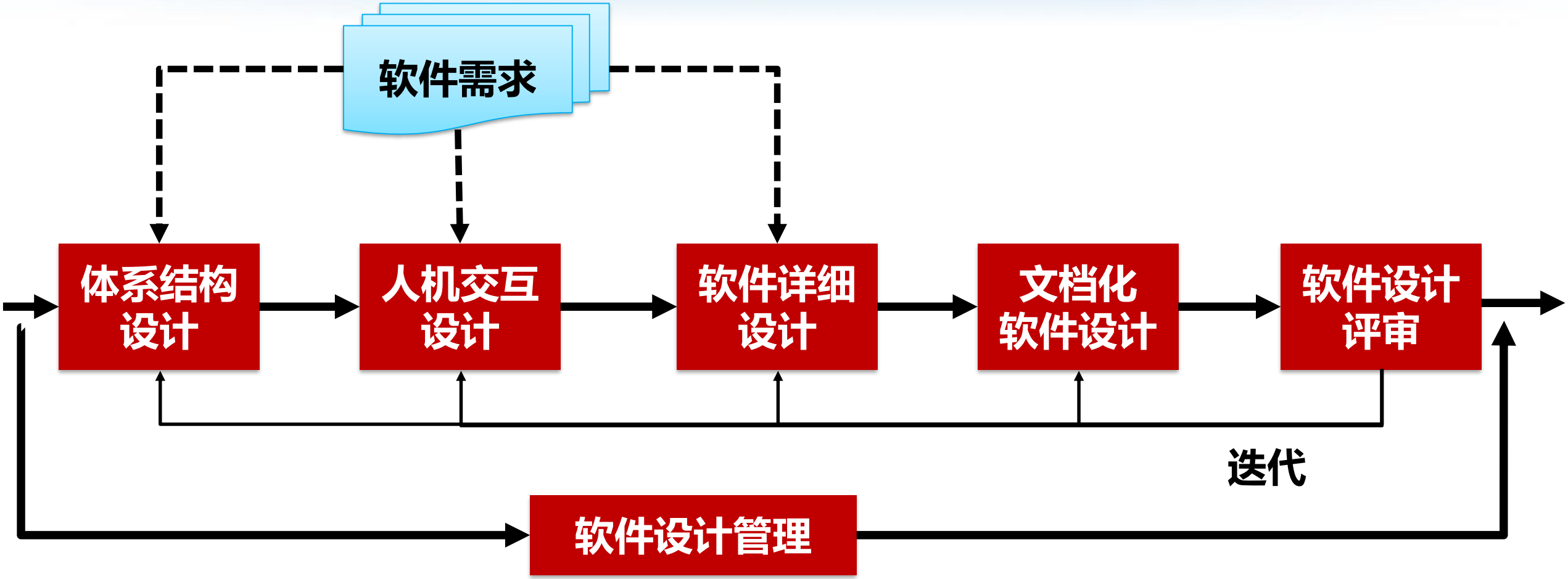
- **正确性**
  - ✓ 正确实现所有的软件需求项；设计元素间无逻辑冲突；在技术平台和软件项目的可用资源约束条件下，采用程序设计语言可完整地实现设计模型
- **充分性**
  - ✓ 所有的设计元素已充分细化，模型易于理解，编程人员勿需再面对影响软件功能和质量的技术抉择或权衡
- **优化性**
  - ✓ 以合理的、充分优化的方式实现软件需求模型，目标软件产品能够表现出良好的软件质量属性，尤其是正确性、有效性、可靠性和可修改性
- **简单性**
  - ✓ 模型中的模块的功能或职责尽可能简明易懂，模块间的关系简单直观，模型的结构尽可能自然地反映待解软件问题的结构

高质量软件设计的特点

- 正确性
- 可靠性
- 可维护性
- 可重用性
- 可追踪性
- 可移植性
- 可互操作性
- 有效性
- 安全性 ……

- 设计不仅要满足需求，还要有好的质量！
- 要从多个利益相关者的角度来理解设计的“质量”
  - ✓ 用户、开发者、维护人员等
- 设计要内外兼修
  - ✓ 内在质量和外在质量

