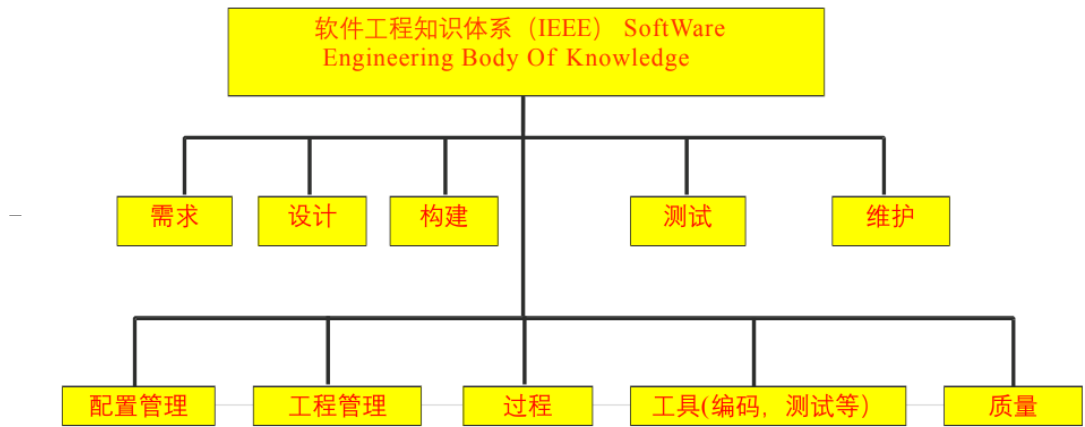


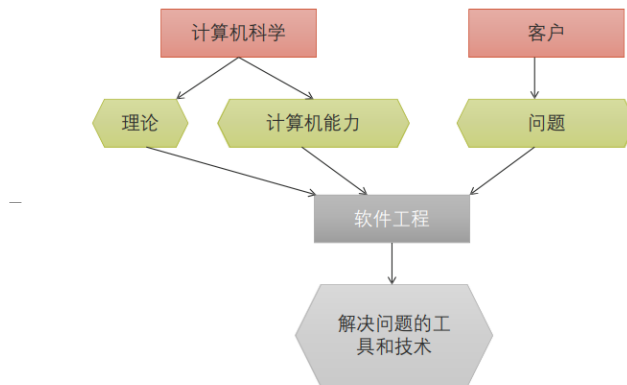
软件工程

1 软件工程介绍

- 软件的定义
 - 软件是计算机程序以及运行计算机系统可能需要的相关数据和文档。
 - 软件=程序+数据+文档
- 软件的分类
 - 按软件功能
 - 系统软件
 - 支撑软件
 - 应用软件
 - 按服务对象
 - 项目软件
 - 产品软件
- 本质特性* (p1)
 - 复杂性
 - 软件在规模上可能比任何由人类创造的其他实体都要复杂
 - 开发问题也会增加复杂性
 - 一致性
 - 软件必须遵从人为的惯例并适应已有的技术和系统
 - 可变性
 - 不可见性
- 软件危机
 - 在计算机软件的开发和维护过程中所遇到的一系列严重问题：效率和质量下降
 - 原因
 - 软件本质特征
 - 不正确的开发方法
- 软件工程的定义
 - 应用系统化的、学科化的、定量的方法，来开发、运行和维护软件，即将工程应用到软件；以及对各种方法的研究。
 - 目的：软件工程的目标是在给定的时间和预算内，按照用户的需求，开发易修改、高效、可靠、可维护、适应力强、可移动、可重用的软件。
- 软件质量
 - 软件质量是软件产品与明确的和隐含的需求相一致的 程度
 - 质量特性
 - 可维护性
 - 可依赖性
 - 有效性
 - 可用性
- 知识体系



- 软件工程 VS 计算机科学



2 软件过程

- 软件生命周期

- 软件产品或系统从设计、投入使用到被淘汰的全过程
- 问题定义、可行性研究、需求分析、总体设计、详细设计、编码、测试、维护

- 软件过程

- 构建过程中，所需完成的工作活动、动作和任务的集合

- 传统软件过程模型

- 瀑布模型

- 缺点：增加工作量、开发风险大、早期错误发现晚、不适应需求变化
- 适用场景：系统需求明确且稳定、技术成熟、 工程管理较严格的场合

- V模型

- 原型模型

- 优点：减少需求不明确带来的风险
- 缺点：快速建立起来的系统加上连续的修改可 能导致原型质量低下、设计者在质量和原型中进行折中、客户意识不到一些质量问题
- 适用场合：
 - 客户不清楚具体输入输出
 - 开发者不确定算法效率、软件操作系统以及客户与计算机的交互方式

