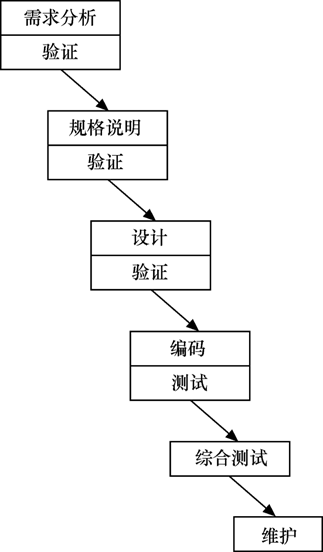
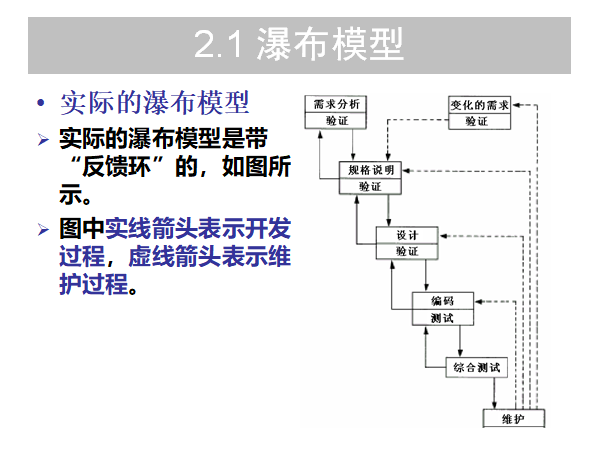
# 软件工程概论1-2

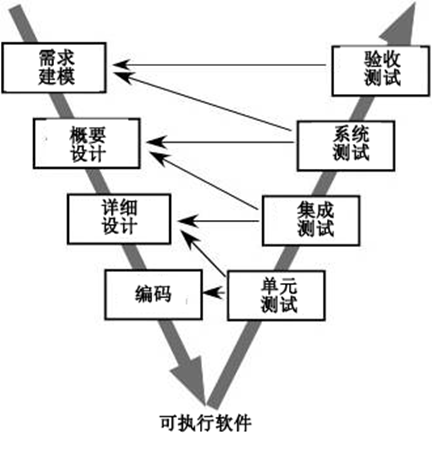
1. 一、软件与软件工程的概念
   1. 1.软件的概念、特点和分类
      1. 软件的概念
         1. 软件是计算机系统中与硬件相互依存的另一部分，它是包括程序，数据及其相关文档的完整集合。
         2. 程序是按事先设计的功能和性能要求执行的指令序列。
         3. 数据是使程序能正常操纵信息的数据结构。
         4. 文档是与程序开发，维护和使用有关的图文材料。
      2. 软件的特点
         1. (1)形态特性: 软件是无形的、不可见的逻辑实体。度量常规产品的几何尺寸、物理性质和化学成分对它却是毫无意义的。
         2. (2)智能特性：软件是复杂的智力产品，它的开发凝聚了人们的大量脑力劳动，它本身也体现了知识实践经验和人类的智慧，具有一定的智能。它可以帮助我们解决复杂的计算、分析、判断和决策问题。
         3. (3)开发特性: 尽管已经有了一些工具（也是软件）来辅助软件开发工作，但到目前为止尚未实现自动化。软件开发中仍然包含了相当份量的个体劳动，使得这一大规模知识型工作充满了个人行为和个人因素。
         4. (4)质量特性：目前还无法得到完全没有缺陷的软件产品 。
         5. (5)生产特性:与硬件或传统的制造业产品的生产完全不同，软件一旦设计开发出来，如果需要提供多个用户，它的复制十分简单，其成本也极为有限 。
         6. (6)管理特性：由于上述的几个特点，使得软件的开发管理显得更为重要，也更为独特 。
         7. (7)环境特性:软件的开发和运行都离不开相关的计算机系统环境，包括支持它的开发和运行的相关硬件和软件。软件对于计算机系统的环境有着不可摆脱的依赖性。
         8. (8)维护特性：软件投入使用以后需要进行维护，但这种维护与传统产业产品的维护概念有着很大差别。
         9. (9)废弃特性: 与硬件不同，软件并不是由于被“用坏”而被废弃的 。
         10. (10)应用特性：软件的应用极为广泛，如今它已渗入国民经济和国防的各个领域，现已成为信息产业、先进制造业和现代服务业的核心，占据了无可取代的地位。
      3. 软件的分类
         1. （1） 系统软件
            1. 操作系统
            2. 数据库管理系统
            3. 设备驱动程序
            4. 通信和网络处理程序等
         2. （2）支撑软件（工具软件）
            1. 纵向支撑软件：分析、设计、编码、测试工具等
            2. 横向支撑软件：项目管理工具，配置管理工具等
         3. （3）应用软件
            1. 工程与科学计算软件
            2. 商业数据处理软件
            3. ERP软件
            4. 计算机辅助设计／制造软件
            5. 系统仿真软件
            6. 智能产品嵌入软件
            7. 事务管理、办公自动化软件
         4. （4）可复用软件
            1. 标准函数库、类库、构件库等
   2. 2.软件危机与软件工程
      1. 软件危机主要有以下一些典型表现:
         1. 对软件开发成本和进度的估计常常很不准确。
         2. 用户对“已完成的”软件系统不满意的现象经常发生。
         3. 软件产品的质量往往靠不住。
         4. 软件常常是不可维护的。
         5. 软件通常没有适当的文档资料。
         6. 软件成本在计算机系统总成本中所占的比例逐年上升。
         7. 软件开发生产率提高的速度，既跟不上硬件的发展速度，也远远跟不上计算机应用迅速普及深入的趋势。
      2. 除了软件本身的特点，软件危机发生的主要原因有：
         1. 缺乏软件开发的经验和有关软件开发数据的积累，使得开发工作的计划很难制定。
         2. 软件人员与用户的交流存在障碍，使得获取的需求不充分或存在错误。
         3. 软件开发过程不规范。如，没有真正了解用户的需求就开始编程序。
         4. 随着软件规模的增大，其复杂性往往会呈指数级升高。需要很多人分工协作，不仅涉及技术问题，更重要的是必须有科学严格的管理。
         5. 缺少有效的软件评测手段，提交用户的软件质量不能完全保证。
      3. 如何摆脱软件危机?
         1. 彻底消除“软件就是程序”的错误观念。
         2. 充分认识到软件开发应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。
         3. 推广和使用在实践中总结出来的开发软件的成功技术、方法和工具。
         4. 按工程化的原则和方法组织软件开发工作。
   3. 3.软件工程的目标
      1. 基本目标：
         1. 付出较低的开发成本
         2. 达到要求的软件功能
         3. 取得较好的软件性能
         4. 开发的软件易于移植
         5. 需要较低的维护费用
         6. 能按时完成开发工作，及时交付使用
      2. 软件的质量特性：功能性、可靠性、可使用性、效率、可维护性和可移植性。
         1. 功能性是指软件所实现的功能达到它的设计规范和满足用户需求的程度；
         2. 可靠性是指在规定的时间和条件下，软件能够正常维持其工作的能力；
         3. 可使用性是指为了使用该软件所需要的能力；
         4. 效率是指在规定的条件下用软件实现某种功能所需要的计算机资源的有效性；
         5. 可维护性是指当环境改变或软件运行发生故障时，为了使其恢复正常运行所做努力的程度；
         6. 可移植性是指软件从某一环境转移到另一环境时所做努力的程度。
   4. 4.软件生存期
      1. 软件生存期分为三个时期
         1. 软件定义时期
            1. 问题定义
            2. 可行性研究
            3. 需求分析
         2. 软件开发时期
            1. 概要设计
            2. 详细设计
            3. 编码（程序编码和单元测试）
            4. 测试（集成测试和系统测试）
         3. 运行维护时期
            1. 改正性维护，也就是诊断和改正在使用过程中发现的软件错误；
            2. 适应性维护，即修改软件以适应环境的变化；
            3. 完善性维护，即根据用户的要求改进或扩充软件，使它更完善；
            4. 预防性维护，即修改软件为将来的维护活动预先做准备
   5. 5.软件工程方法三要素及作用
      1. 概念
         1. 软件工程包含技术和管理两方面的内容，是技术和管理紧密结合所形成的工程学科。
         2. 通常将软件开发全过程中使用的一整套技术方法的集合称为方法学(methodology)，也称为范型(paradigm)。
         3. 目前使用最广泛的软件工程方法学：传统方法学（结构化方法学），面向对象方法学。
      2. 三要素：方法、工具和过程。
         1. 软件工程方法为软件开发提供了 “如何做” 的技术;
         2. 软件工具为软件工程方法提供了自动的或半自动的软件支撑环境;
         3. 过程是为了获得高质量的软件所需要完成的一系列任务框架，它规定了完成各项任务的工作步骤。
      3. 结构化方法学
         1. 也称为生命周期方法学或结构化范型。将软件生命周期的全过程依次划分为若干个阶段，采用结构化技术来完成每个阶段的任务。
         2. 特点：
            1. (1) 强调自顶向下顺序地完成软件开发的各阶段任务;
            2. (2) 结构化方法要么面向行为，要么面向数据，缺乏使两者有机结合的机制。
      4. 面向对象方法学
         1. 是将数据和对数据的操作紧密地结合起来的方法。
         2. 软件开发过程是多次反复迭代的演化过程。
         3. 面向对象方法在概念和表示方法上的一致性，保证了各项开发活动之间的平滑过渡。
         4. 对于大型、复杂及交互性比较强的系统，使用面向对象方法学更有优势。
2. 二、软件生存期模型
   1. **1.瀑布模型**
      1. 特点：
         1. （1）阶段间具有顺序性和依赖性。其中包含两重含义：
            1. ① 必须等前一阶段的工作完成之后，才能开始后一阶段的工作；
            2. ② 前一阶段的输出文档就是后一阶段的输入文档。
         2. （2）推迟实现的观点
            1. ① 瀑布模型在编码之前设置了系统分析和系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。
            2. ② 清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想。
         3. （3）质量保证的观点
            1. ① 每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。
            2. ② 每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。
      2. 图
         1. 瀑布模型



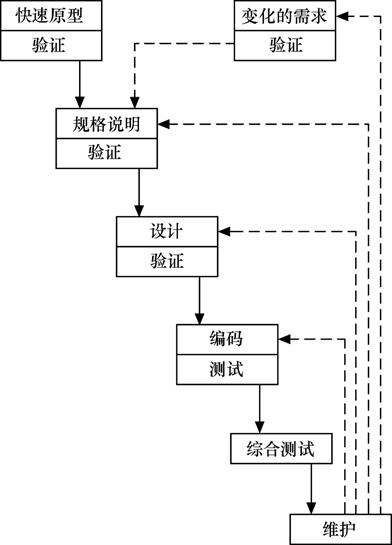
* + - 1. 实际的瀑布模型



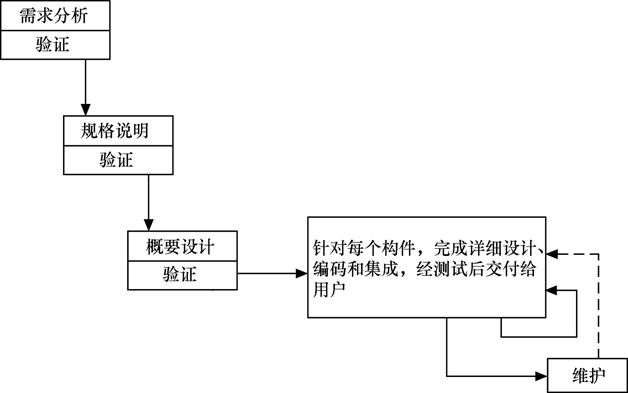
* + 1. V模型：瀑布模型的一个变体
       1. V模型描述了测试阶段的活动与开发阶段相关活动（包括需求建模、概要设计、详细设计、编码）之间的关系。



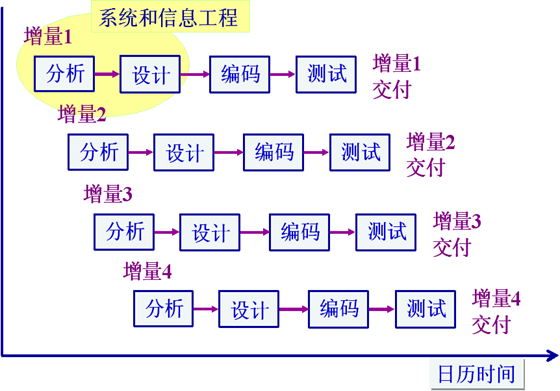
* + 1. 优点：
       1. 可强迫开发人员采用规范化的方法。
       2. 严格地规定了每个阶段必须提交的文档。
       3. 要求每个阶段交出的所有产品都必须是经过验证的。
    2. 缺点：
       1. 瀑布模型的缺点
       2. 由于瀑布模型几乎完全依赖于书面的规格说明，很可能导致最终开发出的软件产品不能真正满足用户的需要。如果需求规格说明与用户需求之间有差异，就会发生这种情况。
       3. 瀑布模型只适用于项目开始时需求已确定的情况。
  1. **2.快速原型模型**
     1. 特点：
        1. 快速原型是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。
     2. 图：



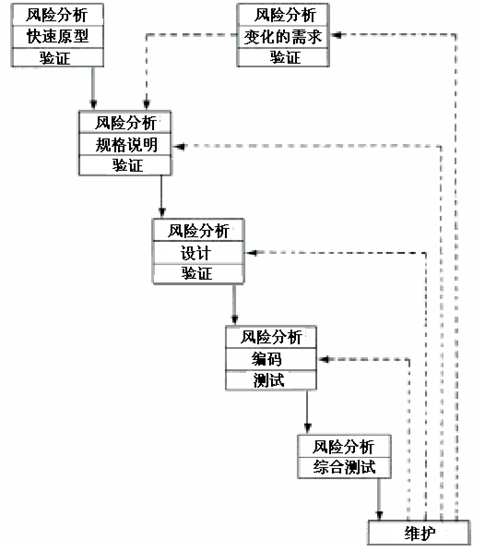
* + 1. 优点：
       1. (1)有助于满足用户的真实需求。
       2. (2)原型系统已经通过与用户的交互而得到验证，据此产生的规格说明文档能够正确地描述用户需求。
       3. (3)软件产品的开发基本上是按线性顺序进行。
       4. (4)因为规格说明文档正确地描述了用户需求，因此，在开发过程的后续阶段不会因为发现规格说明文档的错误而进行较大的返工。
       5. (5)开发人员通过建立原型系统已经学到了许多东西，因此，在设计和编码阶段发生错误的可能性也比较小，这自然减少了在后续阶段需要改正前面阶段所犯错误的可能性。
       6. (6) 快速原型的突出特点是“快速”。开发人员应该尽可能快地建造出原型系统，以加速软件开发过程，节约软件开发成本。
       7. 原型的用途是获知用户的真正需求，一旦需求确定了，原型可以抛弃，当然也可以在原型的基础上进行开发。
  1. **3.增量模型**
     1. 特点：
        1. 增量模型也称为渐增模型，是Mills等于1980年提出来的。
        2. 使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码、集成和测试。
        3. 每个构件由多个相互作用的模块构成，并且能够完成特定的功能。
     2. 图：



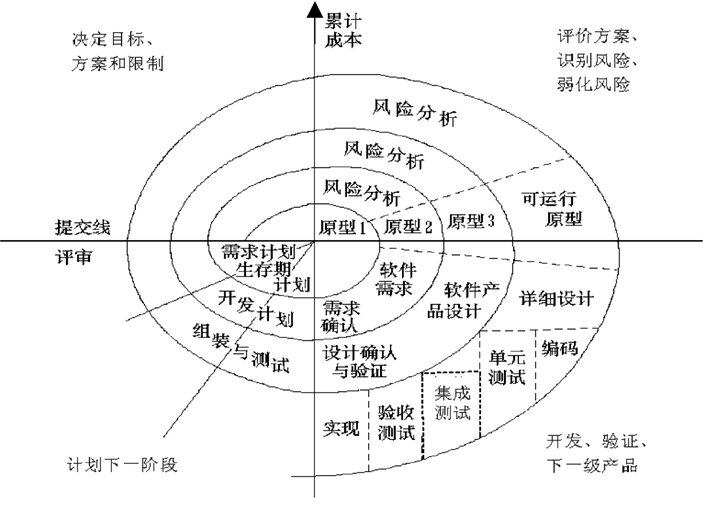
* + 1. 优点：
       1. (1)能在较短时间内向用户提交可完成一些有用的工作产品，即从第1个构件交付之日起，用户就能做一些有用的工作。
       2. (2)逐步增加产品的功能可以使用户有较充裕的时间学习和适应新产品，从而减少一个全新的软件可能给用户组织带来的冲击。
       3. (3)项目失败的风险较低，虽然在某些增量构件中可能遇到一些问题，但其他增量构件将能够成功地交付给客户。
       4. (4)优先级最高的服务首先交付，然后再将其他增量构件逐次集成进来。因此，最重要的系统服务将接受最多的测试。
    2. 问题：
       1. (1)在把每个新的增量构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。
       2. (2)软件体系结构必须是开放的，即向现有产品中加入新构件的过程必须简单、方便。
       3. 因此，采用增量模型比采用瀑布模型和快速原型模型更需要精心的设计。
    3. 增量构件开发
       1. 每个增量构件应当实现某种系统功能，因此增量构件的开发可以采用瀑布模型的方式，如图所示。



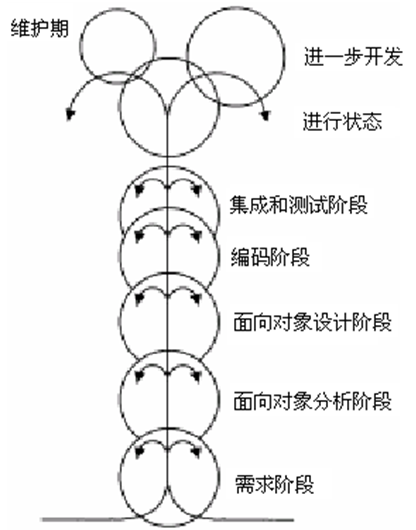
* 1. **4.螺旋模型**
     1. 特点：
        1. 螺旋模型最初是Boehm于1988年提出来的。
        2. 该模型将瀑布模型与快速原型模型结合起来，并且加入两种模型均忽略了的风险分析。
        3. 螺旋模型的基本思想是，使用原型及其他方法来尽量降低风险。
        4. 理解这种模型的一个简便方法，是把它看做在每个阶段之前都增加了风险分析过程的快速原型模型。
     2. 图
        1. 图



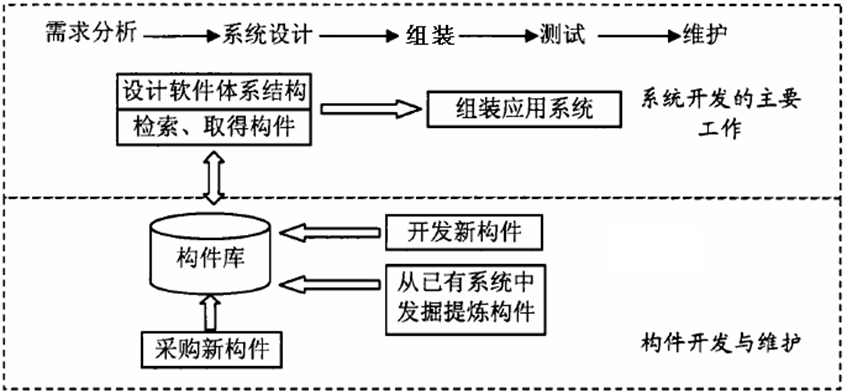
* + - 1. 完整的螺旋模型



* + 1. 完整的螺旋模型
       1. 在螺旋模型中，软件过程表示成一个螺线，而不是像以往的模型那样表示为一个具有回溯的活动序列。
       2. 在螺线上的每一个循环表示过程的一个阶段。
       3. 每个阶段开始时的任务是确定该阶段的目标、为完成这些目标选择方案及设定这些方案的约束条件。接下来的任务是，从风险角度分析上一步的工作结果，努力排除各种潜在的风险，通常用建造原型的方法来排除风险。如果成功地排除了所有风险，则启动下一步开发步骤，在这个步骤的工作过程相当于纯粹的瀑布模型。最后是评价该阶段的工作成果并计划下一个阶段的工作。
    2. 螺旋模型的4项活动 ：
       1. 螺线上的每一个循环可划分为4个象限，分别表达了4个方面的活动。
       2. (1)目标设定——定义在该阶段的目标，弄清对过程和产品的限制条件，制订详细的管理计划，识别项目风险，可能还要计划与这些风险有关的对策。
       3. (2)风险估计与弱化——针对每一个风险进行详细分析，设想弱化风险的步骤。
       4. (3)开发与验证——评价风险之后选择系统开发模型。
       5. (4)计划——评价开发工作，确定是否继续进行螺线的下一个循环。如果确定要继续，则计划项目的下一个阶段的工作。
    3. 优点：
       1. 对可选方案和约束条件的强调有利于已有软件的重用，也有助于把软件质量作为软件开发的一个重要目标。
       2. 减少了过多测试或测试不足所带来的风险。
       3. 在螺旋模型中维护只是模型的另一个周期，因而在维护和开发之间并没有本质区别。
    4. 缺点：
       1. 螺旋模型是风险驱动的，因此要求软件开发人员必须具有丰富的风险评估经验和这方面的专门知识，否则将出现真正的风险：当项目实际上正在走向灾难时，开发人员可能还以为一切正常。
  1. **5.喷泉模型**
     1. 特点：
        1. 喷泉模型是典型的面向对象生命周期模型。
        2. “喷泉”一词体现了迭代和无间隙特性。图中代表不同阶段的圆圈相互重叠，这明确表示两个活动之间存在重叠。
     2. 图：



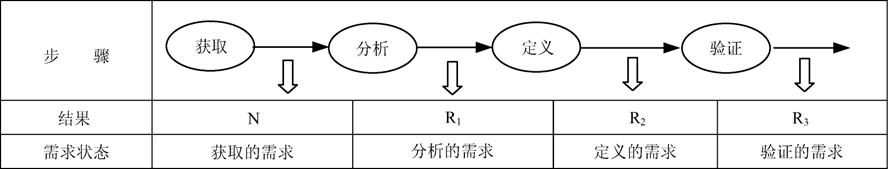
* 1. 6.统一过程
     1. 统一过程的工作流
        1. ① 业务建模工作流。用商业用例为商业过程建立文档。
        2. ② 需求工作流。目标是描述系统应该做什么，确保开发人员构建正确的系统。为此，需明确系统的功能需求和非功能需求（约束）。
        3. ③ 分析和设计工作流。其目标是说明如何做。结果是分析模型和设计模型。
        4. ④ 实现工作流。用分层的方式组织代码的结构，用构件的形式来实现类，对构件进行单元测试，将构件集成到可执行的系统中。
        5. ⑤ 测试工作流。验证对象之间的交互、是否所有的构件都集成了、是否正确实现了所有需求、查错并改正。
        6. ⑥ 部署工作流。制作软件的外部版本、软件打包、分发、为用户提供帮助和支持。
     2. 统一过程的阶段：统一过程有4个阶段，分别是初始阶段、细化阶段、构造阶段和移交阶段。
        1. ① 初始阶段。初始阶段主要关注项目计划和风险评估，其目的是确定是否值得开发目标信息系统。
        2. ② 细化阶段。细化阶段关心定义系统的总体框架，其目标是：细化初始需求（用况）、细化体系结构、监控风险并细化它们的优先级、细化业务案例以及制订项目管理计划。
        3. ③ 构造阶段。构造阶段是建立系统，构造信息系统的第1个具有操作质量的版本，以能够交付给客户进行b测试的版本结束，有时称为测试版本。
        4. ④ 移交阶段。移交阶段包含测试时期，以发布完整的系统而终止，其目标是确保信息系统真正满足客户的需求。
  2. 7.基于构件的开发模型
     1. 基于构件的开发模型如下图



