软件工程基础知识

软件生命周期与软件开发模型

软件危机与软件工程

- 软件危机
- 软件工程：软件工程是将系统化的、严格约束的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件
- 软件工程方法学：
    1. 方法：完成软件开发的各项技术方法
    2. 工具：为了应用开发方法所使用的支持环境，如.net环境
    3. 过程：为了完成高质量软件而进行的一系列任务框架，如：需求分析等

软件生命周期

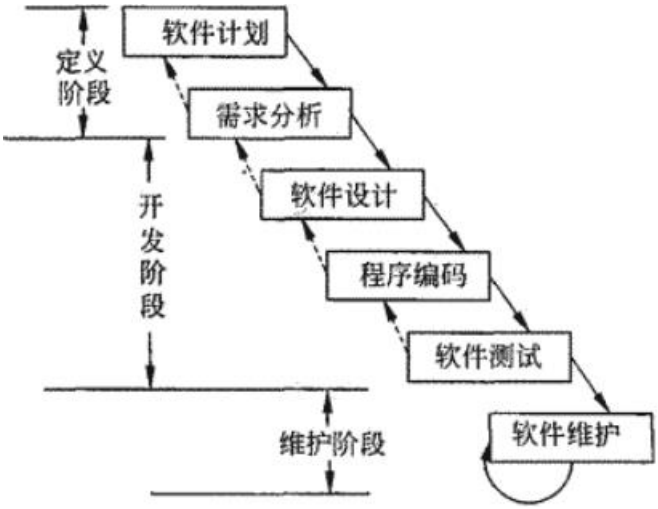
- 是指软件的产生直到报废的生命周期，包括：
    - 问题定义
    - 可行性分析
    - 需求分析
    - 业务需求

      ⬇
    - 用户需求

      ⬇
    - 系统需求：有利于开发的表达
        - 功能需求
        - 性能需求（非功能需求）
        - 设计约束（使用的开发工具等，与功能与性能无关）
    - QFD:质量展开模型
        - 基本需求：用户要求的需求，**必须完成**
        - 希望需求：用户没有明确要求，但是应该完成，这也是开发团队**必须完成**的
        - 兴奋需求：用户没有提出、也没觉得需要去做，但却被完成，且非常符合用户的需求，用户会开心；风险较大、成本会增加，**开发过程中不推荐**
    - 总体设计
    - 详细设计
    - 编码
    - 测试
    - 运行维护

软件开发模型

- 常见的开发模型：
    - 瀑布模型

    - 增量模型
    - 螺旋模型
    - 喷泉模型
    - 智能模型
    - V模型：与瀑布模型接近，但是更加强调测试，测试要在每个阶段进行测试，测试要贯穿软件开发的始终
        - 需求分析制定系统测试的计划
        - 概要设计制定集成测试的计划
        - 详细设计制定单元测试的计划
    - 快速应用开发模型
    - 构件组装模型
    - 敏捷方法
    - 统一过程
- 瀑布模型：最早的软件开发模型，最早的结构化开发模型
    - 瀑布模型也称为生命周期法，是结构化方法中最常用的开发模型，它把软件开发的过程分为软件计划、需求分析（属于定义阶段）、软件设计、程序编码、软件测试<产出需求说明书>（属于开发阶段）和运行维护（属于维护阶段）6个阶段
    - 

- 优点：
    1. 为项目提供了按阶段划分的检查点
        2. 当前一阶段完成后，只需要去关注后续阶段
    2. 它提供了一个模板，这个模板使得分析、设计、编码、测试和支持的方法可以在该模板下有一个共同的指导
    - 缺点：
        1. 各个阶段之间产生大量的文档，极大地增加了工作量
        2. 由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发风险
        3. 不适应用户需求的变化，并且在需求分析阶段不可能完全获取
        4. 在软件开发前期未发现的错误传到后面的开发活动中时，可能会扩散，进而可能会导致整个软件项目开发失败
- 适用：需求明确或很少变更的项目；二次开发
- 快速原型模型：补全了瀑布模型对需求分析无法完全的问题而产生的一种模型，主要针对需求不明确的情况
    - 快速原型是利用原型辅助软件开发的一种新思想；快速原型模型是“抛弃式”的原型方法

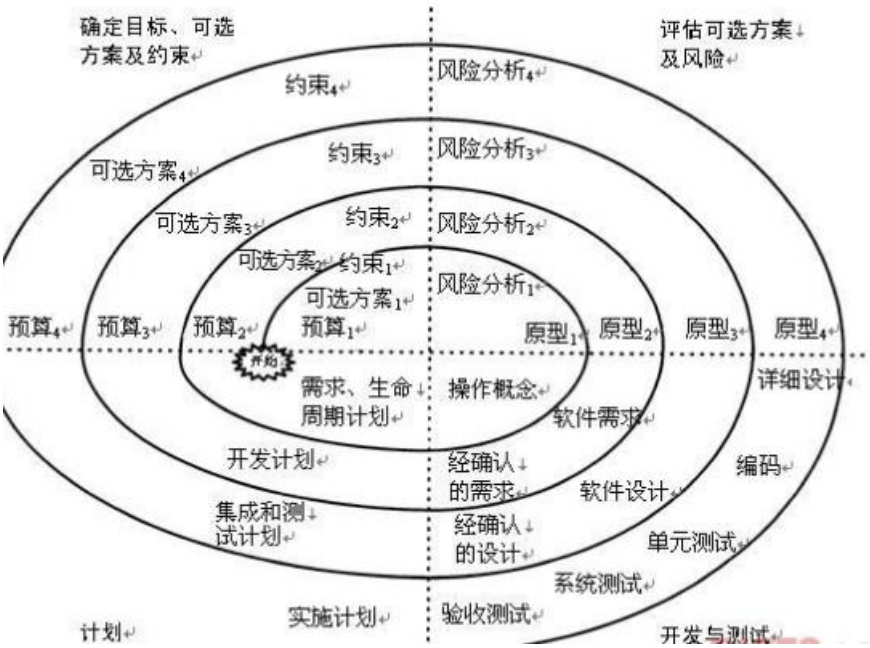
- 经过简单快速分析，快速建造一个可以运行的软件原型 ，以便理解和澄清问题，使开发人员与用户达成共识，最终在确定的客户需求基础上开发客户满意的软件产品
- 原型分类：
    1. 探索型原型：
    - 主要用于**需求分析阶段**，目的是要弄清用户的需求，并探索各种方案的可行性
    - 适用：它主要针对开发目标模糊，用户与开发人员对项目都缺乏经验的情况，通过对原型的开发来明确用户的需求
    2. 实验型原型：
    - 对于大型系统，若对设计方案心中没有把握时，可通过这种原型来证实设计方案的正确性
    - 适用：主要用于**设计阶段**，**考核实现方案是否合适**，能否实现
    3. 演化型原型（演化模型）：它将原型的思想扩展到软件开发的全过程
    - 该原型系统或者包含系统的框架，或者包含系统的主要功能，在得到用户的认可后，将原型系统不断扩充演变为最终的软件系统
    - 适用：主要用于及早向用户提交一个原型系统
- 适用：**适用于小型的开发项目；主要应用于需求分析阶段**

