西安电子科技大学
XIDIAN UNIVERSITY

软件设计基础

内容

1. 何为软件设计

✓设计的对象、任务和过程以及产生的设计元素

2. 软件设计原则

✓抽象、求精、模块、隐藏、多视点、分离

3. 面向对象软件设计方法学

✓基本思想、特点和优势

4. 软件设计输出及评审

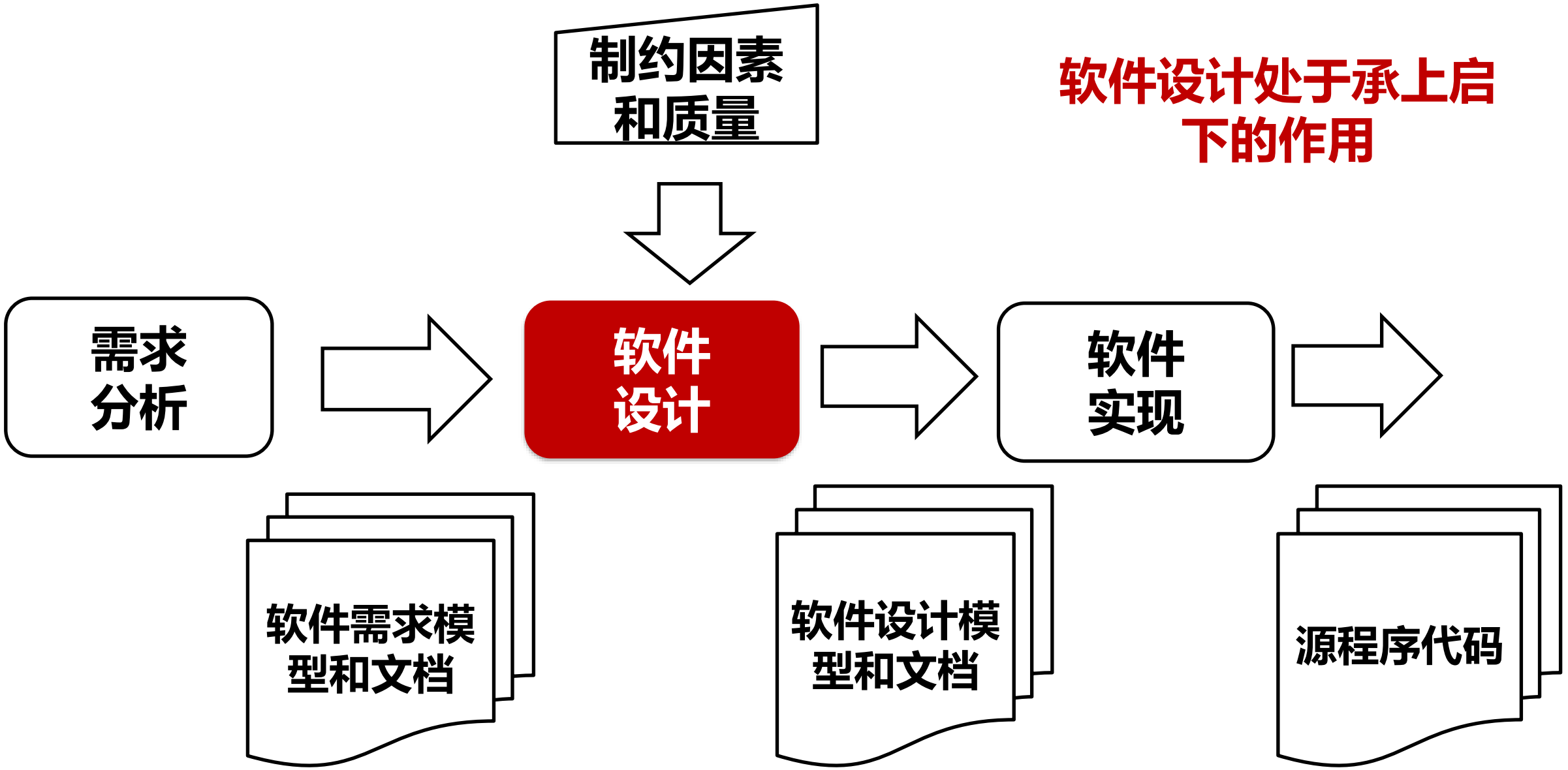
✓软件设计软件制品、软件设计缺陷及评审要求

1.1 何为软件设计?

- **软件设计**
  - ✓连接需求工程和软件实现的桥梁
  - ✓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实现的解决方案

1.2 需求分析、软件设计、软件实现间的关系



软件设计处于承上启下的作用

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

1.3 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

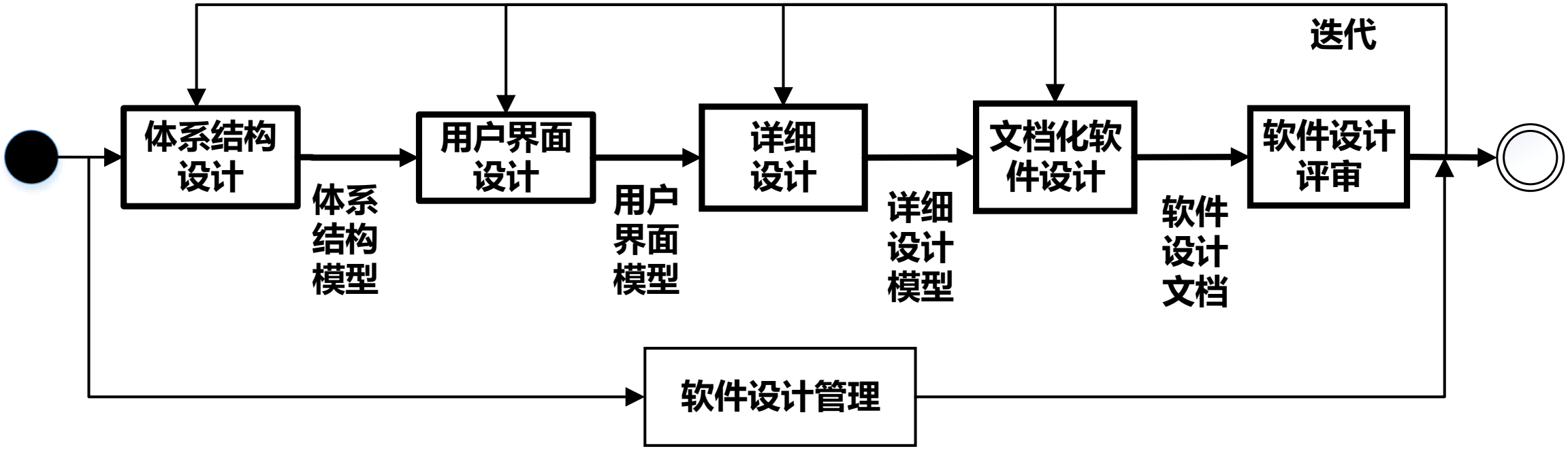
设计结果n

- □ 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
- **详尽性**
  - ✓ 既有高层次的概貌性设计信息，也有低层次的细节性设计信息，各个设计元素得到充分细化
- **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