* + 1. 开发步骤：
       1. 不考虑构件的开发技术，基于构件的开发模型由以下步骤组成：
       2. (1)对于该问题领域的基于构件的可用产品进行研究和评估。
       3. (2)考虑构件集成的问题。
       4. (3)设计软件架构以容纳这些构件。
       5. (4)将构件集成到架构中。
       6. (5)进行充分的测试以保证功能正常。
    2. 典型的构件模型
       1. (1)OMG/CORBA。对象管理组织发布了公共对象请求代理体系结构（OMG/CORBA），一个对象请求代理提供了多种服务，使得可复用构件（对象）可以与其他构件通信。
       2. (2)Microsoft COM/DCOM/.NET。微软公司开发了构件对象模型（COM），此模型提供了构件的规格说明,在Windows操作系统，一个应用系统中可以使用不同厂商生产的构件。
       3. (3)Sun JavaBean构件。JavaBean构件系统是一个可移植的、平台独立的、使用Java程序设计语言开发的CBSE基础设施。
  1. 8.敏捷过程模型
     1. 敏捷原则 ：
        1. (1)我们最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。
        2. (2)即使在开发的后期，也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。
        3. (3)经常交付可运行软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。
        4. (4)在整个项目开发期间，业务人员和开发人员必须天天都在一起工作。
        5. (5)围绕有积极性的个人构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。
        6. (6)在团队内部，最富有效果和效率的信息传递方法是面对面交谈。
        7. (7)可运行软件是进度的首要度量标准。
        8. (8)敏捷过程提倡可持续的开发速度。责任人、开发者和用户应该能够保持一种长期、稳定的开发速度。
        9. (9)不断地关注优秀的技能和好的设计会增强敏捷能力。
        10. (10)简单是必要的。
        11. (11)好的架构、需求和设计出自于自组织团队。
        12. (12)每隔一定时间，团队会反省如何才能更有效地工作，并相应调整自己的行为。

# 软件工程概论3-4

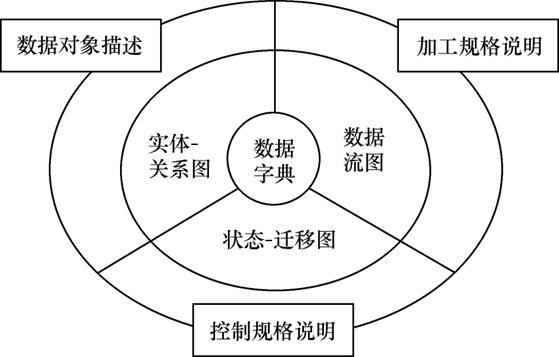
1. 三、软件需求获取与结构化分析方法
   1. 1.需求获取的任务和原则
      1. 任务：
         1. (1) 发现和分析问题，并分析问题的原因/结果关系。
         2. (2) 与用户进行各种方式的交流，并使用调查研究方法收集信息。
         3. (3) 按照三个成分观察问题的不同侧面：即数据、过程和接口。
         4. (4) 将获取的需求文档化，形式有用例、决策表、需求表等。
      2. 原则：
         1. (1) 深入浅出的原则。就是说，需求获取要尽可能全面、细致。获取的需求是个全集，目标系统真正实现的是个子集。
         2. (2) 以流程为主线的原则。在与用户交流的过程中，应该用流程将所有的内容串起来。如信息、组织结构、处理规则等。这样便于交流沟通。流程的描述既有宏观描述，也有微观描述。
   2. 2.需求获取的过程
      1. 1. 开发高层的业务模型
      2. 2. 定义项目范围和高层需求
      3. 3. 识别用户类和用户代表
         1. 系统的不同用户之间在很多方面存在差异，例如：
            1. (1) 使用产品的频率；
            2. (2) 用户在应用领域的经验和使用计算机系统的技能；
            3. (3) 所用到的产品功能；
            4. (4) 为支持业务过程所进行的工作；
            5. (5) 访问权限和安全级别
      4. 4. 获取具体的需求
         1. 确定了项目范围和高层需求，并确定了用户类及用户代表后，就需要获取更具体、完整和详细的需求。具体需求的来源可以来自以下几种典型的途径。
            1. (1) 与用户进行交流。
            2. (2) 现有产品或竞争产品的描述文档。
            3. (3) 系统需求规格说明。
            4. (4) 当前系统的问题报告和改进要求。
            5. (5) 市场调查和用户问卷调查。
            6. (6) 观察用户如何工作。
      5. 5. 确定目标系统的业务工作流
         1. 具体到当前待开发的应用系统，确定系统的业务工作流和主要的业务规则，采取需求调研的方法获取所需的信息。例如，针对信息系统的需求调研方法如下：
            1. (1) 调研用户的组织结构、岗位设置、职责定义，从功能上区分有多少个子系统，划分系统的大致范围，明确系统的目标。
            2. (2) 调研每个子系统的工作流程、功能与处理规则，收集原始信息资料，用数据流来表示物流、资金流、信息流三者的关系。
            3. (3) 对调研内容事先准备，针对不同管理层次的用户询问不同的问题，列出问题清单。将操作层、管理层、决策层的需求既联系又区分开来，形成一个需求的层次。
      6. 6. 需求整理与总结
         1. 必须对上面步骤取得的需求资料进行整理和总结，确定对软件系统的综合要求，即软件的需求。
         2. 并提出这些需求实现条件，以及需求应达到的标准。
         3. 这些需求包括功能需求、性能需求、环境需求、可靠性需求、安全保密要求、用户界面需求、资源使用需求、软件成本消耗与开发进度需求等。
   3. 3.软件需求分析阶段的任务
      1. 可以把软件需求分析阶段的工作分为4个步骤，即获取需求、分析需求、定义需求和验证需求，如图所示。



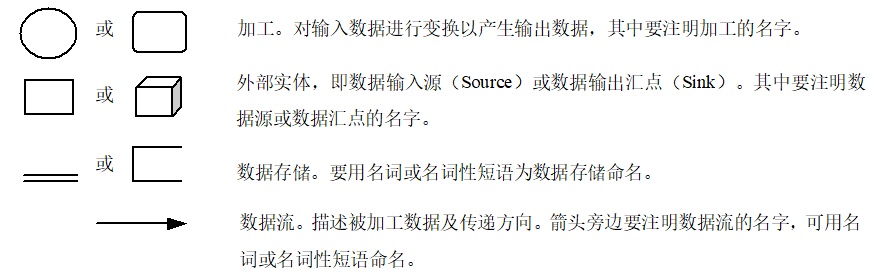
* + 1. 1. 需求获取
       1. 通过启发、引导从客户（或用户）那里得到的原始需求是他们的业务要求（needs），简称为N。
       2. 这是分析之前获取的需求，其中可能存在一些实际问题，这些问题只有通过分析才能得到解决，直接把获取的需求作为软件设计阶段的依据将会导致严重的后果。
    2. 2. 需求分析
       1. 认真研究获取的需求，必须考虑以下几方面：



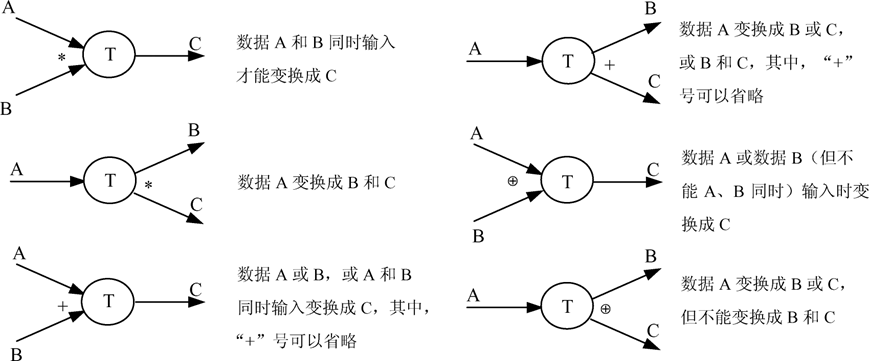
* + - 1. (1) 完整性：每项获取的需求都应给出清楚的描述，使得软件开发工作能够取得设计和实现该功能所需要的全部必要信息。
      2. (2) 正确性：获取的每项需求必须是准确无误的，并且需求描述无歧义性。
      3. (3) 合理性：各项需求之间、软件需求与系统需求之间应是协调一致的，不应存在矛盾和冲突。
      4. (4) 可行性：包括技术可行性 、经济可行性 、社会可行性 。
      5. (5) 充分性：获取的需求是否全面、周到。
    1. 3. 需求定义
       1. 将已经过分析的需求清晰、全面、系统、准确地描述成为正式的文档，这一步定义需求的工作就是编写需求规格说明。
    2. 4. 需求验证
       1. 为了确保已定义的需求（需求规格说明）准确无误，并能为客户（或用户）理解和接受，需要对其进行严格的评审。
  1. 4.结构化分析方法
     1. 传统的分析建模方法称为结构化分析（structured analysis，SA）方法。结构化分析方法是一种建模技术，它建立的分析模型如图所示。



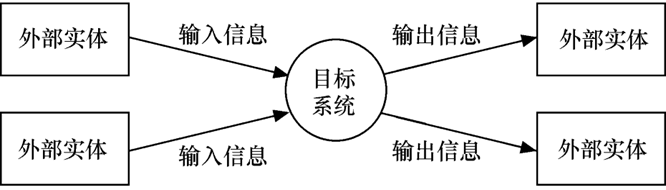
* + 1. 1.功能建模
       1. 概念： 功能建模的思想就是用抽象模型的概念，按照软件内部数据传递、变换的关系，自顶向下逐层分解，直到找到满足功能要求的所有可实现的软件为止。功能模型用数据流图来描述。
       2. 基本图形符号：



* + - 1. 多个数据流之间的关系 ：



* + - 1. 环境图：、
         1. 环境图（context diagram）也称为顶层数据流图（或0层数据流图），它仅包括一个数据处理过程，也就是要开发的目标系统。环境图的作用是确定系统在其环境中的位置，通过确定系统的输入和输出与外部实体的关系确定其边界。
         2. 典型的环境图



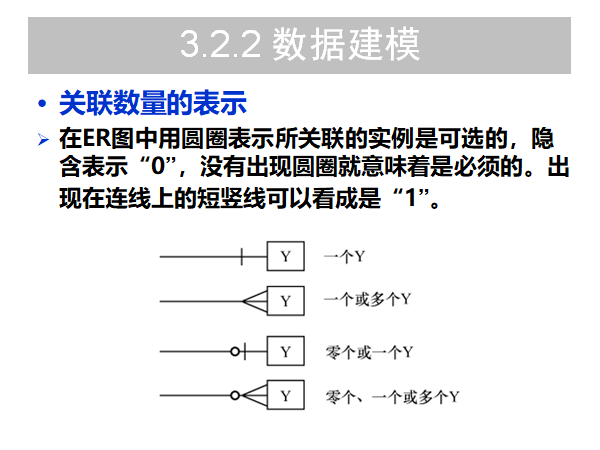
* + 1. 2.数据建模
       1. 概念
          1. 在结构化分析方法中，使用实体—关系建模技术来建立数据模型。这种技术是在较高的抽象层次（概念层）上对数据库结构进行建模的流行技术。实体—关系模型表示为可视化的实体—关系图（entity-relationship diagram，ERD），也称为ER图。
          2. ER图中仅包含3种相互关联的元素：数据对象（实体）、描述数据对象的属性及数据对象彼此间相互连接的关系。
       2. 数据对象：
          1. 数据对象是目标系统所需要的复合信息的表示，所谓复合信息是具有若干不同属性的信息。在ER图中用矩形表示数据对象。
          2. 在实际问题中，数据对象（实体）可以是外部实体、事物、角色、行为或事件、组织单位、地点或结构等。
       3. 属性：
          1. 属性定义数据对象的特征，如数据对象学生的学号、姓名、性别、专业等，课程的课程编号、课程名称、学分等。在ER图中用椭圆或圆角矩形表示属性，并用无向边将属性与相关的数据对象连接在一起。
       4. 关系：
          1. 不同数据对象的实例之间是有关联关系的，在ER图上用无向边表示。
          2. 在无向边的两端应标识出关联实例的数量，也称为关联的重数。
          3. 从关联重数的角度可以将关联分为３种。

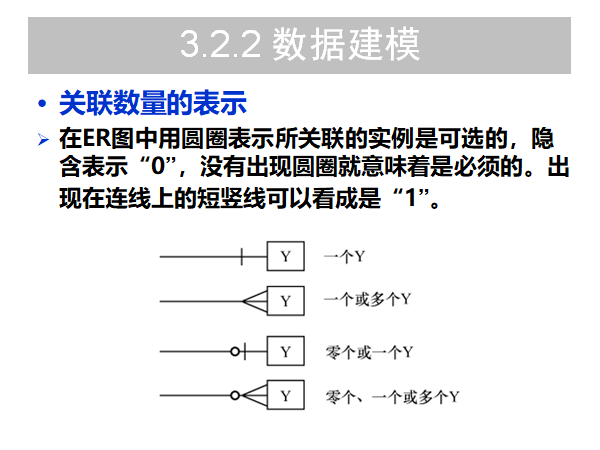
(1) 一对一（1:1）关联

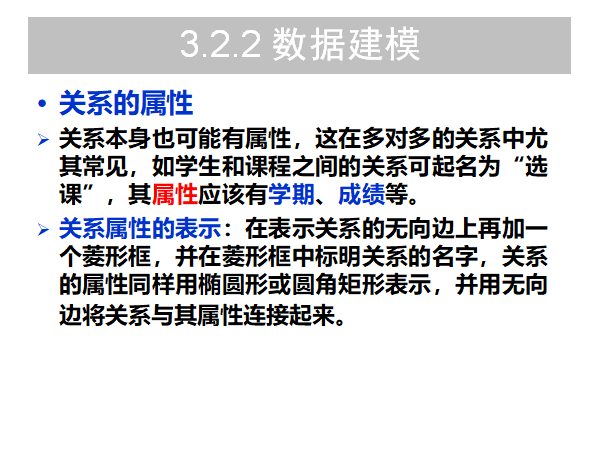
(2) 一对多（1:m）关联

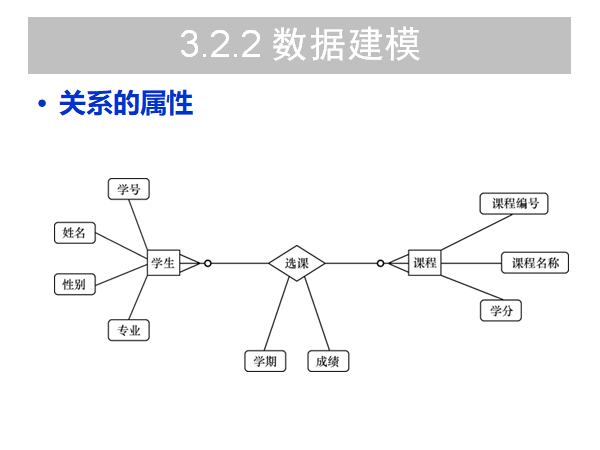
(3) 多对多（m:n）关联

* + - * 1. 实例关联还有“必须”和“可选”之分。

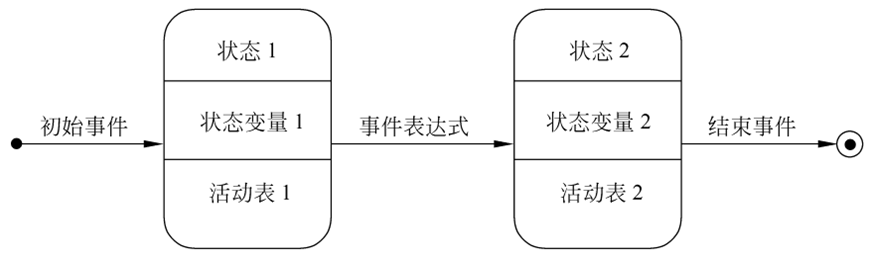




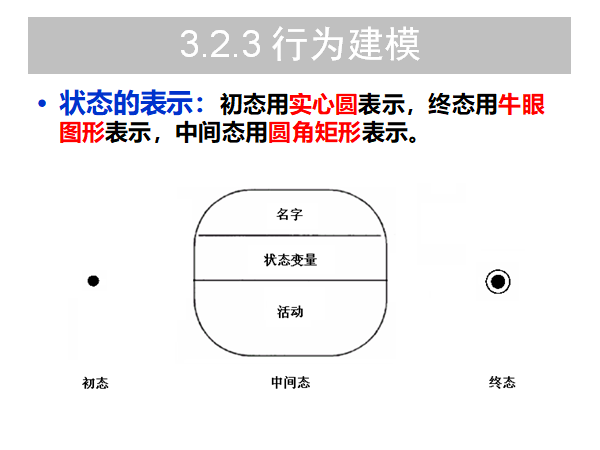




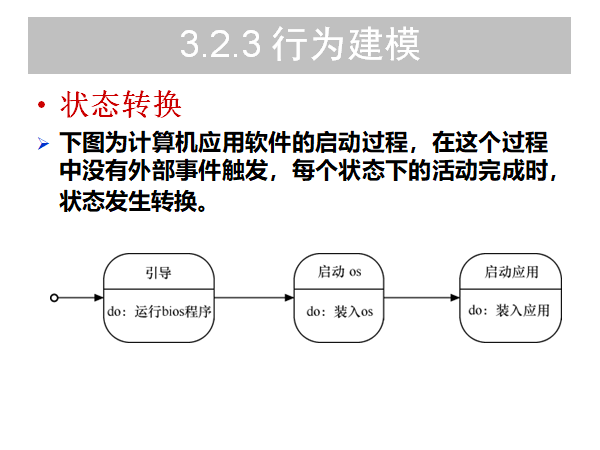
* + 1. 3.行为建模
       1. 概念：
          1. 状态转换图（简称状态图）通过描绘系统的状态及引起系统状态转换的事件，来表示系统的行为。状态图中使用的主要符号如图所示。



* + - 1. 状态：
         1. 状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式，状态规定了系统对事件的响应方式。
         2. 状态可能有：初态（初始状态）、终态（最终状态）和中间态。
         3. 在一张状态图中只能有一个初态，而终态则可以有多个，也可以没有。



* + - 1. 状态转换：
         1. 状态图中两个状态之间带箭头的连线称为状态转换。
         2. 状态的变迁通常是由事件触发的，在这种情况下应在表示状态转换的箭头线上标出触发转换的事件表达式。
         3. 如果在箭头线上未标明事件，则表示在源状态的内部活动执行完之后自动触发转换。



* + - 1. 事件：
         1. 事件是在某个特定时刻发生的事情，它是对引起系统做动作或从一个状态转换到另一个状态的外部事件的抽象。事件表达式的语法如下：
         2. 事件说明（守卫条件）/动作表达式
         3. (1) 事件说明的语法如下：事件名（参数表）
         4. (2) 守卫条件是一个布尔表达式。如果同时使用守卫条件和事件说明，则当且仅当事件发生且布尔表达式成立时，状态转换才发生。如果只有守卫条件没有事件说明，则只要守卫条件为真，状态转换就发生。
         5. (3) 动作表达式是一个过程表达式，当状态转换开始时执行该表达式。
    1. 4.数据字典
       1. 词条描述:
          1. 含义：

对于在数据流图中每一个被命名的图形元素均加以定义；

其内容包括图形元素的名字，图形元素的别名或编号，图形元素类别（如加工、数据流、数据文件、数据元素、数据源点或数据汇点等）、描述、定义、位置等。

* + - * 1. （1）数据流词条：数据流是数据结构在系统内传播的路径，数据流词条应包括以下几项内容。

①数据流名：要求与数据流图中该图形元素的名字一致。

②简述：简要介绍它产生的原因和结果。

③组成：数据流的数据结构。

④来源：数据流来自哪个加工或作为哪个数据源的外部实体。

⑤去向：数据流流向哪个加工或作为哪个数据汇点的外部实体。

⑥流通量：单位时间数据的流通量。

⑦峰值：流通量的极限值。

* + - * 1. （2）数据元素词条：数据流图中的每个数据结构都是由数据元素构成的，数据元素是数据处理中最小的、不可再分的单位，它直接反映事物的某一特征。

① 类型：数据元素分为数字型与文字型。数字型又分为离散值和连续值，文字的类型用编码类型和长度区分。

② 取值范围：离散值的取值或是枚举的（如3，17，21），或是介于上下界的一组数（如2..100）；连续值一般是有取值范围的实数集（如0.0..100.0）。对于文字型，文字的取值需加以定义。

③ 相关的数据元素及数据结构。

* + - * 1. （3）数据存储文件词条：数据存储文件是数据保存的地方。一个数据存储文件词条应有以下几项内容。

① 文件名：要求与数据流图中该图形元素的名字一致。

② 简述：简要介绍存放的是什么数据。

③ 组成：文件的数据结构。

④ 输入：从哪些加工获取数据。

⑤ 输出：由哪些加工使用数据。

⑥ 存取方式：分为顺序、直接、关键码等不同存取方式。

⑦ 存取频率：单位时间的存取次数。

* + - * 1. （4）加工词条：加工可以使用诸如判定表、判定树、结构化语言等形式表达，主要描述如下。

① 加工名：要求与数据流图中该图形元素的名字一致。

② 编号：用以反映该加工的层次和父子关系。

③ 简述：加工逻辑及功能简述。

④ 输入：加工的输入数据流。

⑤ 输出：加工的输出数据流。

⑥ 加工逻辑：简述加工程序和加工顺序。

* + - * 1. （5）数据源点及数据汇点词条：对于一个数据处理系统来说，数据源点和数据汇点应比较少。

① 名称：要求与数据流图中该外部实体的名字一致。

② 简述：简要描述是什么外部实体。

③ 有关数据流：该实体与系统交互时涉及哪些数据流。

④ 数目：该实体与系统交互的次数。

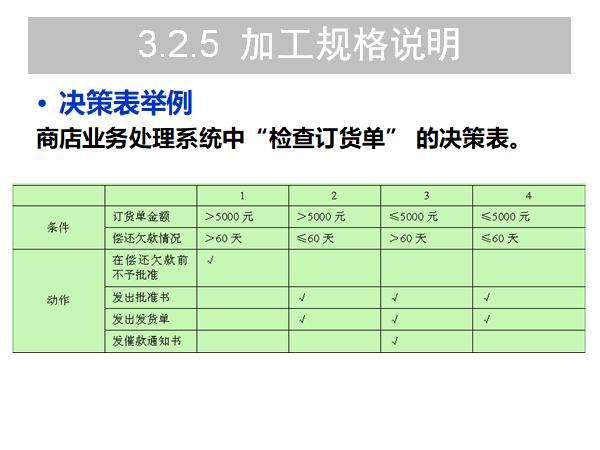
* + - 1. 数据结构描述：
         1. 在数据字典的编制中，分析员最常用的描述数据结构的方式有定义式、Warnier图等。
         2. （1）定义式。在数据流图中，数据流和数据文件都具有一定的数据结构，因此，必须以一种清晰、准确、无二义性的方式来描述数据结构。
         3. （2）Warnier图。Warnier图是表示数据结构的另一种图形工具，它用树形结构来描绘数据结构
      2. 决策表：
         1. 决策表由4个部分组成：

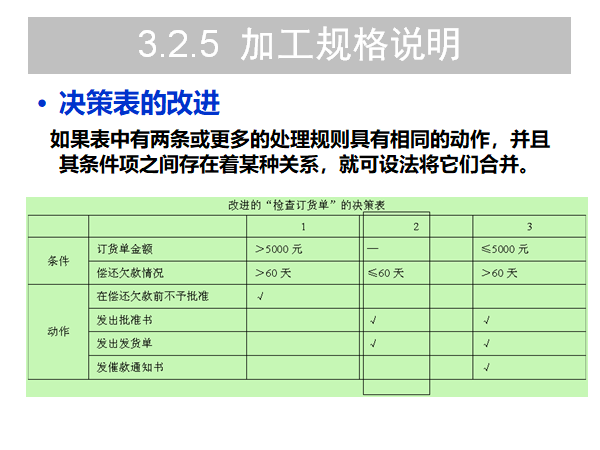
左上部分是条件茬，在此区域列出了各种可能的单个条件；

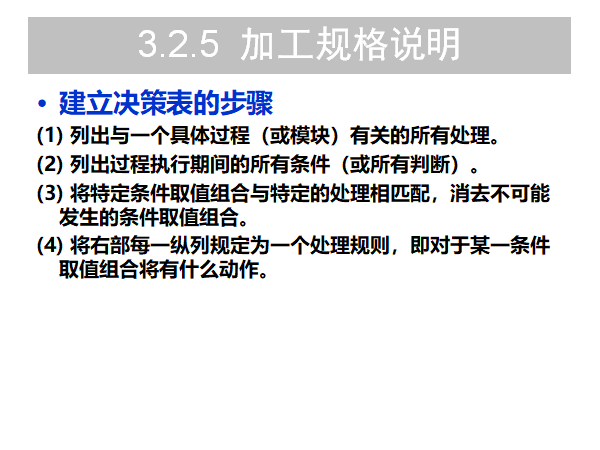
左下部分是动作茬，在此区域列出了可能采取的单个动作；

右上部分是条件项，在此区域列出了针对各种条件的每一组条件取值的组合；

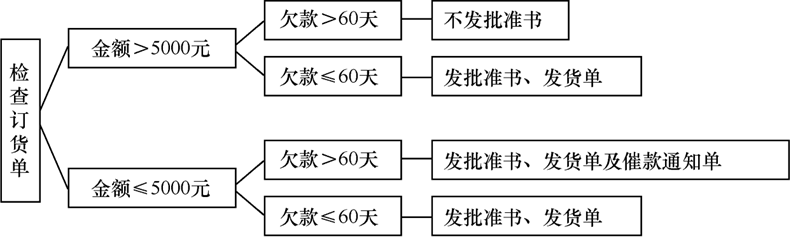
右下部分是动作项，这些动作项与条件项紧密相关，它指出了在条件项的各组取值的组合情况下应采取的动作。







