南京大学

设计模式概述

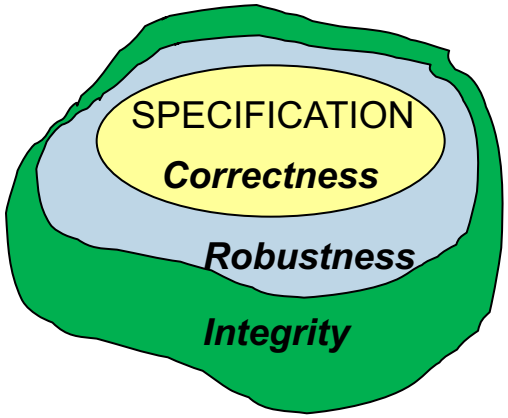
Design Patterns

和前面课程的关系-软件外部质量因素

- Correctness 正确性
  - 依据规约 完成任务
- Robustness 鲁棒性
  - 异常情况 合理反应
- Integrity 完整性
  - 非法访问或修改 合理反应

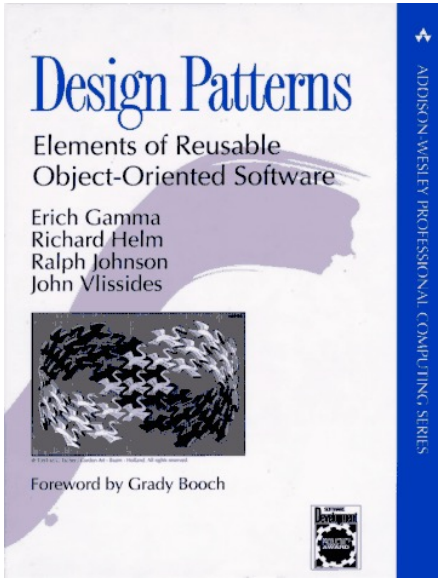
Reliability 可靠性

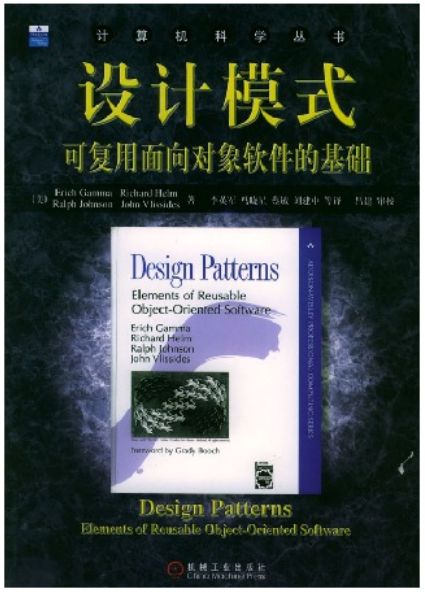
- Extendibility 易扩展性
  - 软件产品 应 规约改变 而 改变
- Reusability 易复用性
  - 软件模块 用于构建多种不同应用



