Summarized by LPX

1软件工程概述（2*）

1.1软件工程的发展历程

软件危机（1*）

定义：软件危机是指在计算机软件的开发和维护的过程中所遇到的一系列严重的问题。

表现：

1.软件开发进度和成本难以控制 eg.伦敦股票交易系统

2.软件产品难以满足用户需求。 eg.美国军方购买软件

3. 软件质量难以保证。 eg.美苏火箭爆炸

4.软件产品难以进行维护。 eg.软件维护费用的占比上升

5.软件的文档资料难以管理。

6.软件的生产率难以得到提高

出现的原因

一方面是软件自身的特点

另一方面是开发软件和使用软件的人员

1.2软件工程的概念（2*）

软件工程的定义：软件工程是把系统的，规范的，可度量的途径用于软件开发、运行和维护的全过程，以及对上述方法的研究。

软件工程三要素：方法、工具、过程

涉及软件开发的 工程学和软件生产过程

软件工程的目标：跟踪最新的软件技术发展，修改和制定新的软件开发活动规则，提高和规范软件管理的效率和可操作性，确保软件质量，提高软件生产率，开发出满足用户需求，并最终实现软件的工业化生产。

软件的概念：软件是计算机中与硬件相互依存的另一部分，它包括程序、数据以及相关文档的完整集合。

1.3软件与软件过程

软件过程的概念（2*）

软件过程是由组织或项目使用的，用以计划、管理、执行、监控和改进其软件相关活动的过程或者过程集合。

软件生命周期

需求分析（问题定义与可行性分析）

设计（概要设计、详细设计、实现与测试）

应用

维护（灭亡）

1.4软件过程模型（3*）

1.瀑布模型

特点：自顶向下，每一阶段的开始是上一阶段的结束（单向），

优点：简单，总揽全局，明确需求，严格规范，过程严谨

缺点：难以世应需求的变更

2.原型模型

特点：快速，用户不需要等软件全部实现就能够看见使用软件；符合用户预期。

优点：通过对原型的开发和讨论，逐步明确问题定义和系统需求

缺点：原型的开发会增加系统成本

3.增量模型

特点：非整体开发模型，将整体分为若干部分，每一部分都按照瀑布模型进行开发。

优点：系统核心功能明确，实现最小可用为前提，再后续扩展。

缺点 ：需要各类人员广泛参与，增加了管理的难度。

4.螺旋模型

特点：原型模型、瀑布模型的结合体，首次引入风险分析机制（每阶段结束都要进行风险分析），适合大型复杂项目的开发

四个象限

制定计划

风险分析

实施工程

客户评估

优点：增加风险分析机制，最大限度降低风险

缺点：风险分析策略需要分析人员有经验，有一定的难度。

5.喷泉模型

特点：每个阶段相互重叠，不严格的阶段划分，对象驱动，适合面向对象软件开发，支持软件重用

缺点：各阶段并行，导致较多的软件人员参与到了开发中，给管理过程带来了一定的困难

6.敏捷过程模型

是对一类软件开发过程的总称，只要符合敏捷价值观，就是敏捷过程模型

敏捷过程价值观

个体和交互胜过过程和工具

可以工作的软件胜过面面俱到的文档

客户合作胜过合同谈判

相应变化胜过遵循计划

优点：快速开发，及时响应用户反馈

缺点：缺乏细致的准备，会使后续的维护更加困难

7.渐进交付迭代模型

优点：通过不断地演进，往复，完成软件系统的实施过程

缺点：对大型系统难以做到最小应用开发，也不适用于需求分析明确的软件

MSF模型

同步和稳定的策略推动项目和技术的成功

缺点：较少在微软之外的公司推广

2软件需求工程（4*）

2.1软件需求的基本概念

软件需求是准确回答”系统必须做什么“的问题

2.1.1软件需求的基本概念（2*）

需求分析的基本任务是：
- 定义软件的适用领域和必须满足的约束；
- 确定系统功能、性能、领域等内容；
- 确定软件与其他部分间的接口和通信；
- 建立数据模型、功能模型和行为模型；
- 最终完成需求规格说明，并经过技术审查和管理复审，用作评价确认测试和质量评估的依据；

软件需求工程：
- 1.确定系统将要实现的各项需求　系统中最主要，最基本的要求是功能需求，是用户体验中最重要的部分。
- 2.数据分析　将数据从物理实体映射为逻辑实体，分析逻辑数据间的连接和视图模型
- 3.定义逻辑模型　把问题域问题转化为信息域。系统的逻辑模型与软件开发方法有关
  - 结构化开发方法
    - 数据流图
    - 数据字典
  - 面向对象开发方法
    - 用例图
    - 类-对象图
- 4.适应需求变更　基线是需求变更的界线。
  - 基线定义之前，能够随时进行需求变更
  - 基线定义之后变更，则需要重新进行审查和复审

2.1.2需求分析的原则

1.软件人员要从用户的角度考虑软件需求
2.以流程为主线
3.尽量重用
- 重用能提高开发效率，节约成本，更重要的是能提高软件质量
- 需求分析阶段，重用设计模式是考虑的重点。

4.划分需求的优先级
5.需求变更要及时反馈

2.1.3需求分析的内容

功能需求
- 描述系统提供的服务和在特定条件下的行为，包括系统登录、输入输出、响应、异常等。
- 有时还需说明系统不该做什么。
- 软件的功能描述应该满足完整性和一致性
  - 完整性：需求分析覆盖用户提出的所有需求
  - 一致性：要求功能需求中不能出现矛盾之处

性能需求　软件系统必须满足时间上和空间上的约束
领域需求　与软件系统的具体应用范围有关，是对需求中的功能或数据在领域上需要的特别是先，具有特殊性
其他需求　与软件系统有关的外在约束，如法律需求、道德需求、外部数据交换需求、预期需求。

2.2可行性分析（1*）

力求用最小的代价、在尽可能短的时间内确定问题是否能够被解决

经济可行性
- 回答”系统的经济效益能超过开发成本吗“。
- 运用软件成本预算估计、成本/效益分析等方法

软件成本估计技术
- 代码行技术
- 功能点技术

技术可行性
- 回答”现有技术能实现这个系统吗“
- 从问题的复杂性、现有技术、技术所需代价（如开发环境）、技术风险等四个方面出发

操作可行性　回答”系统的操作方式符合用户操作流程和方式吗“
法律可行性

2.3需求工程的过程

需求工程在进行可行性研究之后，通过需求获取、需求分析与建模、需求评审的迭代过程，对需求工程的不断地演进，最终得到确认的**需求规格说明。**

需求工程
- 1.需求获取　采用需求获取技术，得到用自然语言描述的问题的陈述和功能陈述
- 2.需求分析与建模　以需求获取文档为基础，逐步细化软件功能。
- 3.需求评审　需求规格说明需要进行技术审查和管理复审