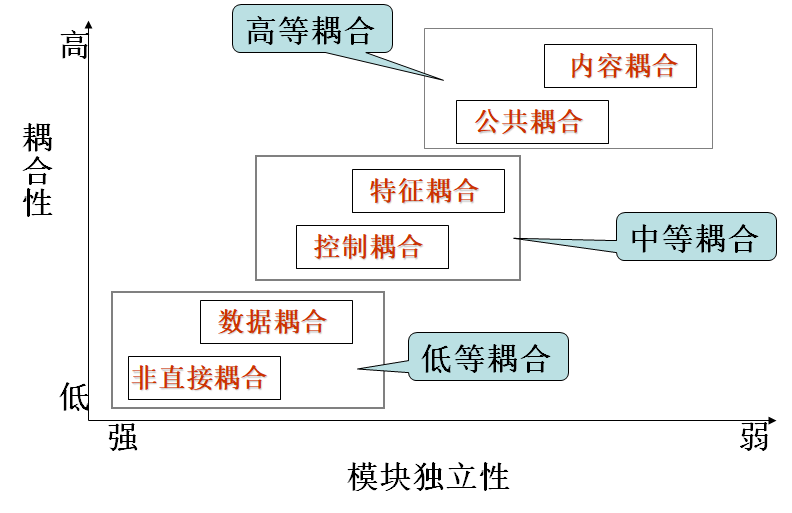
* + - 1. 决策树：
         1. 决策树（decision tree）也是用来表达加工逻辑的一种工具，有时侯它比决策表更直观。
         2. 检查订货单的决策树



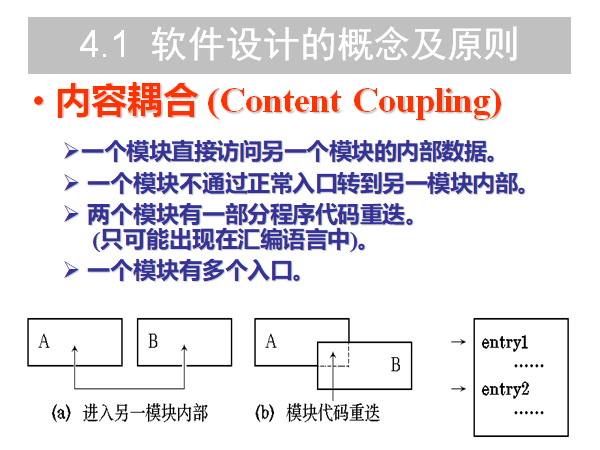
1. 四、结构化设计方法
   1. 1.软件设计的原则
      1. (1) 分而治之
         1. 分而治之是人们解决大型复杂问题时通常采用的策略。将大型复杂的问题分解为许多容易解决的小问题，原来的问题也就容易解决了。软件的体系结构设计、模块化设计都是分而治之策略的具体表现。
         2. 尽管模块分解可以简化要解决的问题，但模块分解并不是越小越好。当模块数目增加时，每个模块的规模将减小，开发单个模块的成本确实减少了；但是，随着模块数目增加，模块之间关系的复杂程度也会增加，设计模块间接口所需要的工作量也将增加，如图所示。



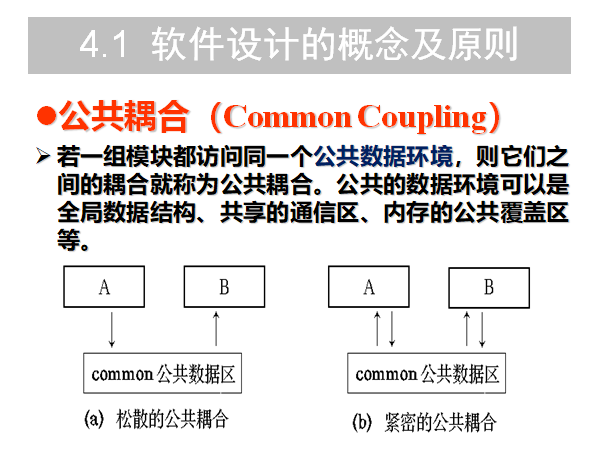
* + 1. (2) 模块独立性
       1. 定义: 是指软件系统中每个模块只涉及软件要求的具体的子功能, 而和软件系统中其它模块的接口是简单的。
       2. 有效的模块化使软件便于分工协作开发。独立的模块比较容易测试和维护。
    2. (3) 提高抽象层次
       1. 抽象是指忽视一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。
       2. 当我们进行软件设计时，设计开始时应尽量提高软件的抽象层次，按抽象级别从高到低进行软件设计。
    3. (4) 复用性设计
       1. 复用是指同一事物不做修改或稍加修改就可以多次重复使用。将复用的思想用于软件开发，称为软件复用。
       2. 我们将软件的重用部分称为软构件。
       3. 也就是说，在构造新的软件系统时不必从零做起，可以直接使用已有的软构件即可组装（或加以合理修改）成新的系统。
    4. (5) 灵活性设计
       1. 保证软件灵活性设计的关键是抽象。
       2. 面向对象系统中的类结构类似一座金字塔，越接近金字塔的顶端，抽象程度就越高。
       3. “抽象”的反义词是“具体”。理想情况下，一个系统的任何代码、逻缉、概念在这个系统中都应该是唯一的，也就是说不存在重复的代码。
       4. 在设计中引入灵活性的方法有：
          1. 降低耦合并提高内聚（易于提高替换能力）；
          2. 建立抽象（创建有多态操作的接口和父类）；
          3. 不要将代码写死（消除代码中的常数）；
          4. 抛出异常（由操作的调用者处理异常）；
          5. 使用并创建可复用的代码。
  1. 2.模块间的耦合与内聚
     1. 模块独立性的度量准则:
        1. 耦合:是模块之间的互相连接的紧密程度的度量。
        2. 内聚:是模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量。
        3. 模块独立性比较强的模块应是高内聚低耦合的模块。



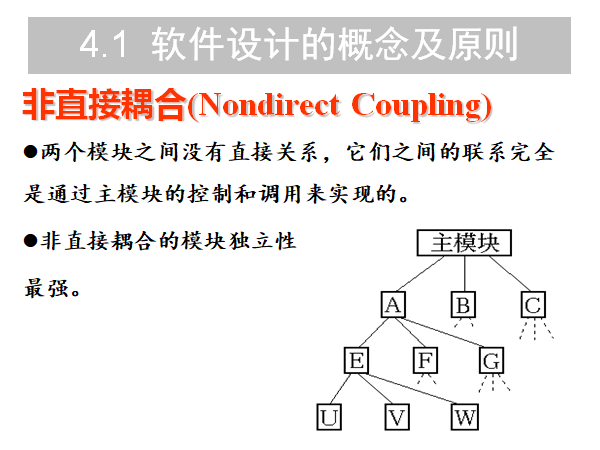
* + 1. 耦合
       1. (1)内容耦合



* + - 1. (2)公共耦合



* + - 1. (3)外部耦合
         1. 若一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递全局变量的信息，则称之为外部耦合
      2. (4)控制耦合
         1. 如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。
      3. (5)标记耦合
         1. 一组模块通过参数表传递记录信息，就是特征耦合。这个记录是某一数据结构的子结构，而不是简单变量。
      4. (6)数据耦合
         1. 一个模块访问另一个模块时，彼此之间是通过简单数据参数 (不是控制参数、公共数据结构或外部变量) 来交换输入、输出信息的。也是较理想的耦合。
      5. (7)非直接耦合



* + 1. 内聚
       1. (1)巧合内聚（偶然内聚）
          1. 当模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为偶然内聚模块，内聚程度最低。
          2. 缺点：

1）内容不易理解，很难描述其功能。

2）把完整的程序分割到多个模块中， 在程序运行时会频繁地互相调用

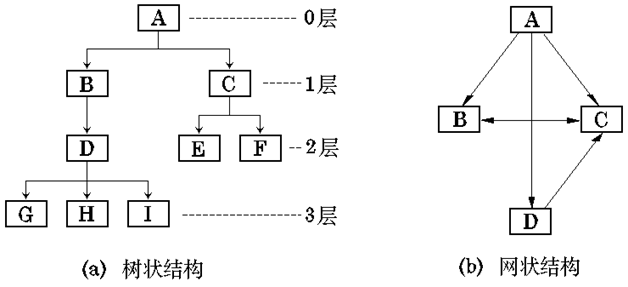
* + - 1. (2)逻辑内聚
         1. 把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪个功能。
         2. 缺点

1）不易修改，因包含多个功能

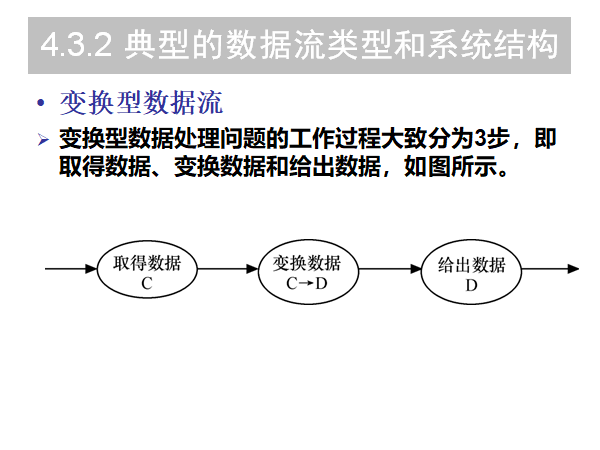
2）需传递控制参数——控制耦合

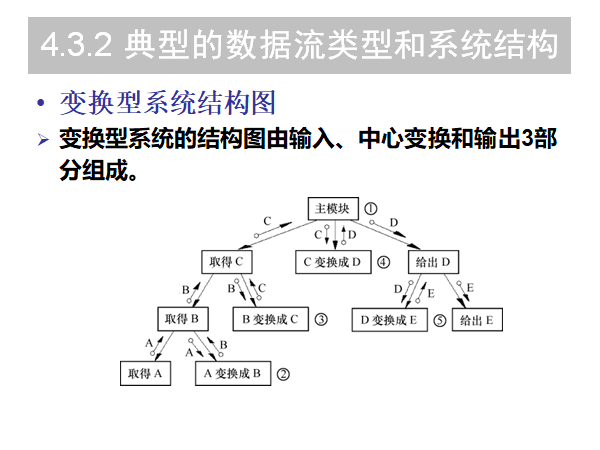
3）未用部分调入内存，影响效率

* + - 1. (3)时间内聚
         1. 时间内聚模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。
      2. (4)过程内聚
         1. 如果一个模块内的处理是相关的，而且必须以特定次序执行，则是过程内聚。
      3. (5)通信内聚
         1. 如果一个模块内各功能部分都使用了相同的输入数据，或者产生了相同的输出数据，则称之为通信内聚模块。
      4. (6)信息内聚
         1. 这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能都有唯一的入口点。
      5. (7)功能内聚
         1. 一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的。则称该模块为功能内聚模块。
         2. 优点：容易修改和维护
      6. (8)顺序内聚
         1. 一个模块中的处理元素和同一功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。
  1. 3.软件设计的阶段与任务
     1. 从工程管理的角度，可以将软件设计分为概要设计阶段和详细设计阶段。
     2. 从技术的角度，传统的结构化方法将软件设计划分为体系结构设计、数据设计、接口设计和过程设计4部分；
     3. 面向对象方法则将软件设计划分为体系结构设计、类设计/数据设计、接口设计和构件级设计4部分。
  2. 4.模块结构及表示
     1. 模块的结构
        1. 模块结构最普通的形式就是树状结构和网状结构，如图所示。

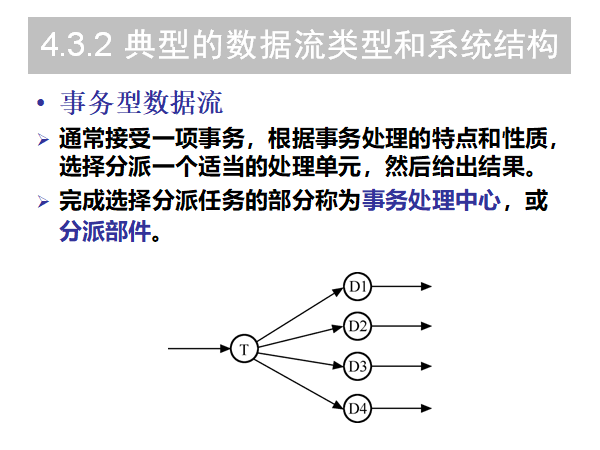


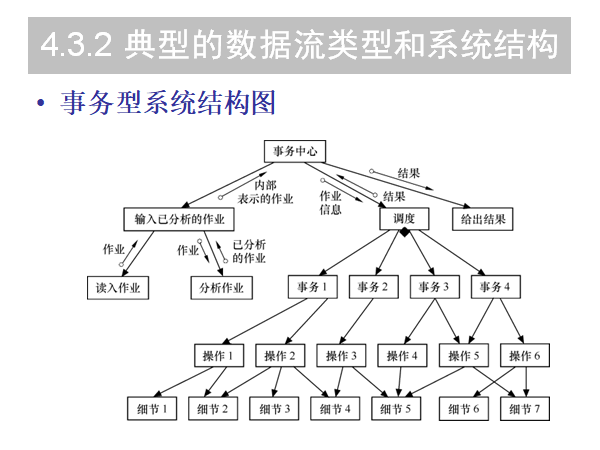
* + 1. 结构图（structure chart，SC）是精确表达模块结构的图形表示工具。
       1. (1) 模块的调用关系和接口：在结构图中，两个模块之间用单向箭头连接。
       2. (2) 模块间的信息传递：当一个模块调用另一个模块时，调用模块把数据或控制信息传送给被调用模块，以使被调用模块能够运行。
       3. (3) 条件调用和循环调用 ：当模块A有条件地调用另一个模块B时，在模块A的箭头尾部标以一个菱形符号；当一个模块A反复地调用模块C和模块D时，在调用箭头尾部则标以一个弧形符号。
       4. (4) 结构图的形态特征。在图中，上级模块调用下级模块，它们之间存在主从关系。
  1. 5.典型的数据流类型和系统结构
     1. (1)变换型数据流

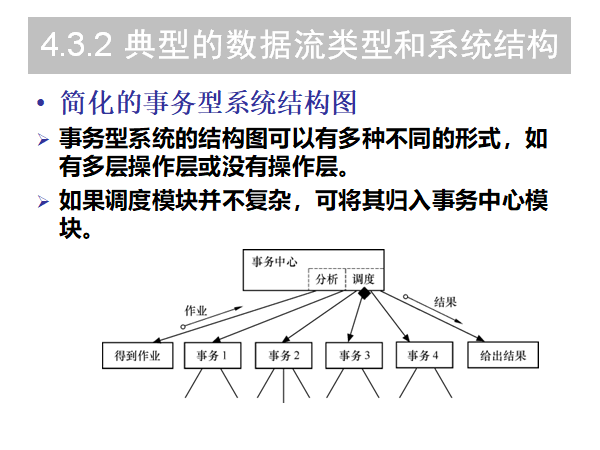




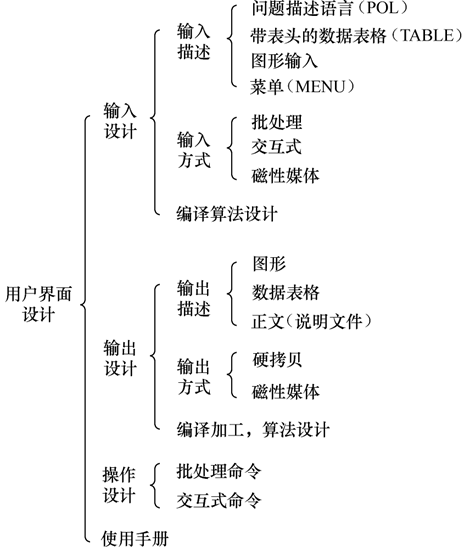
* + - 1. 变换分析方法的步骤
         1. (1) 重画数据流图。在需求分析阶段得到的数据流图侧重于描述系统如何加工数据，而重画数据流图的出发点是描述系统中的数据是如何流动的。
         2. (2)在数据流图上区分系统的逻辑输入、逻辑输出和中心变换部分。
         3. (3) 进行一级分解，设计系统模块结构的顶层和第一层。自顶向下设计的关键是找出系统树形结构图的根或顶层模块。
         4. (4) 进行二级分解，设计中、下层模块。
    1. (2)事务型数据流



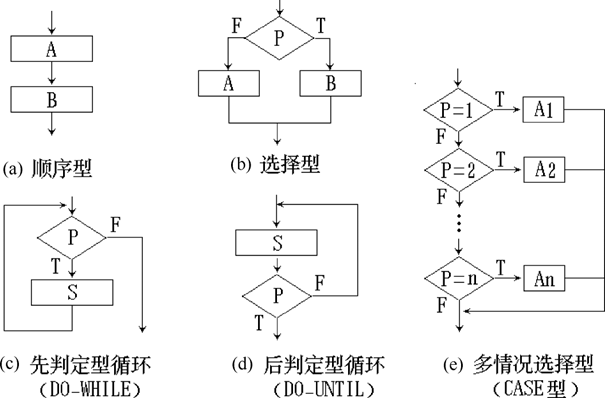




* + - 1. 事务分析方法的步骤
         1. (1) 识别事务源。利用数据流图和数据词典，从问题定义和需求分析的结果中，找出各种需要处理的事务。
         2. (2) 规定适当的事务型结构。在确定了该数据流图具有事务型特征之后，根据模块划分理论，建立适当的事务型结构。
         3. (3) 识别各种事务和它们定义的操作。
         4. (4) 注意利用公用模块。
         5. (5) 建立事务处理模块。对每一事务，或对联系密切的一组事务，建立一个事务处理模块。
         6. (6) 对事务处理模块规定它们全部的下层操作模块。
         7. (7) 对操作模块规定它们的全部细节模块。
    1. (3)软件模块结构的改进方法
       1. (1) 模块功能的完善化。(2) 消除重复功能，改善软件结构。(3) 模块的作用范围应在控制范围之内。(4) 尽可能减少高扇出结构，随着深度增大扇入(5) 避免或减少使用病态连接。(6) 模块的大小要适中。
  1. 6.接口设计
     1. (1)概述：
        1. 接口设计的依据是数据流图中的自动化系统边界。
        2. 接口设计主要包括3个方面：模块或软件构件间的接口设计；软件与其他软硬件系统之间的接口设计；软件与人（用户）之间的交互设计。
        3. 人机交互（用户）界面是人机交互的主要方式
     2. (2)人机交互界面
        1. 用户界面应具备的特性
           1. 可使用性：包括使用简单、界面一致、拥有HELP帮助功能、快速的系统响应和低的系统成本、具有容错能力等。
           2. 灵活性：考虑到用户的特点、能力和知识水平，应当使用户接口满足不同用户的要求。
           3. 可靠性：用户界面的可靠性是指无故障使用的间隔时间。用户界面应能保证用户正确、可靠地使用系统，保证有关程序和数据的安全性。
        2. 用户类型
           1. 外行型：以前从未使用过计算机系统的用户。
           2. 初学型：尽管对新的系统不熟悉，但对计算机还有一些使用经验的用户。
           3. 熟练型：对一个系统有相当多的经验，能够熟练操作的用户。
           4. 专家型：这一类用户了解系统内部的构造，有关于系统工作机制的专业知识，具有维护和修改基本系统的能力。专家型需要为他们提供能够修改和扩充系统能力的复杂界面。
        3. 界面设计类型



* + - * 1. (1) 使用的难易程度：对于没有经验的用户，该界面使用的难度有多大。
        2. (2) 学习的难易程度：学习该界面的命令和功能的难度有多大。
        3. (3) 操作速度：在完成一个指定操作时，该界面在操作步骤、击键和反应时间等方面效率有多高。
        4. (4) 复杂程度：该界面提供了什么功能、能否用新的方式组合这些功能以增强界面的功能。
        5. (5) 控制：人机交互时，是由计算机还是由人发起和控制对话。
        6. (6) 开发的难易程度：该界面设计是否有难度、开发工作量有多大。
      1. 设计详细的交互 :人机交互的设计有若干准则，包括以下内容：
         1. (1) 一致性。采用一致的术语、一致的步骤和一致的活动。
         2. (2) 操作步骤少。使击键或点击鼠标的次数减到最少，甚至要减少做某些事所需的下拉菜单的距离。
         3. (3) 不要“哑播放”。
         4. (4) 提供Undo功能。
         5. (5) 减少人脑的记忆负担。不应该要求人从一个窗口中记住某些信息，然后在另一个窗口中使用。
         6. (6) 提高学习效率。为高级特性提供联机帮助，以便用户在需要时容易找到。
    1. (3)过程设计
       1. 结构程序设计的主要原则
          1. (1)使用语言中的顺序、选择、重复等有限的基本控制结构表示程序逻辑。(2)选用的控制结构只准许有一个入口和一个出口。(3)程序语句组成容易识别的块（Block），每块只有一个入口和一个出口。(4)复杂结构应该用基本控制结构进行组合嵌套来实现。(5)语言中没有的控制结构，可用一段等价的程序段模拟， 但要求该程序段在整个系统中应前后一致。(6) 严格控制GOTO语句(7) 在程序设计过程中，尽量采用自顶向下(Top－Down)、逐步细化(Stepwise Refinement)的原则，由粗到细，一步步展开。
       2. 程序流程图的基本控制结构



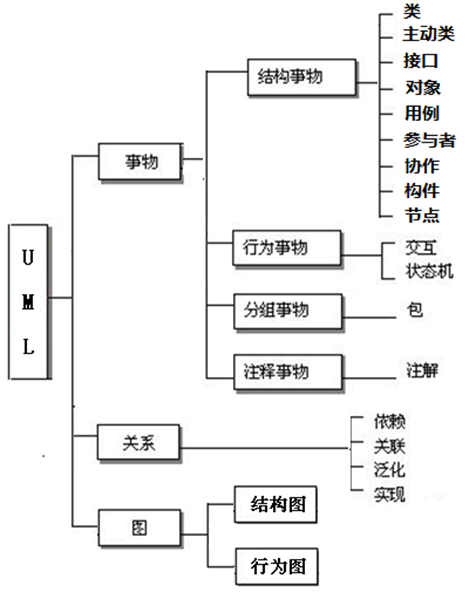
* + - * 1. (1) 顺序型：几个连续的加工步骤依次排列构成。
        2. (2) 选择型：由某个逻辑判断式的取值决定选择两个加工中的一个。
        3. (3) 先判定（while）型循环：在循环控制条件成立时，重复执行特定的加工。
        4. (4) 后判定（until）型循环：重复执行某些特定的加工，直至控制条件成立。
        5. (5) 多情况（case）型选择：列举多种加工情况，根据控制变量的取值，选择执行其一。

# 软件工程概论5-6

1. 五、面型对象方法和UML
   1. 1.面向对象的概念
      1. 面向对象＝对象＋类＋继承＋消息通信，如果一个系统是使用这样4个概念设计和实现的，则可认为这个系统是面向对象的。
      2. 对象
         1. 对象是包含现实世界物体特征的抽象实体，它反映了系统为之保存信息和（或）与它交互的能力。