1.4 软件设计的过程

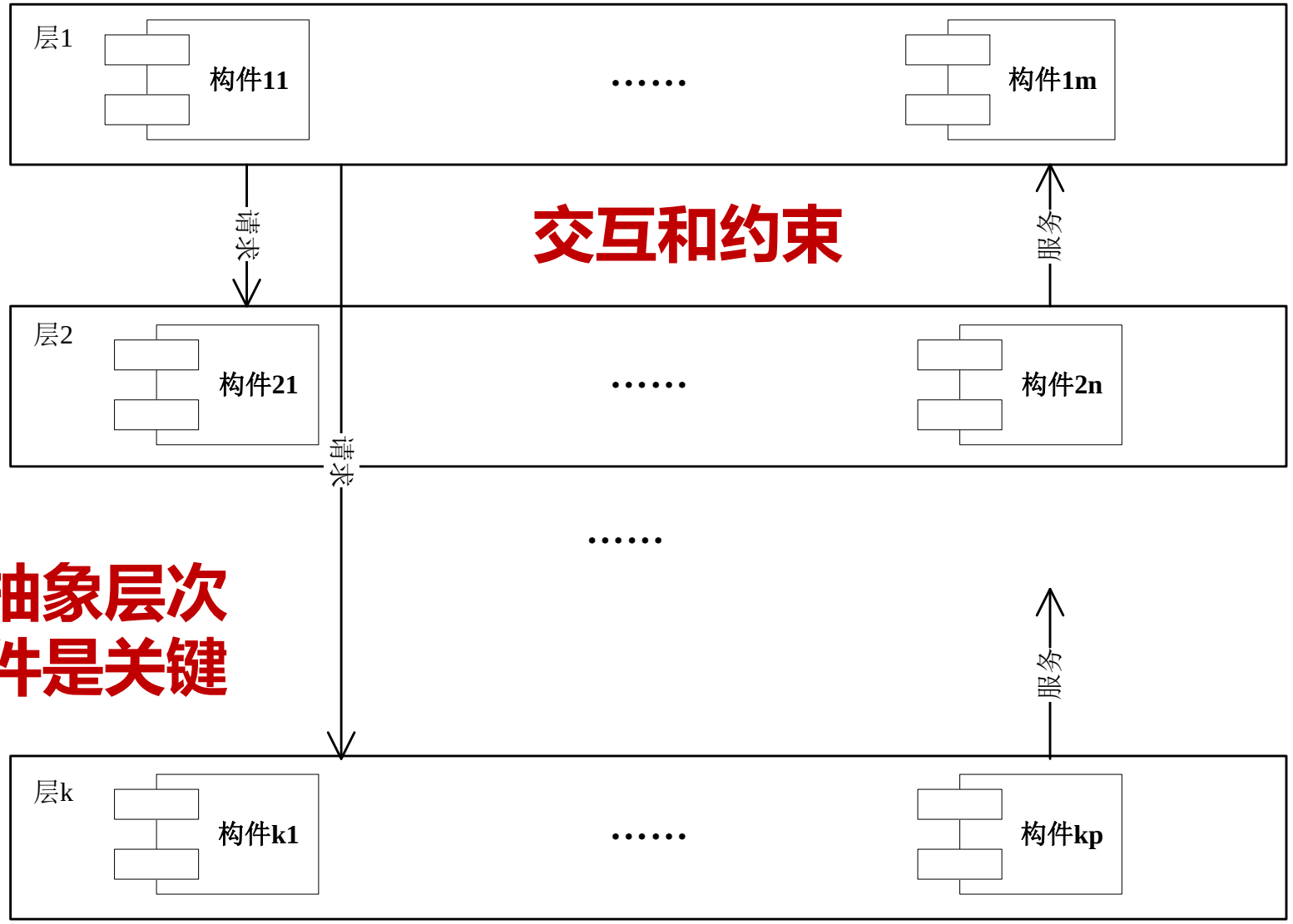

迭代
体系结构设计
用户界面设计
详细设计
文档化软件设计
软件设计评审
体系结构模型
用户界面模型
详细设计模型
软件设计文档
软件设计管理

1.4.1 软件设计的过程 – 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**

示例：分层体系结构风格



界面

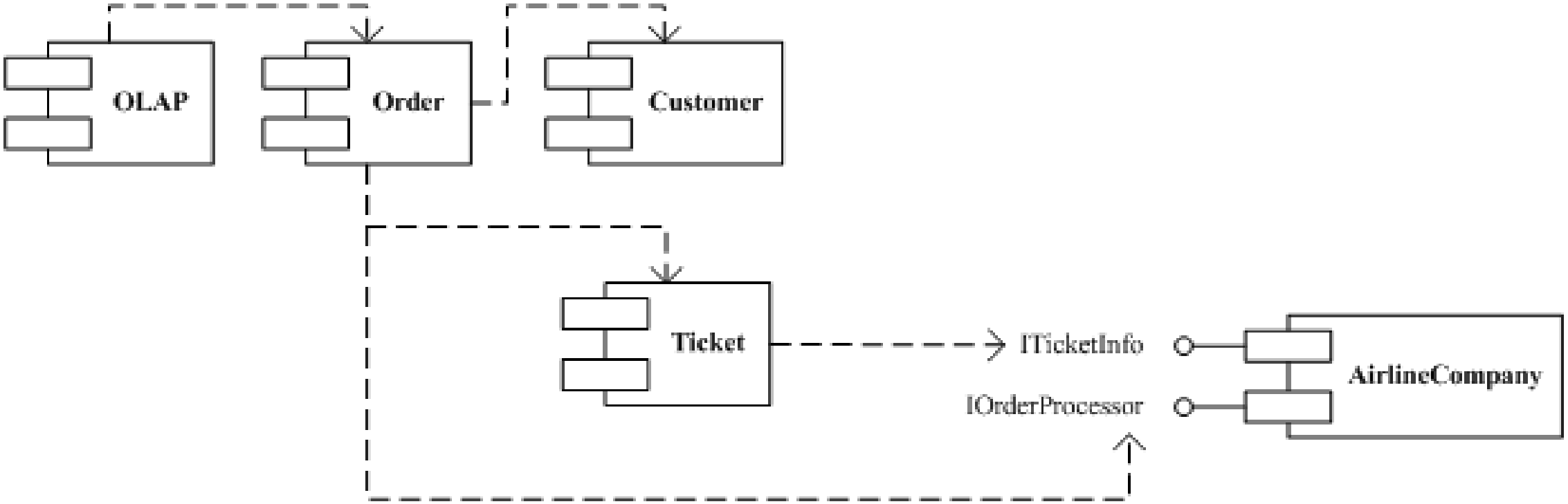
业务

服务

合理地设计抽象层次和组织软构件是关键

示例：软件体系结构设计

❑软构件及其之间的关系



1.4.2 软件设计的过程 – 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

1.4.3 软件设计的过程 – 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**
  - ✓**数据设计**：信息描述 → 计算机可以处理的数据描述
  - ✓**接口设计**：构件提供的交互接口
  - ✓**类设计**：细化各个类设计，包括属性、操作、状态等
  - ✓**算法设计**：实现特定功能的具体执行流程和算法