需求工程管理两大难题
- 需求确认困难
- 需求不断变更

图 2-9　结构化需求分析与数据字典

2.5结构化分析与建模（4*）

结构化需求分析和建模的主要目的是为了减少需求分析时的错误，通过**自顶向下**建立系统逻辑模型，降低系统设计时的复杂性，提高系统的可维护性。

结构化需求分析的核心是数据。数据包括在分析、设计和实现中涉及的概念、术语、属性等所有内容，并把这些内容定义到了数据字典中。

围绕数据字典
- 实体关系模型描述数据建模过程，刻画系统的静态特征
- 数据流图完成对功能、操作流程的抽象和分解，完成功能建模
- 状态转换图是系统的行为建模，针对系统中的某个核心数据（对象）通过外部事件或条件的触发，导致该数据的状态改变。

面向数据的数据建模
- 数据建模回答软件开发中与各部分设计有关的所有数据对象。数据对象包括实体、实体属性和实体间的关系。
- 实体关系模型（ER图）是结构化建模的可视化图形工具，它描述了数据对象（实体）、对象属性以及对象间关系和基数。

面向数据流的功能建模

数据流图（DFD）是结构化建模中最流行的功能建模工具。
- 数据源：表明数据来源、结果去向，不同层次DFD中，结果是不能变的
- 数据加工：系统的变换部分，表明不同数据是通过哪些功能完成的变换。
- 数据存储：在DFD中保存数据，数据结果既可以是临时文件，又可以是持久文件。
- 数据流：有方向，表示数据变换是可以追溯的过程。

DFD的分解
- 最上层DFD称为顶层/0层，又称为语境模型，它反映的是与系统交互的外部系统或用户
- DFD的分层思想：自外向内、自顶向下、柱层析话、逐步精化

面向状态转换的行为建模
- 状态转换图STD
- 过程描述语言PDL

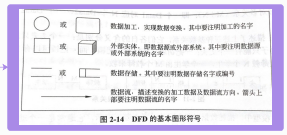


图 2-14　DFD 的基本图形符号

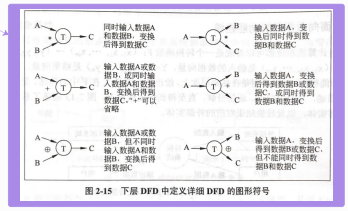


图 2-15　下层 DFD 中定义详细 DFD 的图形符号

2.6数据字典（1*）

数据字典是表达系统逻辑模型中所有数据详细情况的工具，它严格的定义了整个系统逻辑模型中所涉及到的基本数据、数据结构、文件形式，每一个元素作为一个词条，在词条解释中对该元素的定义、描述、使用特点、与其他数据元素的关系等等都进行了准确详细的表达。

数据字典的形式定义：表格、范式、Wornior图等形式

编辑数据字典需注意
- 完整性：在DFD中的数据流名称在DD中必定有相应的名称
- 一致性：在DFD数据流名与在DD中定义的数据语义必须一致
- 正确性：数据定义真实的描述了用户领域的数据、结构、文件等信息。

表 2-2　数据词典定义式中的符号意义

符号	含　义	解　释
=	被定义为	
+	与	例如，x=a+b，表示 x 由 a 和 b 组成
[,,,,,,]	或	例如，x=[a,b]，x=[ab]，表示 x 由 a 或由 b 组成
{...}...	或	
{ ... }	重复	例如，x={a}，表示 x 由 0 个或多个 a 组成
m{...}n	重复	例如，x=3{a}8，表示 x 中至少出现 3 次 a，至多出现 8 次 a，也可表示为{...}n
(...)	可选	例如，x=(a)，表示 a 可在 x 中出现，也可不出现
“...”	基本数据元素	例如，x= “a”，表示 x 为取值为 a 的数据元素
..	连接符	例如，x=1..9，表示 x 可取 1～9 的任一值

表 2-3　用数据表表示数据对象

数据编号	属　性	英文名称	备　注
USER001	姓名	Name	
USER002	所属部门编号	DepartmentID	
USER003	参加工作时间	StartDate	
AUTHORITY	权限	Authority	对应“授权”数据对象
USER004	专业	Major	
USER005	特长	Skill	
RESUME	简历	Resume	对应“简历”数据对象

3.软件设计基础（3*）

3.1软件设计概述

软件设计的目标：构造一个高内聚、高可靠性、高可维护性和高效的软件模型，为提高软件质量打下坚实基础。

软件设计阶段

- 概要设计
  - 又称为总体设计，主要任务是基于数据流图和数据字典，确定系统的整体软件结构，划分软件体系的各个子系统或模块，确定他们之间的关系。
  - 确切的说概要设计是要完成
    - 体系结构设计
    - 界面设计
    - 数据设计
- 详细设计
  - 在概要设计的基础上，具体实现各部分的细节，又叫过程设计
  - 过程设计包括了确定软件各模块内部的具体实现过程与局部数据结构
    - 结构化设计：模块的地理性约束了数据结构与算法相分离的情况
    - 面向对象设计：类的封装性较好的体现了算法和数据结构的内部性

软件设计的原则

- 分而治之
- 重用设计模式
- 可跟踪性
- 灵活性
- 一致性

3.2i软件体系结构设计（1*）

- 软件体系的设计概述
  - 提供软件设计师能预期的体系结构描述
  - 数据结构、文件组织、文件结构体现了软件设计的早期抉择，这些抉择将他极大的影响着后续的开发人员，决定着软件产品的最后成功
- 以数据为中心的数据仓库模型
- 客户端、服务器模式的分布式结构
- 层次模型
- MVC模型

3.3模块化设计（5*）

模块是程序语句集合，它拥有独立的命名，以及明确的输入输出范围

1.模块化

模块化是结构化设计的基本理论之一。模块是指具有**输入、输出、逻辑功能、内部数据**这四个基本属性的指令集合。模块化的摸底就是为了降低软件自身的复杂度，减少开发所需要的工作量。在模块数量适中的某一个值时，软件开发的总工作量最少，成本也最低。

2.抽象

- 实体抽象：又称数据抽象，是对需求陈述中实体的归纳
- 接口抽象：接口抽象的目的是通过统一的接口，设计不同的实现过程。
- 设计模式抽象：即为设计模型，它是对有相同数据组织，行为，结构的系统的指导性框架。

3.信息隐藏和局部化

- 信息隐藏：每一个模块内部与其他模块无关的数据，其他模块都无法访问；每个模块只完成一个相对对立的功能；模块之间的交换仅仅是为了实现系统功能儿必须交换的信息
- 局部化：是指应该把可能相互作用的过程和数据尽量放在一个模块内

模块独立性升高

耦合是对一个软件结构不同模块之间互连程度的度量

耦合

- 非直接耦合：模块间没有直接的相互调用关系，是最松散的耦合关系
- 数据耦合：信息交换仅发生在接口上
- 特征耦合：交换的信息是一个复杂的数据结构
- 控制耦合：交换的信息是控制信息
- 公共耦合：多个模块访问全局变量、结构、文件等公共信息都称为公共耦合。
- 内容耦合：一个模块直接访问另一个模块内部的数据，模块之间的叠加。