软件设计过程



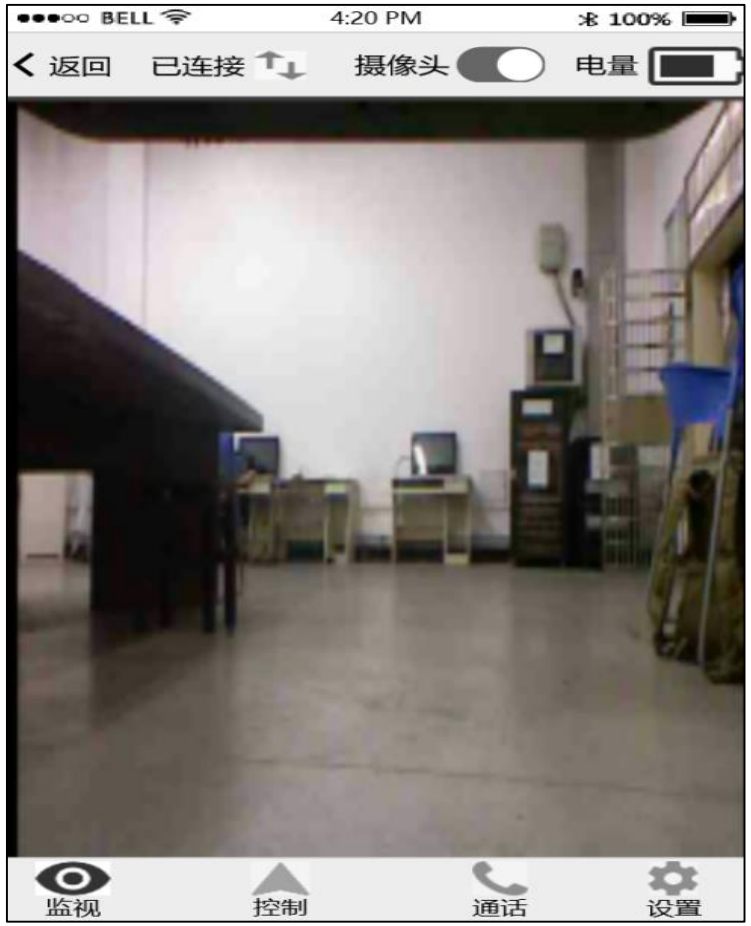
软件设计过程 – 软件体系结构设计

- **从全局和宏观视角、站在最高抽象层次来设计软件系统**
  - ✓构成要素及其关系
  - ✓职责分派、接口定义
  - ✓相互交互及协作行为
- **每个模块为“黑盒子”**
- **设计关注的质量要素**
  - ✓可扩展、可维护、可重用、可移植、可互操作等等