* + - 1. 对象有两个层次的概念：
         1. (1) 现实生活中对象指的是客观世界的实体。可以是可见的有形对象，如人、学生、汽车、房屋等；也可以是抽象的逻辑对象，如银行帐号，生日。
         2. (2) 程序中对象就是一组变量和相关方法的集合，其中变量表明对象的状态，方法表明对象所具有的行为。
      2. 可以将程序中的对象分为5类：物理对象，角色，事件，交互，规格说明。
         1. (1)物理对象（Physical Objects）── 物理对象是最易识别的对象，通常可以在问题领域的描述中找到，它们的属性可以标识和测量。
         2. (2) 角色（Roles）── 一个实体的角色也可以抽象成一个单独的对象。角色对象的操作是由角色提供的技能。
         3. (3) 事件（Events）── 一个事件是某种活动的一次“出现”。
         4. (4) 交互（Interactions）── 交互表示了在两个对象之间的关系，这种类型的对象类似于在数据库设计时所涉及的“关系”实体。
    1. 类:
       1. 可以将现实生活中的对象经过抽象，映射为程序中的对象。对象在程序中是通过一种抽象数据类型来描述的，这种抽象数据类型称为类（Class）。
    2. 封装:
       1. 面向对象的封装特性与其抽象特性密切相关。封装是一种信息隐蔽技术，就是利用抽象数据类型将数据和基于数据的操作封装在一起。用户只能看到对象的封装界面信息，对象的内部细节对用户是隐蔽的。
    3. 继承
       1. 继承是一种联结类的层次模型，为类的重用提供了方便，它提供了明确表述不同类之间共性的方法。
       2. 我们将公共类称为超类(superclass)、父类（father class）、祖先（ancestor）或基类（base class），而从其继承的类称为子类(subclasses)、后代（deslendane）或导出类（derived class）。
    4. 面向对象软件开发方法的特征
       1. 方法的唯一性即方法是对软件开发过程所有阶段进行综合考虑而得到的。
       2. 从生存期的一个阶段到下一个阶段的高度连续性，即生存期后一阶段的成果只是在前一阶段成果的补充和修改。
       3. 将面向对象分析(OOA)、面向对象设计(OOD)和面向对象程序设计(OOP)集成到生存期的相应阶段。
  1. 2.UML
     1. UML的基本模型

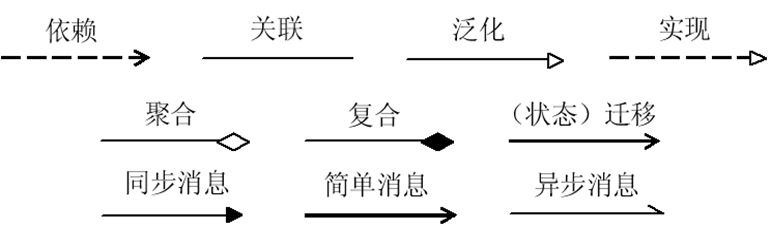


* + 1. UML的事物
       1. 事物是对模型中最具代表性成分的抽象，在UML中，可以分为结构事物、行为事物、分组事物和注释事物4类。
       2. 结构事物:
          1. 结构事物是UML模型的静态部分，主要用来描述概念的或物理的元素，包括类、主动类、接口、对象、用例、参与者、协作、构件和节点等。
       3. 行为事物:
          1. 行为事物是UML模型的动态部分，包括两类：

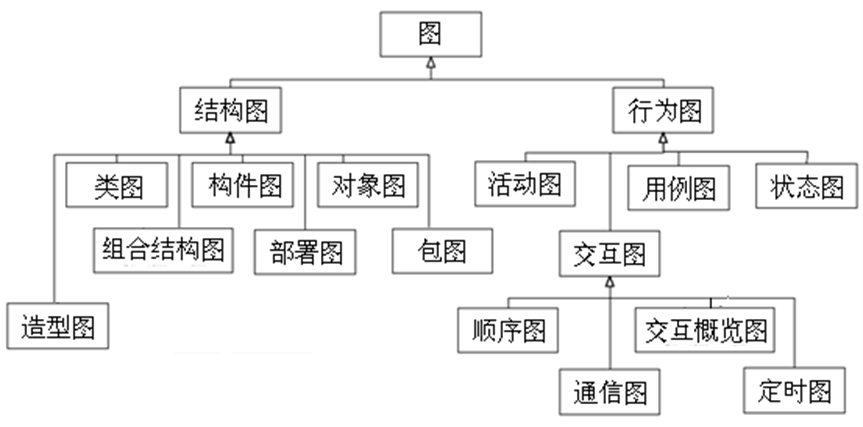
（1）交互（interaction）── 交互由在特定的上下文环境中共同完成一定任务的一组对象之间传递的消息组成。如图所示。交互涉及的元素包括消息、动作序列（由一个消息所引起的行为）和链（对象间的连接）。

（2）状态机（state machine）── 描述了一个对象或一个交互在生存周期内响应事件所经历的状态序列，单个类或者一组类之间协作的行为都可以用状态机来描述。 状态机涉及到状态、变迁和活动，其中状态用圆角矩形来表示。

* + - 1. 分组事物
         1. 分组事物是UML模型的组织部分。它的作用是为了降低模型复杂性。UML中的分组事物是包（package）。包是把模型元素组织成组的机制，结构事物、行为事物甚至其他分组事物都可以放进包内。
      2. 注释事物
         1. 注释事物是UML模型的解释部分，它们用来描述和标注模型的任何元素。通常可以用注释修饰带有约束或者解释的图。
    1. UML的关系
       1. 画图



* + - 1. 依赖关系
         1. 依赖(Dependency)是两个事物之间的语义关系，其中一个事物发生变化会影响到另一个事物的语 义，它用一个虚线箭头表示。
         2. 虚线箭头的方向从源事物指向目标事物，表示源事物依赖于目标事物。
      2. 关联关系
         1. 关联(association)是一种结构关系，它描述了两个或多个类的实例之间的连接关系，是一种特殊的依赖。
         2. 关联分为普通关联（**二元关联和多元关联**）、限定关联、关联类，以及聚合与复合。
      3. 泛化关系
         1. 泛化(generalization)关系就是一般类和特殊类之间的继承关系。
         2. 在UML中，一般类亦称泛化类，特殊类亦称特化类。
         3. 泛化针对类型而不针对实例，因为一个类可以继承另一个类，但一个对象不能继承另一个对象。
         4. 泛化可进一步划分成普通泛化和受限泛化（交叠、不相交、完全和不完全 ）两类。、
      4. 实现关系
         1. 实现(implement)是泛化关系和依赖关系的结合，也是类之间的语义关系，通常在以下两种情况出现实现关系：
         2. (1)接口和实现它们的类或构件之间；
         3. (2)用例和实现它们的协作之间。
    1. UML的图
       1. 图



* + - 1. 用例图
         1. 在UML中，一个用例模型由若干个用例图来描述，用例图的主要元素是用例和执行者。用例图是包括执行者、由系统边界（一个矩形）封闭的一组用例，执行者和用例之间的关联、用例间关系以及执行者的泛化的图。
         2. 用例之间可以有泛化、扩展、使用（包含）三种关系。

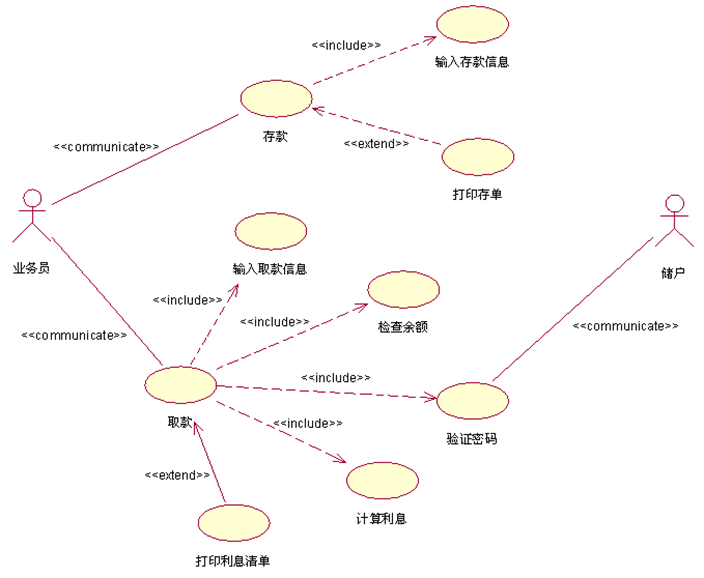
(1) 泛化关系：用例泛化是指一个用例可以被特别列举为一个或多个子用例。

(2) 扩展关系：向一个用例中加入一些新的动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者通过继承前者的一些行为得来，通常把后者称为扩展用例。

(3) 使用（包含）关系：当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。当有一大块相似的动作存在于几个用例，又不想重复描述该动作，将重复的部分分离为一个用例，两用例间关系称为使用关系。

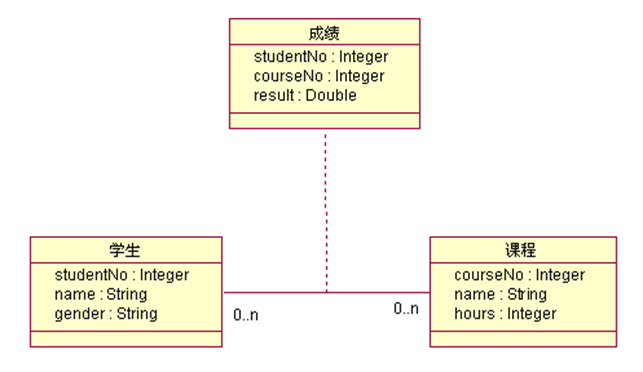
* + - * 1. 例子

银行储蓄系统的用例图



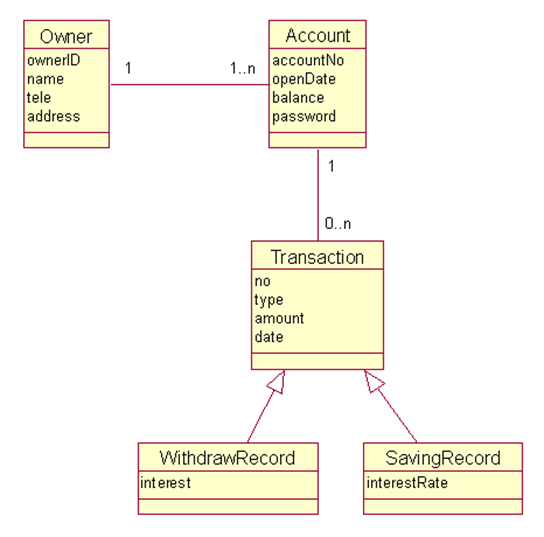
* + - 1. 类图
         1. 类图描述类和类与类之间的静态关系，它是从静态角度表示系统的，因此类图属于一种静态模型。类图是构建其他图的基础，没有类图就没有状态图、协作图等其他图，也就无法表示系统其他方面的特性。
         2. 类图显示了类（及其接口）、类的内部结构以及与其他类的联系。联系是指类元之间的联系，在类的建模中可以使用关联、聚合和泛化（继承）关系。
         3. 关联类

关联类是指表示其他类之间关联关系的类。当一个关联具有自己的属性并需要存储它们时，就需要用关联类建模。关联类用虚线连接在两个类之间的联系上

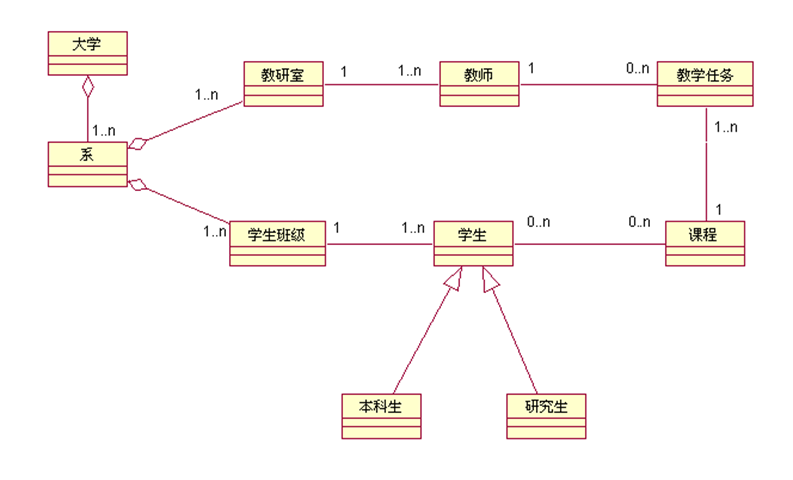


* + - * 1. 例子

银行储蓄系统的核心类图

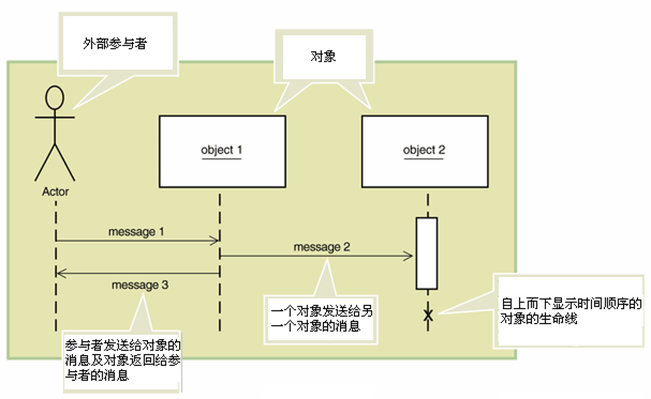


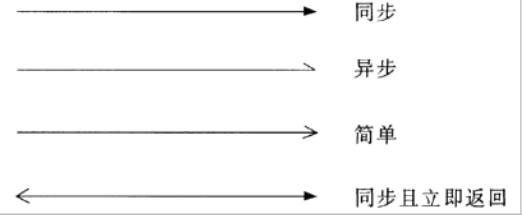
教学管理系统的类图



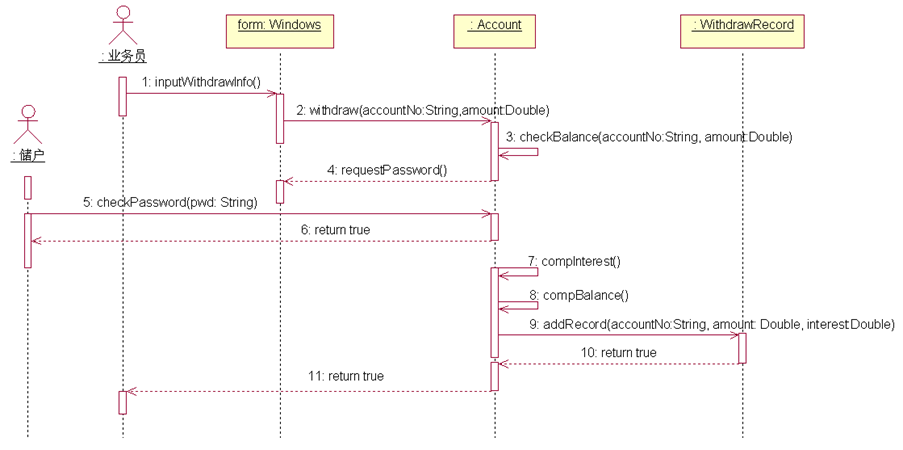
* + - 1. 交互图
         1. UML中有两种类型的交互图：顺序图和协作图。
         2. 顺序图

顺序图描述对象之间的动态交互关系，着重表现对象间消息传递的时间顺序。顺序图中的符号如下:



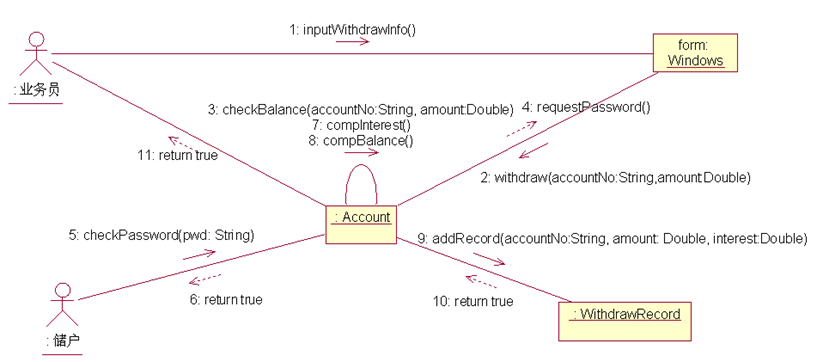


取款用例的顺序图

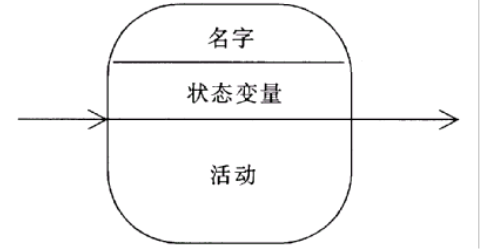


* + - * 1. 通信图

通信图是顺序图的一种变化形式，用于描述相互协作的对象间的交互关系和链接关系。

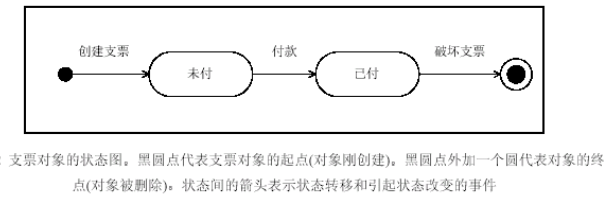


* + - 1. 状态图
         1. 状态图描述一个特定对象的所有可能的状态以及引起状态转换的事件。大多数面向对象技术都用状态图表示单个对象在其生命期中的行为。一个状态图包括一系列状态、事件以及状态之间的转移。
         2. 所有对象都具有状态，状态是对象执行了一系列活动的结果。当某个事件发生后，对象的状态将发生变化。在状态图中定义的状态可能有：初态(初始状态)、终态(最终状态)、中间状态和复合状态。
         3. 在一张状态图中只能有一个初态，而终态则可以有多个。
         4. 中间状态用圆角矩形表示，可能包含三个部分，第一部分为状态的名称；第二部分为状态变量的名字和值，这部分是可选的；第三部分是活动表，这部分也是可选的。

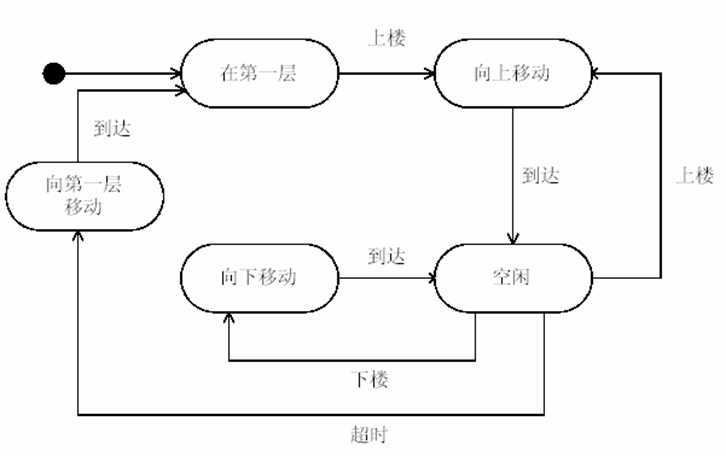


* + - * 1. 例子

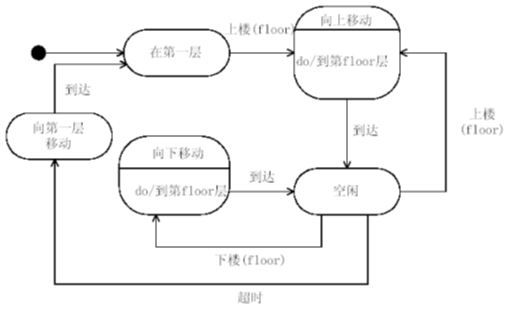
支票对象的状态图



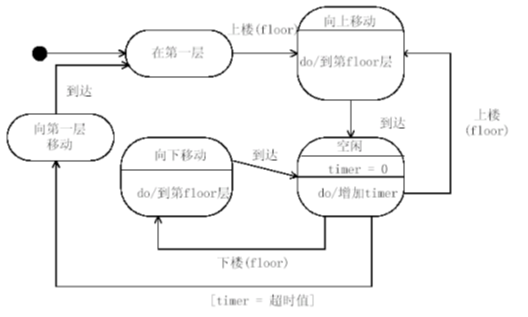
电梯的状态图(本状态图没有终点)



带有事件说明的状态转换

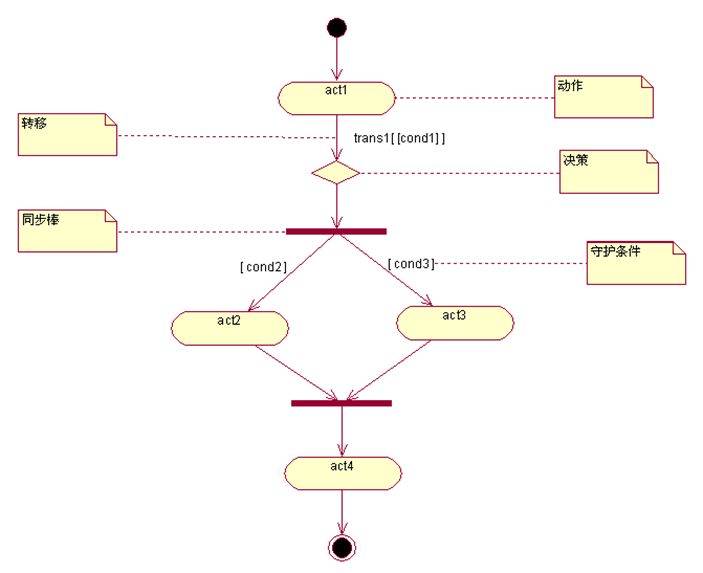


加上属性的状态转换

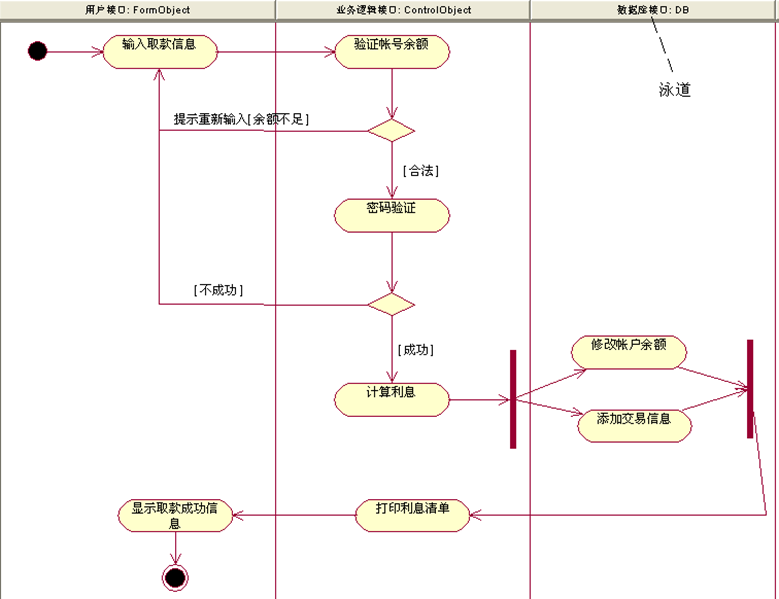


* + - 1. 活动图
         1. 活动图用来捕捉用例的活动，使用框图的方式显示动作及其结果。活动图是一个流图，描述了从活动到活动的流。它是另一种描述交互的方式，它描述采取何种动作，动作的结果是什么(动作状态改变)，何时发生(动作序列)，以及在何处发生(泳道)。
         2. 例子

活动图中的符号



取款用例的活动图



* + - 1. 构件图
         1. 构件图描述软件构件及构件之间的依赖关系，显示代码的静态结构。
         2. 构件是逻辑架构中定义的概念和功能(例如，类、对象及它们之间的关系)在物理架构中的实现。典型情况下，构件是开发环境中的实现文件。
         3. 软件构件可以是下述的任何一种构件。

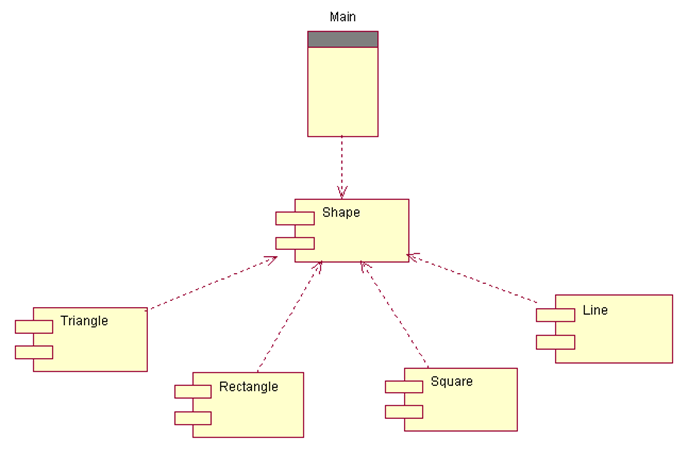
源构件：源构件仅在编译时才有意义。典型情况下，它是实现一个或多个类的源代码文件。