耦合度升高

模块独立性升高

内聚是指一个模块内的处理和数据之间的关联程度

内聚

- 偶然内聚：模块内的功能因为一些偶然的因素被聚集在一个模块
- 逻辑内聚：在逻辑上相关功能放在统一模块内，用参数区别执行哪一个
- 时间内聚：在同一时间内运行，或者在逻辑上有先后顺序
- 过程内聚：按照过程描述自上而下组织任务
- 通信内聚：模块内部操作同一数据区域/结构
- 顺序内聚：同一功能、按照顺序执行
- 功能内聚：模块是一个整体，完成一个独立的功能

内聚性升高

模块独立性升高

启发式规则

1.改进软件结构，提高模块独立性　　尽量降低耦合度，提高内聚性

2.模块规模适中

3.软件结构的宽度、深度、扇出度和扇入度都应该适中

- 宽度：软件结构中同一层次上模块数目的最大值
- 深度：软件结构中同意模块能控制的最深的层数
- 扇出度：同意模块能直接调用其他模块的数目
- 扇入度：软件结构中模块被直接调用的模块数目。扇入度越大，独立性越强

4.模块的作用域应该在模块控制域之内

5.设计单入口、单出口的模块，并力争降低模块接口的复杂度。

6.模块功能可以预测

7.内聚性高的指标不能绝对

8.根据关键任务优化模块区分

”先让系统正确运行起来，再让系统好起来“

3.4界面设计（2*）

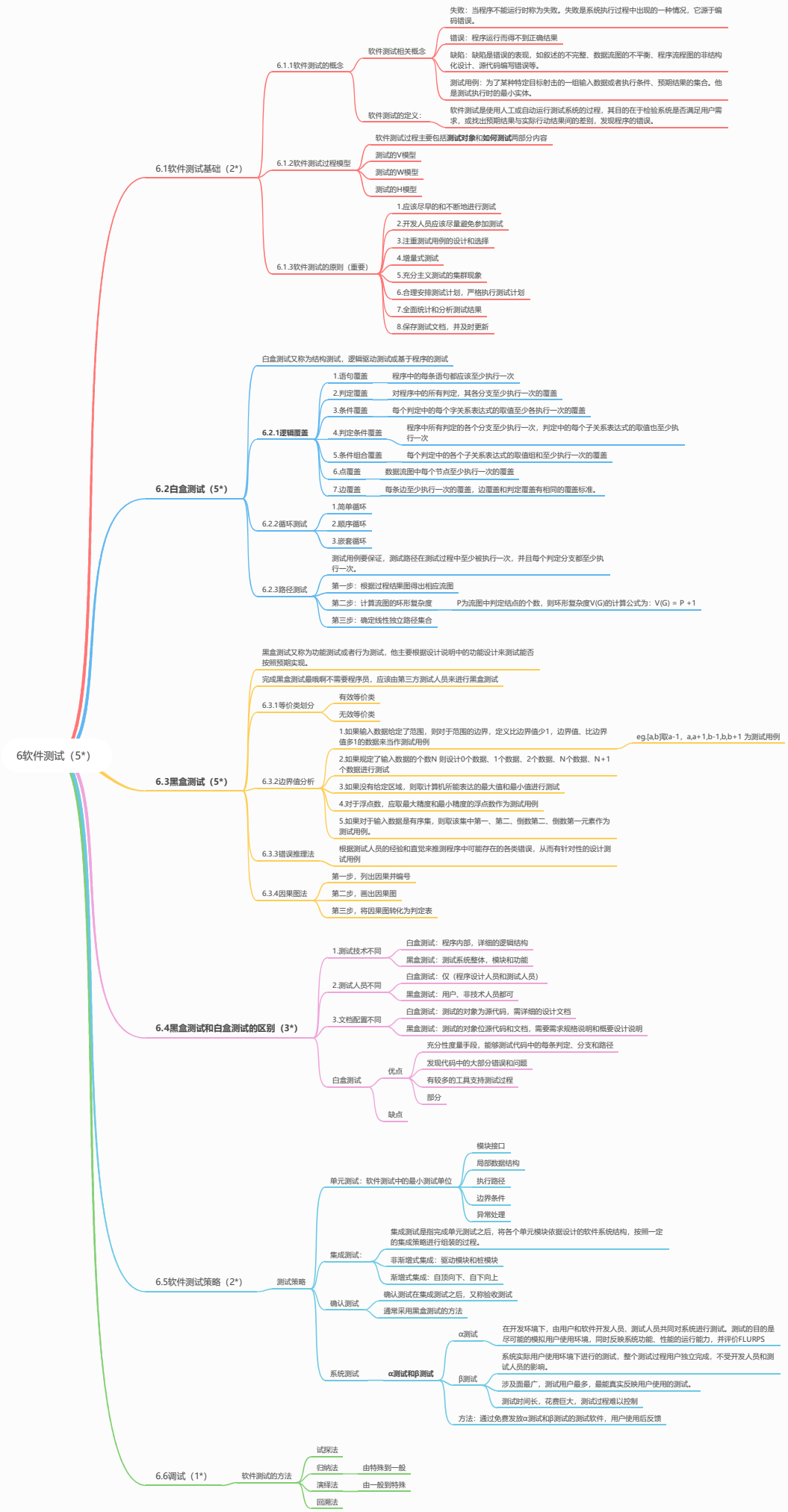
界面设计的任务

- 用户特性分析　　用户对计算机的接受程度
- 用户工作分析　　对系统内部功能自顶向下分解
- 界面任务分析　　根据用户工作分析的数据操作，设计用户界面
- 界面类型确定　　考虑操作的难易程度
- 界面原型评估　　评估修改确认