大纲

- 引言
- 设计模式的诞生与发展
- 设计模式的定义与分类
- GoF设计模式简介
- 设计模式的优点





Free version：
https://gameprogrammingpatterns.com/

引言

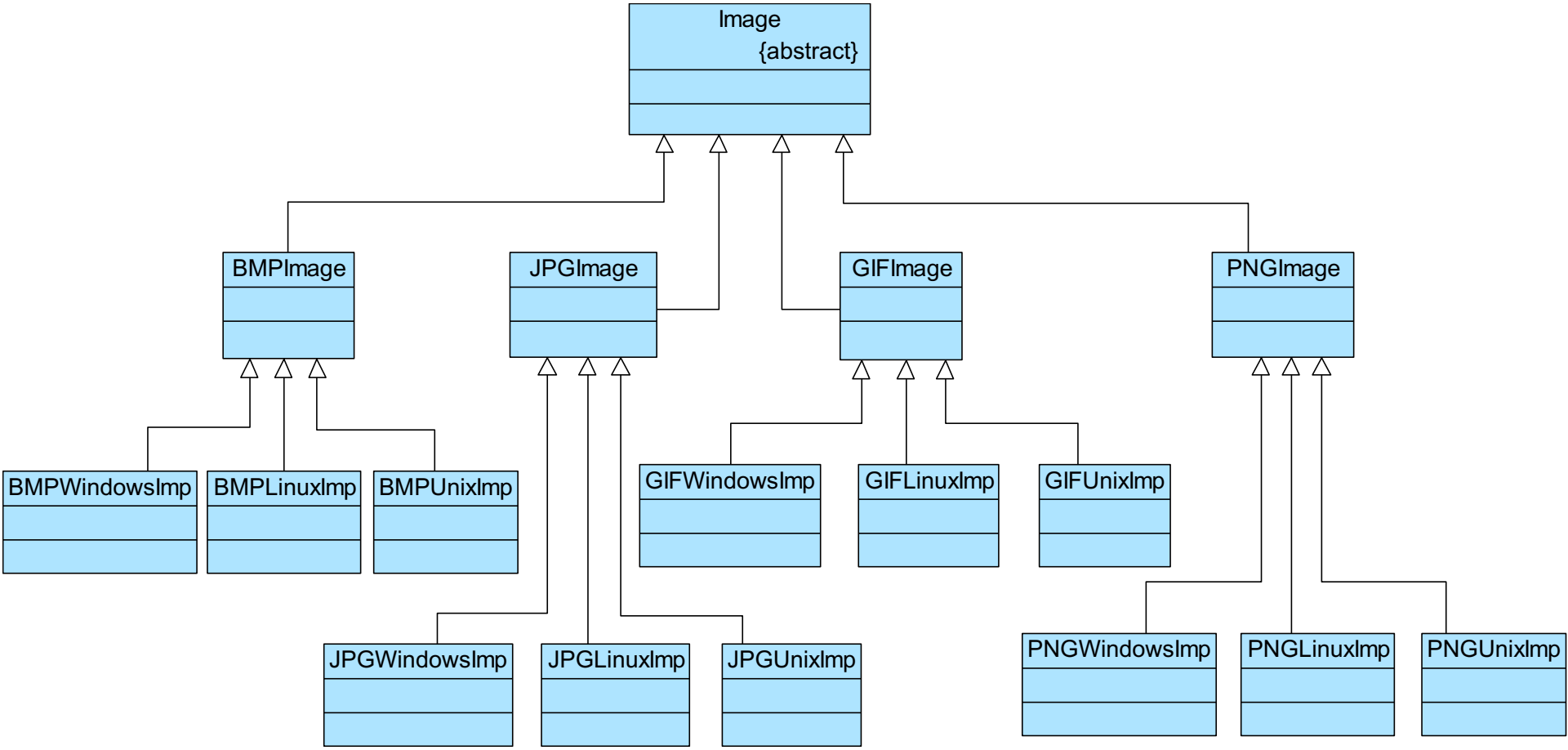
- 从三个实例说起……

引言

- 庞大的跨平台图像浏览系统
  - 实例说明
    - 某软件公司要开发一个跨平台图像浏览系统，要求该系统能够显示BMP、JPG、GIF、PNG等多种格式的文件，并且能够在Windows、Linux、Unix等多个操作系统上运行。系统首先将各种格式的文件解析为像素矩阵(Matrix)，然后将像素矩阵显示在屏幕上，在不同的操作系统中可以调用不同的绘制函数来绘制像素矩阵。

引言

- 庞大的跨平台图像浏览系统
  - 初始设计方案



引言

- 庞大的跨平台图像浏览系统
  - 问题
    - (1) 采用了多层继承结构，导致系统中类的个数急剧增加，具体层的类的个数 = 所支持的图像文件格式数×所支持的操作系统数
    - (2) 系统扩展麻烦，无论是增加新的图像文件格式还是增加新的操作系统，都需要增加大量的具体类，这将导致系统变得非常庞大，增加运行和维护开销

引言

- 不够灵活的影院售票系统
  - 实例说明
    - 某软件公司为某电影院开发了一套影院售票系统，在该系统中需要为不同类型的用户提供不同的电影票打折方式，具体打折方案如下：
      - (1) 学生凭学生证可享受票价8折优惠；
      - (2) 年龄在10周岁及以下的儿童可享受每张票减免10元的优惠（原始票价需大于等于20元）；
      - (3) 影院VIP用户除享受票价半价优惠外还可进行积分，积分累计到一定额度可换取电影院赠送的奖品。
    - 该系统在将来可能还要根据需要引入新的打折方式。

```
public class MovieTicket {
    private double price;

    //compute the price
    public double calculate(String type) {
        //student ticket
        if(type.equalsIgnoreCase("student")) {
            return this.price * 0.8;
        }
        //children ticket
        else if(type.equalsIgnoreCase("children") && this.price >= 20 ) {
            return this.price - 10;
        }
         //VIP ticket
        else if(type.equalsIgnoreCase("vip")) {
            //add points, code is omitted
            return this.price * 0.5;
         }
        else {
            return this.price;
        }
    }
}
```