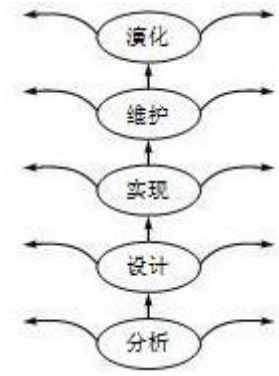
- 演化模型：经过原型的不断演化形成最终的产品，也称为变换模型，根据用户的基本需求，通过快速分析构造出一个初始可运行版本(原型)，然后根据用户在使用原型的过程中提出的意见和建议对原型进行改进，获得原型的新版本。重复这一过程，最终可得到令用户满意的软件产品
    - 演化模型是“渐进式”原型方法
    - 演化模型特别适用于对软件需求缺乏准确认识的情况
    - 优点：
        1. 很早就可以验证是否符合产品需求
        2. 风险管理可以在早期就获得项目进程数据，可据此对后续的开发进度作出比较切实的估算。增加项目成功的机率
        3. 经验教训能反馈于本产品的下一个循环过程，提高质量效率
        4. 心理上，开发人员早日见到产品的雏型，是一种鼓舞
        5. 使用户可以在新的一批功能开发测试后，立即参加验证，以便提供非常有价值的反馈
    - 缺点：
        1. 产品需求在一开始并不完全弄清楚的话，会给总体设计带来困难及削弱产品设计的完整性，并影响产品性能的优化
        2. 如果**缺乏严格的过程管理**，这个生命周期模型可能退化为一种原始的无计划的“试 - 错 - 改”模式
        3. 用户接触开发中的尚未测试稳定的功能，可能对用户都产生负面的影响
    - 适用：**对软件需求缺乏准确认识的情况**

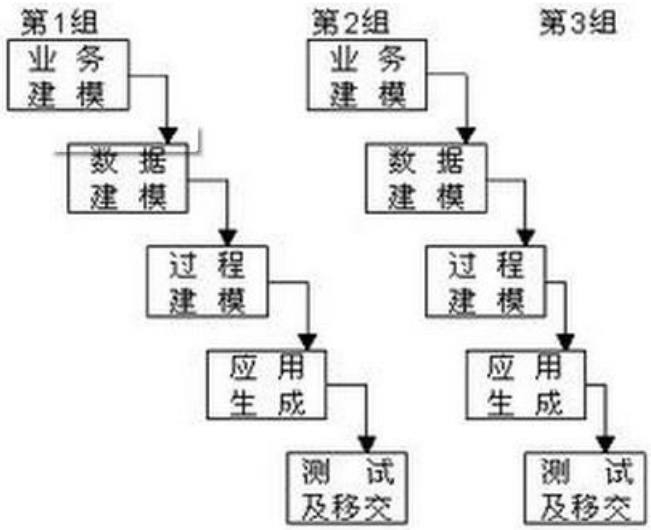
- 增量模型：融合了瀑布模型与原型模型的思想，形成了增量模型；融合了瀑布模型的基本成分和原型实现的迭代特征，是第三种原型化开发方法
    - 不是“抛弃式”的，也不是“渐进式”的
    - 增量模型把软件产品划分为一系列的增量构件，第一个增量往往是核心的产品，即第一个增量实现了基本的需求
    - 客户对每一个增量的使用和评估都作为下一个增量发布的新特征和功能，这个过程在每一个增量发布后不断重复，直到产生了最终的完善产品
    - 增量模型与原型实现模型和其他演化方法一样，本质上是迭代的，但与原型实现不一样的是其强调每一个增量均发布一个可操作产品
    - 特点：
        1. 引进了**增量包的概念**，无须等到所有需求都出来，只要某个需求的增量包出来即可进行开发

2. **每个增量均发布一个可以操作的产品**

- 优点：
  1. **人员分配灵活，初期不用太大投入**
  2. 每隔一小段时间就提交用户部分功能，用户可以直观感受项目进展，及时试用产品功能
  3. **有利于风险的把控**
  4. 核心模块尽早的交付用户，风险小很多
- 适用：增量模型将功能细化、分别开发的方法适应于**需求经常改变的软件开发过程**

- 螺旋模型：结合了瀑布模型、原型模型、演化模型的优点形成的模型
  - 将**瀑布模型和演化模型相结合**，综合了两者的优点，**并增加了风险分析**
  - 以原型为基础，沿着螺线自内向外旋转，每旋转一圈都要经过制订计划、风险分析、实施工程及客户评价等活动，并开发原型的一个新版本。经过若干次螺旋上升的过程，得到最终的系统
  - 

