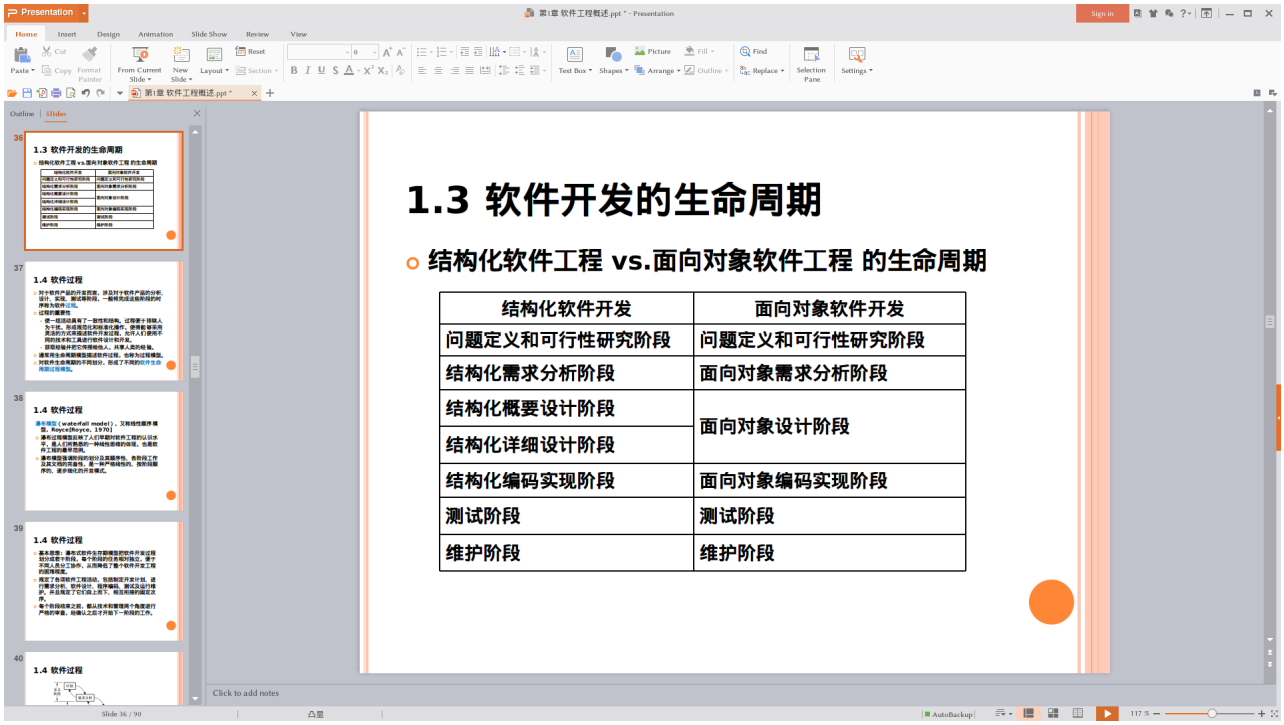
1. 软件工程概述

- **什么是软件？**
  - 能够完成预定功能和性能的可执行的计算机程序和使程序正常执行所需要的数据，加上描述程序的操作和使用的文档
- 什么是软件危机？软件危机的表现有哪些？产生软件危机的原因是什么？
  - 软件危机指在计算机软件的开发和维护中所遇到的一系列严重问题，概括来讲包含两个方面：如何开发软件和如何维护软件
  - **软件危机的表现**
    1. 对软件开发成本和进度的估计常常很不准确。（成本增加、项目延期、项目中止）
    2. 用户对“已完成的”软件系统不满意的现象经常发生。（与需求不符）
    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. 软件通常没有相关的文档资料。
    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跟不上计算机应用迅速普及深入的趋势。（供不应求）
  - 软件危机产生的原因
    1. 软件本身的特点
       - 软件开发进展情况较难衡量
       - 软件开发质量难以评价
       - 管理和控制软件开发过程相当困难
       - 软件没有“磨损”概念，软件维护通常意味着改进或修改原来的设计（软件维护费用数倍于开发费用）
    2. 软件开发人员的错误观点
       - 忽视软件需求分析
       - 认为软件开发就是编写程序
       - 忽视软件维护
- **什么是软件工程?**
  - 采用工程的概念、原理、技术和方法来开发和维护计算机软件，将工程管理技术的成功经验和思想与具体的软件开发过程、研究技术相结合，形成一整套适合于计算机软件开发的方法、规范和技术。
- 软件工程的原则
  - 抽象 信息隐蔽 模块化 局部化 确定性 一致性 完备性 可验证性
- **结构化软件工程与面向对象软件工程的概念以及异同点** 优缺点
  1. 结构化软件工程指采用结构化技术(结构化分析、设计、实现)来完成软件开发的各项任务，这种方法把软件生命周期的全过程划分为若干个阶段，然后顺序地完成每个阶段的任务
     - 优点：每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作
     - 缺点：当软件规模庞大时，把数据和操作人为的分为两个独立的部分，增加了软件开发与维护的难度
  2. 面向对象软件工程指
     - 优点：设计较好地的实现了“解空间”与“问题空间”的一致性。“对象 + 消息”的机制取代了“数据结构 + 算法”的思路；有效地降低了软件的复杂性，简化了软件的开发; 有利于维护
- **什么是软件生命周期**，七个阶段是什么、四阶段是什么
  - 软件生命周期：软件从决定进行开发到最终退役所经历的一系列步骤与过程。表示一个软件产品从形成概念开始，经过开发过程，交付使用，在使用中不断维护，直到最后被淘汰，让位于新的软件产品的全周期。是包含多个子过程和多个阶段的一系列相关活动的全周期。