引言

- 不够灵活的影院售票系统
  - 问题
    - (1) MovieTicket类的calculate()方法非常庞大，它包含各种打折算法的实现代码，在代码中出现了较长的条件转移语句，不利于测试和维护
    - (2) 在增加新的打折算法或者对原有打折算法进行修改时必须修改MovieTicket类的源代码，系统的灵活性和可扩展性较差
    - (3) 算法的复用性差，如果另一个系统需要重用某些打折算法，只能通过对源代码进行复制粘贴来重用，无法单独重用其中的某个或某些算法

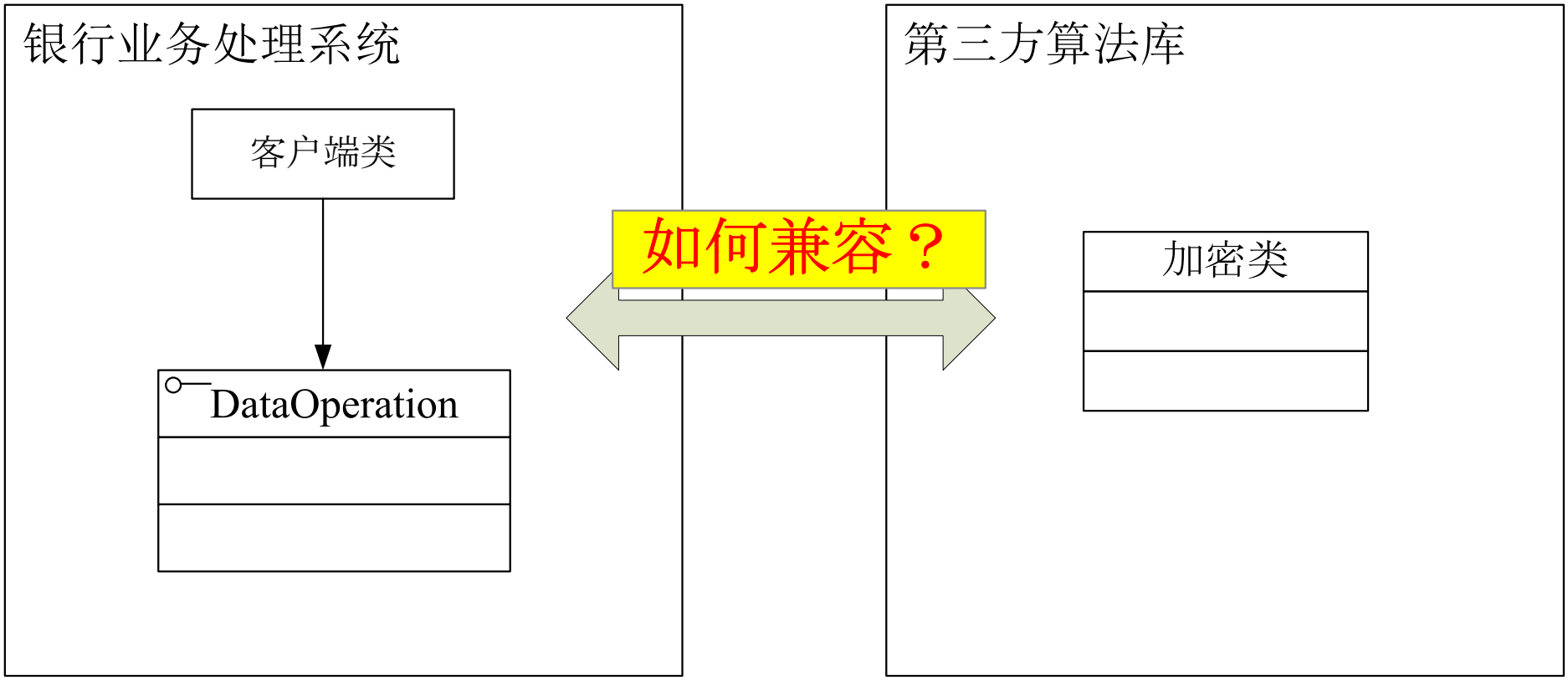
引言

- 重用第三方算法库时面临的问题
  - 实例说明
    - 某软件公司在开发一个银行业务处理系统时需要对其中的机密数据进行加密处理，通过分析发现，用于加密的程序已经存在于一个第三方算法库中，但是没有该算法库的源代码。在系统初始设计阶段，已定义数据操作接口DataOperation，且该接口已被很多同事使用，对该接口的修改势必导致大量代码需要产生改动。

引言

- 重用第三方算法库时面临的问题
  - 问题
    - 如何在既不修改现有接口又不需要算法库源代码的基础上能够实现第三方算法库的重用是该软件公司开发人员必须面对的问题。