- **要素：函数、方法、类、程序包**
- **关系：依赖、交互**
- **区别：粒度**

软件设计过程 – 用户界面设计

- 设计软件对外展示以及与用户进行交互的界面，关注软件如何与用户进行交互
  - ✓**输出**：告诉给用户的信息
  - ✓**输入**：需要用户提供的信息



- 设计关注的质量要素
  - ✓直观、友好、易于操作和理解等

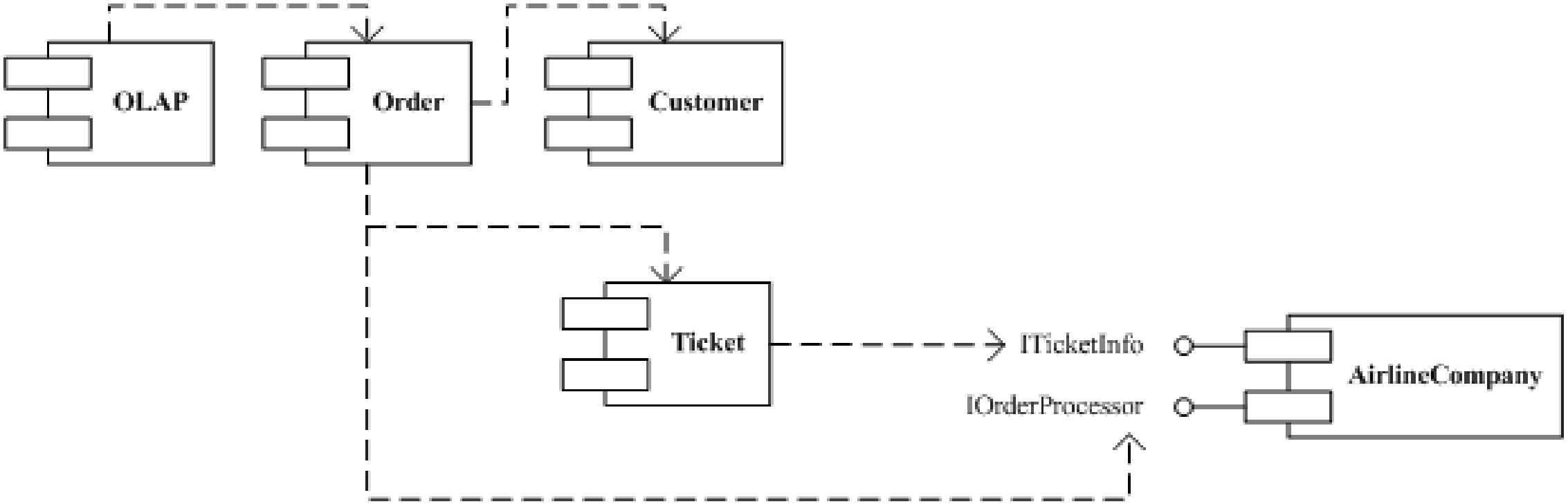
软件设计过程 – 软件详细设计

- **对体系结构设计和人机交互设计成果进行细化和精化，获得高质量的、充分细化的软件设计模型**
  - ✓**构件和类设计**：细化各个构件和类设计，如属性、操作、状态等
  - ✓**接口设计**：构件和类等提供的交互接口
  - ✓**算法设计**：实现特定功能的具体执行流程和算法
  - ✓**数据设计**：信息描述 → 计算机可以处理的数据描述

- **设计关注的质量要素**
  - ✓有效、高效、可靠、易于维护等等

示例：软构件及接口设计

❑软构件及其之间的关系



示例：类设计

public class Contact {

- ✓private static HashMap<String, String> sContactCache;
- ✓private static final String TAG = "Contact";
- ✓private static final String CALLER_ID_SELECTION;
- ✓public static String getContact(Context context, String phoneNumber)
- ✓……

}

- **给出类层次的设计信息**
  - **属性**
  - **方法及其算法等**

软件设计过程 – 其它工作

❑撰写设计文档

- ✓基于软件设计及其成果，按照软件设计规格说明书的规范和要求，撰写软件设计文档，详细记录软件设计的具体信息

❑评审软件设计

- ✓对软件设计制品（包括设计模型和文档）进行评审，验证软件设计是否实现了软件需求，分析软件设计的质量，发现软件设计中存在的缺陷和问题，并与多方人员一起协商加以解决