- **设计关注的质量要素**
  - ✓有效、高效、可靠、易于维护等等

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

软件设计的过程 – 其它的工作

❑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

1.5 软件设计元素

❑设计类

- ✓类既是最基本的设计单元，也是最基本的模块单元

❑软构件

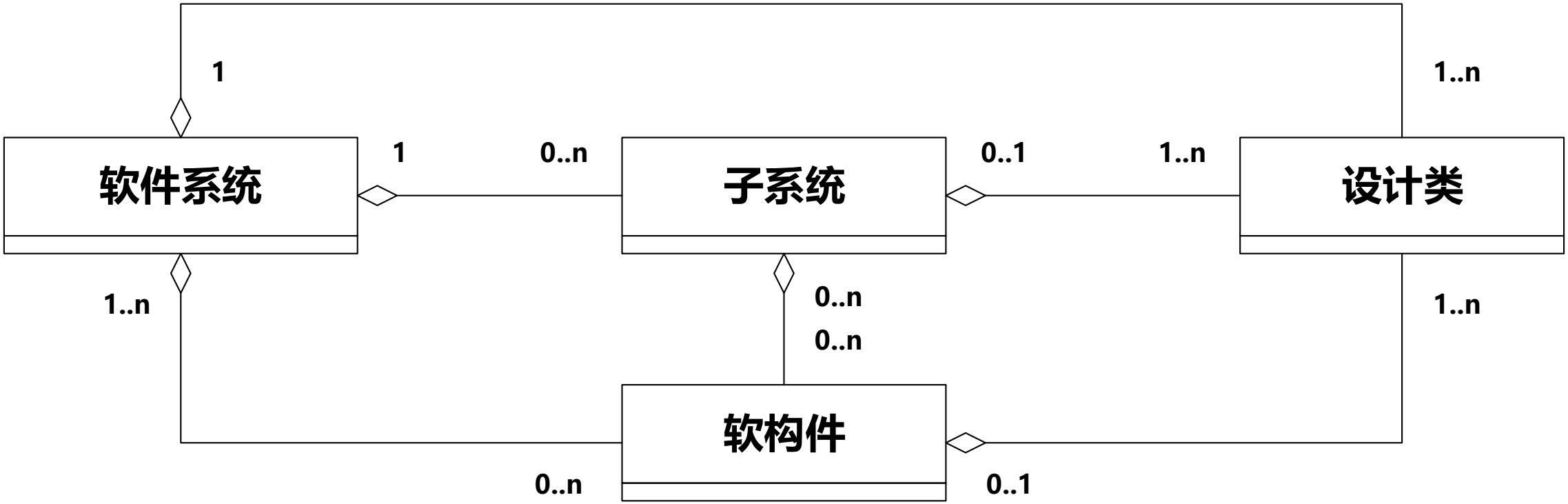
- ✓可分离、可独立部署和执行、可单独重用的一类设计元素
- ✓如动态链接库（.DLL）、可运行的Java JAR包、微服务镜像等就属于软构件

❑子系统

- ✓完成特定功能、逻辑上相互关联的一组模块集合
- ✓有助于管理软件系统的复杂度，简化软件设计和实现

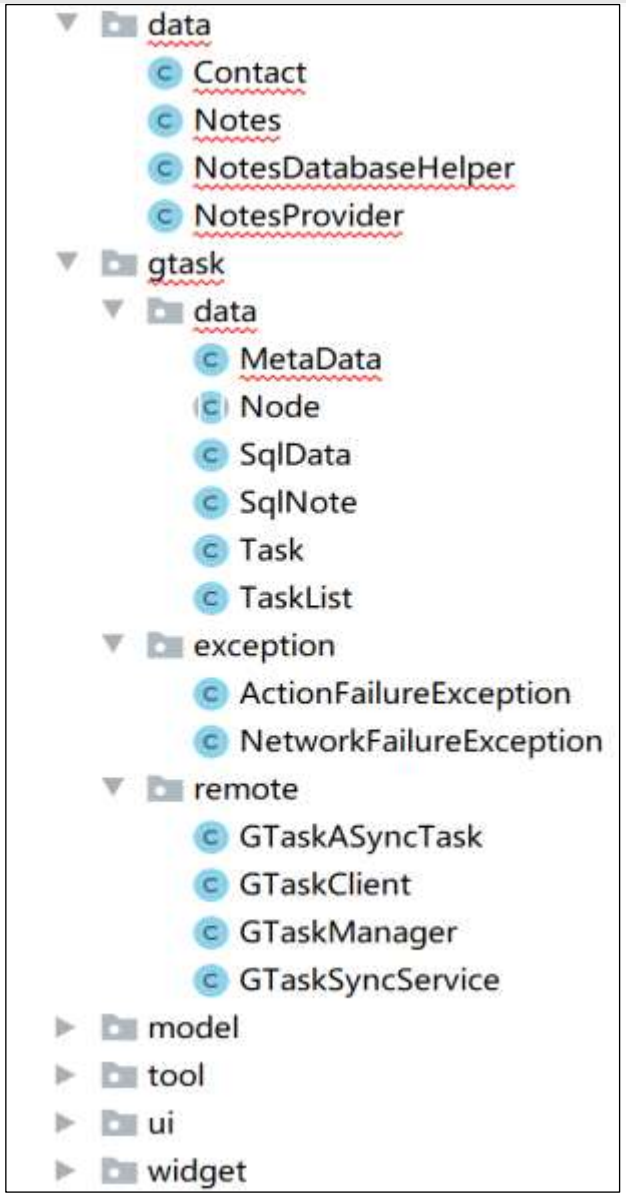
所有的设计元素在编码实现时都有相应的对应物

软件设计元素之间的关系



示例：软件设计元素

体现了哪些设计元素?




```
package net.micode.notes.data;


import ...


public class NotesProvider extends ContentProvider {
    private static final UriMatcher mMatcher;

    private NotesDatabaseHelper mHelper;

    private static final String TAG = "NotesProvider";

    private static final int URI_NOTE            = 1;
    private static final int URI_NOTE_ITEM       = 2;
    private static final int URI_DATA            = 3;
    private static final int URI_DATA_ITEM       = 4;

    private static final int URI_SEARCH          = 5;
    private static final int URI_SEARCH_SUGGEST  = 6;

    static {
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Notes.AUTHORITY, "note", URI_NOTE);
        mMatcher.addURI(Notes.AUTHORITY, "note/#", URI_NOTE_ITEM);
```

内容

1. 软件设计概述
   - ✓设计元素、任务和过程
2. **软件设计原则**
   - ✓抽象、求精、模块、隐藏、多视点、分离
3. **面向对象软件设计方法学**
   - ✓基本思想、特点和优势
4. **软件设计输出及评审**
   - ✓软件设计软件制品、软件设计缺陷及评审要求

2.1 软件设计要考虑的因素

- **满足需求**
  - ✓正确、一致、可行、完整、无冗余
- **权衡抉择**
  - ✓多种设计方案，明确优缺点，综合考虑多方因素
  - ✓关注质量要求
- **应对变化**
  - ✓易于理解、扩展、高效

如何才能得到高质量的软件设计呢？==》设计原则

2.2 软件设计基本原则

① 抽象与逐步求精
② 模块化，高内聚度、低耦合度
③ 信息隐藏
④ 多视点和关注点分离
⑤ 软件重用
⑥ 迭代设计
⑦ 可追踪性

2.2.1 抽象原则

❑何为抽象?