引言

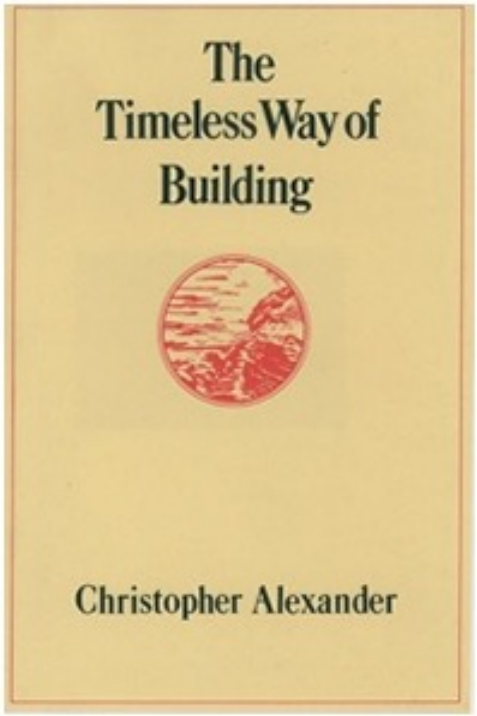


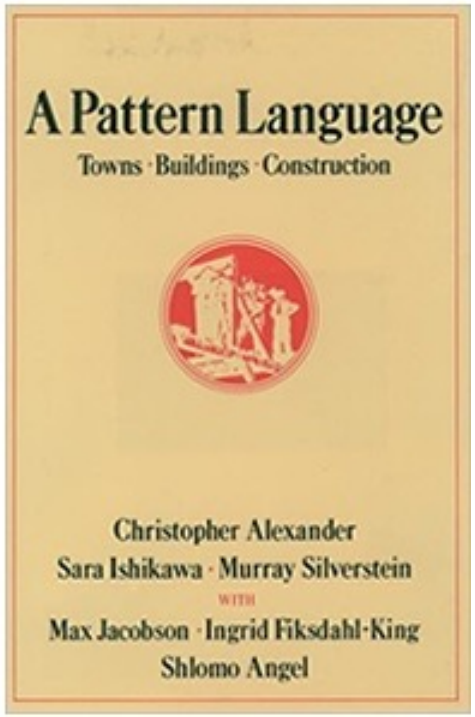
设计模式的诞生与发展

- 模式的诞生与定义
  - 模式(Pattern)起源于建筑业而非软件业
  - 模式之父——美国加利佛尼亚大学环境结构中心研究所所长Christopher Alexander博士
  - 《A Pattern Language: Towns, Buildings, Construction》——253个建筑和城市规划模式
  - 模式
    - Context（模式可适用的前提条件）
    - Theme或Problem（在特定条件下要解决的目标问题）
    - Solution（对目标问题求解过程中各种物理关系的记述）

设计模式的诞生与发展

Christopher Alexander





设计模式的诞生与发展

- 模式的诞生与定义
  - ✓ Alexander给出了关于模式的经典定义：
    - 每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心，通过这种方式，人们可以无数次地重用那些已有的解决方案，无须再重复相同的工作