❑软件设计管理

- ✓对软件设计变化以及相应的软件设计制品进行有效的管理，包括追踪软件设计变化、分析和评估软件设计变化所产生的影响、对变化后的软件设计制品进行配置管理等等

2.2.2

CATALOGUE

面向对象的设计

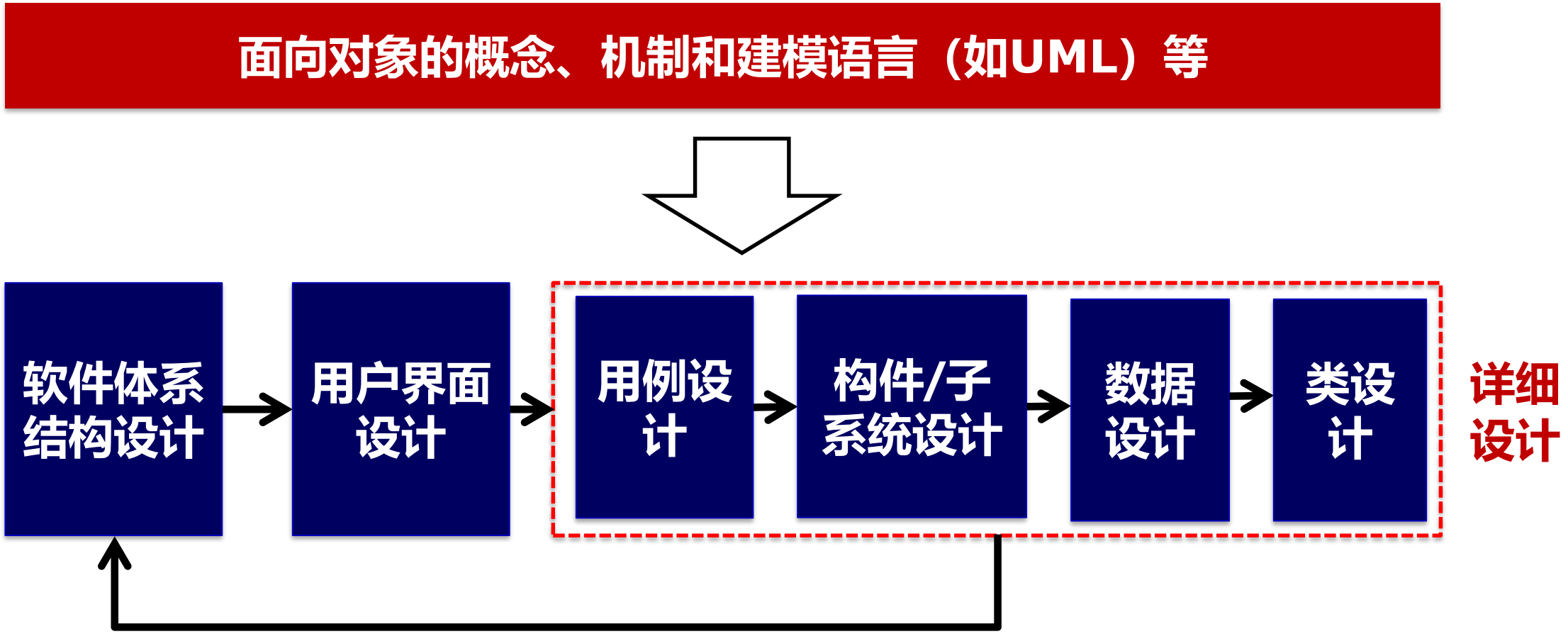
1 面向对象软件设计方法学的基本思想

- 针对面向对象需求分析所得到的**软件需求模型**（如用例图、交互图、分析类图），对其进行不断**精化**（而非转换），获得软件系统的**各类软件设计元素**，如子系统、构件、设计类等，产生不同视角、不同抽象层次的**软件设计模型**，如软件体系结构图、用例设计交互图、设计类图、活动图等，形成软件系统**完整和详尽的设计方案**