- ✓在认识事物、分析和解决问题的过程中，忽略那些与当前研究目标不相关的部分，以便将注意力集中于与当前目标相关的方面
- ✓关注与软件体系结构相关的要求，有哪些子系统、构件和设计类，以及它们之间的关系，不要过早地考虑细节
- ✓抽象是管理和控制复杂性的基本策略

❑抽象在软件设计中的应用

- ✓软件开发就是一个**从高层抽象到低层抽象逐步过渡过程**
- ✓先建立关于问题及其解的高层次抽象，然后以此为基础，通过精化获得更多的细节，建立问题和系统的低层次抽象

软件设计抽象层次的变化

- 结构性
- 全局性
- 关键性

→

- 过程性
- 局部性
- 细节性

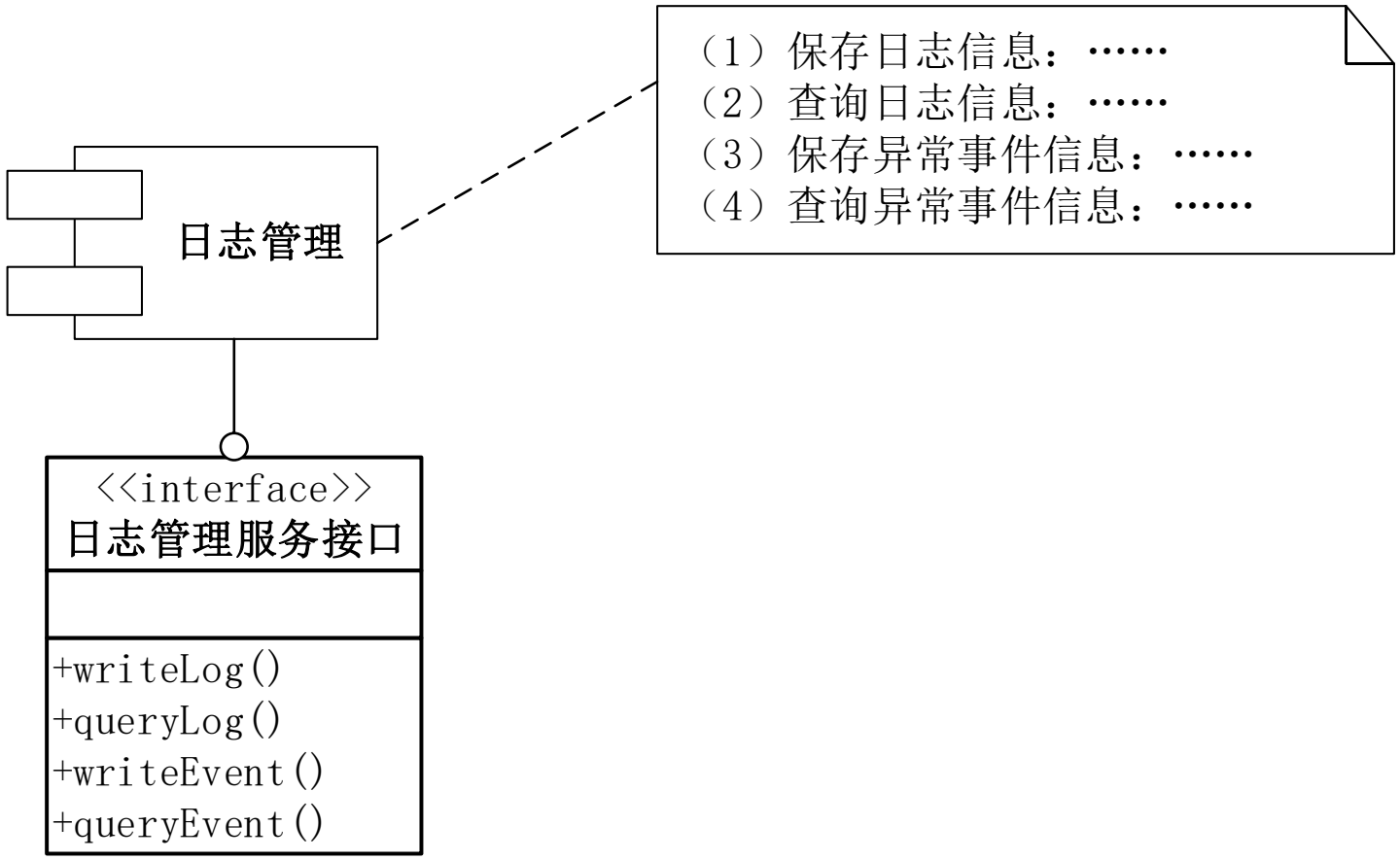
体系结构设计抽象 → 类设计抽象 → 算法设计抽象

逐步求精

软件设计过程中不采用抽象的原则会产生什么样的结果?