- 增量模型

- 增量：满足用户需求的一个子集，能够完成一定功能、小而可用的软件
- 增量与原型的区别：原型的功能是部分开发的，而增量模型已开发部分功能是完整的

- 优点：
 - 不需要提供完整的需求
 - 软件能够更早投入市场
 - 开放式体系结构，便于维护
 - 产品逐步交付，软件开发能够较好地适应需求的变化
 - 缺点
 - 软件必须具备开放式体系结构（困难）
 - 易退化成边做边改的方式，使软件过程控制失去整体性
 - 这使得开发者很难根据客户需求给出大小适合的增量
 - 适用场景：需求可能发生变化、具有较大风险、或者希望尽早进入市场
- 螺旋模型
 - 开发过程分成若干次迭代，每次迭代代表开发的一个阶段，对应模型中一条环线，每次迭代分成四个方面的活动
 - 优点
 - 强调原型的可扩充性和可修改性
 - 为项目管理人员及时调整管理决策提供了方便
 - 缺点
 - 迭代次数过多，将会增加成本推迟交付时间
 - 型需要有相当丰富的风险评估经验和专门知识
 - 适用场合：需求不明确或者需求可能发生变化的大型复杂的软件系统
- 现代软件过程模型
 - 基于构件的开发模型
 - 考虑的焦点是集成，而非实现
 - 优点：软件重用，降低开发成本和风险，加快开发进度，提高软件质量
 - 缺点：商业构件不能修改，会导致修改需求
 - Rational统一过程模型
 - 一种以用例驱动、以体系结构为核心、迭代及增量的软件过程模型
 - 6条最佳实践：迭代式开发、管理需求、基于构件体系结构、可视化建模、验证软件质量、控制软件变更
 - 敏捷软件开发
 - 特点：个体交互、可工作软件、客户合作、响应变化
 - 极限编程
 - 优点：快速响应、可持续开发速度、适应商业竞争
 - 缺点：测试驱动开发违背用户期望、重构而不降低质量困难
 - 适合场景：适用于需求模糊且经常改变的场合，适合商业竞争环境下的项目
- 模型的选择
 - 对于完成多个独立功能开发的情况，可在需求分析阶段就进行功能**并行**，每个功能内部都尽量遵循**瀑布模型**
 - 全新系统的开发必须在总体设计完成后才开始增量或并行
 - 编码人员经验较少的情况下，尽量不要采用敏捷或迭代

3 需求分析

- 软件需求的不同层次
 - 功能需求：描述系统应该提供的功能或服务，通常涉及用户或外部系统与该系统之间的交互
 - 业务需求、用户需求
 - 非功能需求：响应时间、数据精度、可靠性、开发过程的标准
- 需求分析概述
 - 需求获取
 - 需求提炼：为问题涉及的信息、功能及系统行为建立模型
 - 需求描述：需求规格说明书，包括功能性需求和非功能性需求
 - 需求验证：有效性检查、一致性检查、完备性检查、现实性检查
 - 需求变更
- 软件需求规格文档编制

3.1 需求分析模型

	面向过程的需求分析	面向对象的需求分析
数据模型	实体-联系图 (ERD) 数据字典 (DD)	类图、类关系图
功能模型	数据流图 (DFD)	用例图
行为模型	状态变迁图 (STD)	活动图、时序图、状态图

3.1.1 面向过程的分析方法

- 结构化分析方法：用抽象模型的概念，自顶向下逐层分解
- 数据流图
 - 图形元素：数据加工、数据源/终点、数据流、数据存储文件
 - 加工
 - 表示对数据的操作
 - 顶层的加工名就是整个系统项目的名字
 - 尽量最好使用动宾词组
 - 外部实体
 - 系统位于数据源点与终点之间
 - 数据流
 - 数据存储
 - 绘制步骤
 1. 绘制顶层（第0层）
 2. 绘制第1层，根据工作流程绘制加工框
 3. 绘制第2层，细化加工框
 4. 合成总体数据流图
 5. 检查数据流图
 - 改进
 - 正确性检查：数据守恒（遗漏、多余）、数据存储（读+写）、父图与子图平衡
 - 提高易理解性：减少数据流、分解均匀、命名
 - 重新分解：重新连接为整体、重新拆分、重新绘制父图与子图