- 七个阶段：问题定义和可行性研究 需求分析 概要设计 详细设计 编码 测试 维护
- 四个阶段：软件定义 软件开发 软件测试 软件使用与维护

- **结构化软件工程和面向对象软件工程生命周期**

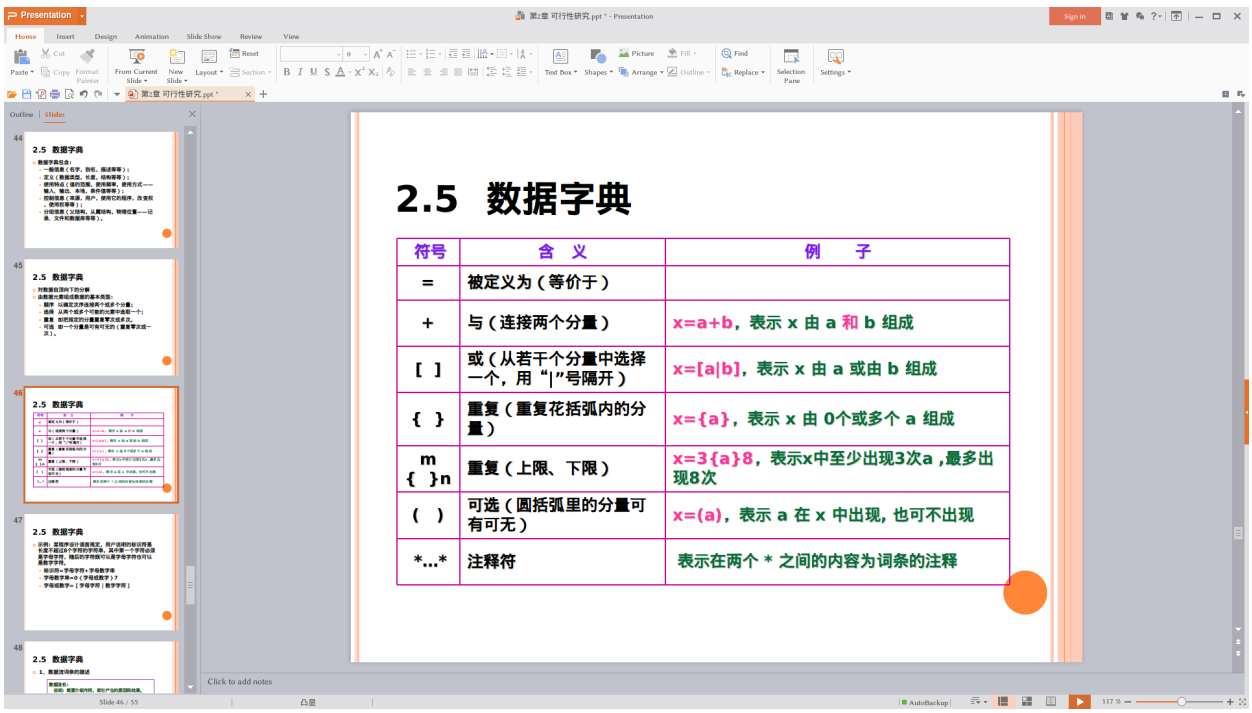


- 什么是软件过程、
  - 对于软件产品的开发而言，涉及对于软件产品的分析、设计、实现、测试等阶段，一般将完成这些阶段的时序称为软件过程
- **瀑布模型、螺旋模型、V模型的概念、特点与优缺点、每种模型的适用范围**