- 优点：
  1. 设计上灵活，可以在项目的各个阶段进行变更；以小的分段来构建大型系统，使成本计算变得简单容易；**客户始终参与每个阶段的开发**，保证了项目不偏离正确方向
  2. 随着项目推进，客户始终**掌握项目的最新信息，从而能够和管理层有效地交互**
- 缺点：
  1. **需要具有相当丰富的风险评估经验和专门知识，如果未能够及时标识风险，势必造成重大损失**
  2. 过多的迭代次数会**增加开发成本，延迟提交时间**
- 喷泉模型：早期的面向对象的模型。一种以用户需求为动力，以对象为驱动的模型，主要用于描述面向对象的软件开发过程，该模型认为软件开发过程自下而上的，各阶段是**相互迭代**和**无间隙**的。
  - **无间隙**是指在开发活动中，分析、设计和编码之间不存在明显的边界
  - 

- 基于构件的开发模型：将整个系统模块化，并在一定构件模型的支持下复用构件库中的一个或多个软件构件，通过组合手段高效率、高质量地构造应用软件系统的过程
    - 基于构件的开发模型由软件的需求分析和定义、体系结构设计、构件库建立、应用软件构建以及测试和发布5个阶段组成
    - 优点：
        1. **构件复用，提高了软件开发的效率**
        2. 构件可由一方定义其规格说明，被另一方实现。然后供给第三方使用，构件组装模型允许多个项目同时开发，降低了费用，提高了可维护性，可实现分步提交软件产品
    - 缺点：
        1. 缺乏通用的组装结构标准，因而引入了**较大的风险**
        2. 可重用性和软件高效性不易协调，**需要精干的有经验的分析和开发人员**
        3. 客户的满意度低，并且由于过分依赖于构件，所以**构件库的质量影响着产品质量**
- 快速应用开发模型（RAD）：适用IDE开发的过程
    - 是一个增量型的软件开发过程模型
    - **强调极短的开发周期**
    - RAD模型是瀑布模型的一个"高速"变种，通过大量使用可复用构件，采用基于构件的建造方法赢得快速开发
    - 如果需求理解得好且约束了项目的范围（业务建模），随后是**数据建模**、**过程建模**、**应用生成**、**测试**及**交付**
    - 

- 构件组装模型（CBSD）：
    - 需求分析和定义
    - 软件架构设计
    - 构件库的建立：使用以前稳定的构件进行使用，这是稳定的关键。能用的直接用，用不了的改改再用，实在不能用，直接做新的。
        - 构建标准
    - 应用软件的构建
    - 测试与发布

软件开发方法

敏捷开发方法

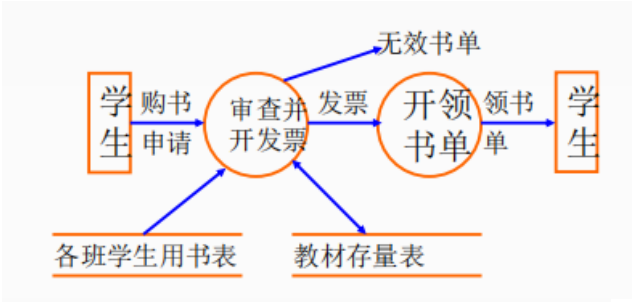
- 是一组开发方法
- ……

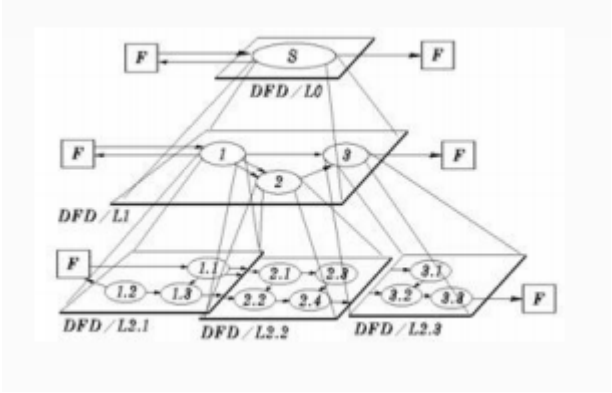
原型开发方法

- 适合需求不明确的开发，用来补充结构化开发的短板

结构化分析和设计

- 结构化**分析**(SA)：一种面向数据流的需求分析方法，利用图形表达用户需求，常用工具有**数据流图、数据字典**
  - 数据流图（DFD）：用来描述数据流从输入到输出的变换流程
  - 

    1. 加工： 
    2. 数据的源点或终点： 
    3. 数据的流向：
    4. 数据文件或数据库： 
  - 例图：
    

    1. DFD不同于程序流程图
    2. DFD可以表现大到整个系统，小到一个模块
    3. 分层数据流图
  - 数据字典（DD）：对软件中的每个数据规定一个定义条目，以保持数据在系统中的一致性
    - 包含内容：
      1. 数据项：只含一个数据，又称为数据元素
      2. 数据流：由多个相关数据项组成

        - 例图：

发票=(学号)+姓名+{书号+单价+数量+总价}+书费合计

数据流名：	发票
别　名：	购书发票
组　成：	学号+姓名+{书号+单价+数量+总价}+书费合计
备　注：	

        3. 数据文件：数据库
        4. 基本加工

- 结构化设计(SD)：一种面向数据流的设计方法，以分析阶段产生的**文档**（**数据流图**、**数据字典**、**软件需求说明书**）为基础，**逐步求精**和**模块化**（高内聚、低耦合、复杂度）的过程
    - 结构化设计通常可以分为**概要设计**和**详细设计**：
        - 概要设计：概要设计也称为结构设计或总体设计
            1. 基本任务：设计**软件系统结构**，进行**模块划分**，确定**每个模块的功能**、**接口**、**模块间的调用**关系
            2. 设计工具：结构图、数据字典（DD）、判定树和判定表
        - 详细设计：为每个模块设计其实现的细节
            - 设计工具：
                1. 程序流程图
                2. 盒图(NS 图)
                3. 问题分析图(PAD)
                4. 程序设计语言 (PDL)