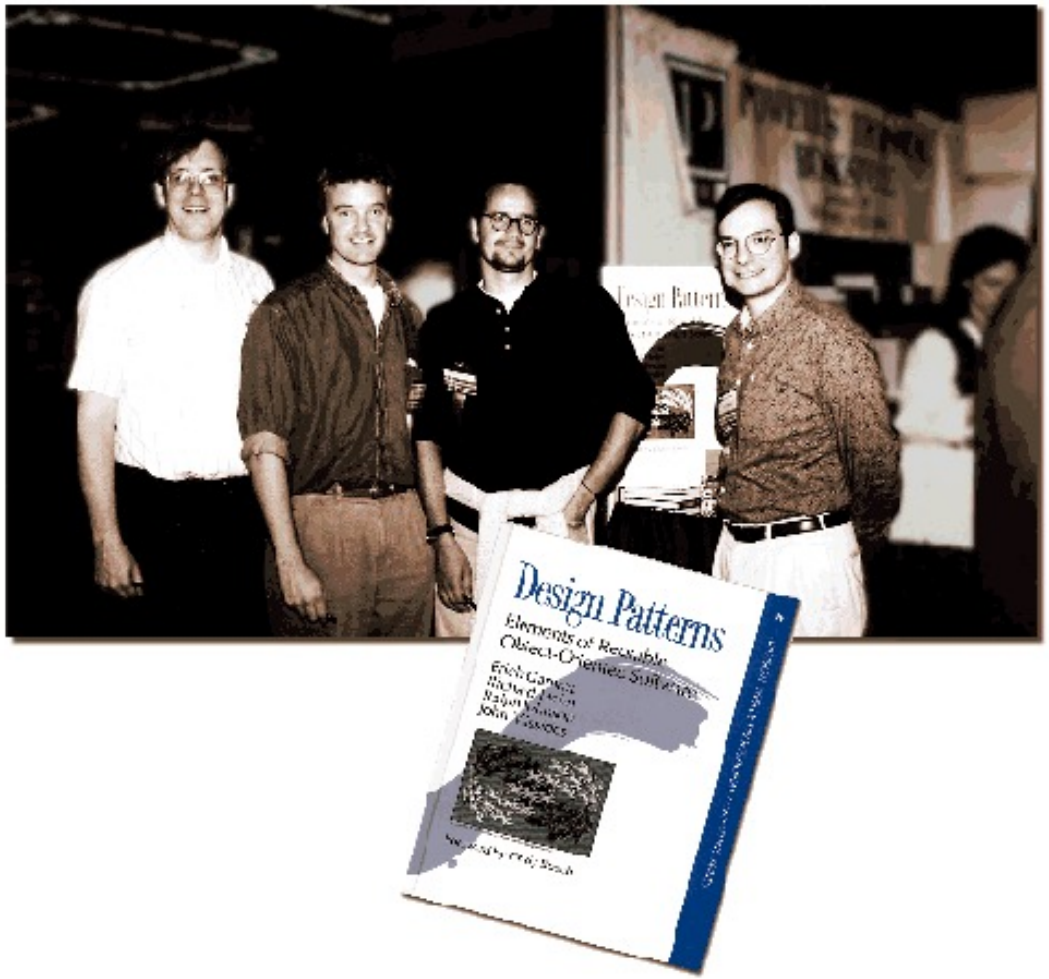
模式是在特定环境下人们解决某类重复出现问题的一套成功或有效的解决方案。

A pattern is a successful or efficient solution to a recurring problem within a context.

设计模式的诞生与发展

- 软件模式概述
  - ✓ 20世纪80年代末，软件工程界开始关注Christopher Alexander等在这一住宅、公共建筑与城市规划领域的重大突破
  - ✓ “四人组(Gang of Four，GoF，分别是Erich Gamma, Richard Helm, Ralph Johnson和John Vlissides)”于1994年归纳发表了23种在软件开发中使用频率较高的设计模式，旨在用模式来统一沟通面向对象方法在分析、设计和实现间的鸿沟

设计模式的诞生与发展



Gang of Four (GoF)