1. 瀑布模型：把软件开发过程划分成若干阶段，每个阶段的任务相对独立，便于不同人员分工协作，从而降低了整个软件开发工程的困难程度。
   - 优点：简单、历史悠久，应用广、有一套成熟的开发方法与工具
   - 缺点：任何一个阶段都不可能在下一阶段开始之前完全结束; 确定了需求分析的绝对重要性，但是在实践中要想获得完善的需求说明是非常困难的; 反馈信息慢； 开发者常常被不必要地耽搁。
   - 适用范围：很小的项目、有熟悉的类似项目。当需求确定，工作能够按线性方式完成时，该模型很有用
2. 螺旋模型：在瀑布模型的每一个开发阶段前引入一个非常严格的风险识别、风险分析和风险控制，把软件项目分解成一个个小项目。每个小项目都标识一个或多个主要风险，直到所有的主要风险因素都被确定。螺旋模型4个象限的活动包括制定计划、风险分析、实施工程、客户评估。**沿螺旋线自内向外每旋转一圈，便开发出一个更为完善的、新的软件版本。依据前一个版本的结果构造新的版本，这个不断重复迭代的过程形成了一个螺旋上升的路径。**
   - 优点：每次迭代中收集过程中产生的各种度量数据; 将软件质量作为目标; 解决了做太多测试或未做足够测试所带来的点风险; 软件维护
   - 缺点：管理过程 、文档一致性; 限制条件：适用于内部软件开发
   - 适用范围：大规模软件项目
3. V模型：
   - 特点：实现了测试设计和测试执行相分离
   - 优点：非常明确的标明了测试过程中存在的不同等级，并且清晰的描述了这些测试阶段和开发过程期间各阶段的对应关系
   - 缺点：仅仅把测试作为在编码之后的一个阶段，是针对程序进行的寻找错误的活动，而忽视了测试活动对需求分析，系统设计等活动的验证与确认的功能; 仅注重动态测试，未涉及静态测试技术

2. 可行性研究

- 为什么需要进行可行性研究
  - 用最小的代价在最短的时间内确定问题是否能够解决
- 可行性研究包括哪些任务？
  1. 技术可行性
  2. 经济可行性
  3. 操作可行性：
  4. 社会可行性
- 什么是数据流图？ **两种表示法熟练掌握**
  - 数据流图：一种图形化技术，它描绘信息流和数据从输入移动到输出的过程中所经受的变换，是系统逻辑功能的图形表示
  - 数据流图中的基本元素：外部实体、过程、数据流、数据存储四种
  - *数据字典通常被用来描述DFD数据流的详细内容，数据存储的详细内容通常也是用数据字典来进行描述的。数据存储是处于静止状态的数据，数据流是处于运动中的数据
  - **数据流图(会画图)**分层：上下文图(顶层图) 0层图 1层图。。。
  - **上下文图：将整个系统看做是一个过程，这个过程实现系统的所有功能 ，是系统功能的最高抽象（1.存在且仅存在一个过程，通常编号为0； 2. 需要表示出所有和系统交互的外部实体，并描述交互的数据流，包括系统输入和系统输出 3.不会出现数据存储实例 ）**
  - **0层图：是上下文图中单一过程的细节描述，是对该单一过程的第一次功能分解，0层图应该被描述的简洁、清晰**
  - **1层图：在低于0层图的子图上通常不显示外部实体 ，子图中过程的编号要以父过程的编号为前缀：1.2、1.2.1.....**
  - **特别需要注意的点：顶层图中过程编号为0， 0层图过程标号为1 2 3 4 ， 1层图编号为1.1 1.2 。。。 2.1 2.1.。。 ； 顶层图中没有数据存储实例，高层图必然包含低层图的所有输入与输出，N>=1时，去掉所有外部实体**
- 数据流图和数据字典共同构成系统的逻辑模型。 **数据字典的作用是在软件分析和设计的过程中提供关于数据的描述信息。只需掌握**