示例：体系结构层次的设计抽象

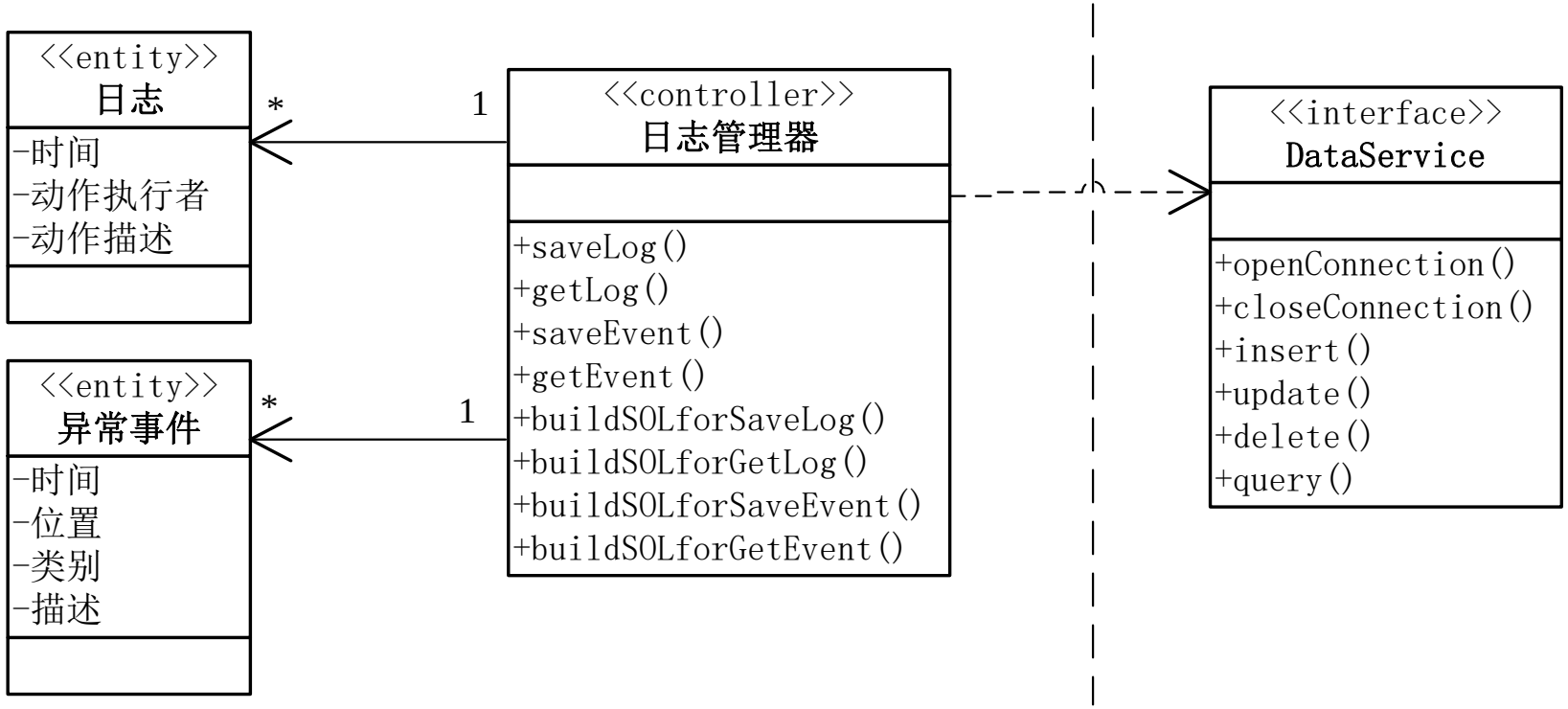
❑关注构件的职责和接口



外部的功能和服务

示例：构件层次的设计抽象

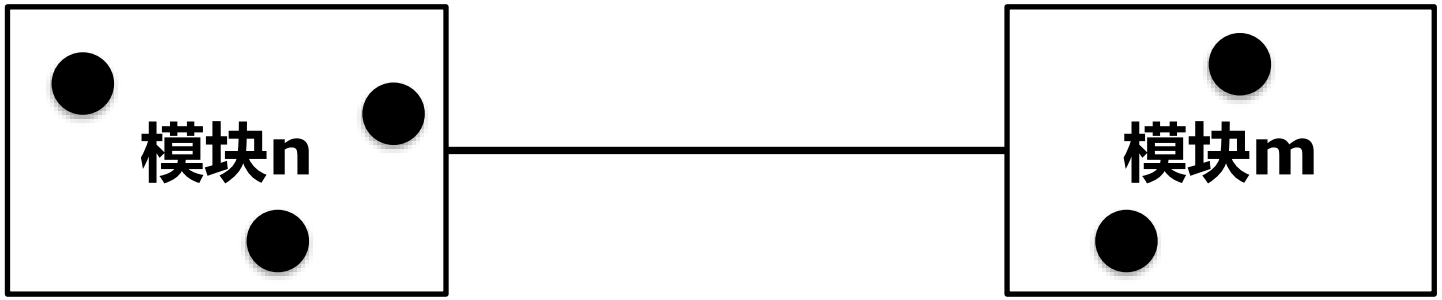
❑考虑构件内部设计元素的职责和协同



内部的结构和构成

2.2.2 模块化、高内聚度和低耦合度原则

- 将软件系统的整体结构分解为一组相关模块
  - ✓模块：包、子系统、构件、类、方法等等
  - ✓**每个模块实现单一的功能，接口明确，相对独立**
  - ✓通过模块之间的交互来组装模块，形成整体框架
  - ✓体现了“分而治之”思想
- 如何达成模块化
  - ✓模块内部强内聚
  - ✓模块之间低耦合



不按照模块化的原则来进行软件设计会怎样?

高内聚度原则

❑何为模块的内聚度?