设计模式的诞生与发展

Gang of Four

Erich Gamma

苏黎世大学计算机科学博士，是Eclipse、 JUnit 等项目的负责人

Richard Helm

墨尔本大学计算机科学博士，原IBM研究员，现供职于波士顿顾问集团

Ralph Johnson

康奈尔大学计算机科学博士，伊利诺伊大学教授

John Vlissides

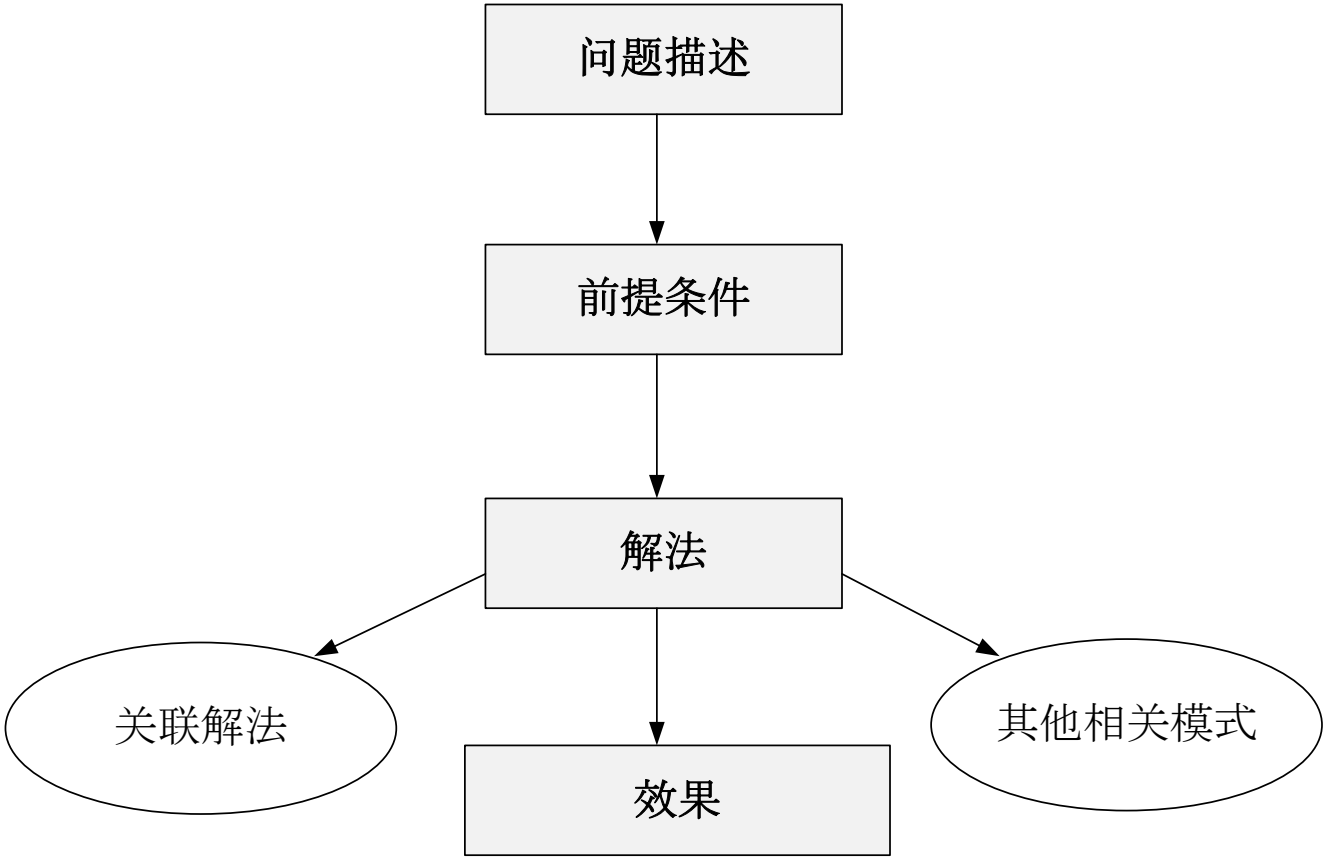
斯坦福大学计算机科学博士，原IBM研究员，于2005年11月24日因脑瘤去世，享年44岁

设计模式的诞生与发展

- 软件模式概述
  - 软件模式：在一定条件下的软件开发问题及其解法
    - 问题描述
    - 前提条件（环境或约束条件）
    - 解法
    - 效果

设计模式的诞生与发展

- 软件模式概述



设计模式的诞生与发展

- 软件模式概述
  - 大三律(Rule of Three)
    - 只有经过3个以上不同类型（或不同领域）的系统的校验，一个解决方案才能从候选模式升格为模式

设计模式的定义与分类

- 设计模式的定义
  - 设计模式(Design Pattern)
    - 一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结
    - 是一种用于对软件系统中不断重现的设计问题的解决方案进行文档化的技术
    - 是一种共享专家设计经验的技术
    - 目的：为了可重用代码、让代码更容易被他人理解、提高代码可靠性

设计模式的定义与分类

- 设计模式的定义

设计模式是在特定环境下为解决某一通用软件设计问题提供的一套定制的解决方案，该方案描述了对象和类之间的相互作用。

Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.