3. 需求分析

- 需求分析的任务
  - 完全弄清用户（顾客）对软件系统的确切要求，用规范的格式表达出来。给出一个将要用软件来解决的一个问题的初始定义
- 需求分为两种类型：功能需求与非功能需求
  - 功能需求分三种：业务需求 用户需求 系统需求
- 可以将整个软件需求工程研究领域划分为需求开发和需求管理两部分，需求开发可进一步分为：需求获取（elicitation）、需求分析（analysis）、需求规格说明（specification）和需求验证（verification）四个阶段
- 需求获取的方法：面谈 问卷调查 专题讨论会等
- **实体关系图(会画图)**：
  - 基本元素 实体（具有相同特征和属性的实例集的类别描述） 关系 属性
  - 属性是实体的特征，不是数据。属性会以一定的形式存在，这种存在才是数据，被称为属性的值
  - ppt上有个例子需要看一下

4. 概要设计/总体设计

- 软件设计是把软件需求变为软件的具体方案， 包括两个阶段：概要设计（总体设计）和详细设计
- 概要设计的任务
  - 确定软件的总体结构。在概要设计中，将软件系统分解为多个模块，并确定每个模块的功能和模块之间的外部接口
- **什么是模块？**
  - 模块是软件设计的最小单位。（单元）， 是具有一定功能的可以用名词调用的程序语句集合。具有一些基本属性，如：明确的功能、规格定义，与其他部分明确的接口定义等，可以清晰地与同一程序的其他部分划分开来
- **什么是模块化， 优点是什么？**
  - 把程序划分成若干个模块，分别实现。 对每个问题都存在着某个最佳模块数目，它能使得软件成本最小
  - 优点：
    - 一个复杂问题分割成若干个容易解决、容易管理的小问题后更易于求解。（**分而治之**）
    - 可以使软件结构清晰，容易设计、容易阅读和理解、容易测试和调试。
    - 提高软件的可靠性。
    - 有助于软件开发工程的组织管理。
- 模块独立性的度量标准：**耦合（coupling）和内聚（cohesion）的概念**
  - 耦合是模块之间相对独立性的量度（是对软件程序结构中各个模块之间相互关联程度的一种度量。）低耦合更好。
  - 内聚是模块功能相对强度的量度(反映一个模块内部各成分彼此结合的紧密程度)
- **七种耦合性是什么？**
  - 间接耦合：两模块中任一个都不依赖对方而能独立工作，也就是说两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。
  - 数据耦合：两模块间通过参数交换信息，而传递的信息仅限于数据，或者说一个模块访问另一个模块时，彼此之间是通过数据参数（而不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的。
  - 特征耦合/标记耦合: 两个模块都要使用同一数据结构的一部分，不是采用全程公共数据区共享，而是通过模块接口传递数据结构的一部分（不是简单的数据）。
  - 控制耦合: 两模块间通过参数交换信息，而传递的信息中含有控制信息（控制参数）。
  - 外部耦合（external coupling）：若干模块均与同一个外部环境关联。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该全局变量的信息。
  - 公共耦合（common coupling）：若干模块通过全局的数据环境相互作用时（一组模块都访问同一个公共数据环境）。

    - 内容耦合: 一个模块直接访问另一个模块的内部数据、一个模块不通过正常入口转到另一模块内部等
- **设计软件时应尽量使用数据耦合，减少控制耦合，限制环境耦合和公共耦合，杜绝内容耦合。（低耦合）**
- **七级内聚性指什么？**

  低级内聚

    - 偶然性内聚（coincidental cohesion）：一个模块内各成分为完成一组功能而组合在一起，它们相互之间即使有关系，也很松散或者模块内各部分之间就没有联系。
    - 逻辑性内聚（logical cohesion）：一个模块完成的诸任务逻辑上相关。
    - 时间性内聚/经典内聚（temporal cohesion）：一个模块包含的诸任务必须在同一时间段内执行。

  中级内聚

    - 过程性内聚（procedural cohesion）：模块内各个组成部分的处理动作各不相同、彼此相关，并且受同一控制流支配，必须按特定的次序执行。
    - 通信性内聚（communicational cohesion）：一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据。