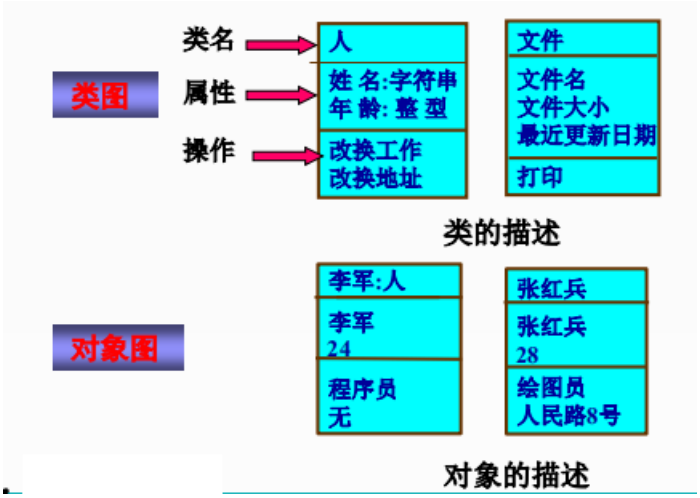
面向数据结构的设计

- 根据**输入/输出数据结构**导出程序结构
- Jackson方法和Warnier方法是最著名的两个面向数据结构的设计方法

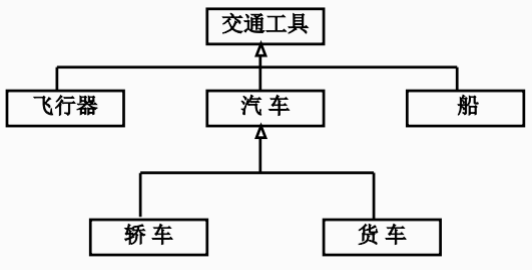
面向对象的分析与设计

- 面向对象的方法是一种运用**对象**、**类**、**继承**、**封装**、**聚合**、**消息传送**、**多态性**等概念来构造系统的软件开发方法
- **面向对象**=**对象**(object)+**类**(classification)+**继承**(inheritance)+**通信**(communication with messages)
- 对象（object）：对象是系统中用来描述客观事物的一个实体，是构成系统的一个基本单位
    - 包含内容：
        1. 属性（attribute）：也称为状态或数据，用来描述对象的静态特征
        2. 操作（operation）：（也称方法或服务）规定了对象的行为，表示对象所能提供的服务
        3. 封装（encapsulation）：是一种信息隐蔽技术，用户只能看见对象封装界面上的信息，对象的内部实现对用户是隐蔽的
- 类（class）：类是一组具有相同属性和相同操作的对象的集合。一个类中的每个对象都是这个类的一个实例（instance）。类是创建对象的模板，从同一个类实例化的每个对象都具有相同的结构和行为
- 对象和类的描述：对象和类一般采用“对象图” 和 “类图” 来描述

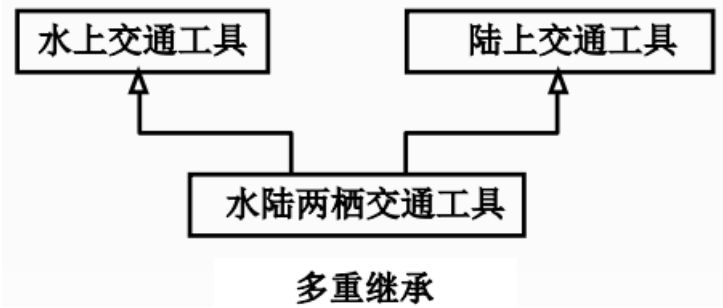
- 例图：



- 继承（inheritance）：继承是指特殊类（子类）的对象拥有其一般类（父类）的全部属性与服务。**父类中定义了其所有子类的公共属性和操作**，在**子类中除了定义自己特有的属性和操作外**，可以**继承其父类（或祖先类）的属性和操作**，还可以**对父类（或祖先类）中的操作重新定义其实现方法**



  - 单一继承：一个子类只有唯一一个父类
  - 多重继承：一个子类有一个以上的父类



- 多态性：是指同一个操作作用于不同的对象上可以有不同的解释，并产生不同的执行结果
- 消息（message）：消息传递是对象间通信的手段，一个对象通过向另一个对象发送消息来请求其服务
  - 一个消息通常**包括接收对象名、调用的操作名**和**适当的参数**（如果有必要的话）
  - 消息只告诉接收对象需要完成什么操作，但并不指示接收者怎样完成操作
  - **消息完全由接收者解释执行**
- 面向对象方法的优点：
  1. 与人类习惯的思维方法一致
  2. 稳定性好
  3. **可重（复）用性好**
  4. 较易开发大型软件产品
  5. 可维护性好

软件测试与软件维护

- 软件测试：是指在规定的条件下对程序进行操作，以**发现程序错误**，**衡量软件质量**，并**对其是否能满足设计要求进行评估的过程**

- 软件的正确性证明尚未得到根本的解决，软件测试仍是发现软件错误和缺陷的主要手段

软件测试基础

- 测试用例的构成：1、测试数据　2、预期结果
- 为了发现程序中的错误，**应竭力设计能暴露错误的测试用例**
- **好的测试用例**是**发现至今为止尚未发现的错误**
- 一次**成功的测试**是**发现了至今尚未发现的错误**的测试