面向对象软件设计方法学

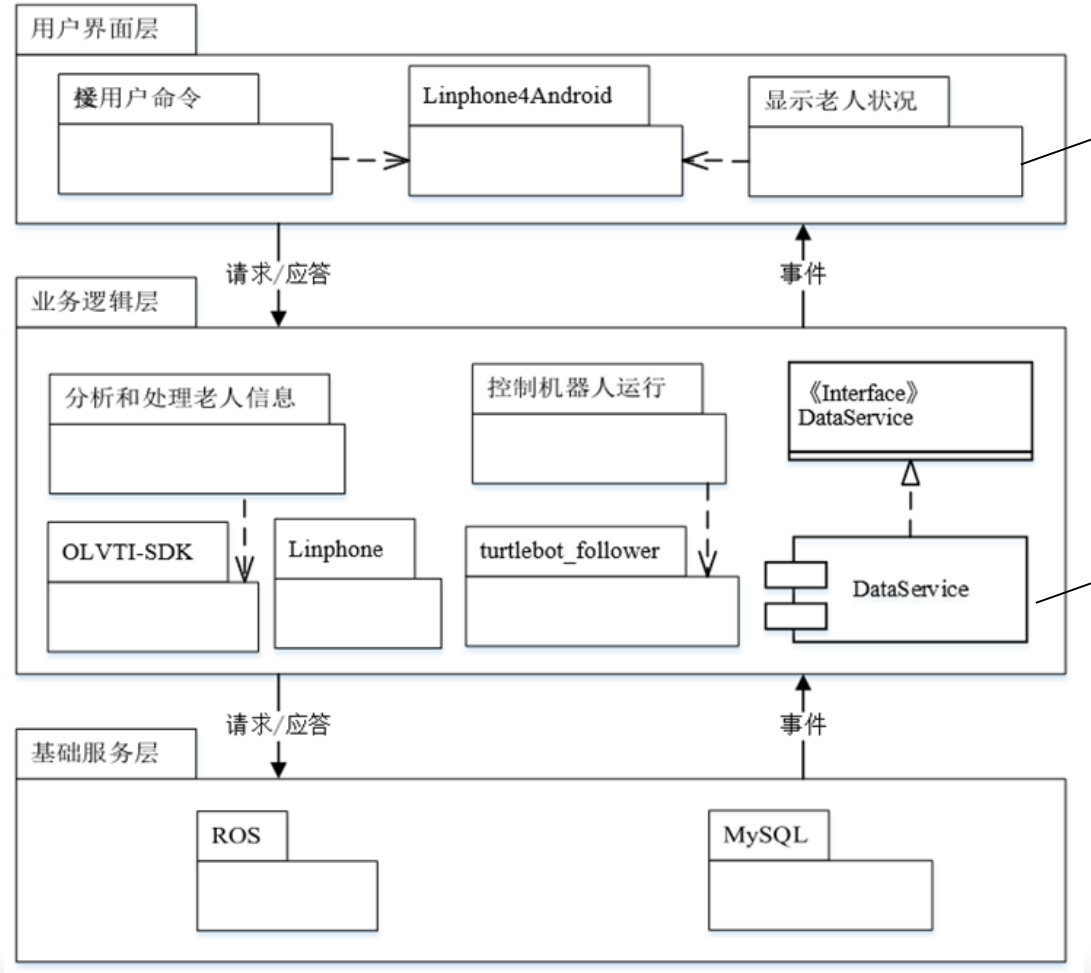
提供了概念、机制、过程、策略等来支持OO软件设计，产生高质量软件设计

2 面向对象软件设计过程



示例：面向对象的软件设计表示

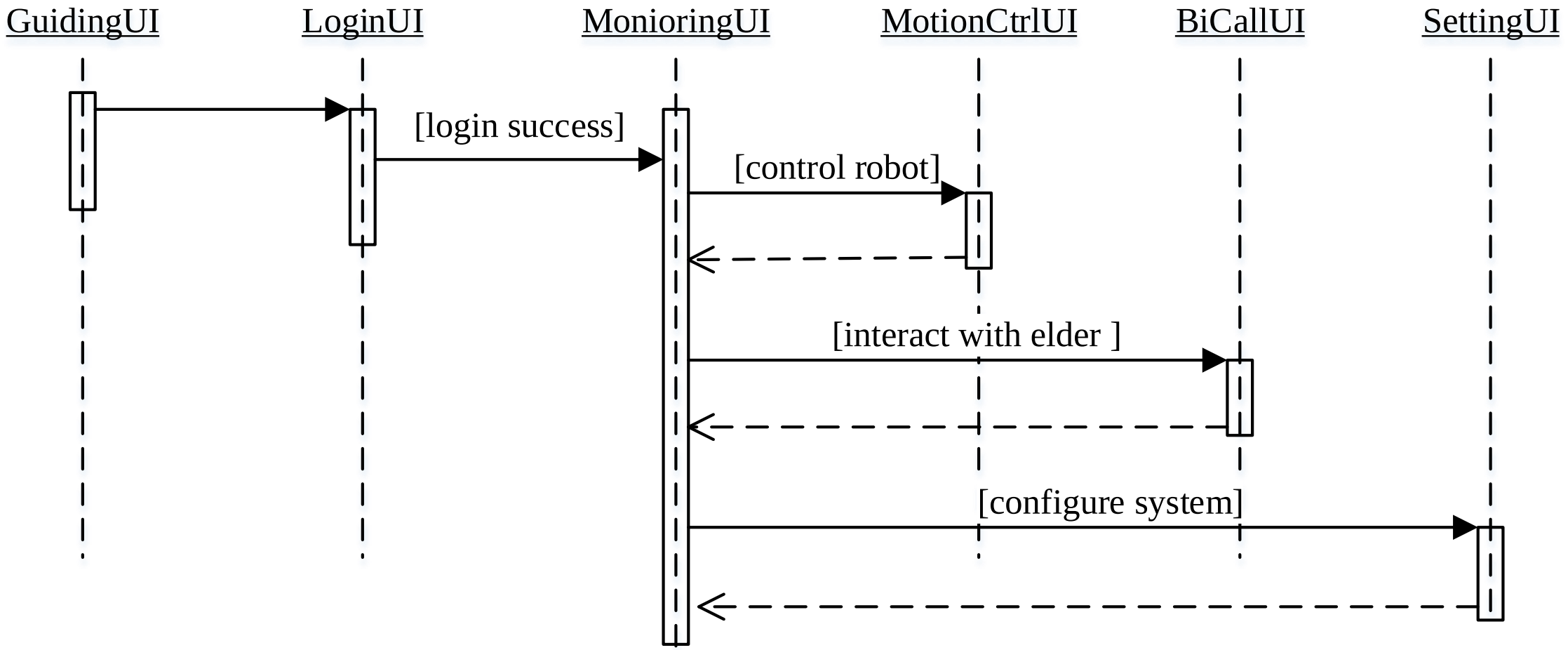
❑用包图表示的软件体系结构设计



包来表示体系结构要素

软构件来表示体系结构要素

示例：用交互图来表示用户界面设计



顺序图用于表示对象之间的交互时序及内容

面向对象软件设计的优势（1/2）

❑高层抽象和自然过渡

- ✓面向对象概念更加**贴近于现实世界**，有助于对应用问题以及软件系统的直观理解和建模
- ✓采用相同的一组抽象和概念来进行描述和分析，基于模型的精化手段来实现软件设计，极大简化了软件设计工作
- ✓面向对象模型更易于为人们所接受，可减少软件工程师与用户之间的交流鸿沟，有助于支持大型复杂软件系统的开发