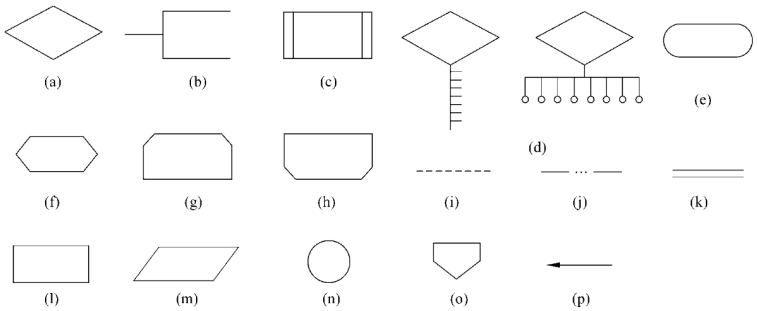
  高级内聚

    - 顺序性内聚（sequential cohesion）：一个模块内的各个组成部分顺序执行几个处理动作，前一个处理动作产生的输出数据是下一个处理工作的输入数据。
    - 功能性内聚（functional cohesion）：模块内所有成分形成一个整体，完成单个功能。
- **设计软件应尽可能使模块达到功能内聚**
- **概要设计只要求sc图, 会把数据流图转化为sc图**
- **内聚性与耦合性的异同点比较？**
- **什么是变换分析？变换分析的步骤**
    - 把具有变换流特点的数据流图按预先确定的模式映射成软件结构。
    -

5. 详细设计/过程设计

- 什么是详细设计
    - 在概要设计阶段成果的基础上，考虑如何实现定义的软件系统，直到对系统中的每个模块给出足够详细的过程描述。
    - 在详细设计中，把一个模块的功能逐步分解细化为一系列具体的处理步骤，即确定模块采用的算法和块内数据结构。
- 详细设计的任务
    1. 为每个模块确定采用的算法 ，即程序流程设计
    2. 确定每个模块使用的数据结构 ，即数据设计
    3. 确定模块接口的细节
    4. 其他设计：人机界面设计、（数据库）物理设计等
    5. 制定测试计划
    6. 编写文档：“详细设计说明书”
    7. 复审
- **4种图之间的转换**
    1. 程序流程图
        - 方框表示一个处理步骤
        - 菱形表示一个逻辑条件
        - 箭头表示控制流向

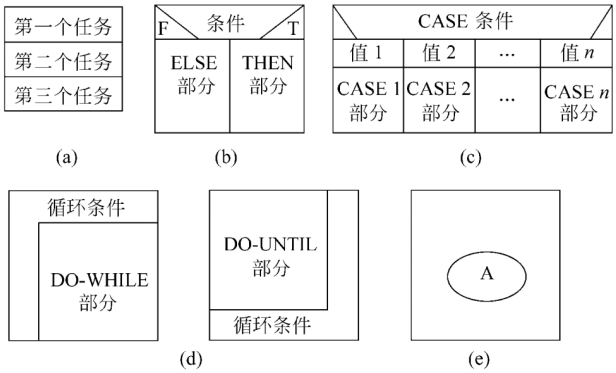
5.2 详细设计的工具



2. N-S流程图、

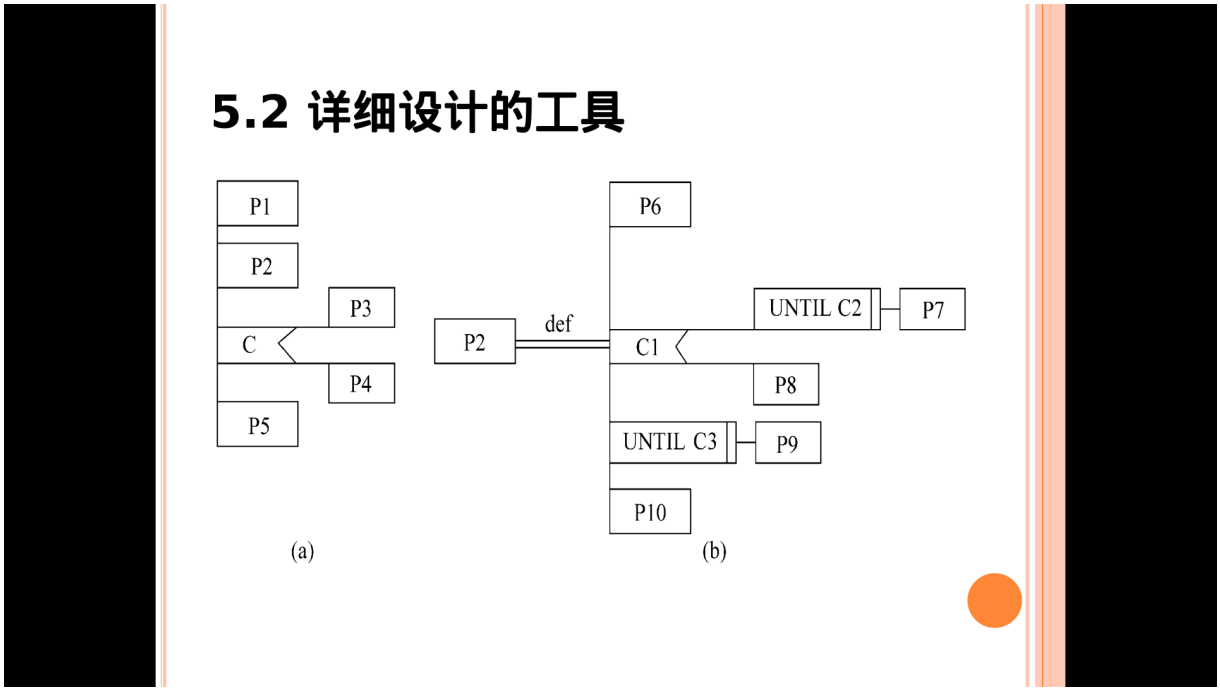
- 每个处理步骤都用一个矩形框来表示，这些处理步骤可以是语句或语句序列。
- 在需要时，矩形框中还可以嵌套另一个矩形框