界面设计的原则

- 系统所有界面达到统一
- 提供系统运行过程中必要的反馈信息
- 提供快捷方式和回滚操作

- 4结构化设计方法（5*）
  - 4.1结构化设计方法概述
    - 结构化设计的基础是模块
    - 结构化设计的基本思想是：基于模块独立性和信息隐藏原则，自顶向下，逐步求精，分解和抽象相结合，并应用结构化程序设计技术而进行的软件设计。
    - 1.变换分析法
    - 2.事务分析法
    - 3.混合分析法
  - 4.2面向数据流的设计方法（5*）
    - 4.2.1层次图和结构图
      - 层次图
      - 结构图
        - 在层次图的基础上增加了对连线的数据流描述
        - 传入模块接收系统的输入数据，也表示系统内部有先后顺序的模块间的数据传递
        - 传出模块传递系统的输出数据，也表示系统内部有先后顺序的模块间的数据传递
        - 变换模块是系统的功能模块，实现数据的改变
        - 协调模块是系统控制或数据传递模块
    - 4.2.2变换分析法
      - 第一步：复审并精细化数据流图
        - （1）命名时尽量使用有明确含义的词、短语、术语和领域词汇，减少出现数据流图歧义。
        - **（2）上下层图（父子图）在输入、输出以及文件访问数据流之间的平衡。** 错例
          - （1）根据数据守恒原则，外部实体和外部实体，外部实体和数据存储之间不能存在数据流，，存储与存储之间也不应该有数据流，数据流必须跟加工有关，没有加工数据流不可能流来流去的。
          - （2）对于加工，输入是A，输出还是A，也违反了数据守恒原则，输入与输出一样，加工没有作用。
          - （3）对于加工，只有输入没有输出，违反了数据守恒原则。比如，人不可能只吃饭，不大小便。嘿嘿
          - （4）对于加工，只有输出没有输入，违反了数据守恒原则。比如，人不可能一直大小便，但不吃饭。嘿嘿
        - **（3）上下层图（父子图）的层次编号要一致，正确反映数据流图的分解过程。**
        - （4）对于每层数据流的分解，可以用逻辑运算符*+⊕增加数据流中各变换部分间的部分
        - （5）精化数据流图，使其能使用正确、完整的描述用户需求，因为这将决定软件结构图的逻辑框架正确与否。
        - **（6）第一步工作依赖于软件需求规格说明**
      - 第二步：划分自动化边界，确定数据流特征，判断数据流是变换流还是事务流　　变换流的特征是数据有明显的输入和输出，处理部分没有过多的控制和判断。
      - 第三步：划分数据输入输出边界，分理出处理部分
      - **第四步：执行一级分解**
        - 导出软件逻辑结构最上层的两层关系，顶层为主控模块，第二层为根据自动化边界的划分
        - 分为以下三个模块
          - 输入模块
          - 输出模块
          - 控制模块
        - 一级分解的总原则是抽象，在能明确表示软件总体框架和各部分逻辑子系统的前提下，模块数应尽量少。
      - **第五步：执行二级分解**
        - 二级分解的任务是把一层分级得到的各子系统模块按照各层的数据流图逐层细分，得到系统结构图的原型
      - **第六步：采用启发式规则，精化所得到的初步软件结构，以模块独立性为原则**，合并、分解、抽取各个模块，**得到**一个高内聚、低耦合、易实现、易测试、易维护的**软件结构图**
    - 4.2.3事务分析法
      - 当数据例图具有事务特征，即能找到事务中心和对应的多条活动路径的，则采用事务分析法。
      - 事务流的特征是在数据的输入、处理和输出的过程中，处理部分有明显的控制或判断中心，后续的数据流有较多的活动路径。
    - 4.2.4混合分析法
  - 4.4结构化详细设计的工具（5*）
    - 程序流程图
      - 优点：易懂，使用广泛
      - 缺点：
        - 对程序流程图中的控制流无法约束其转向，造成设计的随意性，并可能造成非结构化设计的出现。
        - 难以表达数据结构
    - 盒图
      - 没有控制流，控制域明晰
      - 盒图中方框的互相嵌套，准确的反映了过程设计时模块间的层次关系
    - 问题分析图（PAD）
    - 判定树
    - 判定表
    - 重点在于会画图


6软件测试（5*）
6.1软件测试基础（2*）
6.1.1软件测试的概念
软件测试相关概念
失败：当程序不能运行时称为失败。失败是系统执行过程中出现的一种情况，它源于编码错误。
错误：程序运行而得不到正确结果
缺陷：缺陷是错误的表现，如叙述的不完整、数据流图的不平衡、程序流程图的非结构化设计、源代码编写错误等。
测试用例：为了某种特定目标射击的一组输入数据或者执行条件、预期结果的集合。他是测试执行时的最小实体。
软件测试的定义：
软件测试是使用人工或自动运行测试系统的过程，其目的在于检验系统是否满足用户需求，或找出预期结果与实际行动结果间的差别，发现程序的错误。
6.1.2软件测试过程模型
软件测试过程主要包括测试对象和如何测试两部分内容
测试的V模型
测试的W模型
测试的H模型
6.1.3软件测试的原则（重要）
1.应该尽早的和不断地进行测试
2.开发人员应该尽量避免参加测试
3.注重测试用例的设计和选择
4.增量式测试
5.充分主义测试的集群现象
6.合理安排测试计划，严格执行测试计划
7.全面统计和分析测试结果
8.保存测试文档，并及时更新
6.2白盒测试（5*）
白盒测试又称为结构测试，逻辑驱动测试或基于程序的测试
6.2.1逻辑覆盖
1.语句覆盖 程序中的每条语句都应该至少执行一次
2.判定覆盖 对程序中的所有判定，其各分支至少执行一次的覆盖
3.条件覆盖 每个判定中的每个字关系表达式的取值至少各执行一次的覆盖
4.判定条件覆盖 程序中所有判定的各个分支至少执行一次，判定中的每个子关系表达式的取值也至少执行一次
5.条件组合覆盖 每个判定中的各个子关系表达式的取值组和至少执行一次的覆盖
6.点覆盖 数据流图中每个节点至少执行一次的覆盖
7.边覆盖 每条边至少执行一次的覆盖，边覆盖和判定覆盖有相同的覆盖标准。
6.2.2循环测试
1.简单循环
2.顺序循环
3.嵌套循环
6.2.3路径测试
测试用例要保证，测试路径在测试过程中至少被执行一次，并且每个判定分支都至少执行一次。
第一步：根据过程结果图得出相应流图
第二步：计算流图的环形复杂度 P为流图中判定结点的个数，则环形复杂度V(G)的计算公式为：V(G) = P +1
第三步：确定线性独立路径集合
6.3黑盒测试（5*）
黑盒测试又称为功能测试或者行为测试，他主要根据设计说明中的功能设计来测试能否按照预期实现。
完成黑盒测试最哦啊不需要程序员，应该由第三方测试人员来进行黑盒测试
6.3.1等价类划分
有效等价类
无效等价类
6.3.2边界值分析
1.如果输入数据给定了范围，则对于范围的边界，定义比边界值少1，边界值、比边界值多1的数据来当作测试用例
eg.[a,b]取a-1，a,a+1,b-1,b,b+1 为测试用例
2.如果规定了输入数据的个数N 则设计0个数据、1个数据、2个数据、N个数据、N+1个数据进行测试
3.如果没有给定区域，则取计算机所能表达的最大值和最小值进行测试
4.对于浮点数，应取最大精度和最小精度的浮点数作为测试用例
5.如果对于输入数据是有序集，则取该集中第一、第二、倒数第二、倒数第一元素作为测试用例。
6.3.3错误推理法
根据测试人员的经验和直觉来推测程序中可能存在的各类错误，从而有针对性的设计测试用例
6.3.4因果图法
第一步，列出因果并编号
第二步，画出因果图
第三步，将因果图转化为判定表
6.4黑盒测试和白盒测试的区别（3*）
1.测试技术不同
白盒测试：程序内部，详细的逻辑结构
黑盒测试：测试系统整体，模块和功能
2.测试人员不同
白盒测试：仅（程序设计人员和测试人员）
黑盒测试：用户、非技术人员都可
3.文档配置不同
白盒测试：测试的对象为源代码，需详细的设计文档
黑盒测试：测试的对象位源代码和文档，需要需求规格说明和概要设计说明
白盒测试
优点
充分性度量手段，能够测试代码中的每条判定、分支和路径
发现代码中的大部分错误和问题
有较多的工具支持测试过程
部分
缺点
6.5软件测试策略（2*）
测试策略
单元测试：软件测试中的最小测试单位
模块接口
局部数据结构
执行路径
边界条件
异常处理
集成测试：
集成测试是指完成单元测试之后，将各个单元模块依据设计的软件系统结构，按照一定的集成策略进行组装的过程。
非渐增式集成：驱动模块和桩模块
渐增式集成：自顶向下、自下向上
确认测试
确认测试在集成测试之后，又称验收测试
通常采用黑盒测试的方法
系统测试
α测试和β测试
α测试
在开发环境下，由用户和软件开发人员、测试人员共同对系统进行测试。测试的目的是尽可能的模拟用户使用环境，同时反映系统功能、性能的运行能力，并评价FLURPS
β测试
系统实际用户使用环境下进行的测试，整个测试过程用户独立完成，不受开发人员和测试人员的影响。
涉及面最广，测试用户最多，最能真实反映用户使用的测试。
测试时间长，花费巨大，测试过程难以控制
方法：通过免费发放α测试和β测试的测试软件，用户使用后反馈
6.6调试（1*）
软件测试的方法
试探法
归纳法 由特殊到一般
演绎法 由一般到特殊
回溯法