软件测试准则

- **应该尽早地、不断的进行测试**，软件**测试贯穿于开发过程的始终**
- 所有**测试**都应该能**追溯到用户需求**
- 应该**从小规模测试开始，并逐步进行大规模测试**
- 应该远在测试之前就制定出测试计划（**制定测试计划要超前**）
- 80%的错误可能出现在20%的程序模块中
- 应该由独立的第三方从事测试工作
- **对非法和非预期的输入数据也要像合法数据一样编写测试用例**
- 检查软件**是否做了应该做的事**仅是成功一半，另一半是看**软件是否做了不该做的事**
- 在规划测试时不要设想程序中不会查出错误
- 测试**只能证明软件中有错误**，**不能证明软件中没有错误**

软件测试分类

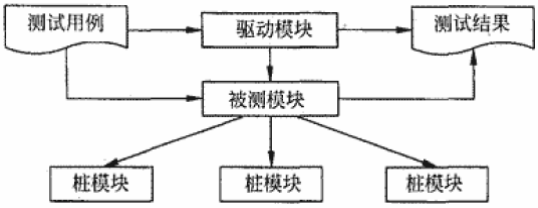
- 从测试阶段分：单元测试、集成测试、确认测试和系统测试
- 从测试方法分：白盒测试、黑盒测试
- 回归测试：是指修改了旧代码后，重新进行全部或部分以前的测试用例，以确认修改没有引入新的错误或导致其他代码产生错误

测试的阶段

单元测试（由模块的开发人员完成）：单元测试的计划是在软件详细设计阶段完成的

- 单元测试又称为模块测试，是针对软件设计的**最小单位（模块）进行正确性检验的测试工作**
- 测试内容：每个程序单元能否正确实现详细设计说明中的
    1. 模块功能
    2. 性能、
    3. 接口
    4. 设计约束等
    5. 各模块内部可能存在的各种错误
- 单元测试通常**由开发人员自己负责**
- 单元测试要**借助驱动模块**（相当于用于测试模拟的主程序）和**桩模块**（子模块）来完成
- **单元测试的计划是在软件详细设计阶段完成的**
- 单元测试一般**使用白盒测试方法**
- 辅助模块：在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块
    - 分类：
        1. 驱动模块：相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后输出实测结果

2. 桩模块：用于代替被测模块调用的子模块。桩模块可以做少量的数据操作，不需要把子模块所有功能都带进来，但不允许什么事情也不做



集成测试：集成测试计划是在软件概要设计阶段完成的

- 集成测试也称为组装测试、联合测试
- 它将已通过单元测试的模块集成在一起，主要测试模块之间的协作性
- 从组装策略而言，可以分为一次性组装和增量式组装（包括自顶向下、自底向上及混合式）两种
- **集成测试计划是在软件概要设计阶段完成的**
- 集成测试一般采用黑盒测试方法
- 自顶向下进行组装，不需要驱动模块
- 自底向上进行组装，不需要桩模块
- 在每个版本提交时，都需要进行“冒烟”测试，即对程序主要功能进行验证

  冒烟测试也称为版本验证测试或提交测试

确认测试：确认测试计划是在需求分析阶段完成的

- 确认测试也称为有效性测试，主要包括验证软件的功能、性能及其他特性是否与用户要求（需求）一致
- **确认测试计划是在需求分析阶段完成的**
- 分类：
  1. 内部确认测试：由软件开发组织内部按软件需求说明书进行测试
  2. Alpha测试：由用户在开发环境下进行测试
  3. Beta测试：由用户在实际使用环境下进行测试

系统测试：系统测试计划在系统分析阶段（需求分析阶段）完成

- 是将已经确认的软件、计算机硬件、外设、网络等其他元素结合在一起，进行信息系统的各种组装测试和确认测试
- 系统测试是**针对整个产品系统**进行的测试
- 目的：是验证系统**是否满足了需求规格**的定义，找**出与需求规格不符或与之矛盾的地方**，从而**提出更加完善的方案**
- **系统测试计划在系统分析阶段（需求分析阶段）完成**
- 测试内容：
  1. 功能测试
  2. 性能测试
  3. 健壮性测试
  4. 用户界面测试
  5. 安全性测试
  6. 安装与反安装测试

测试的类型

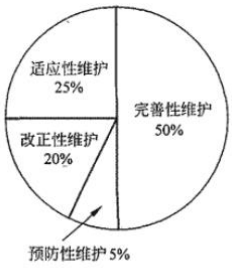
动态测试：动态测试指通过运行程序发现错误

- 分类：
  - 黑盒测试法：黑盒测试又称为功能测试或数据驱动测试。把被测试对象看成一个黑盒子，测试人员完全不考虑程序的内部结构和处理过程，只在软件的接口处进行测试，依据需求规格说明书，检查程序是否满足功能要求
    - 常用的黑盒测试用例的设计方法：
      1. 等价类划分
      2. 等价类划分
      3. 错误推测
      4. 因果图
  - 白盒测试法：又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试。把测试对象看做一个打开的盒子，测试人员必须了解程序的内部结构和处理过程，以检查处理过程的细节为基础，对程序中尽可能多的逻辑路径进行测试，检验内部控制结构和数据结构是否有错，实际的运行状态与预期的状态是否一致
    - 常用的白盒测试用例设计方法：差错能力逐渐增强
      1. 语句覆盖：每条语句至少执行一次
      2. 判定覆盖：每个判定的每个分支至少执行一次
      3. 条件覆盖：每个判定的每个条件应取到各种可能的值
      4. 条件判定覆盖：每个判定的每个条件应取到各种可能的值
      5. 条件组合覆盖：每个判定中各条件的每一种组合至少出现一次
      6. 路径覆盖：使程序中每一条可能的路径至少执行一次
  - 灰盒测试法：灰盒测试是一种介于白盒测试与黑盒测试之间的测试，它关注输出对于输入的正确性，同时也关注内部表现，但这种关注不像白盒测试那样详细且完整，而只是通过一些表征性的现象、事件及标志来判断程序内部的运行状态。灰盒测试结合了白盒测试和黑盒测试的要素，考虑了用户端、特定的系统知识和操作环境，在系统组件的协同性环境中评价应用软件的设计