5.2 详细设计的工具



3. PAD图

- 图中最左纵线是程序的主干线，即程序的第一层结构。
- 其后每增加一个层次，图向右扩展一条纵线。

- 

4. 过程设计语言PDL

- 判定表与判定数了解
- **面向数据结构的设计方法与面向数据流的设计方法异同点对比**
  - 共同点：都是数据信息驱动的，都试图将数据表示转换成软件表示
  - 不同点：面向数据结构的设计不利用数据流图，而根据数据结构的表示来设计。面向数据结构的设计方法定义了一组以数据结构为指导的映射过程，它根据输入、输出的数据结构，按一定的规则映射成软件的过程描述，最终目标是生成软件过程的描述，即程序结构，而不是软件的体系结构。而面向数据流的设计方法是根据数据流图来设计，最终目标是建立软件体系结构的描述，即软件模块的层次结构。

6. 编码 不考

7. 测试 重点

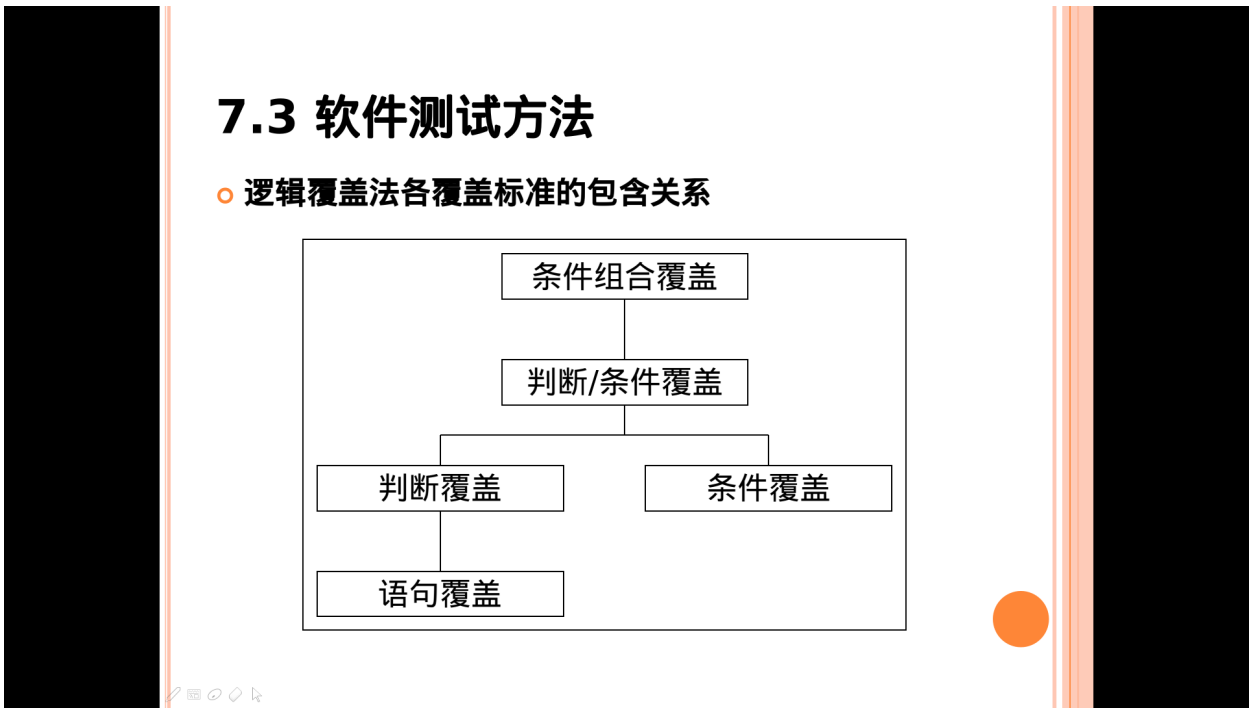
- **什么是软件测试**
  - 软件测试是使用人工和自动手段来运行或测试某个系统的过程，其目的在于检验它是否满足规定的需求或弄清楚预期结果与实际结果之间的差别。
- **软件测试方法有哪些分类， 之间有什么异同点**
  - 按测试技术
    - 白盒测试：通过对程序内部结构的分析、检测来寻找问题
    - 黑盒测试：通过软件的外部表现来发现其缺陷和错误， 即测试人员只依据程序的需求规格说明书和用户手册，检查程序的功能是否符合它的功能说明，以及性能是否满足用户的要求
  - 按测试方式
    - 静态测试：不执行被测试软件，而对需求分析说明书、软件设计说明书、源程序做结构检查、流程图分析等找出软件缺陷
    - 动态测试：执行被测程序，通过执行结果分析软件可能出现的缺陷
  - 按测试阶段
    - 单元测试：又称为模块测试，对软件设计的最小单元——模块进行正确性检验的测试工作
      - 目的：测试模块在语法、格式和逻辑上的缺陷。
    - 集成测试：又称为组装/联合测试，按设计要求把通过单元测试的各个模块组装在一起之后所进行的测试。