7统一建模语言UML（5*）

UML是通过图形化的表示机制进行**面向对象分析和设计**，并进行统一的、标准化的视图、图、模型元素和通用机制来刻画面向对象方法。

7.1UML的发展

UML的构成（四部分）

1.视图　从某个角度观察到的系统叫视图

2.图

视图由图组成

UML有9类图

- 用例图
- 类图（对象图）
- 包图
- 状态图
- 活动图
- 顺序图
- 协作图
- 构建图
- 部署图

3.模型元素　表示面向对象中的概念，如类、对象、接口、消息和组件等

4.通用机制　通用机制用于描述系统的其他信息，例如注释、通用系统的语义拓展

7.4UML的图和模型元素（5*）

7.4.1用例图

由参与者、用例、他们之间的关系共同构成，用于描述系统功能的图。

从系统外部描述系统功能及其功能间关系

主要用于子系统、包、类等功能行为描述

用例图不负责描述功能实现的细节和性能的约束

图形元素

- 参与者　不仅仅指用户，系统外部的、所有与系统交互的角色。
- 用例　对一组动作序列的抽象描述，系统执行动作序列，产生响应结果。
- 系统边界
- 关系　描述用例图之间的关联
  - <<use>>：使用关系，表示参与者对用例的操作，一般可省略
  - <<include>>:包含关系，表示一个用例行为包括另一个用例的行为。前者基本用例，后者为拓展用例
  - <<extend>>：扩展关系，表示扩展用例对基本用例的特殊服务。
  - 泛化关系：表示不同参与者或者不同用例间的继承关系

7.4.2类图

类图用于描述类的属性、方法和类间关系。属性和方法是类的内部结构，关系是类间的关联。

它们用于定义UML的静态模型。

类的内部结构：类名、属性、方法以及他们的可见性

7.4.3包图

对UML中的用例图、类图、UML关系等模型元素的封装，用于描述具有相似功能的模型元素的组合。

包之间的关系有两类：

- 依赖关系：一个包中引入另一个包的输出关系
- 泛化关系：定义包的继承关系，体现 具有与类库相似的包的家族

7.4.4状态图

7.4.5活动图

描述用例或场景的活动顺序，或者描述从一个活动到另一个活动的控制流

7.4.6顺序图

横坐标表示不同对象，纵坐标表示时间，说明某一时刻对象是否还存在

消息：

- 简单消息：控制流，描述对象间控制的转换
- 同步消息，：等待-继续“控制流
- 异步消息：“不等待”控制流

顺序图不仅仅描述系统交互或者操作过程，对于复杂的系统交互或操作过程，还可以增加结构化控制。

7.4.7协作图

描述类和类间关系，反映的是通过一组类的共同合作来完成的系统功能，又称为合作图

7.4.8构件图

7.4.9配置图

7.5UML的关系（4*）

7.5.1关联关系

描述类与类之间的关系构成。

关联关系通常是双向的，关联的多个类之间的彼此都能相互通信。

1.普通关联　最常见的关联关系，有连接关系就能通用普通关联表示

2.限定关联　限定关联用于描述一对多、多对多的关联关系，通过限定关联，可以将多对多关系转化为多个一对多关系，又可以将多个一对多关系转化为多个一对一的关系。

限定词放在限制的类旁边，用矩形或者( )，作为对类的约束。

3.关联类　虚线

4.递归关联　类间关系发生在单个类自身上，类和它自身有关联关系。

5.聚合关联　又称聚集，描述多个类之间是整体和部分的关联关系

- 共享聚合　”部分“类的对象可以同时成为多个”整体“类的对象
- 复合聚合　“部分”类的对象完全参加一个”整体“类的对象

7.5.2泛化关系

泛化关系用于描述一个类自动具有另一个类的属性和方法的机制

又被成为继承关系，带有空心的三角形的直线连接

1.普通泛化　一般意义上的继承

2.受限泛化　对泛化关系加约束，强化泛化关系的语义信息

- 交叠泛化
- 不相交泛化
- 完全泛化
- 不完全泛化

7.5.3依赖关系

虚线箭头表述

7.5.4实现关系

描述同一模型间的不同细化过程，体现的是类间的语义关联。

7.6UML的通用机制（3*）

常见的通用机制包含：修饰，注释，规格说明

1.修饰　用于增加UML模型元素的语义信息

2.注释　用缺角矩形表示，通过虚线与被注释的连接

3.规格说明

规格说明是对UML图形的一个标准化描述，既增加了事物的图形文字内容，又使得UML提供了可视化视图以及与之有关的语法和描述。

非标准方式

4.扩展机制

UML的修饰、注释、规格说明体现的就是UML的扩展机制，主要是对事物外部表象的扩展

为了既适应大型项目开发建模过程，又适应描述具体的功能、组织和个人，UML还提供了语义信息更加丰富的扩展机制。

分类

- 构造型　用于UML已有的模型元素的基础上，通过增加语义信息或说明来建立一种新的元素。
- 标签型　通过增加”属性-值“来进一步描述问题中的事物
- 约束型　在UML的基础上，增加对事物或者事物间关系因该满足的规则或语义。