静态测试：静态测试指被测试程序不在机器上运行，而采用人工检测和计算机辅助静态分析的手段对程序进行检测

- 人工测试的主要方法
  - 桌前检查(Desk Checking)：由**程序员自己检查自己编写的程序。**程序员在程序通过编译之后，进行单元测试设计之前，对源程序代码进行分析、检验，并补充相关的文档，目的是发现程序中的错误
  - 代码审查：代码审查是由若干程序员和测试员组成一个会审小组，通过阅读、讨论和争议，对程序进行静态分析的过程
    - 步骤：
      1. 第一步，小组负责人提前把设计规格说明书、控制流程图、程序文本及有关要求、规范等分发给小组成员，作为评审的依据。小组成员充分阅读这些材料
      2. 第二步，召开程序审查会。在会上，首先由程序员逐句讲解程序的逻辑。在此过程中，程序员或其他小组成员可以提出问题，展开讨论，审查错误是否存在
  - 代码走查：代码走查与代码审查基本相同
    - 步骤
      1. 第一步，把材料先发给走查小组成员，认真研究程序，再开会
      2. 第二步，开会的程序与代码会审不同，不是简单地读程序和对照错误检查表进行检查，而是**让与会者“充当”计算机。让测试用例沿程序的逻辑运行一遍，随时记录程序的踪迹，供分析和讨论使用。**数据标准化、数据命名合理、文件格式转换、数据库格式转换等。

软件维护

- 软件可维护性是指维护人员对该软件进行维护的难易程度，具体包括理解、改正、改动和改进软件的难易程度
- 衡量程序可维护性的因素：**可理解性**、**可测试性**和**可修改性**等
- 软件维护占整个软件生命周期的60%～80%，维护的类型有
    1. 改正性维护：指改正在系统开发阶段已发生而系统测试阶段尚未发现的错误
    2. 适应性维护：指使用软件适应信息技术变化和管理需求变化而进行的修改
    3. 完善性维护：为扩充功能和改善性能而进行的修改，主要是指对已有的软件系统增加一些在系统分析和设计阶段中没有规定的功能与性能特征
    4. 预防性维护：为了改进应用软件的可靠性和可维护性，为了适应未来的软硬件环境的变化，应主动增加预防性的新的功能，以使应用系统适应各类变化而不被淘汰
- 

各种维护所占的比例
- 影响维护工作量的主要因素：
    1. 系统大小
    2. 程序设计语言
    3. 系统年龄
    4. 数据库技术的应用
    5. 先进的软件开发技术

软件工具与软件开发环境

软件质量保证

- 软件质量是“软件与**明确的**和**隐含的**定义**的需求相一致的程度**”。具体地说，软件质量是软件符合明确叙述的功能和性能需求、文档中明确描述的开发标准、以及所有专业开发的软件都应具有的隐含特征的程度
- 质量要点：
    1. **用户需求**是度量软件质量的**基础**（满足即可，不必过剩，成本控制）
    2. 指定的**标准**定义了一组指导软件开发的准则（GB/T-XXXXXXX国家行业标准）
    3. 没有显式描述的隐含需求（如期望软件是容易维护的）

软件质量特性

- 三个层次：质量的度量与评价
    - 质量特性：功能性、可靠性、易用性、效率、可维护性、可移植性
    - 质量子特性：对各个质量特性的描述
    - 度量指标
    -

质量特性	功能性	可靠性	易用性	效率	维护性	可移植性
质量子特性	适合性	成熟性	易理解性	时间特性	易分析性	适应性
	准确性	容错性	易学性	资源利用性	易改变性	易安装性
	互操作性	易恢复性	易操作性		稳定性	共存性
	保密安全性		吸引性		易测试性	易替换性
	依从性	依从性	依从性	依从性	依从性	依从性

- **功能性**：与功能及其指定的性质有关的一组属性

- **适合性**：与规定任务**能否提供一组功能**及这组功能的适合程度有关的软件属性
  - **准确性**：与能否得到**正确**或**相符**的**结果**或效果有关的软件属性
  - **互操作性**：与**其他指定系统**进行**交互**的能力有关的软件属性
  - **保密安全性**：与**防止对**程序及数据的**非授权**的故意或**意外访问**的能力有关的软件属性
  - **功能性的依从性**：使软件**遵循**有关的**标准**、**约定**、**法规**及类似规定的软件属性
- **可靠性**：在规定的时间和条件下，软件维持其性能水平的能力有关的一组属性
- **成熟性**：与由软件故障引起**失效的频度**有关的软件属性
  - **容错性**：与由软件**故障**或**违反指定接口**的情况下，维持规定的性能水平的能力有关的软件属性
  - **可恢复性**：在**失效发生后**，重建其性能水平并恢复直接受影响数据的能力，以及为达此目的所需的**时间和能力有关的软件属性**
- **易用性**：与一组规定或潜在的用户为使用软件所需作的努力和对这样的使用所作的评价有关的一组属性
- **易理解性**：与用户为认识逻辑概念及其应用范围所花的努力有关的软件属性（对于图标的理解）
  - **易学习性**：与用户为学习软件应用所花的努力有关的软件属性（）
  - **易操作性**：与用户为操作和运行控制所花的努力有关的软件属性
- **效率**：与在规定的条件时间特性与软件执行其功能时，软件的性能水平与所用资源量之间关系有关的一组属性
- **时间**特性：响应和处理时间及吞吐量有关的软件属性
  - **资源**特性：使用的资源数量及其使用资源的持续时间有关的软件属性
- **可维护性**：与软件维护的难易程度有关的一组属性
  - **易分析性**：与为诊断缺陷或失效原因及为判定待修改的部分所需努力有关的软件属性（发生错误时）
  - **易修改性**：与进行修改，排除错误或适应环境变化所需努力有关的软件属性（发生错误时）
  - **稳定性**：与修改所造成的未预料结果的风险有关的软件属性
  - **可测试性**：与确认已修改软件所需的努力有关的软件属性（修改后可以测试）
- **可移植性**：与软件可从某一环境转移到另一环境的能力有关的一组属性
- **适应性**：与软件无需采用特殊处理就能适应不同的规定环境有关的软件属性
  - **易安装性**：在指定环境下安装软件的难易程度
  - **一致性**：软件服从与可移植性有关的标准或约定的程度
  - **可替换性**：软件在特定软件环境中用来替代指定的其他软件的可能性和难易程度