设计模式的定义与分类

- 设计模式的基本要素
  - 设计模式一般包含模式名称、问题、目的、解决方案、效果、实例代码和相关设计模式等基本要素，4个关键要素如下：
    - 模式名称 (Pattern Name)
    - 问题 (Problem)
    - 解决方案 (Solution)
    - 效果 (Consequences)

设计模式的定义与分类

- 设计模式的分类
  - 根据目的（模式是用来做什么的）可分为创建型(Creational)，结构型(Structural)和行为型(Behavioral)三类：
    - **创建型模式**主要用于创建对象
    - **结构型模式**主要用于处理类或对象的组合
    - **行为型模式**主要用于描述类或对象如何交互和怎样分配职责

设计模式的定义与分类

- 设计模式的分类
  - 根据范围，即模式主要是处理类之间的关系还是处理对象之间的关系，可分为类模式和对象模式两种：
    - **类模式**处理类和子类之间的关系，这些关系通过继承建立，在编译时刻就被确定下来，是一种静态关系
    - **对象模式**处理对象间的关系，这些关系在运行时变化，更具动态性

GoF设计模式简介

范围\目的	创建型模式	结构型模式	行为型模式
类模式	工厂方法模式	（类）适配器模式	解释器模式 模板方法模式
对象模式	抽象工厂模式 建造者模式 原型模式 单例模式	（对象）适配器模式 桥接模式 组合模式 装饰模式 外观模式 享元模式 代理模式	职责链模式 命令模式 迭代器模式 中介者模式 备忘录模式 观察者模式 状态模式 策略模式 访问者模式

GoF设计模式简介

- 创建型模式
  - ✓ 抽象工厂模式(Abstract Factory) ★★★★★
  - ✓ 建造者模式(Builder) ★★☆☆☆
  - ✓ 工厂方法模式(Factory Method) ★★★★★
  - ✓ 原型模式(Prototype) ★★★☆☆
  - ✓ 单例模式(Singleton) ★★★★☆

GoF设计模式简介

- 结构型模式
  - ✓ 适配器模式(Adapter) ★★★★☆
  - ✓ 桥接模式(Bridge) ★★★☆☆
  - ✓ 组合模式(Composite) ★★★★☆
  - ✓ 装饰模式(Decorator) ★★★☆☆
  - ✓ 外观模式(Facade) ★★★★★
  - ✓ 享元模式(Flyweight) ★☆☆☆☆
  - ✓ 代理模式(Proxy) ★★★★☆

GoF设计模式简介

- 行为型模式
  - ✓ 职责链模式(Chain of Responsibility) ★★☆☆☆
  - ✓ 命令模式(Command) ★★★★☆
  - ✓ 解释器模式(Interpreter) ★☆☆☆☆
  - ✓ 迭代器模式(Iterator) ★★★★★
  - ✓ 中介者模式(Mediator) ★★☆☆☆
  - ✓ 备忘录模式(Memento) ★★☆☆☆
  - ✓ 观察者模式(Observer) ★★★★★
  - ✓ 状态模式(State) ★★★☆☆
  - ✓ 策略模式(Strategy) ★★★★☆
  - ✓ 模板方法模式(Template Method) ★★★☆☆
  - ✓ 访问者模式(Visitor) ★☆☆☆☆

设计模式的优点

- 融合了众多专家的经验，并以一种标准的形式供广大开发人员所用
- 提供了一套通用的设计词汇和一种通用的语言，以方便开发人员之间进行沟通和交流，使得设计方案更加通俗易懂
- 让人们可以更加简单方便地复用成功的设计和体系结构
- 使得设计方案更加灵活，且易于修改
- 将提高软件系统的开发效率和软件质量，且在一定程度上节约设计成本
- 有助于初学者更深入地理解面向对象思想，方便阅读和学习现有类库与其他系统中的源代码，还可以提高软件的设计水平和代码质量