8.面向对象分析（5*）

8.1面向对象分析概述

- 通过面向对象分析，最终建立起对问题域正确的、可理解的、无二义性的需求规格说明
- OOA模型由三类模型构成：功能模型、静态模型、动态模型

8.2功能模型（用例模型）（5*）

- 功能模型通过用例图描述系统的主体轮廓，主要确定系统范围和系统功能
- 建模过程
  - **1.识别参与者：**标识与系统相关的所有外部事物
    - 参与者的识别可以从系统使用人员、外部系统、硬件设备中获得
    - 为主题划分提供标准：相同参与者的用例应该尽可能划分在同一个主题包中
  - **2.识别用例和用例间关系：**用例描述系统将实现的功能，体现系统与参与者的交互
    - **识别用例**
      - 从**参与者**和**系统**的角度识别用例
      - 注意以下几点：
        - 在不影响交流和理解，用例粒度宜粗
        - 重点和难点的用例模型，用例粒度宜细，进行有效分解
        - 需要让分析员、设计员、用户都能看懂
      - **间接用例的取舍**
      - 还要注意
        - 1.一个用例描述一个相对完整的功能
        - 2，一个用例是在有限时间内完成的，不同的时间段不要涉及同一个用例
        - 3.用例图并不反映用例间在时间上或者在空间上的执行顺序
        - 4.一个参与者可以使用多个用例，一个用例也可以被多个参与者使用
        - 5.系统边界分析用例时，注意用户界面不一定是用例，一个界面可以对应多个用例，一个用例也可以对应多个界面
    - **识别用例间的关系**
  - **3.用例描述文档：**用文字信息详细描述了用例的内容
    - 用例文档编制
      - 补充：异常、限制
- 用例的作用
  - 1.获取需求
  - 2.指导设计与测试
  - 3.在软件工程生命周期的其他过程中，对其他任务的展开起到指导作用

依据用例模型导出静态模型

8.3静态模型（对象模型/领域模型）（5*）

描述了类的属性和方法，得到类图

- 静态模型是面向对象建模中最基本、最重要、最耗时的技术活动
- 静态模型任务：构建问题域重点概念模型，将问题域中的实体转化为信息域的类和对象以及他们之间的关系。因此又称对象模型、领域模型
- 通过建立类图以及关系反应领域概念
- **8.3.1识别类与对象**
  - 1.找出候选的类与对象
    - 问题域中的实体：包括可感知到的物理实体，如表格、设备、人员
    - 问题域中的概念：领域词汇、名词、短语，描述的问题
    - 问题中的事件
    - 对用例模型中参与者的分析
  - 2.审查筛选
    - 对类的筛选就是要删除无关的类，合并相同含义的类，抽象内容类似的类，分解需要更细粒度的类
    - 筛选原则：
      - 1.去除无关类：类应该与问题紧密相关
      - 2.合并相同的类：对同一实体会有不同的描述
      - 3.抽象和分解类
      - 4.属性和方法：一个类有多个属性和方法，如果一个类仅有一个属性或一个方法，应考虑将此类与其他有关联的类合并
      - 推迟实现：在OOA阶段，不应该考虑具体的实现
  - 3.类的命名
    - 1.类名应体现领域特征
    - 2.类名应该涵盖所有对象
    - 3.类名应尽量采用名词或名词短语
- **8.3.2划分主题**
  - 划分主题是为了**降低对系统理解的复杂度**，将大型的系统分解为多个不同的主题，每个主题表示一个相对独立的功能、性能、领域需求
  - 每个主题描述一个相对独立的功能、性能、领域需求
  - 主题通过UML包图实现
- **8.3.3确定结构**
  - 确定结构就是确定类间关系，给出系统的类图描述
  - **UML的类间关系主要有：关联关系、泛化关系、依赖关系、实现关系**
  - 确定结构
    - 确定关联关系：确定对象间的静态联系、关联的属性与操作
    - 确定泛化关系：按照知识领域、尝试、类间语义关系、软件重用
    - 确定依赖关系：考虑对象和方法的独立性
    - 确定实现关系：不仅反映类间关系，还反映模型间的关系
- **8.3.4确定属性**
  - 属性一方面作为**类的数据结构的支持方法的实现**，另一方面**表示类间静态关联**
  - 属性确认
    - 按照领域知识定义属性
    - 按照常识定义属性
    - 按照类的操作数据定义属性
    - 按照需求中的特定描述定义属性
- **8.3.5确定服务**
  - 服务即为类提供的方法，它必须定义在类中
  - 类的服务分三类：
    - **功能服务：**实现用户需求的功能、性能、领域需求
    - **关联服务：**实现类间的动态联系，如依赖关系中的友元关系，用友元类的方法可直接读取另一个类的私有成员
    - **通用服务：**这些服务不体现具体的功能或关系，但是为功能和关系提供必不可少的行为，如初始化类的属性、内存空间申请与释放、删除对象等。通用服务不一定在类图中给出

静态模型缺乏对象行为的描述，没有描述对象间如何进行交互完成系统任务．对于大型的复杂系统来说，通过对对象间的动态分析，更有助于明确类的方法的分组、类间关系的完善。

8.4动态模型（5*）

- **8.4.1建立顺序图及其描述文档**
  - 顺序图：顺序图是详细描述对象与参与者之间、对象与对象之间在时间上的交互的图，**强调交互过程中消息发送的顺序**。
  - 1.建立顺序图
    - （1）建立系统一次交互或操作过程的描述。这一描述过程又称为场景
    - （2）按照交互在时间上或者操作上的顺序列出参与者和类-对象
    - （3）根据场景中描述的交互过程及消息元素，画出参与者及类-对象之间的消息传递。
    - （4）根据参与者和类-对象各自的生存周期，确定他们的创建和撤销。
    - （5）根据需要确定是否需要增加结构化控制。
  - 2.顺序图文档编制

图 8-12 给出按上述过程得到的“自动组卷”用例的顺序图。

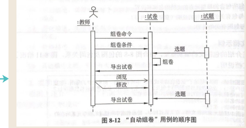


图 8-12 “自动组卷”用例的顺序图

- **8.4.2建立状态图及其描述文档**
  - 通过顺序图可以看出，软件系统的任何对象都有生命周期。同一对象在不同时间会有不同的状态，这就需要状态图来描述。
  - 1.建立状态图
  - 2.状态图文档编制

图 8-14 给出了按上述过程得到的“手动组卷”用例的状态图。CNum 是试卷的试题数目，Proj 是试卷的科目。

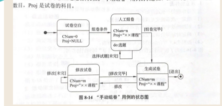


图 8-14 “手动组卷”用例的状态图

- **8.4.3建立协作图及其描述文档**
  - 协作图又称合作图，主要表示对象之间的动态链接关系，并不表示消息发的先后关系
  - 1.建立协作图
  - 2.协作图描述文档

图 8-16 给出了按上述过程得到的“自动组卷”用例的协作图。

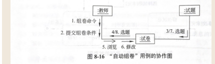


图 8-16 “自动组卷”用例的协作图

- **8.4.4建立活动图及其描述文档**

8.5典例：会议中心系统

题干：

有一个对外提供服务的会议中心，拥有各类不同规格的会议室，并为用户提供以下服务

（1）**会议申请人**能根据会议人数、会议时间预订会议室。临时会议只能预定一次。定期召开的会议可以长期预定。

（2）会议举行之前，会议申请人可以修改会议时间、人数，并重新选择会议室，甚至取消预定的会议

（3）会议预定确认之后，**会议中心**会负责会务管理，包括通过电子邮件，QQ，微信等方式发放会议通知给**参会人员**，制作参会证、宣传海报、横幅。

（4）**系统管理员**会根据会议室的使用情况，有权调整会议室和会议时间，并通知参会人员

建立功能模型

表 8-3 参与者及各自对应的用例

参与者	用　例
会议申请人	预订会议室、修改会议、修改参会人数、重选会议室、取消预订会议
会议中心	发通知（电子邮件、微信、QQ）、制作参会证、宣传海报、横幅
系统管理员	发通知（给参会人员）
参会人员	收到通知