软件质量保证

- 是**为保证**软件系统充分满足**用户要求的质量**而进行有计划、有组织的活动，其目的是生产高质量软件
- 质量保证主要手段：
  1. 开发**初期**制订质量保证**计划**
  2. **开发前**选定制订开发**标准**、开发**规范**
  3. 选择分析设计方法和工具，形成高质量 分析模型 设计模型
  4. 严格执行**阶段评审**，**及时发现问题**
  5. 进行**阶段测试**
  6. 严格执行**变更控制流程**
  7. 有完整**阶段文档**

CMM(软件能力成熟度模型)：是一种用于评价软件承包能力以改善软件质量的方法，侧重于软件开发过程的管理及工程能力的提高与评估

- 五个等级（重点，要背）
  - **初始级**（一级）：软件过程是无序的，有时甚至是混乱的，对过程几乎没有定义，成功取决于个人努力。管理是反应式的
  - **可重复级**（二级）：**建立了基本的项目管理过程**来跟踪费用、进度和功能特性。**制定了必要的过程纪律**，**能重复**早先**类似应用项目取得的成功经验**
  - **已定义级**（三级）：已将**软件管理和工程**两方面的过程文档化、标准化，并**综合成该组织的标准软件过程**。所有项目均使用经批准、剪裁的标准软件过程来开发和维护软件，软件产品生产在整个软件过程是可见的
  - **已管理级**（四级）：软件过程和产品质量的**详细度量数据**，对软件过程和产品都有定量的理解与控制。管理有一个作出结论的客观依据，管理能够在定量的范围内预测性能
  - **优化级**（五级）：过程的量化反馈和先进的新思想、新技术促使过程持续不断改进

CMMI(软件能力成熟度集成模型)

- 五个等级：
  1. 初始级
  2. 可管理级
  3. 严格定义级
  4. 定量管理级
  5. 优化级

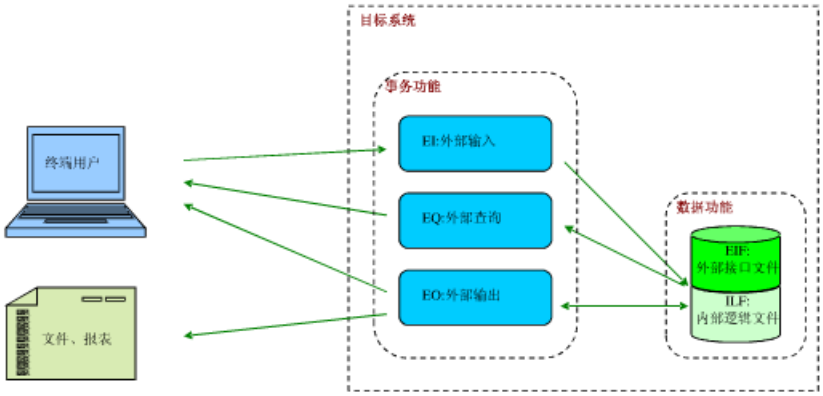
软件项目管理

项目管理内容

- 质量管理
- 进度管理
- 成本管理

软件项目估算

- 估算内容：具有先后顺序
  1. 软件**规模**的估算
  2. 软件**工作量**的估算
  3. **成本**估算
- 软件规模估算方法：
  1. LOC估算法（代码行估算法）：估算软件的**代码行数**。将软件项目划分为小的模块，通过历史数据、开发人员经验对LOC数进行估算
  2. FP估算：FP（功能点）是一种衡量工作量大小的单位
     - 功能点=信息处理规模*技术复杂度
     - 信息处理规模：技术复杂度=0.65+调节因子

    - 外部输入数
    - 外部输出数
    - 外部查询数
    - 内部逻辑文件数
    - 外部接口文件数
  - 

- 软件工作量估算：
  - COCOMO模型是一种精确的、易于使用的工作量估算方法
  - **基本COCOMO模型**：用已估算出来的**代码行数(LOC)**为自变量的**经验函数计算**软件开发工作量
  - **中间COCOMO模型**：在基本模型的基础上，再用**涉及产品、硬件、人员、项目等方面的影响因素****调整**工作量的估算
  - **详细COCOMO模型**：在中间模型的基础上，按**模块层**、**子系统层**、系统层，做出**三张工作量**因素分级表，供不同层次估算使用
- 项目成本估算：