❑多种形式和粗粒度的软件重用

- ✓提供了多种方式来**支持软件重用**，进而有助于提高软件开发的效率和质量

面向对象软件设计的优势（2/2）

❑系统化的软件设计

- ✓系统地支持软件设计阶段的所有工作，包括**体系结构设计、用户界面设计、数据设计、软构件设计、子系统设计、用例设计、类设计等等**

❑支持软件的扩展和变更

- ✓提供了**接口、抽象类、继承、实现**等多种机制，可以设计出易于扩展和变更的软件设计模型

2.3.3

CATALOGUE

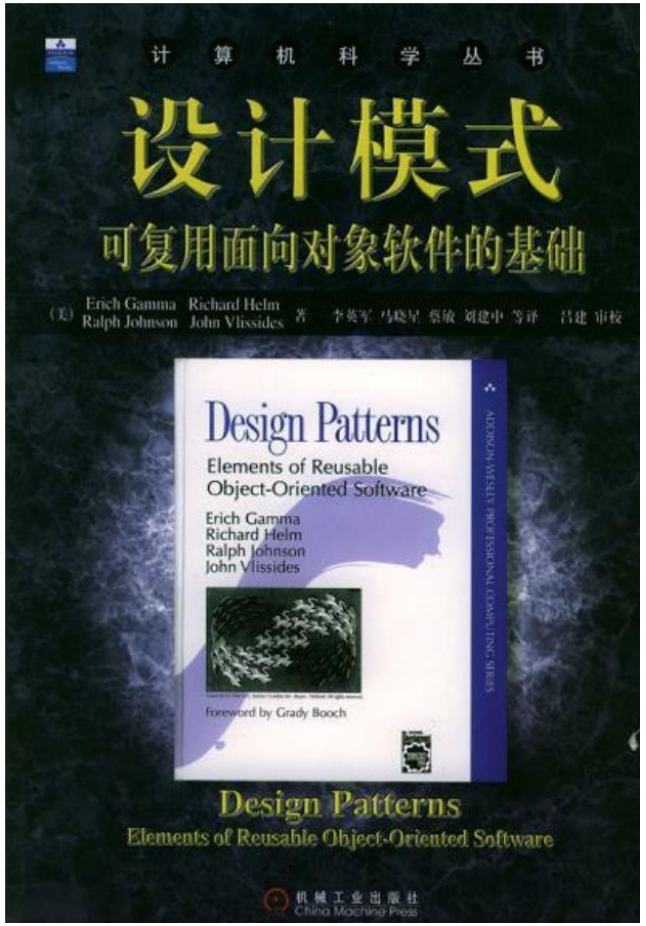
设计模式

什么是设计模式

- “每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次 地使用该方案而不必做重复劳动”。

——Christopher Alexander

GOF 设计模式



- 历史性著作《设计模式：可复用面向对象软 件的基础》一书中描述了23种经典面向对象 设计模式，创立了模式在软件设计中的地位。

- 由于《设计模式》一书确定了设计模式的地 位，通常所说的设计模式隐含地表示“面向 对象设计模式”。但这并不意味“设计模式” 就等于“面向对象设计模式”

Strategy 策略模式

动机（Motivation）

- 在软件构建过程中，某些对象使用的算法可能多种多样，经常改变，如果将这些算法都编码到对象中，将会使对象变得异常复杂；而且有时候支持不使用的算法也是一个性能负担。

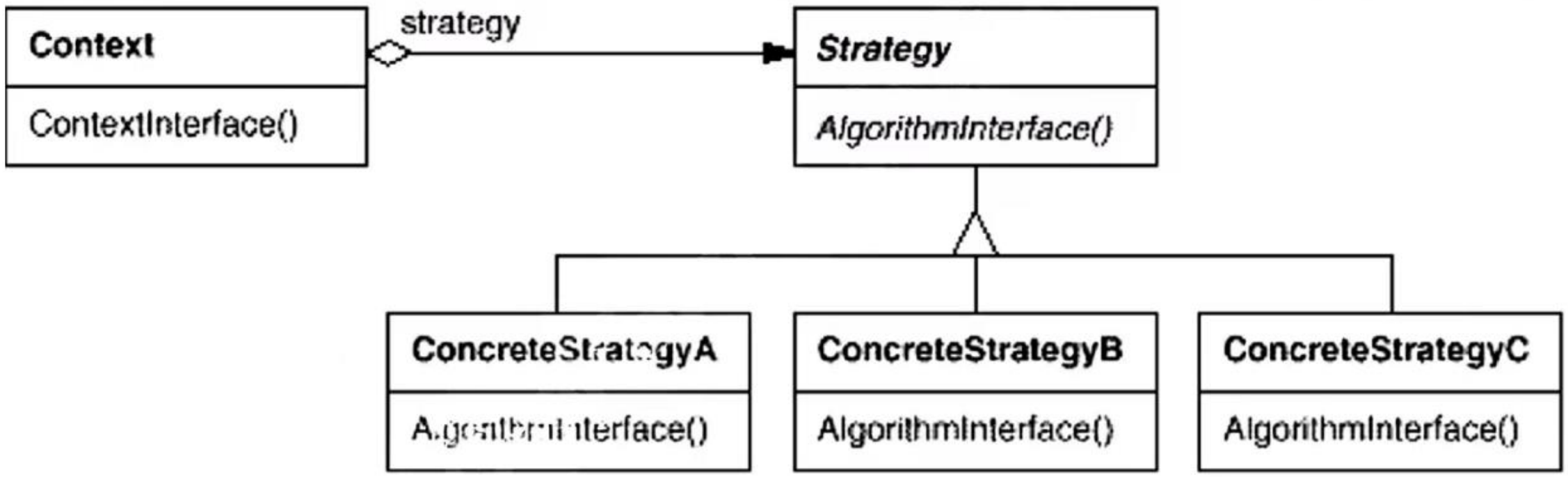
- 如何在运行时根据需要透明地更改对象的算法？将算法与对象本身解耦，从而避免上述问题？