3.1.2 面向对象的分析方法

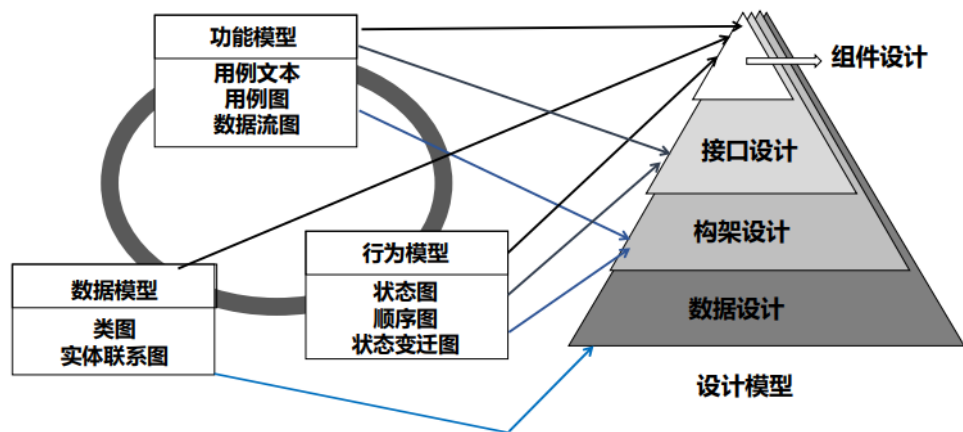
- 对象： 在现实世界中有意义的、与所要解决的问题有关系的任何事物
- 面向对象的软件开发模型：数据模型（类图）、行为模型（活动图、顺序图、状态图）、功能模型（用例图）
- 用例图
 - 图形元素：参与者、用例、执行关联
 - 元素间关系



4 系统设计

- 软件设计的定义：软件系统或组件的架构、构件、接口和其他特征的定义过程及该过程的结果
- 好的设计应该具有如下三个特点
 - 实现需求分析模型中所有明确要求+隐含需求
 - 对编码人员可理解
 - 提供软件完整视图，从实现的角度解决问题
- 设计质量属性
 - 功能性、易用性、可靠性、性能
 - 可支持性：拓展性、适应性、可维护性
- 设计相关的八大概念
 - 抽象、体系结构、设计模式、模块化、信息隐藏、功能独立、精化、重构
 - 抽象
 - 忽略具体的信息将不同事物看成相同事物的过程
 - 数据抽象：描述数据对象的冠名数据集合
 - 过程抽象：具有明确和有限功能的指令序列
 - 体系结构
 - 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
 - 体系结构设计可以使用大量的一种或多种模型来表达
 - 设计模式
 - 在给定上下文环境中一类共同问题的共同解决方案
 - 实体模式、结构模式、行为模式
 - 模块化
 - 软件被划分为命名和功能相对独立的多个组件（通常称为模块），通过这些组件的集成来满足问题的需求
 - 设计标准：分解性、组合性、可理解性、连续性、保护
 - 信息隐藏
 - 模块应该具有彼此相互隐藏的特性
 - 有助于定义构成软件的过程（或信息）实体
 - 功能独立

- 每个模块只负责需求中特定的子功能，并且从程序结构的其他部分看，该模块具有简单的接口
 - 内聚：模块功能强度
 - 耦合：模块之间的相互依赖程度
- 精化
 - 逐步求精的过程
- 重构
 - 不改变组件功能和行为条件下，简化组件设计（或代码）的一种重组技术
- 四类设计技术
 - 数据、架构、接口、组件





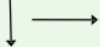
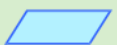
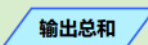



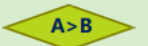


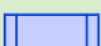





- 数据设计：概念、物理模型
- 体系（架构）结构设计
 - 风格和模式简要分类：数据中心架构、客户机/服务器结构、分层结构、层次结构、P2P、调用和返回架构、面向对象架构（MVC）、数据流体系架构
- 接口设计
- 组件设计
 - 面向过程：函数+模块
 - 面向对象：类+操作

4.2 面向过程的系统设计

- 系统结构图：传入模块、传出模块、变换模块、协调模块
- 变换型系统结构图：取得数据（输入）、变换数据（中心变换）、给出出具（输出）
- 事务型系统结构图：接受事务、分派处理单元、给出结果
- 变换分析
 1. 重画数据流图
 2. 区分输入、输出、中心变换
 3. 设计上层模块
 4. 进一步分解
- 事务分析
 - 事务是可以引发一个或多个处理的数据流
- 混合结构分析
 - 以变换分析为主、事务分析为辅 的方式进行软件结构设计
- 结构化组件设计：也称为过程设计、详细设计

— 流程图

	名 称	意 义	范 例
	开始 (Start) 终止 (End)	表示程序的开始或结束	 
	路径(Path)	表示流程进行的方向	
	输入(Input) 输(Output)	表示数据的输入或结果的输出	
	处理(Process)	表示执行或处理某一项工作	
	名 称	意 义	范 例
	决策判断 (Decision)	针对某一条件进行判断	
	循环 (Loop)	表示循环控制变量的初始值及终值	
	子程序 (Subroutine)	用以表示一群已经定义流程的组合	
	文件(Document) 指输入输出的文件		