图 8-20 是根据问题描述分析得出的用例图。其中，会议中心负责制作的代表证、宣传海报、横幅不属于中心软件系统功能的范畴。

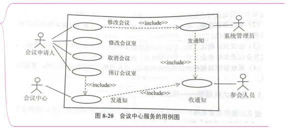


图 8-20 会议中心服务的用例图

建立静态模型（5层）

识别类与对象-类与对象层

图 8-21 是会议中心系统中被初步识别出的类。

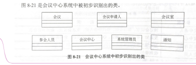


图 8-21 会议中心系统中初步识别出的类

属性和方法-属性与服务层（二层合并）

表 8-4 给出初步的类属性和方法

类	属　性	方　法
会议申请人	会议列表（可以申请多个会议）	
会议	参会人数、时间、会议室、会议类型（临时/长期）	预订（临时/长期）、修改、选择会议室、取消会议
会议中心	参会人员列表	邮寄、发邮件、制作代表证、发通知（确认函）
会议室	编号、容纳人数、预订时间	
参会人员	姓名、性别、电话、Email、微信号、QQ 号	收通知
系统管理员	用户名、密码、权限	调整会议室、发通知（变更函）
通知	名称、类型、日期、会议信息、参会人员	

精化类及类间关系-结构层

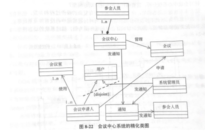


图 8-22 会议中心系统的精化类图

包图-主题层

（3）包之间的逻辑关系、层次关系，也同时反映了系统逻辑关系、层次关系。图 8-23 为会议中心系统的逻辑包图。

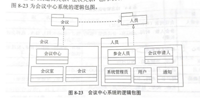


图 8-23 会议中心系统的逻辑包图

建立动态模型（交互行为）

顺序图

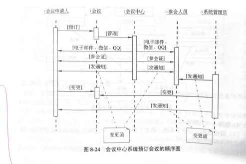


图 8-24 会议中心系统预订会议的顺序图

状态图

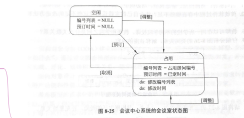
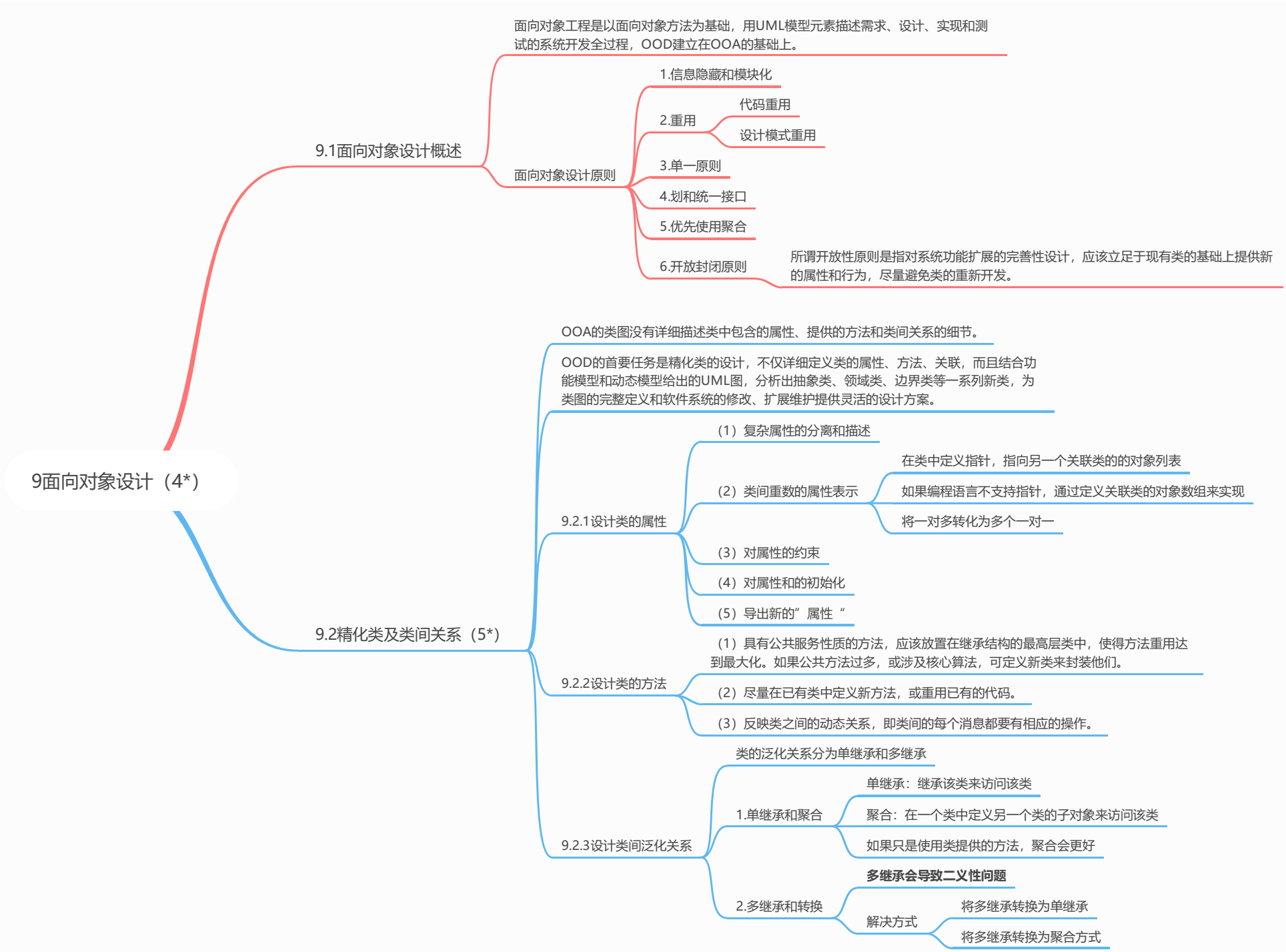
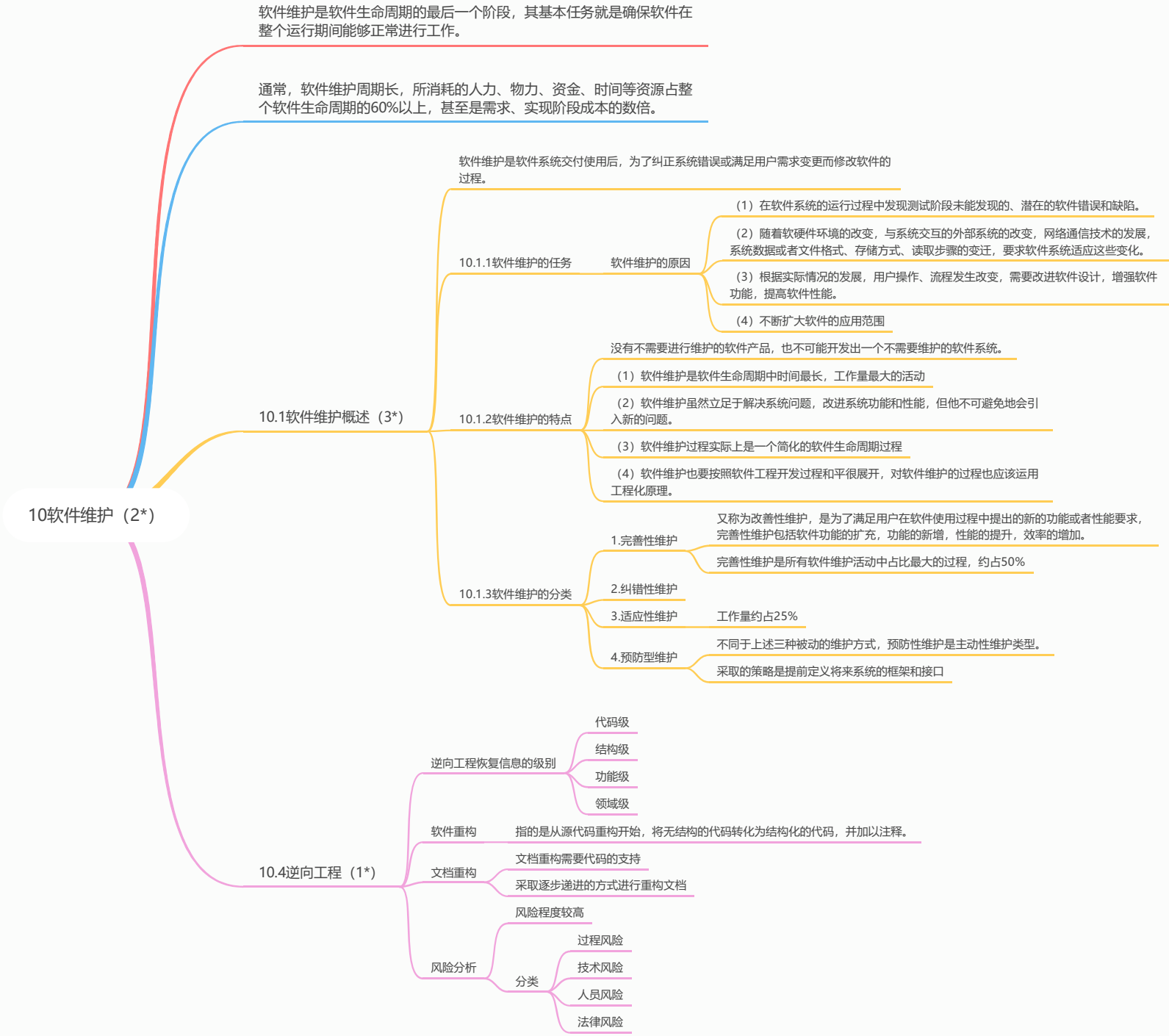