模式定义

定义一系列算法，把它们一个个封装起来，并且使它们可互相替换（变化）。该模式使得算法可独立于使用它的客户程序(稳定)而变化（扩展，子类化）。

——《设计模式》GoF

结构（Structure）



要点总结

- Strategy及其子类为组件提供了一系列可重用的算法，从而可以使得类型在运行时方便地根据需要在各个算法之间进行切换。
- Strategy模式提供了用条件判断语句以外的另一种选择，消除条件判断语句，就是在解耦合。含有许多条件判断语句的代码通常都需要Strategy模式。
- 如果Strategy对象没有实例变量，那么各个上下文可以共享同一个Strategy对象，从而节省对象开销。

Observer 观察者模式

动机（Motivation）

- 在软件构建过程中，我们需要为某些对象建立一种“通知依赖关系”——一个对象（目标对象）的状态发生改变，所有的依赖对象（观察者对象）都将得到通知。如果这样的依赖关系过于紧密，将使软件不能很好地抵御变化。

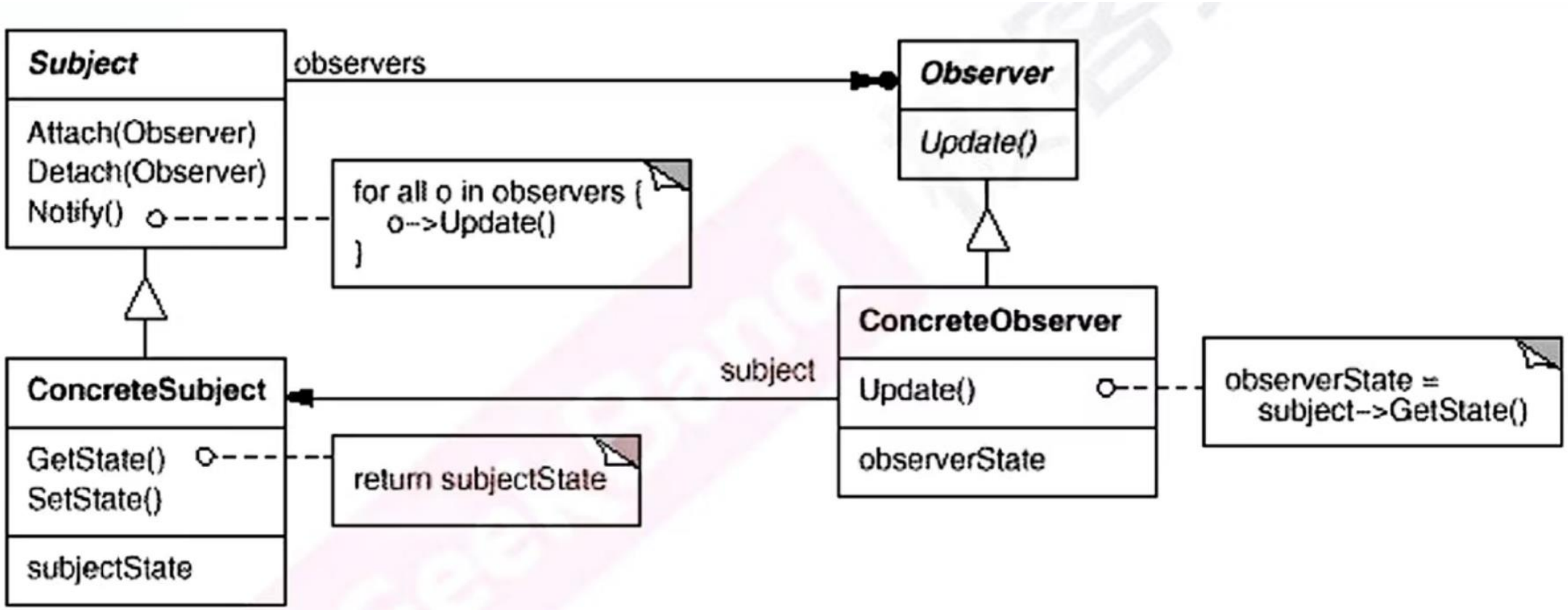
- 使用面向对象技术，可以将这种依赖关系弱化，并形成一种稳定的依赖关系。从而实现软件体系结构的松耦合。