4.3 面向对象的系统设计

- 面向对象设计的活动：框架设计、用例设计、类设计、数据库设计、用户界面设计
- 构架设计
 - 构建物理模型：将功能分配到物理节点、显示节点间拓扑结构、显示节点上软件配置
 - 设计子系统
 - 划分：按功能、物理布局、层次划分
 - 子系统关系：服务-请求、平等关系、依赖关系
 - 非功能需求设计：安全、错误检测与恢复、可移植性、通用性
- 细化用例
 - 类：三个格子的长方形，类名、属性、行为
 - 类间关系：关联、聚合、组合、依赖、泛化
 - 关联：表示两个类之间有关系
 - 聚合：整体与部分可以分开
 - 组合：整体与部分不可以分开
 - 依赖：某个类是另一个类方法的参数
 - 泛化
 - 分析类图
 - 边界类：参与者与用例、用例与用例、用例与系统边界外对象、独立业务之间
 - 实体类：存储信息+相关行为
 - 控制类：对用例场景中动词的分析和定义
 - 详细设计一个类
 - 定义类属性
 - 定义类方法
 - 定义类间关系
- UML顺序图

- 元素：对象、生命线、激活、消息
- 对象
 - 最多两个参与者
 - 三种命名：对象名+类名、对象名、类名
- 生命线
- 激活
- 消息：简单消息、同步消息、异步消息、反身消息、返回消息
- 设计原则
 - 四个层次：整体风格结构、中层、底层、非功能需求
 - 强内聚、弱耦合、耦合方式、可重用性、框架

4.4 设计模式

- 设计模式描述了软件系统设计过程中常见问题的解决方案，它是从大量的成功实践中总结出来的且被广泛公认的实践和知识
- 结构型模式
 - 描述了在软件系统中组织类和对象的常用方法
 - 采用继承机制来组合接口或实现
 - 适配器、桥接、组成、外观（Facade）、代理
- 行为模式
 - 负责分配对象的职责
 - 使用继承机制在类间分配行为
 - 观察者、策略
- 创建型模式
 - 描述了实例化对象的相关技术，解决了与创建对象有关的问题
 - 抽象工厂

5 软件实现

- 模型转换

6 质量保证

- 软件质量保证
 - 软件质量：明确表示是否符合功能和性能要求，明确地记载开发标准和所有专业开发软件的期望的隐性特点
 - 软件质量保证(SQA)：遵照一定的软件生产标准、过程和步骤对软件质量进行评估的活动
 - 软件可靠性
 - 平均无故障时间（MTTF）
 - 平均修复时间（MTTR）
 - 平均故障时间（MTBF）
- 软件测试策略
 - 回归测试
 - 单元测试
 - 测试内容：模块接口、局部数据结构、边界条件、路径、出错处理

- 测试环境
 - 驱动模块：被测模块的上层
 - 桩模块：被测模块的下层，被被测模块调用
- 集成测试
 - 将软件集成起来后进行测试
 - 集成方法
 - 自顶向下：深度、广度优先
 - 自底向上
 - SMOKE方法
- 系统测试
 - 从用户使用的角度进行测试，黑盒测试
 - 内容：压力、性能、功能、恢复、安全
- 验收测试
 - α 测试：用户或模拟用户在开发环境下测试
 - β 测试：用户在实际使用环境下进行测试
- 软件测试技术
 - 相关概念：软件缺陷、验证、确认、软件测试、软件质量保证、测试用例
 - 白盒测试
 - 结构测试、逻辑驱动测试
 - 检查范围
 - 所有独立的执行路径
 - 所有的逻辑判定
 - 循环的边界和运行界限内执行循环体
 - 内部数据结构
 - 逻辑覆盖
 - 语句覆盖：使得所有可执行语句被执行至少一次
 - 分支覆盖：每个分支判断取真和取假至少一次
 - 条件覆盖：每个判断的每个条件分别取真取假一次
 - 条件组合覆盖：每个判断的所有条件组合至少执行一次
 - 循环测试
 - 节点覆盖
 - 边覆盖
 - 路径覆盖：完全覆盖
 - 基本路径覆盖
 - 独立路径数量： $V(G) = e - n + 2$
 - 黑盒测试
 - 功能测试、数据驱动测试
 - 等价类：有效、无效
 - 有效用例覆盖尽可能多的有效等价类
 - 无效用例只覆盖单个无效等价类
 - 边界值分析：选取正好等于，刚刚大于，或刚刚小于边界的值

- 状态测试

7 软件维护

- 软件变更
 - 软件变更不可避免
 - 处理策略：软件维护、软件再工程
- 软件维护
 - 类型：完善性、纠错性、适应性、预防性
 - 维护的困难性
 - 维护过程
- 软件再工程
 - 源代码转换
 - 逆向工程
 - 程序结构改善
 - 程序模块化
 - 数据再工程