✓指该模块内各成分间彼此结合的紧密程度，越高越好，高内聚

❑内聚度分类

✓**偶然性内聚**：模块内各成分为完成一组功能而结合在一起，关系松散

✓逻辑性内聚：模块完成的诸任务逻辑上相关

✓时间性内聚：模块内诸任务必须在同一时间段内执行

✓过程性内聚：模块内各成分相关且必须按特定次序执行

✓通讯性内聚：模块内各成分对数据结构的同一区域操作

✓顺序性内聚：模块内各成分与同一功能相关且顺序执行

✓**功能性内聚：**模块内各成分是一整体，完成单个功能

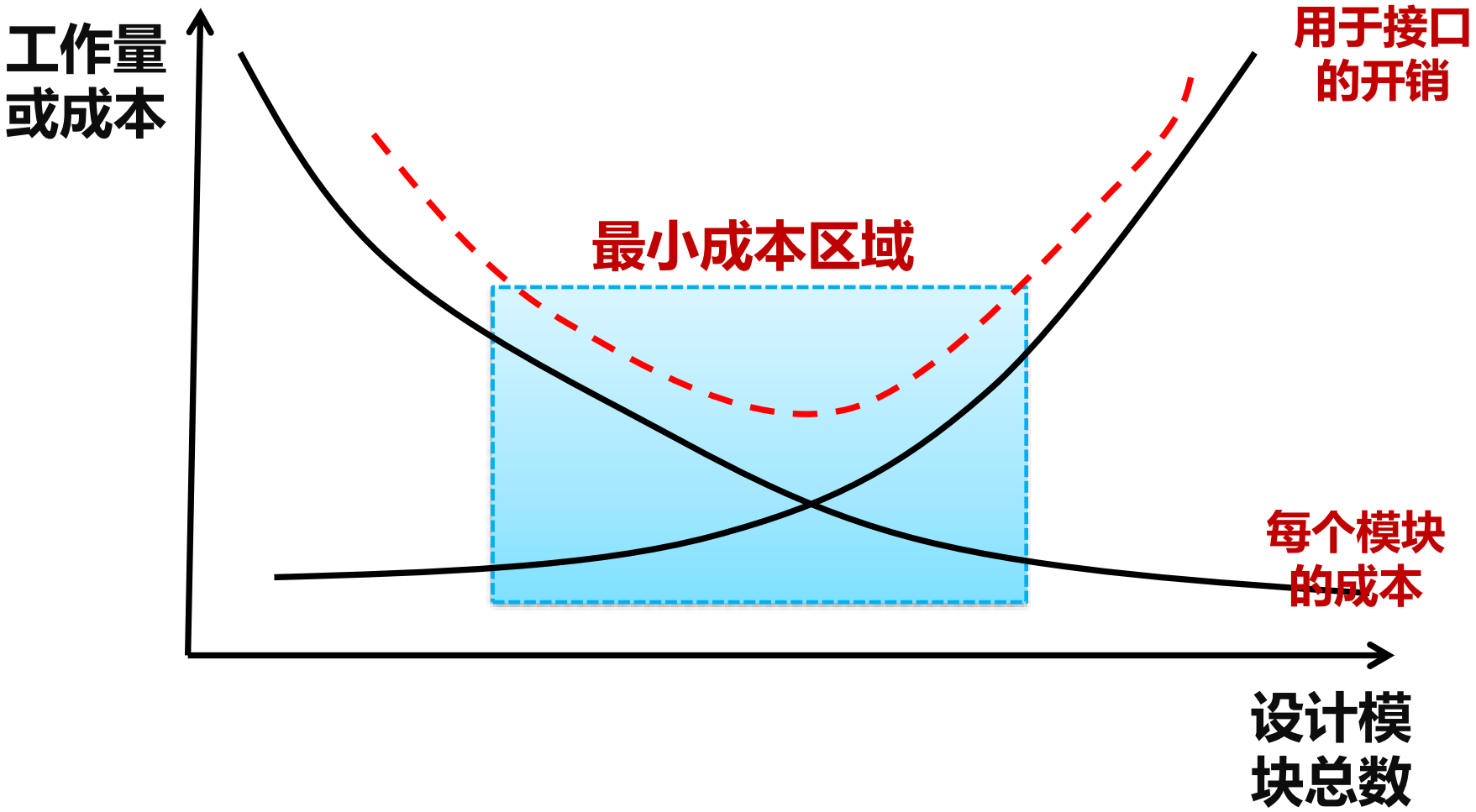
低耦合度原则

- **何为模块间的耦合度?**
  - ✓模块间的相关程度，越低越好，低耦合
- **耦合度分类**
  - ✓**非直接耦合**：二个模块都不依赖对方而独立存在
  - ✓数据耦合：二个模块通过参数交换信息且仅限于数据
  - ✓控制耦合：二个模块通过参数交换信息包含控制信息
  - ✓特征耦合：介于数据耦合和控制耦合之间
  - ✓外部耦合：二个模块与同一外部环境相关联(文件等)
  - ✓公共耦合：模块间通过全局数据环境相互作用
  - ✓**内容耦合**：一个模块使用另一模块内的数据和控制信息，或者直接转移到另一模块内执行

高内聚、低耦合原则指导模块分解合并

- 若各要素紧密程度不高，意味着该模块独立性不好，可能封装实现了多个功能，可以进一步分解
- 如果一组模块之间相关性很强，可以合并，形成一个模块。合并后的模块具有功能单一，内聚度高的特点

模块分解与开发成本之间的关系



2.2.3 信息隐藏原则

❑何为信息隐藏?

- ✓模块应该设计得使其所含的信息对那些不需要这些信息的模块不可访问；模块间仅仅交换那些为完成系统功能所必需交换的信息

❑优点

- ✓模块的独立性更好
- ✓支持模块的并行开发（设计和编码）
- ✓便于测试和维护，减少错误向外传播
- ✓便于增加新的功能

信息隐藏示例

- 模块只提供**对外接口**，不提供内部**实现细节**
- 某些方法或属性设计为“private”

```
package net.micode.notes.data;


import ...


public class NotesProvider extends ContentProvider {
    private static final UriMatcher mMatcher;

    private NotesDatabaseHelper mHelper;

    private static final String TAG = "NotesProvider";

    private static final int URI_NOTE            = 1;
    private static final int URI_NOTE_ITEM       = 2;
    private static final int URI_DATA            = 3;
    private static final int URI_DATA_ITEM       = 4;

    private static final int URI_SEARCH          = 5;
    private static final int URI_SEARCH_SUGGEST  = 6;

    static {
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Notes.AUTHORITY, "note", URI_NOTE);
        mMatcher.addURI(Notes.AUTHORITY, "note/#", URI_NOTE_ITEM);
```

面向对象方法学如何支持信息隐藏的?

2.2.4 关注点分离原则

❑何为关注点

- ✓针对概念、任务和目标的某个部分或者侧面的聚焦
- ✓关注点：结构、行为等

❑何为关注点分离

- ✓设计师将若干性质不同的关注点分离开来，以便在适当的时间处理不同的关注点，随后将这些关注点整合起来，形成局部或者全局性的设计结果
- ✓防止“胡子眉毛一把抓”

关注点不分离会产生什么样的后果?

2.2.5 软件重用原则

- 尽可能地重用已有的软件资产来实现软件系统的功能，同时要确保所开发的软件系统易于为其他软件系统所重用
- 支持软件重用的技术手段
  - ✓封装、接口、继承、多态等
- 软件重用的对象
  - ✓过程和函数、类、软构件、开源软件
  - ✓软件设计模式、软件开发知识

为什么软件常用可以提高软件设计的质量?

2.2.6 软件设计的其它原则

- 设计**可追溯**到分析模型
- 经常关注待建系统的**架构**
- **数据设计**和功能设计同样重要
- 必须设计**接口**
- **用户界面**设计必须符合最终用户要求
- 设计表述要尽可能**易于理解**
- 设计应该**迭代**进行

内容

1. 软件设计概述
   - ✓设计元素、任务和过程
2. 软件设计原则
   - ✓抽象、求精、模块、隐藏、多视点、分离
3. **面向对象软件设计方法学**
   - ✓基本思想、特点和优势
4. **软件设计输出及评审**
   - ✓软件设计软件制品、软件设计缺陷及评审要求

