4.1 软件工程开发方法与软件设计

- 软件工程开发方法
  - 传统开发方法
    - 功能分解法

      以系统需要提供的功能为中心来组织系统。作为早期的建模方法，没有明确的区分分析与设计。
      - 流程
        - 首先定义各种功能，然后把功能分解为子功能
        - 对较大的子功能进一步分解，知道可给出明确的定义
        - 设计功能、子功能所需要的数据结构
        - 定义功能、子功能之间的接口
      - 优点
        - 直接地反映用户的需求，工作容易开始
      - 缺点
        - 不能直接地映射问题域，很难检验结果的正确性
        - 对需求变化的适应能力很差
        - 局部的错误和修改很容易产生全局性的影响
    - 结构化方法
      - 结构化分析

        结构化分析又称数据流法，其基本策略是跟踪数据流，即研究问题域中数据如何流动，以及在各个环节上进行何种处理，从而发现数据流和加工。得到的分析模型是数据流图，主要模型元素是数据流、加工、文件及端点，外加处理说明和数据字典。
      - 结构化设计
        - 基于模块的概念建立设计模型，分为**概要设计**和**详细设计**
        - 从功能的角度设计系统
        - 自顶向下，逐步分解和细化
        - 将大系统分解为若干模块，主程序调用这些模块实现完整的系统功能
      - 结构化编程
        - 具有一个开始和一个结束的程序或程序模块，并且在程序执行中的每一步都由三个部分之一组成：顺序、选择或循环结构
      - 结构化方法的模型
        - 数据流图

          描述系统由哪部分组成，各部分之间联系
        - 数据字典

          定义了数据流图中每一个数据元素
        - 结构化语言

        - 判定表或判定树

          详细描述数据流图中不能再分解的每一个加工的内部处理逻辑
        - 实体联系图
        - 状态转换图
      - 常见问题
        - 需求错误
        - 需求变化
        - 系统结构的崩溃’
        - **根本原因：结构化方法以功能分解和数据流为核心，但是系统功能和数据表示极有可能发生变化**
    - 信息建模法
      - 方法

        由实体-关系法（E-R方法）发展而来，核心概念是实体和关系。实体描述问题域中的事物，关系描述事物之间在数据方面的联系，都可以带有属性。发展之后的方法也把实体称为对象，并使用类型和子类型的概念，作为实体（对象）的抽象描述
      - 与面向对象方法的区别
        - 强调重点是信息建模和状态建模，而不是对象建模
        - 实体中只有属性没有操作
        - 只有属性的继承，不支持操作的继承
        - 没有采用消息通讯
- 面向对象方法

  把系统看做是一起工作来完成某项任务的相互作用的对象的集合
  - 与面向过程的结构化系统的本质区别
    - 面向过程：功能+数据
    - 面向对象：对象+消息
  - 优点
    - 更接近于问题域
    - 反复细化高层模型直到可以实现的程度
    - 将模型组织成对象的集合
  - 面向对象分析
    - 强调对问题和需求的**调查研究**，发现和描述对象
  - 面向对象设计
    - 强调满足需求的概念上的**解决方案**，定义软件对象以及它们如何协作
  - 面向对象编程
  - 面向对象的基本概念
    - 对象

        - 具有**责任**的实体。一个特殊的，自成一体的容器，对象的数据对于外部对象是受保护的
        - 特性：标识符、属性和操作
- 类
    - 具有相同性质、相同行为、相同对象关系，相同语义的对象集合
    - 具有相同属性和操作的一组**对象的抽象**，它为属于该类的全部对象提供了统一的抽象描述
        - 属性
            - 描述对象静态结构特征的一个数据项
            - 共有属性 +
            - 私有属性 -
            - 保护属性 #
- 封装
    - 把对象的属性和操作结合成一个独立的单元，并尽可能对外界隐藏数据的实现过程，隐藏实现细节
    - 一个对象不能直接操作另一个对象内部数据，它也不能使其他对象直接访问自己的数据，所有的交流必须通过 getter 和 setter 方法调用
    - 意义
        - 保护对象，避免用户误用
        - 保护客户端，其实现过程的改变，不会影响到相应客户端的改变
    - 问题
        - 编程麻烦
        - 执行效率的损失
    - 解决办法
        - 不强调严格封装
        - 实行可见性控制
- 继承/泛化
    - 泛化关系是类元的一般描述和具体描述之间的关系，具体描述建立在一般描述的基础之上，并对其进行了扩展。
    - 在共享祖先所定义的成分的前提下允许它自身定义增加的描述，这被称作继承
    - 单一继承与多重继承
- 多态
    - 同一个操作作用于不同给定对象上可以有不同的解释，并产生不同的执行结果
    - 接口定义相同，但是实现形式不同

        - 相同的消息发给不同的对象，每个对象将根据自己所属类中定义的这个操作去执行，从而产生不同的结果
    - 消息
        - 消息是对象间通信的手段
        - 有同步消息和异步消息
- 软件设计
    - 设计的概念
        - 良好的软件设计的三个特征
            - 目标：设计必须是**实现所有**包含在分析模型中的明确需求、以及客户期望的所有隐含**需求**
            - 形态：对开发、测试和维护人员来说，设计必须是可读的、可理解的、可操作的指南
            - 内容：设计必须提供软件的全貌，从实现的角度去说明功能、数据、行为等各个方面
        - 定义

          为问题域的外部课件行为的归约增添实际的计算机系统实现所需的细节，包括人机交互、任务管理和数据管理的细节
        - 设计的目标：质量
            - 软件质量
                - 外部质量
                - 内部质量
    - 设计的原则
        - 设计原则是系统分解和模块设计的基本标准，应用这些原则可以使代码更加灵活、易于维护和扩展
        - 抽象

          抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。
        - 封装

          封装和信息隐藏是指每个软件单元对其他所有单元都隐藏自己的设计决策， 各个单元的特性通过其外部可见的接口来描述
        - 模块化

          模块化是在逻辑和物理上将整个系统分解成多个更小的部分，其实质是“分而治之”，即将一个复杂问题分解成若干个简单问题，然后逐个解决。
        - 系统分解原则
            - 内聚性

              内聚性尽量将同一个功能的模块彼此之间的通信都能放在模块内部封装起来
            - 耦合性

              耦合性两个模块或子系统之间依赖关系的强度
        - 层次化
            - 每一层可以访问下层，不能访问上层

- 封闭式结构：每一层只能访问与其相邻的下一层
- 开放式结构：每一层还可以访问下面更低的层次

- 复用

  是利用某些已开发的、对建立新系统有用的软件元素来生成新的软件系统，其好处在于提高生产效率，提高软件质量

  - 源代码复用
  - 软件体系结构复用
  - 框架复用
  - 设计模式