        - 目的：检查模块间的接口关系，以便发现与接口有关的各种缺陷
    - 系统测试：将已经集成好的软件系统置于实际运行环境中所进行的测试
        - 目的：根据需求分析时确定的标准检验软件是否满足功能、行为、性能和系统协调性等方面的要求
    - 验收测试：又称为确认测试，是软件开发结束后，用户对软件产品投入实际应用前，进行的最后一次质量检验活动。它要回答开发的软件产品是否符合预期的各项要求，以及用户能否接受的问题。包括α测试和β测试
        - 目的：验证软件功能的正确性和需求的符合性
- 按测试实施组织
    - 开发方测试：例如，α测试
    - 用户方测试：例如， β测试
    - 第三方测试：又称为独立测试，由在技术、管理和财务上和开发方和用户方相对独立的组织进行的测试。软件质量工程强调开展独立的验证和确认工作
- **回归测试的定义：指软件系统被修改或扩充后重新进行的测试， 是为了保证对软件所做的修改正确，且没有引入新的错误而重复进行的测试**
- **什么是单元测试**？何时进行单元测试？
    - 单元测试是对软件基本组成单元进行测试。检验程序最小单元（具有基本属性）的实现是否和该单元的说明一致，有无错误
    - 单元测试常常是和代码编写工作同时进行的，在完成了程序编写、复查和语法正确性验证后，就应进行单元测试用例设计
- **单元测试测试哪5个方面**
    - 模块接口、局部数据结构、边界条件、独立的路径、错误处理
- 在单元测试时，如果模块不是独立的程序，要考虑它和外界的联系，需要设置一些辅助测试模块。辅助测试模块有两种
    - 驱动模块：用来模拟被测试模块的上一级模块，相当于被测模块的主程序。它接收数据，将相关数据传送给被测模块，启动被测模块，并打印出相应的结果。
    - 桩模块（Stub）（又称存根模块或连接模块）：用以代替被测模块所调用的子模块，相当于被测模块工作过程中所调用的模块
- **测试环境：被测模块、与被测模块相关的驱动模块和桩模块共同构成了一个测试环境**
- **什么是集成测试？**
    - 集成测试是在单元测试的基础上，将所有模块按照概要设计要求组装成为子系统或系统所进行的测试。
- **集成策略有哪些**？优缺点看ppt
    - 一次性集成方式
        - 大爆炸集成 ：先对每一个子模块进行测试（单元测试阶段），然后将所有模块一次性的全部集成起来进行集成测试 。（整体拼装）
    - 增殖式集成方式（增量式集成）
        - 自顶向下集成：模块集成的顺序是首先集成主控模块（主程序），然后依照控制层次结构向下进行集成。从属于主控模块的模块按深度优先方式（纵向）或者广度优先方式（横向）集成到结构中去
        - 自底向上集成：逐步集成和逐步测试的工作是按结构图自下而上进行的，即从程序模块结构的最底层模块开始集成和测试。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）已经集成并测试完成，所以不再需要使用桩模块进行辅助测试。在模块的测试过程中需要从子模块得到的信息可以直接运行子模块得到
        - 三明治集成：三明治集成是一种混合增殖式测试策略，综合了自顶向下和自底向上两种集成方法，把系统划分成三层，中间一层为目标层，目标层上采用自顶向下集成，目标层下采用自底向上集成
- **验收测试通常有两种alpha、beta测试，它们的定义、异同点？**
    - alpha测试：软件开发公司组织内部人员模拟各类用户行对即将面市软件产品（称为α版本）进行测试，试图发现错误并修正