二进制构件：典型情况下，二进制构件是对象代码，它是源构件的编译结果。

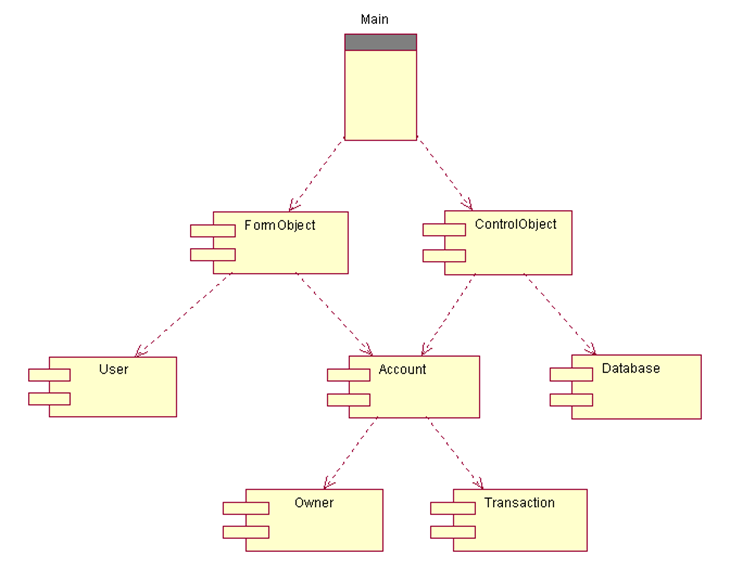
可执行构件：可执行构件是一个可执行的程序文件，它是链接所有二进制构件所得到的结果。一个可执行构件代表在处理器(计算机)上运行的可执行单元。

* + - * 1. 例子

画图系统的构件图



银行储蓄系统的构件图

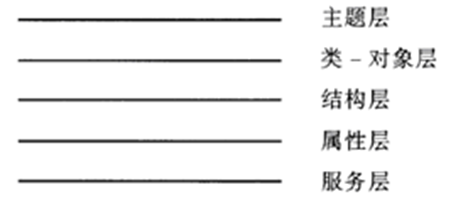


* + - 1. 部署图
         1. 部署图描述处理器、设备和连接，它显示系统硬件的物理拓扑结构及在此结构上执行的软件。
         2. 部署图可以显示计算节点的拓扑结构和通信路径、节点上运行的软件以及软件包含的逻辑单元。
         3. 例子

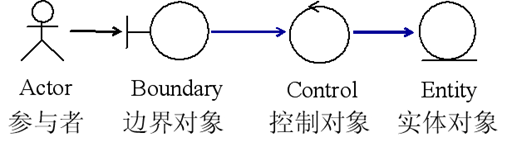
典型的部署图



1. 六、面向对象分析
   1. 1.面向对象分析概述
      1. 在结构化的分析方法中，主要采用数据流图对功能进行建模，建模时首先要建立顶层数据流图来确定系统在环境中的位置及系统的边界。在进行面向对象系统分析时，也同样需要建立环境图来确定系统边界。
      2. 面向对象分析的3种模型
         1. 用例模型：用例和场景表示的功能模型；
         2. 对象模型：用类和对象表示的静态模型；
         3. 交互模型：由状态图和顺序图表示的动态模型。
   2. 2.建立用例模型
      1. 1. 确定业务参与者
      2. 2. 确定业务需求用例
      3. 3. 创建用例图
   3. 3.建立对象模型
      1. 复杂问题(大型系统)的对象模型应该由下述5个层次组成：主题层(也称为范畴层)、类-对象层、结构层、属性层和服务层，如图所示。



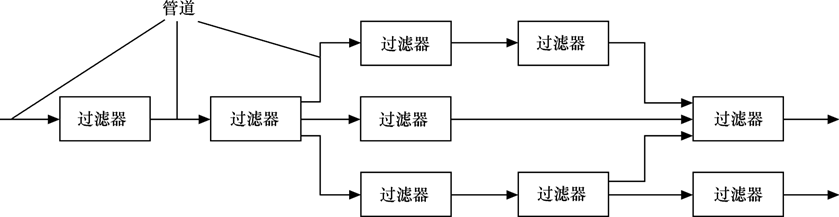
* + 1. 确定类与对象
       1. 1. 找出候选的类与对象
          1. 类与对象是对问题域中有意义的事物的抽象，它们既可能是可见的物理实体，也可能是抽象的概念。我们可以将客观事物分为以下五类：
          2. 可感知的物理实体，如教学楼、教室等。
          3. 人或组织的角色，如教师、计算机系等。
          4. 应该记忆的事件，如演出、交通事故等。
          5. 两个或多个对象的相互作用，通常带有交易或接触的性质，如购买、教学等。
          6. 需要说明的概念，如保险法、政策等。
       2. 2. 筛选出正确的类与对象
       3. 3.区分实体类、边界类和控制类
          1. 实体类表示系统将跟踪的持久信息；边界类表示参与者与系统之间的交互；控制类负责用例的实现。其图形表示如图所示



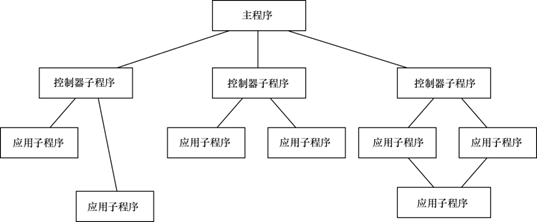
* + 1. 确定结构
       1. 1. 确定泛化（继承）关系
       2. 2. 确定关联关系
    2. 确定属性
    3. 确定服务
       1. 1) 简单的服务。即每一个对象都应具备的服务，这些服务包括：建立和初始化一个新对象，建立或切断对象之间的关联，存取对象的属性值，释放或删除一个对象。这些服务在分析时是隐含的，在图中不标出，但实现类和对象时有定义。
       2. (2) 复杂的服务。它分为两种：
          1. 计算服务：利用对象的属性值计算，以实现某种功能；
          2. 监控服务：处理对外部系统的输入∕输出，外部设备的控制和数据的存取。
  1. 4.建立动态模型
     1. 在UML中动态模型的描述工具有顺序图、通信图和状态图。

# 软件工程概论7

1. 七、软件体系结构与设计模式
   1. 1.软件体系结构的基本概念
      1. 虽然软件体系结构的定义在变化，但其意图是清晰的。体系结构设计是一系列决策和基本原理的集合，这些决策的目标在于开发高效的软件体系结构。
      2. （1）模式
         1. 每个模式都描述了一个在我们的环境中不断出现的问题及该问题解决方案的核心。在软件系统中，可以将模式划分为以下3类。
            1. （1）体系结构模式（architectural pattern）：表达了软件系统的基本结构组织形式或者结构方案，包含了一组预定义的子系统，规定了这些子系统的责任，同时还提供了用于组织和管理这些子系统的规则和向导。典型的体系结构模式如OSI参考模型。
            2. （2）设计模式（design pattern）：为软件系统的子系统、构件或者构件之间的关系提供一个精炼之后的解决方案，描述了在特定环境下，用于解决通用软件设计问题的构件以及这些构件相互通信时的各种结构。有代表性的设计模式是Erich Gamma及其同事提出的23种设计模式。
            3. （3）惯用法（idiom）：是与编程语言相关的低级模式，描述如何实现构件的某些功能，或者利用编程语言的特性来实现构件内部要素之间的通信功能。
      3. （2）风格
         1. 每种风格描述一种系统范畴，该范畴包括：
            1. （1）一组构件（如数据库、计算模块）完成系统需要的某 种功能；
            2. （2）一组连接件，它们能使构件间实现“通信”、“合作”和 “协调”；
            3. （3）约束，定义构件如何集成为一个系统；
            4. （4）语义模型，它能使设计者通过分析系统的构成成分的 性质来理解系统的整体性质。
      4. （3）框架
         1. 随着应用的发展和完善，某些带有整体性的应用模式被逐渐固定下来，形成特定的框架，包括基本构成元素和关系。框架是特定应用领域问题的体系结构模式，框架定义了基本构成单元和关系后，开发者就可以集中精力解决业务逻辑问题。
         2. 在组织形式上，框架是一个待实例化的完整系统，定义了软件系统的元素和关系，创建了基本的模块，定义了涉及功能更改和扩充的插件位置。典型的框架例子有MFC框架和Struts框架。
      5. （4）体系结构的重要作用
         1. （1）体系结构的表示有助于风险承担者（项目共利益者）进行交流。
         2. （2）体系结构突出了早期设计决策。
         3. （3）软件体系结构是可传递和可复用的模型。
   2. 2.典型的软件体系结构风格
      1. (1)数据流风格
         1. 当输入数据经过一系列的计算和操作构件的变换形成输出数据时，可以应用这种体系结构。管道/过滤器、批处理序列都属于数据流风格。
         2. 管道/过滤器结构如下图所示。



* + - 1. 从上图可看出，管道/过滤器结构拥有一组被称为过滤器的构件，这些构件通过管道连接，管道将数据从一个构件传送到下一个构件。每个过滤器独立于其上游和下游的构件而工作，过滤器的设计要针对某种形式的数据输入，并且产生某种特定形式的数据输出。
      2. 如果数据流退化成为单线的变换，则称为批处理序列。这种结构接收一批数据，然后应用一系列连续的构件（过滤器）变换它。
      3. 管道/过滤器风格具有以下优点：
         1. （1）使得软构件具有良好的隐蔽性和高内聚、低耦合的特点。
         2. （2）允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成。
         3. （3）支持软件复用。只要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来。
         4. （4）系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉。
         5. （5）允许对一些如吞吐量、死锁等属性的分析。
         6. （6）支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其他任务并行执行。
      4. 管道/过滤器风格主要缺点如下：
         1. （1）通常导致进程成为批处理的结构。这是因为虽然过滤器可增量式地处理数据，但它们是独立的，所以设计者必须将每个过滤器看成一个完整的从输入到输出的转换。
         2. （2）不适合处理交互的应用。当需要增量地显示改变时，这个问题尤为严重。
         3. （3）因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性。
    1. (2)调用—返回风格
       1. 1．主程序/子程序体系结构
          1. 这种传统的程序结构将功能分解为一个控制层次，其中“主”程序调用一组程序构件，这些程序构件又去调用别的程序构件，如下图所示。这种结构总体上为树状结构，可以在底层存在公共模块。



* + - * 1. 主程序/子程序体系结构的优点如下:

（1）可以使用自顶向下，逐步分解的方法得到体系结构图，典型的拓扑结构为树状结构。基于定义—使用关系对子程序进行分解，使用过程调用作为程序之间的交互机制。

（2）采用程序设计语言支持的单线程控制。

* + - * 1. 其主要缺点如下:

（1）子程序的正确性难于判断。需要运用层次推理来判断子程序的正确性，因为子程序的正确性取决于它调用的子程序的正确性。

（2）子系统的结构不清晰。通常可以将多个子程序合成为模块。

* + - 1. 2．面向对象风格
         1. 系统的构件封装了数据和必须应用到该数据上的操作，构件间通过消息传递进行通信与合作。与主程序/子程序的体系结构相比，面向对象风格中的对象交互会复杂一些。面向对象风格与网络应用的需求在分布性、自治性、协作性、演化性等方面具有内在的一致性。
         2. 面向对象风格具有以下优点:

（1）因为对象对其他对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其他对象。

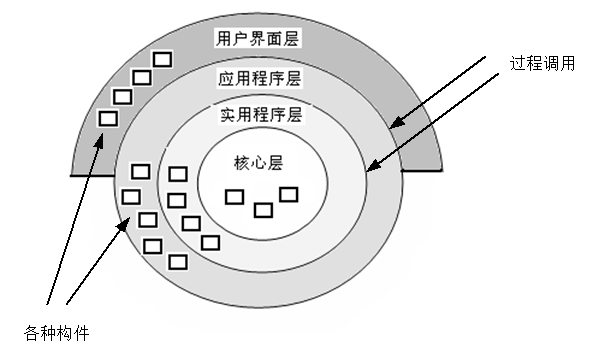
（2）设计者可将一些数据存取操作的问题分解成一些交互的代理程序的集合。

* + - * 1. 其缺点如下:

（1）为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象。

（2）必须修改所有显式调用它的其他对象，并消除由此带来的一些副作用。例如，如果A使用了对象B，C也使用了对象B，那么，C对B的使用所造成的对A的影响可能是料想不到的。

* + - 1. 3．层次结构
         1. 层次结构的基本结构如下图所示。在这种体系结构中，整个系统被组织成一个分层结构，每一层为上层提供服务，并作为下一层的客户。这种风格支持基于可增加抽象层的设计。允许将复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件复用提供了强大的支持。



* + - * 1. 层次结构具有以下优点:

（1）支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解。

（2）支持功能增强，因为每一层至多和相邻的上下层交互，因此，功能的改变最多影响相邻的内外层。

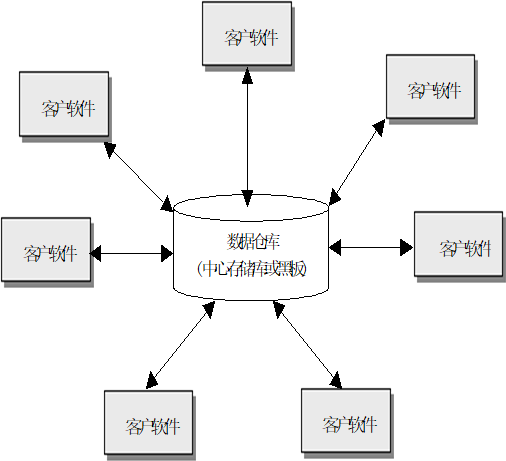
（3）支持复用。只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，从而允许各种不同的实现方法。

* + - * 1. 其缺点如下:

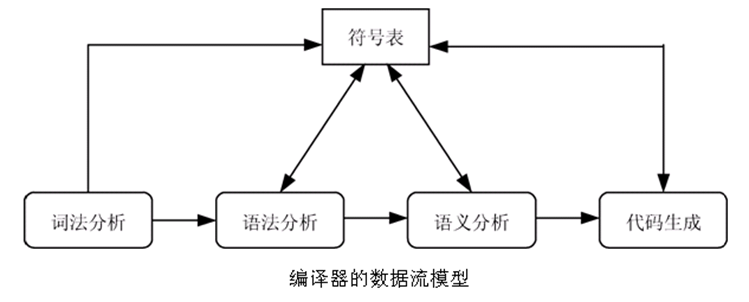
（1）并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来。

（2）很难找到一个合适的、正确的层次抽象方法。

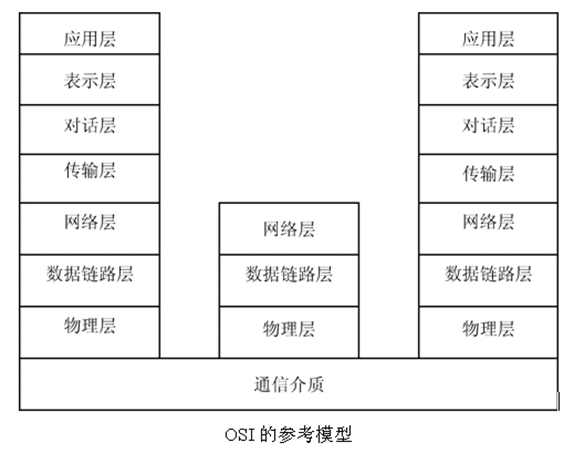
* + 1. (3)仓库风格
       1. 数据库系统、超文本系统和黑板系统都属于仓库风格。在这种风格中，数据仓库（如文件或数据库）位于这种体系结构的中心，其他构件会经常访问该数据仓库，并对仓库中的数据进行增加、修改或删除操作。右图为一个典型的仓库风格的体系结构。



* + - 1. 黑板系统的优点：
         1. 对可更改性和可维护性的支持
         2. 可复用的知识源
         3. 支持容错性和健壮性
      2. 黑板系统的缺点：
         1. 测试困难
         2. 不能保证有好的解决方案
         3. 难以建立好的控制策略
         4. 低效
         5. 昂贵的开发工作
         6. 缺少并行机制的支持
  1. 3.特定领域的软件体系结构
     1. 特定的应用还需要特定的体系结构模型。这些体系结构模型称为领域相关的体系结构。有两种领域相关的体系结构模型：类属模型和参考模型。
     2. 1.类属模型
        1. 类属模型是从许多实际系统中抽象出来的一般模型，它封装了这些系统的主要特征。
        2. 类属模型的一个最著名的例子是编译器模型，由这个模型已开发出了数以千计的编译器。



* + 1. 2.参考模型
       1. 参考模型源于对应用领域的研究，它描述了一个理想化的包含了系统应具有的所有特征的软件体系结构。它是更抽象且是描述一大类系统的模型，并且也是对设计者有关某类系统的一般结构的指导。
       2. 参考模型的典型例子是开放式系统互联（OSI）参考模型。



* + 1. 以上两种不同类型的模型之间并不存在严格的区别，也可以将类属模型视为参考模型。区别之一是类属模型可以直接在设计中复用，而参考模型一般是用于领域概念间的交流和对可能的体系结构做出比较。另外，类属模型通常是经过“自下而上”地对已有系统的抽象，而参考模型是“由上到下”地产生的。
  1. 4.分布式系统结构
     1. 优点
        1. (1) 资源共享。分布式系统允许硬件、软件等资源共享使用。
        2. (2) 经济性。
        3. (3) 性能与可扩展性。
        4. (4) 固有分布性。
        5. (5) 健壮性。
     2. 多处理器体系结构
        1. 分布式系统的一个最简单的模型是多处理器系统，系统由许多进程组成，这些进程可以在不同的处理器上并行运行，可以极大地提高系统的性能。
     3. 客户/服务器体系结构
        1. 客户机/服务器（client/server，C/S）体系结构是基于资源不对等，且为实现共享而提出来的，由服务器、客户机和网络三部分组成。
        2. 在C/S体系结构中，客户机可以通过远程调用来获取服务器提供的服务，因此，客户机必须知道可用的服务器的名字及它们所提供的服务，而服务器不需要知道客户机的身份，也不需要知道有多少台服务器在运行。
        3. 两层C/S体系结构可以有两种形态：
           1. （1）瘦客户机模型。在瘦客户机模型中，数据管理部分和应用逻辑都在服务器上执行，客户机只负责表示部分。
           2. 瘦客户机模型的主要缺点：它将繁重的处理负荷都放在了服务器和网络上，服务器负责所有的计算，这将增加客户机和服务器之间的网络流量。目前个人计算机所具有的处理能力在瘦客户机模型中用不上。
           3. （2）胖客户机模型。在这种模型中，服务器只负责对数据的管理。客户机上的软件实现应用逻辑和与系统用户的交互。胖客户机模型能够利用客户机的处理能力，比瘦客户机模型在分布处理上更有效。但另一方面，随着企业规模的日益扩大，软件的复杂程度不断提高，胖客户机模型逐渐暴露出了以下缺点：

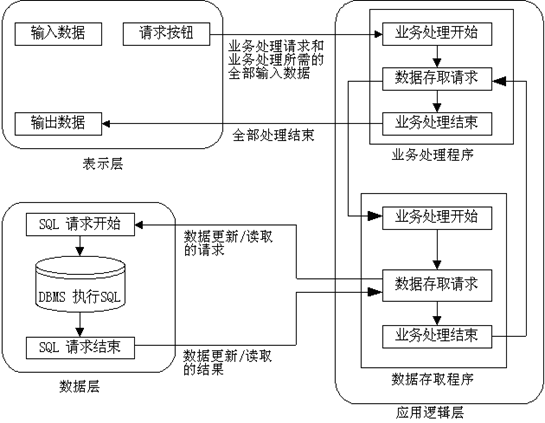
开发成本较高。

用户界面风格不一，使用繁杂，不利于推广使用。

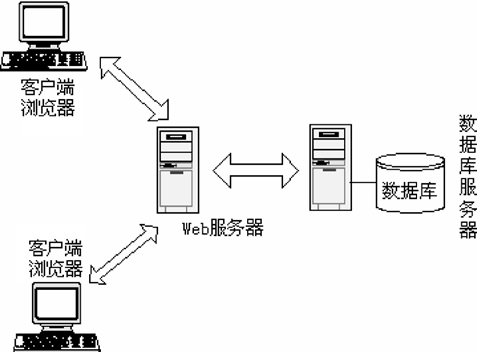
软件移植困难。

软件维护和升级困难。

* + - 1. 三层C/S体系结构将整个系统分成表示层、应用逻辑层和数据层三个部分，其数据处理流程如下图所示。



* + - * 1. （1）表示层：表示层是应用系统的用户界面部分，担负着用户与应用程序之间的对话功能。它用于检查用户从键盘等输入的数据，显示应用程序输出的数据，一般采用图形用户界面（graphic user interface， GUI）。
        2. （2）应用逻辑层：应用逻辑层为应用系统的主体部分，包含具体的业务处理逻辑。通常在功能层中包含有确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。
        3. （3）数据层：数据层主要包括数据的存储及对数据的存取操作，一般选择关系型数据库管理系统（RDBMS）。
        4. 浏览器/服务器（browser/server，B/S）风格是三层体系结构的一种实现方式，其具体结构为浏览器/Web服务器/数据库服务器。B/S体系结构如下图所示。



* + - * 1. B/S结构是一种全新的软件体系结构。
        2. B/S体系结构具有以下优点：

（1）基于B/S体系结构的软件，系统安装、修改和维护全在服务器端解决。

（2）B/S体系结构还提供了异种机、异种网、异种应用服务的联机、联网和统一服务的最现实的开放性基础。

* + - * 1. 与C/S体系结构相比，B/S体系结构也有许多不足之处。

（1）B/S体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能。

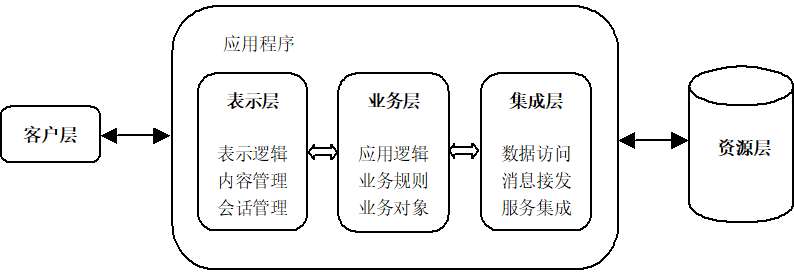
（2）采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远地低于C/S体系结构。

（3）B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理应用。

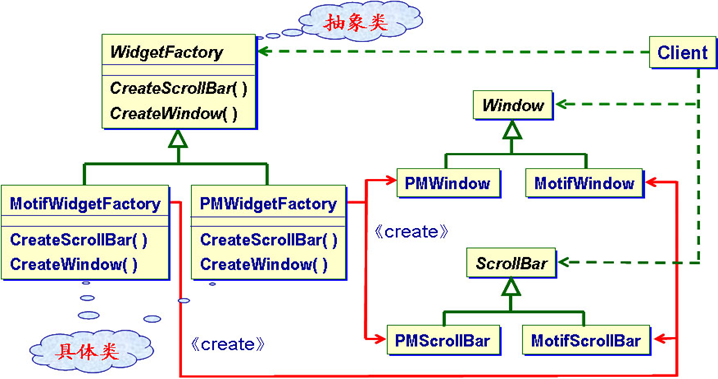
* + 1. 分布式对象体系结构
       1. 分布式对象的实质是在分布式异构环境下建立应用系统框架和对象构件，它将应用服务分割成具有完整逻辑含义的独立子模块（称为构件），各个子模块可放在同一台服务器或分布在多台服务器上运行，模块之间通过中间件互相通信。
       2. 通常将这个中间件称为软件总线或对象请求代理，它的作用是在对象之间提供一个无缝接口。
       3. 分布式对象技术的应用目的是为了降低主服务器的负荷、共享网络资源、平衡网络中计算机业务处理的分配，提高计算机系统协同处理的能力，从而使应用的实现更为灵活。
    2. 代理
       1. 代理可以用于构建带有隔离组件的分布式软件系统，该软件通过远程服务调用进行交互。代理者负责协调通信，诸如转发请求以及传递结果和异常等。
       2. 在ORB（对象请求代理）之上有4个对象接口：
          1. （1）对象服务：定义加入ORB的系统级服务，如安全性、命名和事务处理，它们是与应用领域无关的。
          2. （2）公共设施：水平级的服务，定义应用程序级服务。
          3. （3）领域接口：面向特定的领域。
          4. （4）应用接口：面向指定的现实世界应用。是指供应商或 用户借助于ORB、公共对象服务及公共设施而开发的特定产品。
  1. 5.体系结构框架
     1. MVC框架
        1. MVC框架即模型—视图—控制器（model-view-controller）框架，它强调将用户输入、数据模型和数据表示的方式分开设计，一个交互式应用系统由模型、视图和控制器3个部件组成，分别对应于内部数据、数据表示和输入/输出控制部分。