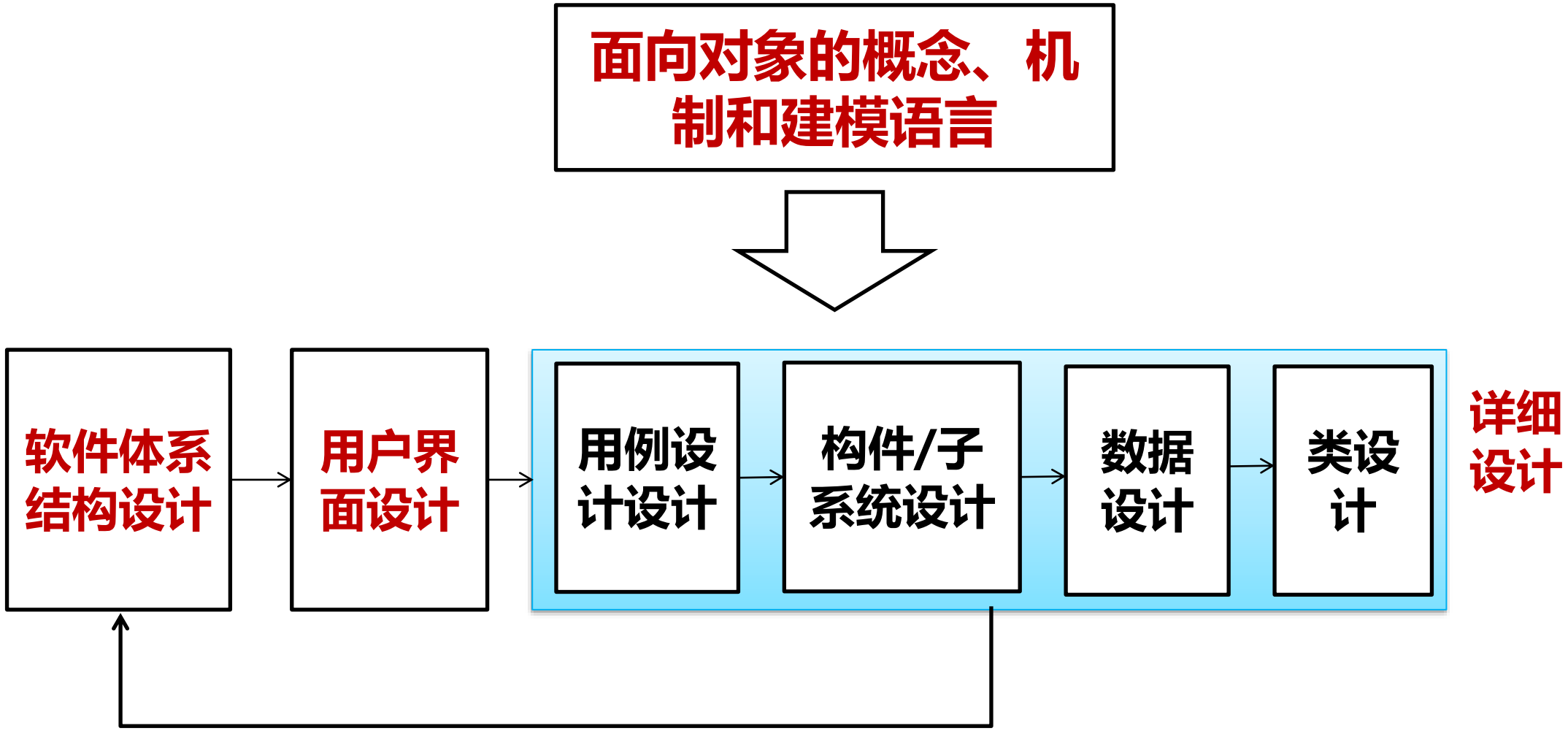
3.1 面向对象软件设计方法学

- 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

提供了概念和机制来落实软件设计原则，支持高质量的软件设计

3.2 面向对象软件设计过程



面向对象的软件设计原则

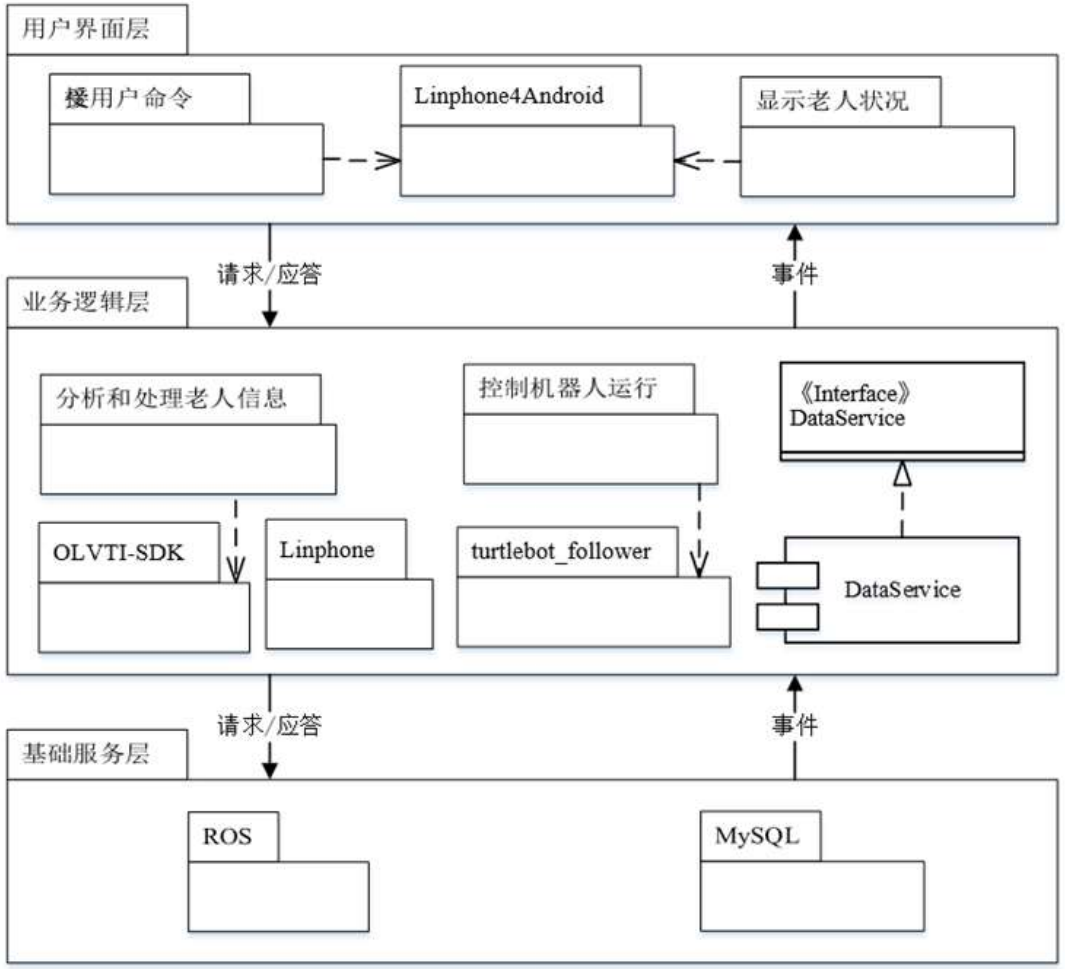
- **单一职责：每个类只承担一项职责，提高类设计的模块化程度以及类的内聚性，降低类的规模和复杂性；**
- **开闭原则：每个类对于扩展是开放的，对于修改是封闭的。当需求更改时，尽可能不修改类的原有设计及源代码，采用扩展的方式。**
  - ✓首先通过接口类或抽象类定义稳定的抽象层，一旦需求或实现变化，可以通过派生具体的类实现功能扩展
- **里氏替换：确保父类的性质在子类中仍成立。**
  - ✓子类可以扩展父类功能，但不能改变父类原有功能。尽量不要重写父类方法