- beta测试：软件开发公司组织各方面的典型用户在日常工作中实际使用β版本，并要求用户报告异常情况、提出批评意见。然后软件开发公司再对β版本进行改错和完善。经过α测试调整的软件产品称为β版本
- α测试：早期的、不稳定的软件版本所进行的验收测试，受控的实验室测试
- β测试：晚期的、更加稳定的软件版本所进行的验收测试，不受控的非实验室测试

- **4种等价类划分方法(会用，可能考大题，画等价类表，看ppt)**

1. 弱一般等价类测试用例设计法
   - 测试用例从每个有效等价类中选取一个值
   - 变量对应有效等价类数的最大值
2. 强一般等价类测试用例设计法
   - 测试用例从有效等价类的笛卡儿积中选取
   - 变量对应有效等价类数的乘积
3. 弱健壮等价类测试用例设计法
   - 对于有效等价类，从每个有效等价类选取一个值
   - 对于无效等价类，使用一个无效值，并保持其余的值都是有效的
   - 变量对应有效等价类数的最大值 +各个变量的无效等价类数之和
4. 强健壮等价类测试用例设计法
   - 测试用例从所有等价类（包括有效和无效等价类）笛卡儿乘积中选取
   - （变量对应有效等价类数+变量的无效等价类数）的乘积

- 什么是**边界值分析法**：对输入或输出的边界值进行测试的一种黑盒测试方法
- 为什么使用边界值分析法：大量的错误是发生在输入或输出范围的边界上（极值附近），而不是发生在输入输出范围的内部
- **逻辑覆盖法**：
  - 语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖
  - 

8. 软件维护

- **什么是软件维护**？
  - 软件维护就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。
- **软件维护的分类有哪四种**？

- 改正性维护
- 适应性维护
- 完善性维护
- 预防性维护。

9. 面向对象方法

- **面向对象(OOA)模型的三个层次**
  - 对象层：发现对象类
  - 特征层：定义属性和服务
  - 关系层：定义对象类之间的关系
- 复杂系统的OOA模型的5个层次
  - 类和对象层
  - 属性层
  - 服务层
  - 结构层
  - 主题层

10. 统一建模语言

- **UML的9种模型图**
  - 类图：Class Diagram
  - 对象图：Object Diagram
  - 构件图： Component Diagram
  - 部署图： Deployment Diagram
  - 用例图：Use Case Diagram
  - 活动图： Activity Diagram
  - 状态图： Statechart Diagram
  - 顺序图： Sequence Diagram
  - 协作图： Collaboration Diagram
- **UML的5种视图， 核心是用例视图**
  - 用例视图：从系统外部的操作者的角度描述系统的功能需求。
  - 逻辑视图：描述系统内部的静态结构和动态行为，即从内部描述如何设计实现系统功能。
  - 构件视图：描述系统由哪些程序构件所组成
  - 并发视图（进程视图）：描述系统的并发性，强调并发系统中存在的各种通信和同步问题。
  - 配置视图（部署视图）：描述系统的软件和各种硬件设备之间的配置关系。
- **用例图与类图的大题**
- 特殊的关联关系
  - 聚集关联（聚合， aggregation ）：用于表示类的对象之间的关系是整体与部分的关系。“部分”对象可以是任意“整体”对象的一部分，表示事物的整体-部分关系较弱的情况
  - 组成关联（组合，composition）：一种更强形式的关联，代表整体的组合对象有管理它的部分对象的特有责任，如部分对象的分配和解除分配。整体拥有各部分，部分与整体共存，如整体不存在了，部分也会随之消失

特征	正常关联	共享聚集	组合聚集
UML标记	实线	加空心菱形	加黑色菱形
拥有关系	无	弱	强
多重性	任意	任意	聚集必为 1
传递性	无	有	有
传递方向	无	整体到部分	整体到部分

- 泛化关联（继承关系， generalization ）：定义了类和包间的“一般-特殊”的分类关系
- 依赖关系：出两个或多个模型元素之间语义上的关系。它表示被依赖元素的变化会要求或指示依赖元素的改变
-
-