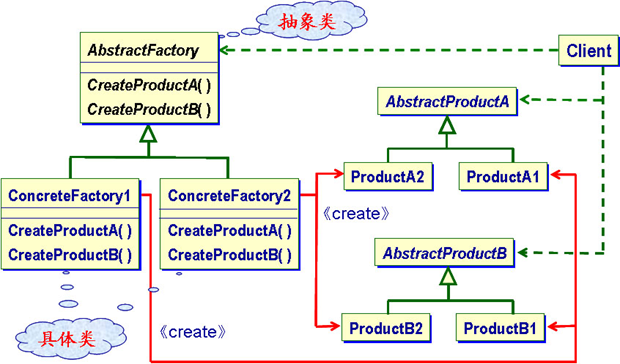
* + 1. J2EE体系结构框架
       1. J2EE的核心体系结构就是在MVC框架的基础上进行扩展得到的，如下图所示。



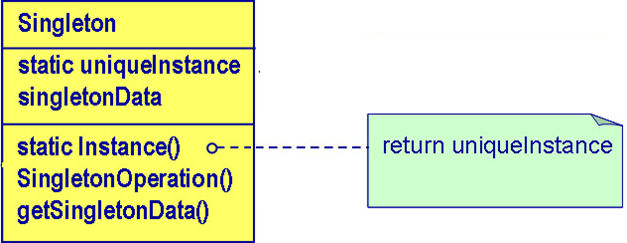
* + 1. PCMEF与PCBMER框架
  1. 6.设计模式
     1. 一般来说，一个模式有4个基本的要素：
        1. 1.模式名称：用于描述模式的名字，说明模式的问题、解决方案和效果。
        2. 2.问题：说明在何种场合使用模式。
        3. 3.解决方案：描述设计的组成成分、它们之间的相互关系、各自的职责和合作方式。
        4. 4.效果：描述了模式使用的效果及使用模式应当权衡的问题。
     2. (1)抽象工厂
        1. (1) 目的：提供一个接口用以创建一个相联系或相依赖的对象族，而无须指定它们的具体类。
        2. (2) 思路：
           1. 例如，在创建可支持多种GUI标准的绘图用户界面工具包时，因为不同的GUI标准会定义出不同外观及行为的“用户界面组件”，如滚动条、按钮、视窗等。为了能够囊括各种GUI标准，应用程序不能把组件写死，不能限制到特定GUI风格的组件类，否则日后很难换成其他GUI风格的组件。
           2. 解决方法是：先定义一个抽象类WidgetFactory（用斜体字区分抽象类），这个类声明了创建各种基本组件的接口，再逐一替各种基本组件定义相对应的抽象类，如 ScrollBar、Window等，让它们的具体子类来真正实现特定的GUI标准。
           3. 可支持多种GUI标准的绘图用户界面工具包的结构图



* + - 1. (3) 结构：抽象工厂模式的结构如图所示



* + - 1. (4) 参与者职责
         1. a) 抽象工厂类（Abstract Factory）：声明创建抽象产品对象的操作的接口。
         2. b) 具体工厂类（Concrete Factory）：实现产生具体产品对象的操作。
         3. c) 抽象产品类（Abstract Product）：声明一种产品对象的接口。
         4. d) 具体产品类（Concrete Product）：定义将被相应的具体工厂类产生的产品对象，并实现抽象产品类接口。
         5. e) 客户（Client）：仅使用由抽象工厂类和抽象产品类声明的接口。
      2. (5) 协作
         1. 在执行时，Abstract Factory将产品交给Concrete Factory创建。Concrete Factory类的实例只有一个，专门针对某种特定的实现标准，建立具体可用的产品对象。如果想要建立其他标准的产品对象，客户程序就得改用另一种Concrete Factory。
    1. (2)单件
       1. (1) 目的：一个类只有一个实例并提供一个访问它的全局访问点。该实例应在系统生存期中都存在。
       2. (2) 思路：例如，通常情况下，用户可以对应用系统进行配置，并将配置信息保存在配置文件中，应用系统在启动时首先将配置文件加载到内存中，这些内存配置信息应该有且仅有一份。应用单件模式可以保证Configure类只能有一个实例。
       3. (3) 结构：单件模式的结构如图所示。



* + - 1. (4) 参与者职责
         1. 单件（Singleton）：能够创建它唯一的实例；同时定义了一个Instance操作，允许外部存取它唯一的实例。Instance是一个静态成员函数
      2. (5) 协作：客户只能通过Singleton的Instance() 存取这唯一的实例。
    1. (3)外观
       1. (1) 目的：给子系统中的一组接口提供一套统一的高层界面，使得子系统更容易使用。
       2. (2) 思路：将系统划分为若干子系统，虽然可以降低整体的复杂性，但还需设法降低子系统之间的通信和相互的依赖性。一种方法就是引进一个外观（facade）对象，为子系统内各种设施提供一个简单的单一界面。
       3. (3) 结构：外观模式的结构如图所示。



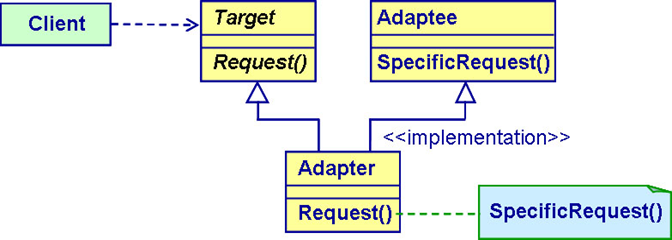
* + - 1. (4)参与者职责
         1. a)外观（Façade）：知道子系统中哪个类负责处理哪种信息；并负责把外界输入的信息转交给适当的子系统对象。
         2. b)子系统中的类（subsystem classes）：实现子系统的功能；处理Facade对象分派的工作；如果不受Facade的控制，则也不会有返回Facade的引用存在。
      2. (5)协作：使用Facade的客户不用直接访问子系统对象。外界想与子系统交互时，把信息传送给Facade，Facade再把这些信息转交给适当的子系统对象。虽然实际处理工作是子系统对象在做，但Facade会居中做接口转换工作。
    1. (4)适配器
       1. 1)目的：适配器模式将一个类的接口转换为客户期望的另一种接口，使得原本不匹配的接口而无法合作的类可以一起工作。
       2. (2)思路：有时要将两个没有关系的类组合在一起使用，一种解决方案是修改各自类的接口，另一种办法是使用Adapter模式，在两种接口之间创建一个混合接口。
          1. 例如，设有一个图形编辑器，可画直线、多边形、文本等。它的接口定义成抽象类Shape，它的子类负责画各种图形。此外，还有一个外购的GUI软件包TextView，用于显示，但它没有Shape功能。
          2. 如何让TextView的接口转换成为Shape的接口，有两种方法：

让TextShape同时继承Shape的接口和TextView的服务（多重继承）；

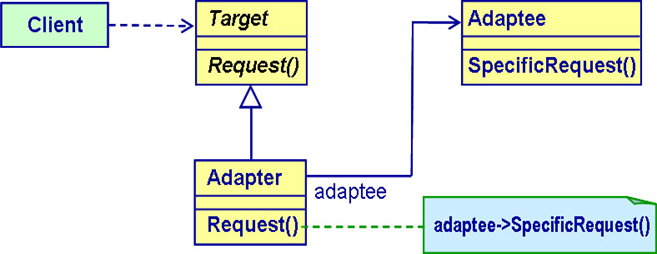
在TextShape中建立TextView的实例，再通过TextView给出TextShape的接口。

前者是适配器的类模式，后者是对象模式。下图就是适配器的对象模式。

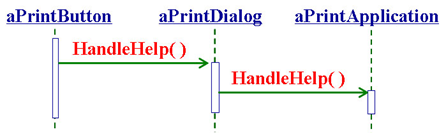
* + - 1. (3) 结构：适配器模式有类适配器模式和对象适配器模式。
         1. 类适配器可以通过多继承方式实现不同接口之间的相容和转换，如图所示。



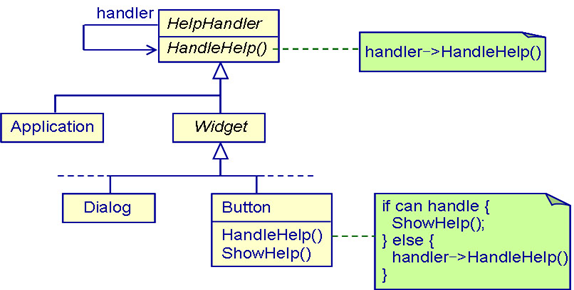
* + - * 1. 而一个对象适配器则依赖对象组合的技术实现接口的相容和转换，如图所示。



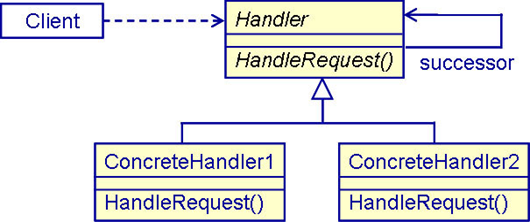
* + - 1. (4) 参与者职责
         1. a) 目标（Target）：定义客户使用的与应用领域相关的接口。
         2. b) 客户（Client）：与具有Target接口的对象合作。
         3. c) 被匹配者（Adaptee）：需要被转换匹配的一个已存在接口。
         4. d) 适配器（Adapter）：将Adaptee的接口与Target接口匹配。
      2. (5) 协作：客户调用Adapter对象的操作，然后Adapter的操作又调用Adaptee对象中负责处理相应请求的操作。
    1. (5)责任链
       1. (1) 目的：通过一条隐式的对象消息链传递处理请求。该请求沿着这条链传递，直到有一个对象处理它为止。其核心是避免将请求的发送者直接耦合到它的接受者。
       2. (2) 思路：以GUI系统的联机帮助系统为例。用户可以在软件中任一位置按下help键，软件就可以根据该信息和当前上下文环境弹出适当的说明。
          1. 如果用户在PrintDialog对话框里“打印”按钮上按了帮助键，帮助信息的顺序图如图所示。



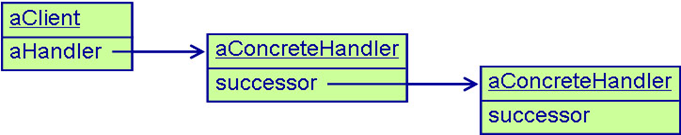
* + - * 1. 联机帮助系统定义了一个抽象类HelpHandler和抽象操作HandleHelp()，所有想处理信息的类可以继承该类。HelpHandler的HandleHelp() 操作的内定做法是把信息传递给后继者去处理，由各个子类分别来实现具体的打印功能。如图所示。



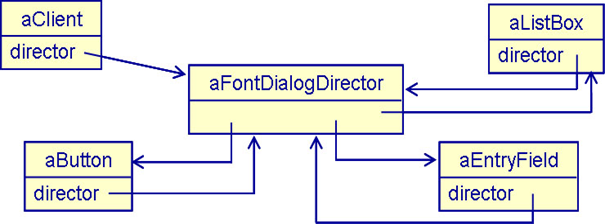
* + - 1. (3) 结构：
         1. 责任链模式的结构如图所示 。



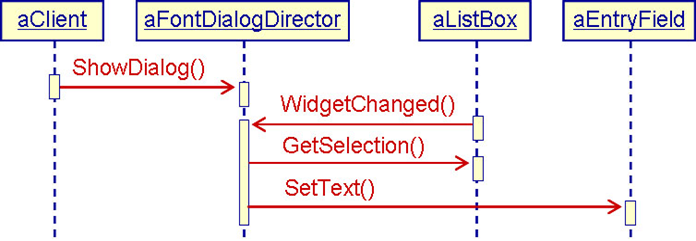
* + - * 1. 典型的对象间的结构如图所示。



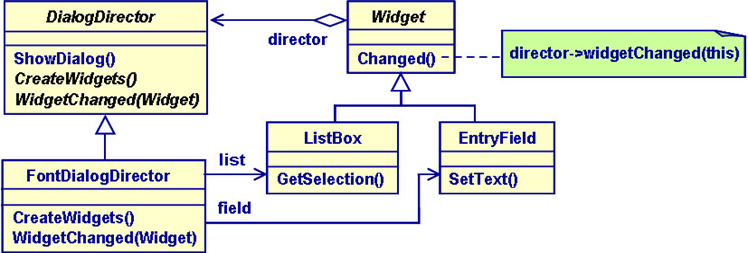
* + - 1. (4) 参与者职责
         1. a) 处理者（Handler）：定义处理请求的接口；实现对后继者的链接（可选）。
         2. b) 具体处理者（ConcreteHandler）：处理它所负责的请求；可访问它的后继；如果它能够处理请求，就处理该请求，否则将请求传送给后继者。
         3. c) 客户（Client）：将处理请求提交给职责链中的ConcreteHandler对象。
      2. (5) 协作：当Client发出请求之后，请求会在责任链中传递，直到有一个ConcreteHandler对象能处理为止。
    1. (6)中介者
       1. (1) 目的：用一个中介对象来封装一系列复杂对象的交互情景。中介者通过阻止各个对象显式地相互引用来降低它们之间的耦合，使得人们可以独立地改变它们之间的交互。
       2. (2) 思路：以GUI系统的对话框为例，对话框中会布置许多窗口组件，如按钮、菜单、文字输入栏等。对话框中各窗口组件之间往往相互牵连。
          1. 为此，可以将这些窗口组件的集体行为封装成一个中介者（mediator）对象。中介者负责居中指挥协调一组对象之间的交互行为，避免互相直接引用。这些对象只认得中介者，因而可降低交互行为的数目。
          2. 例如，可用FontDialogDirector当作对话框内各窗口组件之间的中介者。FontDialogDirector对象认得所有组件，协调彼此之间的交互，如同一个通信枢纽，如图所示。



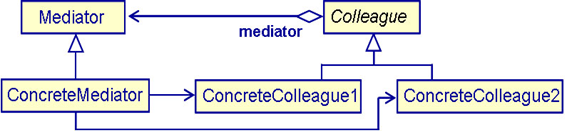
* + - * 1. 描述mediator作用的顺序图 。



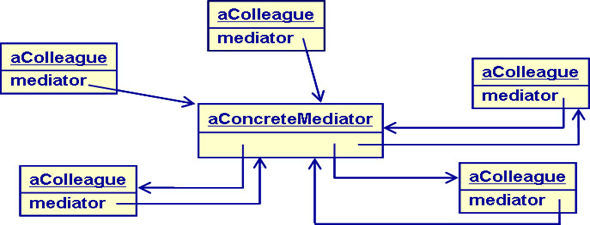
* + - * 1. 下图显示了加入FontDialogDirector后的类结构。



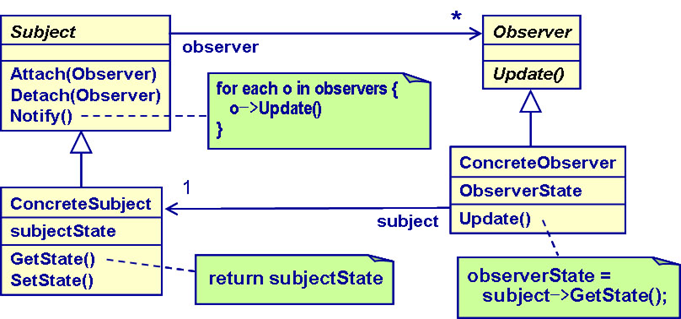
* + - 1. (3) 结构：图(a)给出了中介者的类结构，图(b)给出了典型的对象结构。
         1. （a）



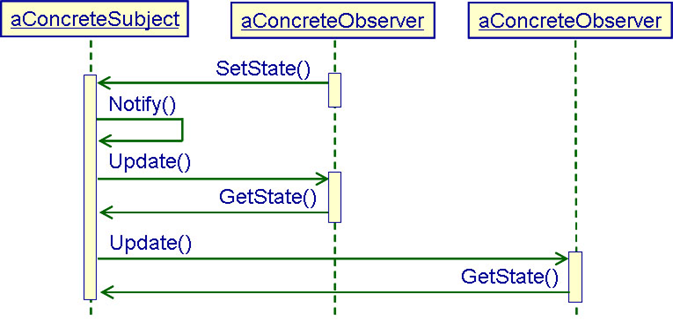
* + - * 1. （b）



* + - 1. (4) 参与者职责
         1. a) 中介者（Mediator）：定义与各个同事（Colleague）对象通信的接口。
         2. b) 具体中介者（ConcreteMediator）：协调各个同事对象，实现协作行为；了解并维护各个同事对象。
         3. c) 同事类（Colleague classes）：这些同事类的对象都了解中介者；一个同事对象与另一个同事对象之间的通信都需要通过中介者来间接实现。
      2. (5) 协作：同事向中介者对象发送或接收请求，中介者则将请求传送给适当的同事对象（一个或多个），协调整体行为。
    1. (7)观察者
       1. (1)目的：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知，并被自动更新。
       2. (2)思路：例如，许多GUI软件包都将数据显示部分与应用程序底层的数据表示分开，以利于分别复用。但这些类也能合作，如图所示的计算表和直方图都是针对同一数据对象的两种不同表示方式。
       3. (3) 结构：Observer模式的结构如图所示。



* + - 1. (4) 参与者职责
         1. a) 主题（Subject）：认得它的观察者。任意数目的观察者对象均可订阅一个主题。另外，提供一个连接观察者对象和解除连接的接口。
         2. b) 观察者（Observer）：定义了一个自我更新的接口。一旦发现主题有变时借助接口通知自己随之改变。
         3. c) 具体主题（ConcreteSubject）：存储具体观察者对象关心的状态；当状态改变时向它的观察者发送通知。
         4. d) 具体观察者（ConcreteObserver）：维持一个对具体主题对象的引用；存储要与主题一致的状态；实现观察者的自我更新接口，确保自己的状态与主题的状态一致。
      2. (5) 协作：当具体主题发生会导致观察者的状态不一致的情况时，就会主动通知所有该通知的观察者。当具体观察者收到通知后，向主题询问，根据所得信息使自己的状态与主题的状态保持一致。下图给出了一个主题和两个观察者对象之间的交互情况。



# 软件工程概论8

1. 八、面向对象设计
   1. 1.面向对象设计准则
      1. (1) 模块化
         1. 传统的面向过程方法中的模块通常是函数、过程及子程序等，而面向对象方法中的模块则是类、对象、接口、构件等。
         2. 在面向过程的方法中，数据及在数据上的处理是分离的；而在面向对象方法中，数据及其上的处理是封装在一起的，具有更好的独立性，也能够更好地支持复用。
      2. (2) 抽象
         1. 面向对象方法不仅支持过程抽象，而且支持数据抽象。类实际上就是一种抽象数据类型。可以将类的抽象分为规格说明抽象及参数化抽象。
         2. 类对外开放的公共接口构成了类的规格说明，即协议。这种接口规定了外部可以使用的服务，使用者无需知道这些服务的具体实现算法。通常将这类抽象称为规格说明抽象。
         3. 参数化抽象是指当描述类的规格说明时并不具体指定所要操作的数据类型，而是将数据类型作为参数。
      3. (3) 信息隐藏
         1. 在面向对象方法中，信息隐藏通过对象的封装性实现。对于类的用户来说，属性的表示方法和操作的实现算法都应该是隐藏的。
      4. (4) 弱耦合
         1. 耦合是指一个软件结构内不同模块之间互连的紧密程度。在面向对象方法中，对象是最基本的模块，因此，耦合主要指不同对象之间相互关联的紧密程度。
      5. (5) 强内聚
         1. 内聚衡量一个模块内各个元素彼此结合的紧密程度。在面向对象设计中存在以下3种内聚：
         2. (1) 服务内聚：一个服务应该完成一个且仅完成一个功能。
         3. (2) 类内聚：设计类的原则是，一个类应该只有一个用途，它的属性和服务应该是高内聚的。类的属性和服务应该全都是完成该类对象的任务所必需的，其中不包含无用的属性或服务。如果某个类有多个用途，通常应该把它分解成多个专用的类。
         4. (3) 一般—特殊内聚：设计出的一般—特殊结构，应该符合多数人的概念，更准确地说，这种结构应该是对相应的领域知识的正确抽取。
      6. (6) 可重用
         1. 软件重用是提高软件开发生产率和目标系统质量的重要途径。
         2. 重用基本上从设计阶段开始。重用有两方面的含义：
         3. 一是尽量使用已有的类(包括开发环境提供的类库，及以往开发类似系统时创建的类)，
         4. 二是如果确实需要创建新类，则在设计这些新类的协议时，应该考虑将来的可重复使用性。
   2. 2.体系结构模块及依赖性
      1. 在面向对象软件中，常见的软件模块有类、接口、包和构件。
      2. 在设计阶段我们往往关注类、接口和包，在实现阶段关注构件，而在部署阶段则关注构件的部署，也就是将构件部署在哪些结点上。
      3. 1.类及其依赖性
         1. 1.类
            1. 在面向对象的程序设计中，类和接口是程序的基本组成单元。
            2. 一个典型程序需要界面类专门负责表示用户界面信息，需要数据库类负责与数据库进行交互，需要有业务逻辑类负责算法计算等。
            3. 在计算机程序中，要设计和实现的所有类都具有唯一的名字，在不同的阶段或从不同的角度可以将它们称为设计类、实现类、系统类、应用类等。
         2. 2．继承依赖性
            1. 依赖性管理中最棘手的问题是由于继承所引起的依赖性。继承是一种在父类和子类之间共享属性和行为的方式，所以运行时可以用一个子类对象代替其父类对象。

（1）多态继承

根据为请求提供服务的对象不同可以得到不同的行为，这种现象称为多态。在运行时对类进行实例化，并调用与实例化对象相应的方法，称为动态绑定、后期绑定或运行时绑定。相应地，如果方法的调用是在编译时确定的，则称为是静态绑定、前期绑定或编译时绑定。多态并不是伴随着继承而出现。如果在子类中不覆盖父类中的任何方法，就不会产生多态行为。很明显，继承会带来类和方法之间的依赖性。继承带来的依赖性有编译时继承依赖性和运行时继承依赖性。

2）无多态继承

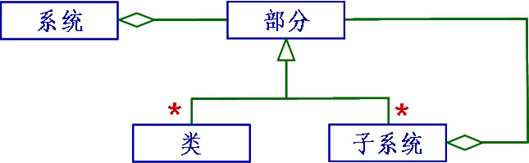
使用继承最简单的方式是子类不覆盖从父类继承来的方法，这样就不存在多态性继承问题。虽然无多态的继承有时并不是十分有用，但理解和管理起来是最容易的。

（3）扩展继承和约束继承

扩展继承是指子类继承父类的属性，并且提供额外属性来增强类定义。子类是父类的一种，如果子类覆盖了父类的方法，那么被覆盖的方法应该实现该方法的定义，并且能够在子类的语境中工作。

当一个类覆盖了继承来的方法，并对一些继承来的功能进行了限制，这时就产生了约束继承。这时，子类不再是父类的一种。有时，限制会造成继承方法的完全禁止。当方法的实现是空时，就会发生这种情况。

* + - 1. 3．交互依赖性
         1. 交互依赖性也称为方法依赖性，是通过消息连接产生的。
    1. 2.接口及其依赖性
       1. 1．接口
          1. 在UML2.0中，接口是不可直接实例化的特性集合的声明，即其对象不能直接实例化，需要通过类来实现，实现接口的类需要实现接口中声明的方法。
          2. 接口与抽象类有相似之处，抽象类是至少包含一个没有实现的方法的类，如果在一个抽象类中所有的方法都没有实现，则称其为纯抽象类，从这一点上，接口和纯抽象类似乎没有区别。但实际上，接口和抽象类还是有着本质的区别。在只支持单继承的语言中，一个类只能有一个直接父类，但是却可以实现多个接口。
       2. 2．实现依赖性
          1. 一个类可以实现多个接口，由类实现的接口集合称为该类的供给接口。在UML 2.0中，将一个类和该类实现的接口之间的依赖性称为实现依赖性
       3. 3．使用依赖性
          1. 一个接口可以为其他类或接口提供服务，同时也可能需要其他接口的服务。一个接口所需要的其他接口所提供的服务称为这个类的需求接口。需求接口详细说明一个类或接口需要的服务，从而可以为其客户提供服务。在UML2.0中，通过类（接口）和它所需接口之间的依赖关系来说明需求接口，这称为使用依赖性。
    2. 3.包及其依赖性
       1. 1. 包
          1. 包（package）又可称为层或子系统，是表示组织类的一种方式，用于划分应用程序的逻辑模型。包是高度相关的类的聚合，这些类本身是内聚的，但相对于其他聚合来说又是松散耦合的。包可以嵌套。外层包可以直接访问包括在它的嵌套包中的任何类。包还可以导入其他包
       2. 2．包依赖性
          1. 本质上，两个包之间的依赖性来自于两个包中类之间的依赖性。类之间的循环依赖性是个特别棘手的问题，好在大多数情况下可以通过重新设计避免循环依赖性。
    3. 4.构件及其依赖性
       1. 一个构件可以是一个编译的类，可以是一组编译的类，也可以是其他独立的部署单元
       2. 为了能够支持复用，软件构件应具有以下特性：
          1. （1）独立部署单元：一个构件是独立部署的，意味着它必须能与它所在的环境及其他构件完全分离。
          2. （2）作为第三方的组装单元：构件必须具备很好的内聚性，必须封装它的实现，并且只通过良好定义的接口与外部环境进行交互。
          3. （3）一个构件不能有任何（外部的）可见状态：即构件不能与自己的拷贝有所区别。
  1. 3.系统分解
     1. （1）子系统和类
        1. 在大型和复杂的软件系统情形，首先根据需求的功能模型（用例模型），将系统分解成若干个部分，每一部分又可分解为若干子系统或类，每个子系统还可以由更小的子系统或类组成，如图所示。