面向对象的软件设计原则

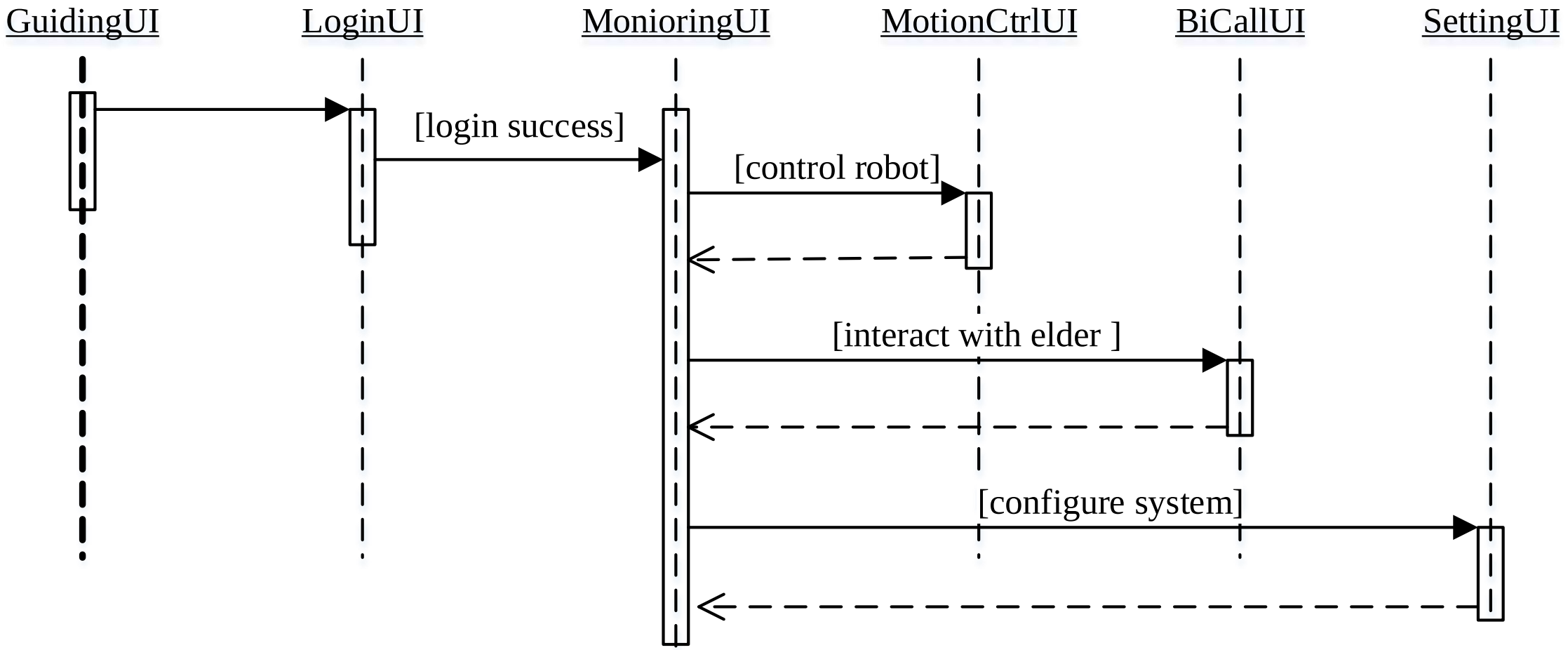
- 接口隔离：类对另一个类的依赖应建立在最小的接口上，要为各个使用类建立需要的专用接口，而不失去建立一个庞大的接口供所有使用类去访问
- 依赖倒置：高层模块不应该依赖底层模块，两者都应该依赖其抽象；抽象不应该依赖细节，细节应依赖抽象。要通过抽象、借助接口和抽象类，使得各个模块实现独立性，减低类耦合
- 最少知识：类只对于自己存在耦合关系的类交互，尽可能少地与其他不相关的类交互。比如“明星”至于自己最亲近的“经纪人”交互，不与大量的“粉丝”和“媒体公司”交互

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



示例：用交互图来表示用户界面设计



3.3 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

3.4 软件设计的CASE工具

- **软件设计文档撰写工具**
  - ✓如借助于Microsoft Office、WPS等
- **软件设计建模工具**
  - ✓如Microsoft Visio、StarUML、Argo UML等工具
- **软件设计分析和转换工具**
  - ✓如IBM Rational Rose等软件工具
- **配置管理工具和平台**
  - ✓如Git、Github、Gitlab、PVCS、Microsoft SourceSafe等，支持软件需求制品（如模型、文档等）的配置、版本管理、变化跟踪等

内容

1. 软件设计概述
   - ✓设计元素、任务和过程
2. 软件设计原则
   - ✓抽象、求精、模块、隐藏、多视点、分离
3. 面向对象软件设计方法学
   - ✓基本思想、特点和优势
4. **软件设计输出及评审**
   - ✓软件设计软件制品、软件设计缺陷及评审要求

4.1 软件设计的输出

❑软件设计模型

- ✓它从多个不同的视角、不同的抽象层次描述了软件的设计信息，并采用诸如UML、模块图、层次图等图形化的方式来加以刻画

❑软件设计文档

- ✓它采用自然语言的形式，结合软件设计模型，详细描述软件系统的各项设计，包括体系结构设计、子系统和构件设计、用户界面设计、用例设计、数据设计等等

4.2 软件设计文档的规范

- 文档概述
- 系统概述
- 设计目标和原则
- 设计约束和现实限制
- 体系结构设计
- 用户界面设计
- 子系统/构件设计
- 用例设计
- 类设计
- 数据设计
- 接口设计

4.3 软件设计中的缺陷

- **设计未能满足需求**
  - ✓对软件需求的理解存在偏差，未能正确地理解用户的软件需求，导致所设计的软件无法满足用户的需要
- **设计质量低下**
  - ✓设计过程中未能遵循设计原则、缺乏设计经验，导致软件设计质量低下，如设计的软件不易于维护和扩展
- **设计存在不一致**
  - ✓不同软件设计制品对同一个设计有不同的描述，或者存在不一致甚至相冲突的设计内容；多个不同软件设计要素之间存在不一致
- **设计不够详尽**
  - ✓未能提供设计细节性信息，导致程序员无法根据设计来开展编码工作

4.4 软件设计的评审

❑谁参与评审

✓程序员、软件测试工程师、用户、质量保证人员、设计工程师等

❑评审什么内容

✓**文档规范性**，软件设计文档是否符合软件设计规格说明书

✓**设计制品的可理解性**，是否简洁、易于理解

✓**设计内容的合法性**，设计结果是否符合相关的标准、法律和法规

✓**设计的质量水平**，软件设计是否遵循设计原则，质量如何

✓**设计是否满足需求**，设计是否完整和正确地实现了软件需求

✓**设计优化性**，软件设计是否还有待优化的内容

4.5 软件设计的管理

❑软件设计的变更管理

- ✓明确哪些方面发生了变更、这些变化反应在软件设计模型和文档的哪些部分、导致软件设计模型和文档的版本发生了什么样变化

❑软件设计的追溯管理

- ✓搞清楚是什么原因导致了软件设计的变更，评估设计变更的影响域，评估设计变更对软件项目开发带来的影响

❑软件设计的基线管理

- ✓一旦软件设计模型和文档通过了评审，纳入到基线库中

小结

- **软件设计是要给出软件需求的实现解决方案**
  - ✓设计既要满足需求，也要关注质量；设计用于指导实现和编码
- **软件设计有其过程，要循序渐进地开展设计**
  - ✓从体系结构设计、用户界面设计、详细设计
- **软件设计要遵循一系列的基本原则**
  - ✓模块化、信息隐藏、逐步求精、多视点等
- **面向对象软件设计的特点**
  - ✓基于面向对象的概念和抽象，系统性的设计支持，具有多种优点
- **对软件设计结果进行文档化和评审**
  - ✓撰写软件设计文档，发现和纠正软件设计中存在的缺陷