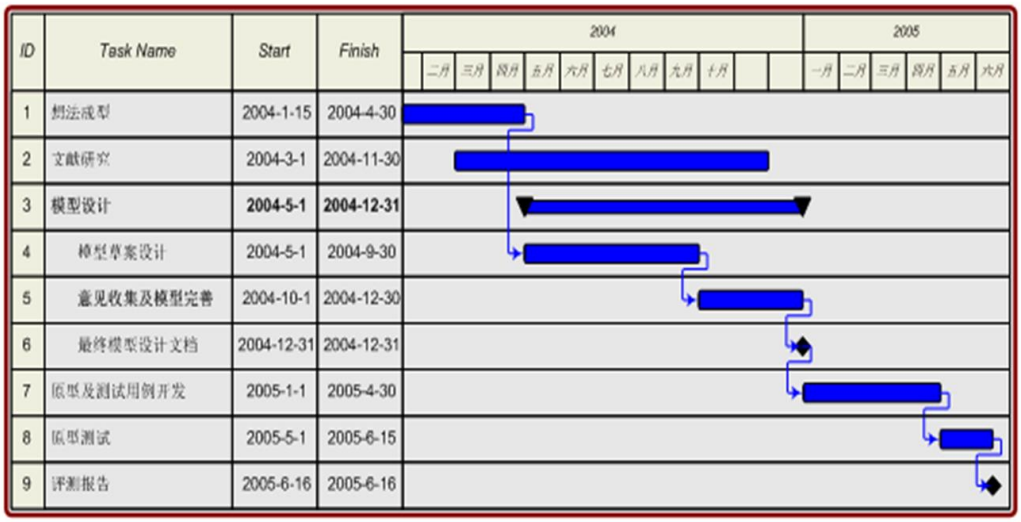
风险管理

风险管理过程：是指对项目风险从**识别**到**分析**乃至采取**应对**措施等一系列过程

1. **编制风险管理计划**：制定风险的识别、分析、应对的策略
2. **风险识别**：识别项目中可能出现的风险
3. **定性风险分析**：风险发生的概率、危害性
4. **定量风险分析**：对定性分析进行量化
5. **风险应对计划的编制**：对不同的风险进行不同的优先级进行处理与不同的应对策略
6. **风险跟踪和监控**：一直监控项目，当有发生风险的苗头时，及时消除风险

- 识别项目中的风险方法：
  1. 头脑风暴法：只记录，不评价，最后进行总结
  2. 专家评估法：聘请专家进行对项目风险进行评估
  3. 风险检查表：在一张风险表中，取出一个风险，检查项目中是否具有该风险
  4. 假设分析：假设风险是否存在
- IT项目中常见的风险：
  1. 需求风险：需求变更的风险
  2. 技术风险：是否拥有这项技术（专利问题），是否具有可以实现这种技术的能力
  3. 团队风险：
  4. 关键人员风险：项目的核心人员离职带来的风险
  5. 预算风险：预算缩水或者扩大的风险
  6. 范围风险：项目范围扩大或者缩小带来的风险

项目计划编排的方法与技术

- 计划评审技术（PERT）：PERT是利用网络分析制定计划以及对计划予以评价的技术。它能协调整个计划的各道工序，合理安排人力、物力、时间、资金，加速计划的完成
    - 乐观时间：**任何事情都顺利的情况**下，完成某项工作的时间
    - 最可能时间：**正常情况下**，完成某项工作的时间
    - 悲观时间：**最不利的情况下**，完成某项工作的时间
    - 活动期望时间=（乐观时间+4*最可能时间+悲观时间）/6
    - 方差=（悲观时间-乐观时间）^2/36
    - 标准差=方差开方
- 甘特图：甘特图也叫**横道图**，它以**横线来表示每项活动的起止时间**
    - 优点：简单、明了、直观、易于编制的，因此到目前为止仍然是**小型项目中常用的工具**
    - 缺点：**不能系统地表达各项工作之间的复杂关系**，难以进行定量的计算和分析，以及计划的优化，同时也**没有指出影响项目寿命周期的关键**所在
    - 适用：在**大型工程项目**中，它是**高级管理层了解全局、基层安排进度**时有用的工具
    - 

- 关键路径法(CPM)是借助网络图和各活动所需时间（估计值），计算每一活动的最早或最迟开始和结束时间
    - CPM算法的核心思想是将工作分解结构（WBS)分解的活动按逻辑关系加以整合，统筹计算出整个项目的工期和关键路径
    - 和时间相关的参数：
        1. **最早开始时间（ES)**：某项活动能够开始的最早时间
        2. **最早结束时间（EF)**：某项活动能够完成的最早时间，EF=ES+工期估计
        3. 最迟结束时间（LF)：为了使项目按时完成，某项工作必须完成的最迟时间
        4. 最迟开始时间（LS)：为了使项目按时完成，某项工作必须开始的最迟时间，LS=LF-工期估计
    - 两个规则:
        1. 规则1 :某项活动的最早开始时间必须相同或晚于直接指向这项活动的最早结束时间中的最晚时间
        2. 规则2 :某项活动的最迟结束时间必须相同或早于该活动直接指向的所有活动最迟开始时间的最早时间
    - 最早完工时间
        - 步骤：
            1. 从网络图始端向终端计算
            2. 第一任务的开始为项目开始
            3. 任务完成时间为开始时间加持续时间
            4. 后续任务的开始时间根据前置任务的时间和搭接时间而定
            5. 多个前置任务存在时，根据最迟任务时间来定

- 最晚完工时间
  - 步骤：
    1. 从网络图终端向始端计算
    2. 最后一个任务的完成时间为项目完成时间
    3. 任务开始时间为完成时间减持续时间
    4. 前置任务的完成时间根据后续任务的时间和搭接时间而定
    5. 多个后续任务存在时，根据最早任务时间来定
- 关键路径
  - 步骤：
    1. 总持续时间最长的线路称为关键线路
    2. 总时差最小的工作组成的线路为关键线路
- 总时差是指一项工作在不影响总工期的前提下所具有的机动时间
  1. 总时差=该工作最迟完工时间-该工作最早完工时间
  2. 总时差=该工作最迟开工时间-该工作最早开工时间
- 自由时差指一项工作在不影响其紧后工作最早开始时间的条件下，本工作可以利用的机动时间
  1. 自由时差=该工作的紧后工作最早开时间-该工作最早完工时间