模式定义

定义对象间的一种一对多（变化）的依赖关系，以便当一个对象(Subject)的状态发生改变时，所有依赖于它的对象都得到通知并自动更新。

——《设计模式》GoF

结构（Structure）



要点总结

- 使用面向对象的抽象，Observer模式使得我们可以独立地改变目标与观察者，从而使二者之间的依赖关系松耦合。
- 目标发送通知时，无需指定观察者，通知（可以携带通知信息作为参数）会自动传播。
- 观察者自己决定是否需要订阅通知，目标对象对此一无所知。
- Observer模式是基于事件的UI框架中非常常用的设计模式，也是MVC模式的一个重要组成部分。

Decorator 装饰模式

3

动机（Motivation）

- 在某些情况下我们可能会“过度地使用继承来扩展对象的功能”，由于继承为类型引入的静态特质，使得这种扩展方式缺乏灵活性；并且随着子类的增多（扩展功能的增多），各种子类的组合（扩展功能的组合）会导致更多子类的膨胀。

- 如何使“对象功能的扩展”能够根据需要来动态地实现？同时避免“扩展功能的增多”带来的子类膨胀问题？从而使得任何“功能扩展变化”所导致的影响将为最低？

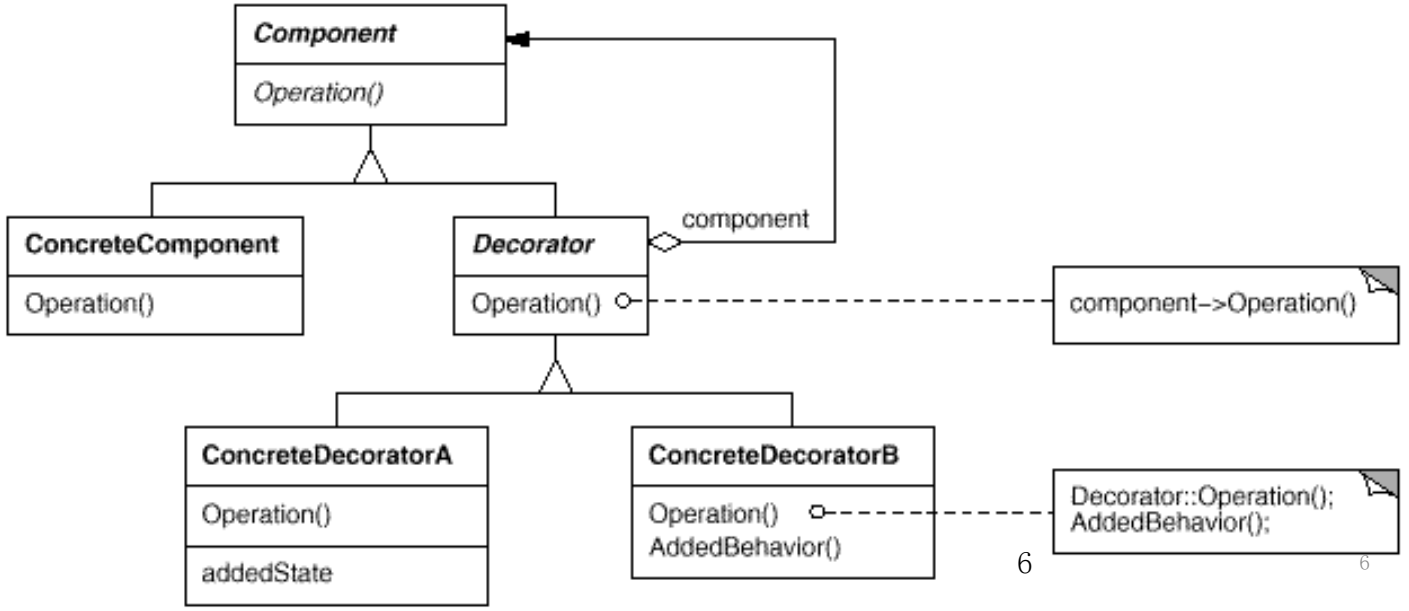
模式定义

动态（组合）地给一个对象增加一些额外的职责。就增加功能而言，Decorator模式比生成子类（继承）更为灵活（消除重复代码 & 减少子类个数）。

——《设计模式》GoF

5

结构（Structure）



6

要点总结

- 通过采用组合而非继承的手法， Decorator模式实现了在运行时动态扩展对象功能的能力，而且可以根据需要扩展多个功能。避免了使用继承带来的“灵活性差”和“多子类衍生问题”。
- Decorator类在接口上表现为is-a Component的继承关系，即Decorator类继承了Component类所具有的接口。但在实现上又表现为has-a Component的组合关系，即Decorator类又使用了另外一个Component类。
- Decorator模式的目的并非解决“多子类衍生的多继承”问题，Decorator模式应用的要点在于解决“主体类在多个方向上的扩展功能”——是为“装饰”的含义。