* + 1. （2）服务和子系统接口
       1. 服务是一组有公共目的的相关操作。而子系统则通过给其他子系统提供服务来发挥自己的能力。与类不同的是，子系统不要求其他子系统为它提供服务。
       2. 供其他子系统调用的某个子系统的操作集合就是子系统的接口。
       3. 子系统的接口包括操作名、操作参数类型及返回值。
       4. 面向对象的系统设计主要关注每个子系统提供服务的定义，即枚举所有的操作、操作参数和行为。
    2. （3）子系统之间的两种交互方式
       1. 客户-供应商关系：在这种关系中，客户子系统调用供应商子系统，后者完成某些服务工作并返回结果。使用这种交互方案，作为客户的子系统必须了解作为供应商的子系统的接口，而后者却无须了解前者的接口。
       2. 平等伙伴关系：在这种关系中，每个子系统都可能调用其他子系统，因此每个子系统都必须了解其他子系统的接口。与第一种方案相比，这种方案中，子系统间的交互更加复杂。
       3. （4）组织系统的两种方案
          1. 分层组织：这种组织方案把软件系统组织成一个层次系统，每层是一个子系统。上层在下层的基础上建立，下层为实现上层功能而提供必要的服务。每一层内所包含的对象，彼此间相互独立，而处于不同层次上的对象，彼此间往往有关联。
          2. 块状组织：这种组织方案把软件系统垂直地分解成若干个相对独立的、弱耦合的子系统，一个子系统相当于一块，每块提供一种类型的服务。
          3. 混合使用层次结构和块状结构，可以成功地由多个子系统组成一个完整的软件系统。
  1. 4 .问题域部分的设计
     1. 典型的面向对象系统一般由三层组成，即数据库层、业务逻辑层及用户界面层
     2. 可能对面向对象分析所得出的问题域模型做以下方面的补充或调整。
        1. （1）调整需求。有两种情况会导致修改通过面向对象分析所确定的系统需求：
           1. 一是用户需求或外部环境发生变化；
           2. 二是分析员对问题理解不透彻，导致分析模型不能完整、准确地反映用户的真实需求。
           3. 从类库选择已有的类，从供应商那里购买商业外购构件，从网络、组织、小组或个人那里搜集适用的遗留软构件，把它们增加到问题域部分的设计中去。
        2. (2) 复用已有的类
           1. 在被复用的已有类和问题域类之间添加泛化（一般化∕特殊化）关系，继承被复用类或构件属性和方法。
           2. 标出在问题域类中因继承被复用的已有类或构件而成为多余的属性和服务。
           3. 修改与问题域类相关的关联。
        3. (3) 把问题域类组合在一起
        4. (4) 增添泛化类以建立类间的协议
        5. (5) 调整继承的支持级别
           1. 1) 针对单继承语言的调整。
           2. 2) 针对无继承语言的调整。
        6. (6) 改进性能
           1. 1) 如果类之间经常需要传送大量消息，可合并相关的类，使得通信成为对象内的通信，而不是对象之间的通信，或者使用全局数据作用域，打破封装的原则，以减少消息传递引起的速度损失。
           2. 2) 增加某些属性到原来的类中，或增加低层的类，以保存暂时结果，避免每次都要重复计算造成速度损失。
        7. (7) 存储对象
           1. 通常的作法是，每个对象将自己传送给数据管理部分，让数据管理部分来存储对象本身。
  2. 5.人机交互部分的设计
     1. 用户界面设计步骤
        1. (1) 从系统的输入、输出及与用户的交互中获得信息，定义界面对象和行为（操作）。
        2. (2) 定义那些导致用户界面状态发生变化的事件，对事件建模。
        3. (3) 描述最终向用户展示的每一个界面的状态，。
        4. (4) 简要说明用户如何从界面提供的界面信息来解释系统状态。
     2. 界面设计目标
        1. (1) 简单性：尽量做到适度和简单，不要在页面上提供太多的东西。
        2. (2) 一致性：这一设计目标几乎适用于设计模型的每一个元素。
        3. (3) 确定性：Web应用系统的美学、界面和导航设计必须与将要构造的应用系统所处的领域保持一致。
        4. (4) 健壮性：在已经建立的确定性的基础上，Web应用系统通常会给用户明确的“承诺”。
        5. (5) 导航性：我们已经在前面提及了导航应该简单和一致，也应该以直观的和可预测的方式来设计。也就是说，用户不必搜索导航链接和帮助就知道如何使用Web应用系统。
        6. (6) 视觉吸引：在所有类型的软件中，Web应用系统毫无疑问是最具有视觉效果的、最生动的、也是最具有审美感的。
        7. (7) 兼容性：Web应用系统会应用于不同的环境(例如，不同的硬件、Internet连接类型、操作系统、浏览器)，并且必须互相兼容。
     3. 界面设计工作流
        1. (1) 回顾那些在分析模型中的信息，并根据需要进行优化。
        2. (2) 开发Web应用系统界面布局的草图。
        3. (3) 将用户目标映射到特定的界面行为。
        4. (4) 定义与每个行为相关的一组用户任务。
        5. (5) 为每个界面行为设计情节串联图板屏像。
        6. (6) 利用从美学设计中的输入来优化界面布局和情节串联图板。
        7. (7) 明确实现界面功能的界面对象。
        8. (8) 开发用户与界面交互的过程表示。
        9. (9) 开发界面的行为表示法。
        10. (10) 描述每种状态的界面布局。
        11. (11) 优化和评审界面设计模型。
  3. 6.任务管理部分的设计
     1. (1) 识别事件驱动任务
     2. (2) 识别时钟驱动任务
        1. 当系统运行时，这类任务的工作过程如下：
        2. 任务设置了唤醒时间后进入睡眠状态；
        3. 等待来自系统的一个时钟中断，一旦接收到这种中断，任务就被唤醒，并做它的工作，通知有关的对象，然后该任务又回到睡眠状态。
     3. (3) 识别优先任务
        1. 根据处理的优先级别来安排各个任务。优先任务可以满足高优先级或低优先级的处理需求。
        2. 高优先级：某些服务具有很高的优先级，为了在严格限定的时间内完成这种服务，可能需要把这类服务分离成独立的任务。
        3. 低优先级：与高优先级相反，有些服务是低优先级的，属于低优先级处理（通常称为后台处理）。设计时可能用额外的任务把这样的处理分离出来。、
     4. (4) 识别关键任务
        1. 关键任务是有关系统成功或失败的关键处理，这类处理通常都有严格的可靠性要求。
        2. 在设计过程中可能用额外的任务把这样的关键处理分离出来，以满足高可靠性处理的要求。
        3. 对高可靠性处理应该精心设计和编码，并且应该严格测试。
     5. (5) 识别协调任务
     6. (6) 审查每个任务
        1. 要使任务数保持到最少。
        2. 对每个任务要进行审查，确保它能满足一个或多个选择任务的工程标准——事件驱动、时钟驱动、优先任务∕关键任务或协调者。
     7. (7) 定义每个任务
        1. 1) 它是什么任务。首先要为任务命名，并对任务做简要描述。为面向对象设计部分的每个服务增加一个新的约束——任务名。如果一个服务被分裂，交叉在多个任务中，则要修改服务名及其描述，使每个服务能映射到一个任务。
        2. 2) 如何协调任务。定义每个任务如何协调工作。指出它是事件驱动的，还是时钟驱动的；对于事件驱动的任务，描述触发该任务的事件；对时钟驱动的任务，描述在触发之前所经过的时间间隔，同时指出它是一次性的，还是重复的事件间隔。
        3. 3) 如何通信。定义每个任务如何通信，任务从哪里取数据及往哪里送数据。
  4. 7.数据管理部分的设计
     1. 在映射时可以按下面的规则进行：
        1. (1) 一个普通的类可以映射为一个表或多个表，当分解为多个表时，可以采用横切和竖切的方法。
           1. 竖切常用于实例较少而属性很多的对象，一般是现实中的事物，将不同分类的属性映射成不同的表。通常将经常使用的属性放在主表中，而将其他一些次要的属性放到其他表中。
           2. 横切常常用于记录与时间相关的对象，如成绩记录、运行记录等。由于一段时间后，这些对象很少被查看，所以往往在主表中只记录最近的对象，而将以前的记录转到对应的历史表中。
        2. (2) 关联关系的映射
           1. 一对一关联的映射：对于一对一关联，可以在两个表中都引入外键，这样两个表之间可以进行双向导航。也可以根据具体情况，将类组合成一张单独的表。
           2. 一对多关联的映射：可以将关联中的“一”端毫无变化地映射到一张表，将关联中表示“多”的端上的类映射到带有外键的另一张表，使外键满足关系引用的完整性。
           3. 多对多关联的映射：由于记录的一个外键最多只能引用另一条记录的一个主键值，因此关系数据库模型不能在表之间直接维护一个多对多联系。为了表示多对多关联，关系模型必须引入一个关联表，将两个类之间的多对多关联转换成表上的两个一对多关联。
        3. (3) 继承关系的映射：通常使用以下两种方法来映射继承关系。
           1. 将基类映射到一张表，每个子类映射到一张表。在基类对应的表中定义主键，而在子类定义的表中定义外键。
           2. 将每个子类映射到一张表，没有基类表。在每个子类的表中包括基类的所有属性。这种方法适用于子类的个数不多，基类属性比较少的情况。
  5. 8.对象设计
     1. 对象设计过程包括使用模式设计对象、接口规格说明、对象模型重构、对象模型优化4组活动。
        1. (1) 使用模式设计对象：设计者可以选择合适的设计模式，复用已有的解决方案，以提高系统的灵活性，并确保在系统开发过程中，特定类不会因要求的变化而被修改。
        2. (2) 接口规格说明：在系统设计中所标识的子系统功能，都需要在类接口中详细说明，包括操作、参数、类型规格说明和异常情况等。
        3. (3) 对象模型重构：重构的目的是改进对象设计模型，提高该模型的可读性和扩展性。如将两个相似的类归并为一个类；将没有明显活动特征的类转为属性；将复杂的类分解为简单的类；重新组合类和操作来增进封装性和继承性等。
        4. (4) 对象模型优化：优化活动是为了改进对象设计模型，以实现系统模型中的性能要求。包括选择更好的算法、提高系统执行的速度、更好地使用存储系统、减少连接中的重数来提高查询的速度、为了增加效率而增加额外的连接、改变执行的顺序等。
     2. 设计模式包括4个要素：
        1. (1) 名字：用来将一设计模式与其他设计模式区分开。
        2. (2) 问题描述：用来描述该设计模式适用于何种情况。通常设计模式所解决的问题是对可更改性、可扩展性设计目标、以及非功能性需求的实现。
        3. (3) 解决方案：描述解决该问题所需要的、结合在一起的类和接口的集合。
        4. (4) 结果：描述将要解决设计目标的协议和可供选择的办法。
     3. 几种常用的优化方法包括：
        1. (1) 增加冗余关联以提高访问效率
        2. (2) 调整查询次序
        3. (3) 保留派生属性

# 软件工程概论10

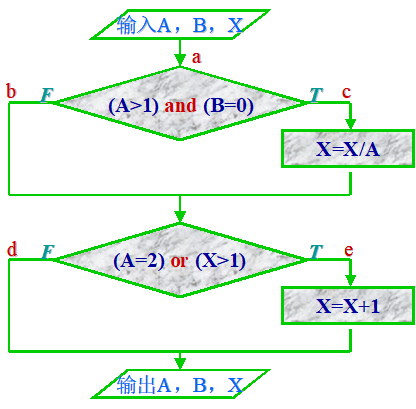
1. 十、软件测试方法
   1. 1.软件测试的目的和原则
      1. 基于不同的立场，存在着两种完全不同的测试目的。
         1. 从用户的角度出发，普遍希望通过软件测试检验软件中隐藏的错误和缺陷，以考虑是否可以接受该产品。
         2. 从软件开发者的角度出发，则希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。
      2. 软件测试目的提出了以下观点。
         1. 测试是程序的执行过程，目的在于发现错误。
         2. 好的测试用例在于能发现至今未发现的错误。
         3. 成功的测试是发现了至今未发现的错误的测试。
      3. 软件测试的原则如下：
         1. （1）应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。
         2. （2）测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。
         3. （3）程序员应避免检查自己的程序。
         4. （4）在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。
         5. （5）充分注意测试中的群集现象。把Pareto原理应用于软件测试。Pareto原理：测试发现的错误中的80%很可能是由程序中20%的模块造成的。
         6. （6）严格执行测试计划，排除测试的随意性。
         7. （7）应当对每一个测试结果作全面检查。
         8. （8）妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。
      4. 机器测试与人工测试
         1. 机器测试：在设定的测试数据上执行被测程序的过程。又称动态测试。
         2. 人工测试：采用人工方法进行，目的在于检查程序的静态结构，找出编译不能发现的错误。
      5. 人工测试的分类
         1. 代码审查：以小组会的形式，发现程序在结构、功能、编码风格等方面存在的问题。可查出30%~70%的错误
         2. 走查：以小组会的形式进行，把测试数据“输入”到被测程序，并在纸上跟踪监视程序的执行情况，让人代替机器沿着程序的逻辑走一遍。
         3. 桌前检查：设计模块时，程序员自己检查。
      6. 机器测试的分类
         1. 黑盒测试
            1. 黑盒测试是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。
            2. 黑盒穷举测试：对所有输入数据的各种可能值的排列组合都进行测试，来检查程序是否都能产生正确的输出。实际上这是不可能的。
         2. 白盒测试
            1. 白盒测试是对软件的过程性细节做细致的检查。
            2. 这一方法是把测试对象看做一个打开的盒子或透明的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。
            3. 通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。
            4. 因此，白盒测试又称为结构测试或逻辑驱动测试
            5. 白盒测试主要是对程序模块进行检查：

对程序模块的所有独立的执行路径至少测试一次；

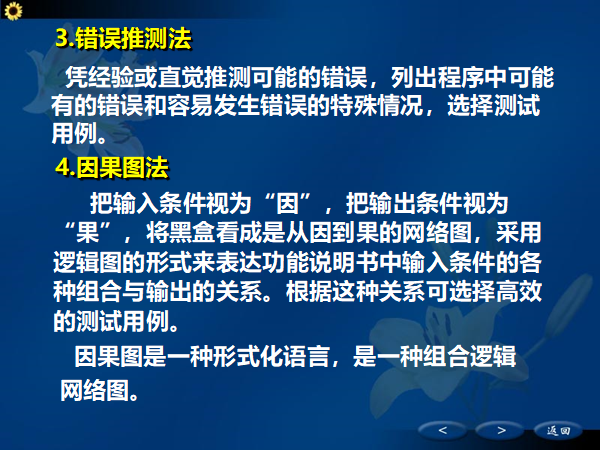
对所有的逻辑判定，取“真”与取“假”的两种情况都能至少测试一次；

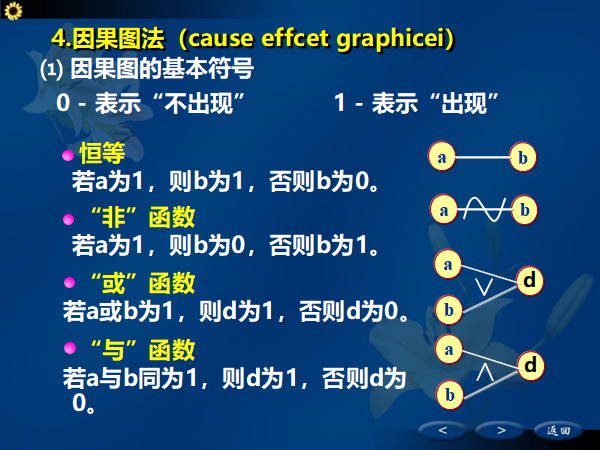
在循环的边界和运行界限内执行循环体；测试内部数据结构的有效性等。

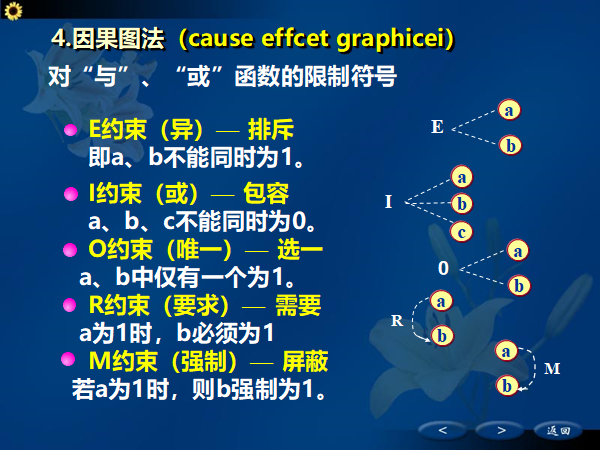
* + - * 1. 白盒穷举测试：对每条通路都应在每种可能的输入数据下执行一次。实际上这是不可能的。
  1. 2.白盒测试的测试用例设计
     1. 逻辑覆盖是以程序内部的逻辑结构为基础的设计测试用例的技术，它属于白盒测试。
     2. 由于覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖、判定覆盖、判定—条件覆盖、条件组合覆盖、路径覆盖。
     3. 最少测试用例覆盖最多的条件

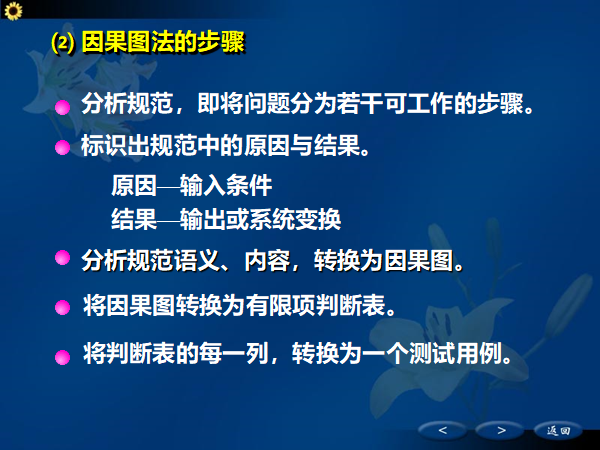


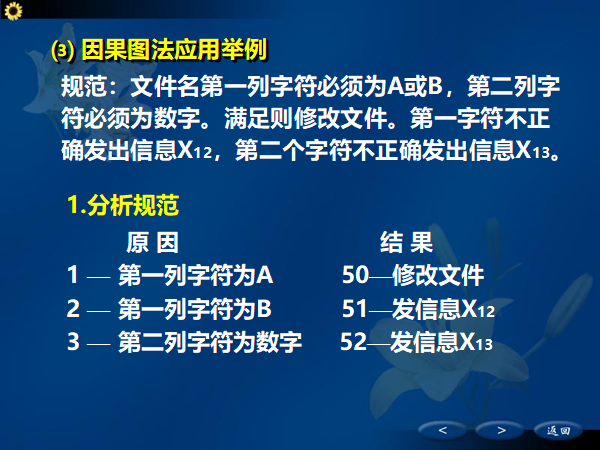
* + 1. 1.语句覆盖
       1. 语句覆盖就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。
       2. 在图例中，正好所有的可执行语句都在路径 ace上，所以选择路径ace设计测试用例，就可以覆盖所有的可执行语句。
    2. 2.判定覆盖
       1. 判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。判定覆盖又称为分支覆盖。(TT、FF)或者（TF、FT）
    3. 3.条件覆盖
       1. 条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。
    4. 4.判定—条件覆盖
       1. 所谓判定-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断本身的所有可能判断结果至少执行一次。
    5. 5.条件组合覆盖
       1. 条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。
    6. 6.路径覆盖
       1. 路径测试是设计足够的测试用例，覆盖程序中所有可能的路径。若仍以最初的图为例，则可以选择如下的一组测试用例，覆盖该程序段的全部路径。
    7. 基本路径覆盖
       1. 基本路径测试是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。
       2. 1. 由过程描述导出控制流图
       3. 2. 由过程描述导出控制流图 ：V（G）=（区域数）=（判定结点数）+1
       4. 3. 确定线性无关的基本路径集
       5. 4. 准备测试用例，确保基本路径集中的每一条路径的执行
  1. 3.黑盒测试的测试用例设计
     1. 1.等价类划分
        1. 等价类划分是一种典型的黑盒测试方法，也是一种非常实用的重要测试方法,它是用来解决如何选择适当的子集，使其尽可能多地发现错误。
        2. 使用这一方法设计测试用例要经历划分等价类（列出等价类表）和选取测试用例两步。
        3. 等价类的划分有两种不同的情况:
           1. （1）有效等价类：是指对于程序的规格说明来说，是合理的、有意义的输入数据构成的集合。利用它，可以检验程序是否实现了规格说明预先规定的功能和性能。
           2. （2）无效等价类：是指对于程序的规格说明来说，是不合理的、无意义的输入数据构成的集合。程序员主要利用这一类测试用例检查程序中功能和性能的实现是否有不符合规格说明要求的地方。
           3. 在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。
     2. 2.边界值分析
        1. 1. 边界值分析方法的考虑
           1. 边界值分析也是一种黑盒测试方法，是对等价类划分方法的补充。人们从长期的测试工作经验中得知，大量的错误是发生在输入或输出范围的边界上，而不是在输入范围的内部。这里所说的边界是指，相当于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。
        2. 2. 选择测试用例的原则
           1. (1)如果输入数据规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。 例如，若输入值的范围是“−1.0～1.0”，则可选取“−1.0”，“1.0”，“−1.001”，“1.001”作为测试输入数据。
           2. (2) 如果输入数据规定了值的个数，则用最大个数、最小个数、比最大个数多1、比最小个数少1的数作为测试数据。例如，一个输入文件有1～255个记录，设计测试用例时则可以分别设计有1个记录、255个记录以及0个记录和256个记录的输入文件。
           3. （3）根据规格说明的每个输出数据，使用前面的原则(1)。
           4. （4）根据规格说明的每个输出数据，使用前面的原则(2)。
           5. （5）如果程序的规格说明给出的输入域或输出域是有序集合（如有序表，顺序文件等），则应选取集合的第一个元素和最后一个元素作为测试用例。
           6. （6）如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。例如，如果程序中定义了一个数组，其元素下标的下界是0，上界是100，那么应选择达到这个数组下标边界的值，如0与100，作为测试用例。
           7. （7）分析规格说明，找出其他可能的边界条件。
        3. 3. 应用边界值分析方法设计测试用例的实例
     3. 3.因果图

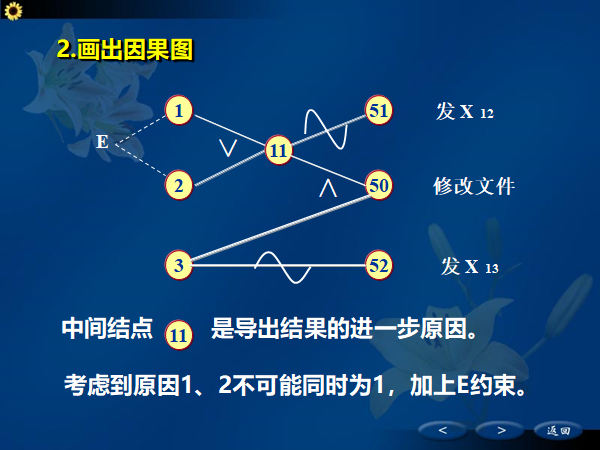






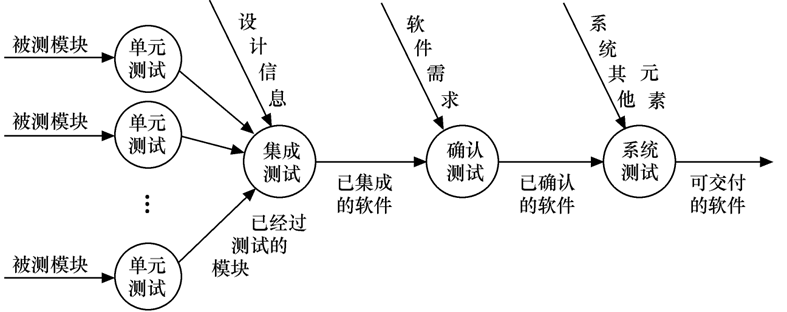








* 1. 4.软件测试的策略
     1. 通常软件测试过程按4个步骤进行，即单元测试、组装测试、确认测试和系统测试。如下图所示。



* + 1. 1.单元测试
       1. 单元测试（unit testing）又称模块测试，是针对软件设计的最小单位—程序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。
       2. 1．单元测试的内容
          1. 单元测试主要采用白盒测试方法设计测试用例，辅之以黑盒测试的测试用例，使之对任何合理的输入和不合理的输入，都能鉴别和响应。在单元测试中进行的测试工作如下图所示，需要在5个方面对被测模块进行检查。

（1）模块接口测试。

（2）局部数据结构测试。

（3）路径测试。

（4）错误处理测试。

（5）边界测试。

* + - 1. 2. 单元测试的步骤
         1. 模块并不是一个独立的程序，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。这些辅助模块分为如下两种。

（1）驱动模块（driver）——相当于被测模块的主程序，它接收测试数据，并把这些数据传送给被测模块，最后再输出实测结果。

2）桩模块（stub）——也叫做存根模块，用以代替被测

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。

* + 1. 2.组装测试
       1. 组装测试（integrated testing）也叫做集成测试或联合测试。通常，在单元测试的基础上，需要将所有模块按照设计要求组装成为系统，把模块组装为系统的方式有两种：一次性组装方式（big bang）和增值式组装方式。
       2. 1．一次性组装方式
          1. 它是一种非增值式组装方式，也叫做整体拼装。使用这种方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。
       3. 2. 增值式组装方式
          1. 这种组装方式又称渐增式组装，首先是对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统，在组装的过程中边连接边测试，以发现连接过程中产生的问题。最后通过增值逐步组装成为要求的软件系统。增值组装有以下3种做法。
          2. （1）自顶向下的增值方式。这种组装方式是将模块按系统程序结构，沿控制层次自顶向下进行组装，其步骤如下：

① 以主模块为被测模块兼驱动模块，所有直属于主模块 的下属模块全部用桩模块代替，对主模块进行测试。

② 采用深度优先（如下图）或宽度优先的策略，逐步用实际模块替换已用过的桩模块，再用新的桩模块代替它们的直接下属模块，与已测试的模块或子系统组装成新的子系统。

③ 进行回归测试（即重新执行以前做过的全部测试或部分测试），排除组装过程中引入新的错误的可能。

④ 判断是否所有的模块都已组装到系统中，若是则结束测试，否则转到②去执行。

* + - * 1. （2）自底向上的增值方式。这种组装方式是从程序模块结构的最底层的模块开始组装和测试。因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块（包括子模块的所有下属模块）已经组装并测试完成，所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以由直接运行子模块得到。

自底向上增值的步骤如下：

① 由驱动模块控制最底层模块的并行测试；也可以把最底层模块组合成实现某一特定软件功能的簇，由驱动模块控制它进行测试。

② 用实际模块代替驱动模块，与它已测试的直属子模块组装成为子系统。

③ 为子系统配备驱动模块，进行新的测试。

④ 判断是否已组装到达主模块。若是则结束测试，否则执行②。

* + - * 1. （3）混合增值式测试。自顶向下增值的方式和自底向上增值的方式各有优缺点。自顶向下增值方式的缺点是需要建立桩模块。自底向上增值方式的缺点是“程序一直未能作为一个实体存在，直到最后一个模块加上去后才形成一个实体”。也就是说，在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。鉴于此，通常是把以上两种方式结合起来进行组装和测试。下面简单介绍3种常见的综合增值方式测试。

① 衍变的自顶向下的增值测试：它的基本思想是强化对输入/输出模块和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增值测试。

② 自底向上—自顶向下的增值测试：它首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统作自顶向下的组装与测试。

③ 回归测试：这种方式采取自顶向下的方式测试被修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否适配。

* + - 1. 3．组装测试的组织和实施
         1. 组装测试是一种正规测试过程，必须精心计划，并与单元测试的完成时间协调起来。
         2. 在制定测试计划时，应考虑如下因素：

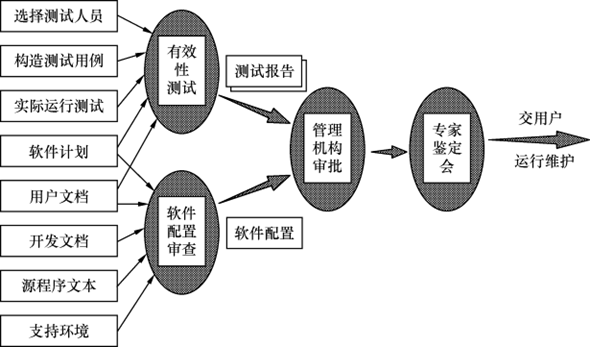
（1）采用何种系统组装方法进行组装测试。

（2）组装测试过程中连接各个模块的顺序。

（3）模块代码编制和测试进度是否与组装测试的顺序一致。

（4）测试过程中是否需要专门的硬件设备

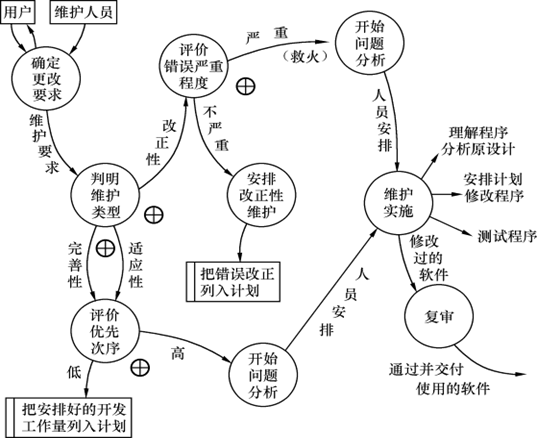
* + 1. 3.确认测试
       1. 确认测试（validation testing）又称有效性测试。它的任务是验证软件的有效性，即验证软件的功能和性能及其他特性是否与用户的要求一致。在确认测试阶段需要做的工作如下图所示。



* + - 1. 1．进行有效性测试（黑盒测试）
      2. 2．软件配置复查
      3. 3．a测试和b测试
      4. 4．验收测试
      5. 5．确认测试的结果
    1. 4.系统测试
       1. 系统测试（system testing）是将通过确认测试的软件，作为整个计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据、人员等其他系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试。
       2. 系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统定义不符合或与之矛盾的地方。系统测试的测试用例应根据系统的需求分析说明书设计，并在实际使用环境下运行。
    2. 5.测试的类型
       1. （1）功能测试（function testing）：功能测试是在规定的一段时间内运行软件系统的所有功能，以验证这个软件系统有无严重错误。
       2. （2）回归测试（regression testing）：这种测试用于验证对软件修改后有没有引出新的错误，或者说，验证修改后的软件是否仍然满足系统的需求规格说明。
       3. （3）可靠性测试（reliability testing）：如果系统需求说明书中有对可靠性的要求，则需进行可靠性测试。通常使用平均失效间隔时间（MTBF）与因故障而停机的时间（MTTR）来度量系统的可靠性。
       4. （4）强度测试（stress testing）：也称压力测试，是要检查在系统运行环境恶劣的情况下，系统可以运行到何种程度的测试。因此，进行强度测试，需要提供非正常数量、频率或总量资源来运行系统。实际上，这是对软件的“超负荷”环境或临界环境的运行检验。
       5. （5）性能测试（performance testing）：是要检查系统是否满足在需求说明书中规定的性能。特别是对于实时系统或嵌入式系统，软件只满足要求的功能而达不到要求的性能是不可接受的，所以还需要进行性能测试。
       6. （6）恢复测试（recovery testing）：恢复测试是要证实在克服硬件故障（包括掉电、硬件或网络出错等）后，系统能否正常地继续进行工作，并不对系统造成任何损害。
       7. （7）启动/停止测试（startup/shutdown testing）：这类测试的目的是验证在机器启动及关机阶段，软件系统正确处理的能力。包括反复启动软件系统（例如，操作系统自举、网络的启动、应用程序的调用等），以及在尽可能多的情况下关机。
       8. （8）配置测试（configuration testing）：这类测试是要检查计算机系统内各个设备或各种资源之间的相互连接和功能分配中的错误。配置测试主要包括以下3种。
          1. ① 配置命令测试：验证全部配置命令的可操作性（有效 性）；特别对最大配置和最小配置要进行测试。软件配置和硬件配置都要测试。
          2. ② 循环配置测试：证明对每个设备物理与逻辑的、逻辑与功能的每次循环置换配置都能正常工作。
          3. ③ 修复测试：检查每种配置状态及哪个设备是坏的，并用自动的或手工的方式进行配置状态间的转换。
       9. （9）安全性测试（security testing）：检验在系统中已经存在的系统安全性和保密性措施是否发挥作用，有无漏洞。为此要了解破坏安全性的方法和工具，并设计一些模拟测试用例对系统进行测试，力图破坏系统的保护机构以进入系统。
       10. （10）可使用性测试（usability testing）：可使用性测试主要从使用的合理性、方便性等角度对软件系统进行检查，以发现人为因素或使用上的问题。
       11. （11）可支持性测试（supportability testing）：验证系统的支持策略对于公司与用户方面是否切实可行。它所采用的方法是试运行支持过程（如对有错部分打补丁的过程，热线界面等），对其结果进行质量分析，评审诊断工具、维护过程、内部维护文档；衡量修复一个明显错误所需的平均最少时间。还有一种常用的方法是，在发行前把产品交给用户，向用户提供支持服务的计划，从用户处得到对支持服务的反馈。
       12. （12）安装测试（installation testing）：安装测试的目的不是查找软件错误，而是查找安装错误。在安装软件系统时，会有多种选择。要分配和装入文件与程序库，布置适用的硬件配置，进行程序的连接。而安装测试是要查找出在这些安装过程中出现的错误。
       13. （13）互连测试（interoperability testing）：验证两个或多个不同的系统之间的互连性。这类测试对支持标准规格说明，或承诺支持与其他系统互连的软件系统有效。
       14. （14）兼容性测试（compatibility testing）：验证软件产品在不同版本之间的兼容性。有两类基本的兼容性测试：向下兼容和交错兼容。向下兼容测试是测试软件新版本，保留它早期版本的功能的情况；交错兼容测试是要验证共同存在的两个相关但不同的产品之间的兼容性。
       15. （15）容量测试（volume testing）：容量测试是要检验系统的能力最高能达到什么程度。
       16. （16）文档测试（documentation testing）：检查用户文档（如用户手册）的清晰性和精确性。用户文档中所使用的例子必须在测试中一一试过，确保叙述正确无误。

# 软件工程概论11

1. 十一、软件维护
   1. 1.软件维护的定义及类型
      1. 软件维护是指在软件运行/维护阶段对软件产品所进行的修改就是所谓的维护。
      2. 根据维护工作的性质，软件维护的活动可以分为以下4种类型:改正性维护、适应性维护、完善性维护、预防性维护
      3. 1．改正性维护
         1. 改正性维护（corrective maintenance）为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，应进行的诊断和改正错误的过程。例如，改正性维护可以是改正原来程序中开关使用的错误；解决开发时未能测试各种可能情况带来的问题等。
      4. 2．适应性维护
         1. 随着信息技术的飞速发展，软件运行的外部环境（新的硬、软件配置）或数据环境（数据库、数据格式、数据输入/输出方式、数据存储介质）可能发生变化，为了使软件适应这种变化，而修改软件的过程叫做适应性维护）。例如，需要对已运行的软件进行改造，以适应网络环境或已升级改版的操作系统要求。
      5. 3．完善性维护
         1. 为了满足新的功能与性能要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性。这种情况下进行的维护活动叫做完善性维护。例如，完善性维护可能是修改一个计算工资的程序，使其增加新的扣除项目；缩短系统的应答时间，使其达到特定的要求等。
      6. 4．预防性维护
         1. 预防性维护（preventive maintenance）是指把今天的方法学用于昨天的系统以满足明天的需要。也就是说，采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编码和测试。
   2. 2.软件维护的策略
      1. 1．改正性维护
      2. 2．适应性维护
         1. 这一类的维护不可避免，但可以采用以下策略加以控制。
         2. （1）在配置管理时，把硬件、操作系统和其他相关环境因素的可能变化考虑在内，可以减少某些适应性维护的工作量。
         3. （2）把与硬件、操作系统，以及其他外围设备有关的程序归到特定的程序模块中。可把因环境变化而必须修改的程序局部于某些程序模块之中。
         4. （3）使用内部程序列表、外部文件，以及处理的例行程序包，可为维护时修改程序提供方便。
         5. （4）使用面向对象技术，增强软件系统的稳定性，易于修改和移植。
      3. 3．完善性维护
   3. 3.软件维护活动
      1. 1.软件维护申请报告
      2. 2.软件维护工作流程



* + 1. 3.维护档案记录
       1. 内容包括程序名称、源程序语句条数、机器代码指令条数、所用的程序设计语言、程序安装的日期、程序安装后的运行次数、与程序安装后运行次数有关的处理故障次数、程序改变的层次及名称、修改程序所增加的源程序语句条数、修改程序所减少的源程序语句条数、每次修改所付出的“人时”数、修改程序的日期、软件维护人员的姓名、维护申请报告的名称、维护类型、维护开始时间和维护结束时间、花费在维护上的累计“人时”数、维护工作的净收益等。对每项维护任务都应该收集上述数据。
    2. 4.维护评价
       1. 评价维护活动可参考的度量值有：
          1. 每次程序运行时的平均出错次数；
          2. 花费在每类维护上的总“人时”数；
          3. 每个程序、每种语言、每种维护类型的程序平均修改次数；
          4. 因为维护，增加或删除每个源程序语句所花费的平均“人时”数；
          5. 用于每种语言的平均“人时”数；
          6. 维护申请报告的平均处理时间；
          7. 各类维护申请的百分比。
  1. 4.程序修改的三个步骤
     1. 为了正确、有效地进行程序修改，需要经历3个步骤：分析和理解程序、实施修改以及重新验证程序。
     2. 1.分析和理解程序
        1. 经过分析，全面、准确、迅速地理解程序是决定维护成败和质量好坏的关键。在这方面，软件的可理解性和文档的质量非常重要。为此必须：
           1. （1）研究程序的使用环境及有关资料，尽可能得到更多的背景信息；
           2. （2）理解程序的功能和目标；
           3. （3）掌握程序的结构信息，即从程序中细分出若干结构成分，如程序系统结构、控制结构、数据结构和输入/输出结构等；
           4. （4）了解数据流信息，即所涉及的数据来自何处，在哪里被使用；
           5. （5）了解控制流信息，即执行每条路径的结果；
           6. （6）如果设计存在，则可利用它们来帮助画出结构图和高层流程图；
           7. （7）理解程序的操作（使用）要求。
        2. 为了容易地理解程序，要求自顶向下地理解现有源程序的程序结构和数据结构，为此可采用如下几种方法。
           1. （1）分析程序结构图。
           2. （2）数据跟踪。
           3. （3）控制跟踪。可采用符号执行或实际动态跟踪的方法，了解数据是如何从一个输入源到达输出点的。
           4. （4）在分析的过程中，应充分阅读和使用源程序清单和文 档，分析现有文档的合理性。
           5. （5）充分使用由编译程序或汇编程序提供的交叉引用表、符号表，以及其他有用的信息。
           6. （6）如有可能，争取参加开发工作。
     3. 2.修改程序
        1. 1．设计程序的修改计划
           1. 程序的修改计划要考虑人员和资源的安排。修改计划的内容主要包括以下几项：

（1）规格说明信息：数据修改、处理修改、作业控制语言修改、系统之间接口的修改等。

（2）维护资源：新程序版本、测试数据、所需的软件系统、计算机时间等。

（3）人员：程序员、用户相关人员、技术支持人员、厂家联系人、数据录入员等。

（4）提供：纸质、计算机媒体等。

* + - 1. 2．修改代码，以适应变化
         1. （1）正确、有效地编写修改代码；
         2. （2）要谨慎地修改程序，尽量保持程序的风格及格式，要在程序清单上注明改动的指令；
         3. （3）不要匆忙删除程序语句，除非完全肯定它是无用的；
         4. （4）不要试图共用程序中已有的临时变量或工作区，为了避免冲突或混淆用途，应自行设置自己的变量；
         5. （5）插入错误检测语句；
         6. （6）保持详细的维护活动和维护结果记录；
         7. （7）如果程序结构混乱，修改受到干扰，可抛弃程序重新编写。
      2. 3. 修改程序的副作用及其控制
         1. 1．修改代码的副作用

在使用程序设计语言修改源代码时，都可能引入新的错误。例如，删除或修改一个子程序、删除或修改一个标号、删除或修改一个标识符、改变程序代码的时序关系、改变占用存储的大小、改变逻辑运算符、修改文件的打开或关闭、改进程序的执行效率，以及把设计上的改变翻译成代码的改变、为边界条件的逻辑测试做出改变时，都容易引入错误。

* + - * 1. 2．修改数据的副作用

在修改数据结构时，有可能造成软件设计与数据结构不匹配，因而导致软件出错。修改数据的副作用是修改软件信息结构导致的结果。例如，在重新定义局部的或全局的常量、重新定义记录或文件的格式、增大或减小一个数组或高层数据结构的大小、修改全局或公共数据、重新初始化控制标志或指针、重新排列输入/输出或子程序的参数时，容易导致设计与数据不相容的错误。数据副作用可以通过详细的设计文档加以控制。

* + - * 1. 3．修改文档的副作用

**3．修改文档的副作用**

**对数据流、软件结构、模块逻辑或任何其他有关特性进行修改时，必须对相关技术文档进行相应修改。如果对可执行软件的修改不反映在文档里，会产生文档的副作用。例如，对交互输入的顺序或格式进行修改，如果没有正确地记入文档中，可能引起重大的问题。过时的文档内容、索引和文本可能造成冲突，引起用户业务的失败和不满。因此，必须在软件交付之前对整个软件配置进行评审，以减少文档的副作用。**

* + - * 1. 为了控制因修改而引起的副作用，要做到：

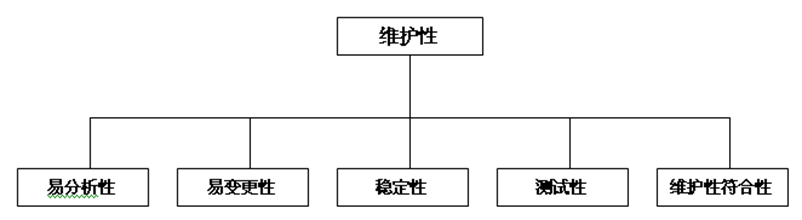
（1）按模块把修改分组；

（2）自顶向下地安排被修改模块的顺序；

（3）每次修改一个模块；

（4）对于每个修改了的模块，在安排修改下一个模块之前，要确定这个修改的副作用，可以使用交叉引用表、存储映象表、执行流程跟踪等。

* + 1. 3.重新验证程序
       1. 1．静态确认
          1. 修改的软件，通常伴随着引起新的错误的危险。为了能够做出正确的判定，验证修改后的程序至少需要两个人参加
       2. 2．确认测试
          1. 在充分进行了以上确认的基础上，要用计算机对修改程序进行确认测试。
          2. （1）确认测试顺序：先对修改部分进行测试，然后隔离修改部分，测试程序的未修改部分，最后再把它们集成起来进行测试。这种测试称为回归测试。
          3. （2）准备标准的测试用例。
          4. （3）充分利用软件工具帮助重新验证过程。
          5. （4）在重新确认过程中，需邀请用户参加。
       3. 3．维护后的验收
          1. 在交付新软件之前，维护主管部门要检验：
          2. （1）全部文档是否完备，并已更新；
          3. （2）所有测试用例和测试结果已经正确记载；
          4. （3）记录软件配置所有副本的工作已经完成；
          5. （4）维护工序和责任是明确的。
  1. 5.软件的维护性
     1. 软件维护性是指当对软件实施各种类型的维护而进行修改时，软件产品可被修改的能力。
     2. 软件维护的子特性：



* 1. 6.提高软件维护性的方法
     1. 1.使用提高软件质量的技术和工具
        1. 1．模块化
           1. 模块化技术的优点是如果需要改变某个模块的功能，则只要改变这个模块，对其他模块影响很小；如果需要增加程序的某些功能，则仅需增加完成这些功能的新的模块或模块层；程序的测试与重复测试比较容易；程序错误易于定位和纠正；容易提高程序效率。
        2. 2．结构化程序设计
           1. 结构化程序设计不仅使得模块结构标准化，而且将模块间的相互作用也标准化了，因而把模块化又向前推进了一步。采用结构化程序设计可以获得良好的程序结构。
        3. 3．使用结构化程序设计技术，提高现有系统的可维护性
           1. （1）采用备用件的方法——当要修改某一个模块时，用一个新的结构良好的模块替换掉整个模块。
           2. （2）采用自动重建结构和重新格式化的工具（结构更新技术）。
           3. （3）改进现有程序的不完善的文档。
           4. （4）使用结构化程序设计方法实现新的子系统。
           5. （5）采用结构化小组。
     2. 2.实施开发阶段产品的维护性审查
        1. 质量保证审查除了保证软件得到适当的质量外，还可以用来检测在开发和维护阶段内发生的质量变化。一旦检测出问题来，就可以采取措施纠正，以控制不断增长的软件维护成本。
        2. 为了保证软件的可维护性，有4种类型的软件审查：检查点审查、验收检查、周期性的维护审查、对软件包进行检查
        3. 1．检查点审查
           1. 保证软件质量的最佳方法是在软件开发的最初阶段就把质量要求考虑进去，并在开发过程每一个阶段的终点，设置检查点进行检查。
           2. 检查的目的是要证实，已开发的软件是否符合标准，是否满足规定的质量需求。
        4. 2．验收检查
           1. 验收检查是一个特殊的检查点的检查，是交付使用前的最后一次检查，是软件投入运行之前保证可维护性的最后机会。以下是验收检查必须遵循的最小验收标准。
           2. （1）需求和规范标准

① 需求应当以可测试的术语进行书写，按优先次序排列和定义。

② 区分必须的、任选的、将来的需求。

③ 包括对系统运行时的计算机设备的需求；对维护、测试、操作，以及维护人员的需求；对测试工具等的需求。

* + - * 1. （2）设计标准

① 程序应设计成分层的模块结构。每个模块应完成唯一的功能，并达到高内聚、低耦合。

② 通过一些知道预期变化的实例，说明设计的可扩充性、可缩减性和可适应性。

* + - * 1. （3）源代码标准

① 尽可能使用程序设计语言的标准版本。

② 所有的代码都必须具有良好的结构。

③ 所有的代码都必须文档化，在注释中说明它的输入、输出，以及便于测试/再测试的一些特点与风格。

* + - * 1. （4）文档标准

文档中应说明程序的输入/输出、使用的方法/算法、错误恢复方法、所有参数的范围、默认条件等。

* + - 1. 3．周期性地维护审查
         1. 检查点复查和验收检查，可用来保证新软件系统的可维护性。对已有的软件系统，则应当进行周期性的维护检查。
         2. 软件在运行期间，必须对软件做周期性的维护审查，以跟踪软件质量的变化。
         3. 周期性维护审查实际上是开发阶段检查点复查的继续，并且采用的检查方法、检查内容都是相同的。
      2. 4．对软件包进行检查
         1. 软件包是一种标准化了的、可为不同单位、不同用户使用的封装软件。
         2. 使用单位的维护人员首先要仔细分析、研究开发商提供的用户手册、操作手册、培训教程、新版本说明、计算机环境要求书，以及开发商提供的验收测试报告等，在此基础上，深入了解本单位的希望和要求，编制软件包的检验程序。该检验程序检查软件包程序所执行的功能是否与用户的要求和条件相一致。
    1. 3.改进文档
       1. 程序文档是对程序总目标、程序各组成部分之间的关系、程序设计策略、程序实现过程的历史数据等的说明和补充。程序文档对提高程序的可理解性有着十分重要作用。在软件维护阶段，利用历史文档，可以大大简化维护工作。
       2. 历史文档有如下3种：
          1. （1）系统开发日志：它记录了项目的开发原则、开发目标、优先次序、选择某种设计方案的理由、决策策略、使用的测试技术和工具、每天出现的问题、计划的成功和失败之处等。
          2. （2）错误记载：它把出错的历史情况记录下来，对于预测今后可能发生的错误类型及出错频率有很大帮助。也有助于维护人员查明出现故障的程序或模块，以便去修改或替换它们。
          3. （3）系统维护日志：记录了在维护阶段有关系统修改和修改目的的信息。包括修改的宗旨、修改的策略、存在的问题、问题所在的位置、解决问题的办法、修改要求和说明、注意事项、新版本说明等信息。