图 8-25 会议中心系统的会议室状态图


9面向对象设计（4*）
9.1面向对象设计概述
面向对象工程是以面向对象方法为基础，用UML模型元素描述需求、设计、实现和测试的系统开发全过程，OOD建立在OOA的基础上。
面向对象设计原则
1.信息隐藏和模块化
2.重用
代码重用
设计模式重用
3.单一原则
4.划和统一接口
5.优先使用聚合
6.开放封闭原则
所谓开放性原则是指对系统功能扩展的完善性设计，应该立足于现有类的基础上提供新的属性和行为，尽量避免类的重新开发。
9.2精化类及类间关系（5*）
OOA的类图没有详细描述类中包含的属性、提供的方法和类间关系的细节。
OOD的首要任务是精化类的设计，不仅详细定义类的属性、方法、关联，而且结合功能模型和动态模型给出的UML图，分析出抽象类、领域类、边界类等一系列新类，为类图的完整定义和软件系统的修改、扩展维护提供灵活的设计方案。
9.2.1设计类的属性
（1）复杂属性的分离和描述
（2）类间重数的属性表示
在类中定义指针，指向另一个关联类的的对象列表
如果编程语言不支持指针，通过定义关联类的对象数组来实现
将一对多转化为多个一对一
（3）对属性的约束
（4）对属性和的初始化
（5）导出新的”属性“
9.2.2设计类的方法
（1）具有公共服务性质的方法，应该放置在继承结构的最高层类中，使得方法重用达到最大化。如果公共方法过多，或涉及核心算法，可定义新类来封装他们。
（2）尽量在已有类中定义新方法，或重用已有的代码。
（3）反映类之间的动态关系，即类间的每个消息都要有相应的操作。
9.2.3设计类间泛化关系
类的泛化关系分为单继承和多继承
1.单继承和聚合
单继承：继承该类来访问该类
聚合：在一个类中定义另一个类的子对象来访问该类
如果只是使用类提供的方法，聚合会更好
2.多继承和转换
多继承会导致二义性问题
解决方式
将多继承转换为单继承
将多继承转换为聚合方式


10软件维护（2*）
软件维护是软件生命周期的最后一个阶段，其基本任务就是确保软件在整个运行期间能够正常进行工作。
通常，软件维护周期长，所消耗的人力、物力、资金、时间等资源占整个软件生命周期的60%以上，甚至是需求、实现阶段成本的数倍。
10.1软件维护概述（3*）
软件维护是软件系统交付使用后，为了纠正系统错误或满足用户需求变更而修改软件的过程。
10.1.1软件维护的任务
软件维护的原因
（1）在软件系统的运行过程中发现测试阶段未能发现的、潜在的软件错误和缺陷。
（2）随着软硬件环境的改变，与系统交互的外部系统的改变，网络通信技术的发展，系统数据或者文件格式、存储方式、读取步骤的变迁，要求软件系统适应这些变化。
（3）根据实际情况的发展，用户操作、流程发生改变，需要改进软件设计，增强软件功能，提高软件性能。
（4）不断扩大软件的应用范围
10.1.2软件维护的特点
没有不需要进行维护的软件产品，也不可能开发出一个不需要维护的软件系统。
（1）软件维护是软件生命周期中时间最长，工作量最大的活动
（2）软件维护虽然立足于解决系统问题，改进系统功能和性能，但他不可避免地会引入新的问题。
（3）软件维护过程实际上是一个简化的软件生命周期过程
（4）软件维护也要按照软件工程开发过程和平很展开，对软件维护的过程也应该运用工程化原理。
10.1.3软件维护的分类
1.完善性维护
又称为改善性维护，是为了满足用户在软件使用过程中提出的新的功能或者性能要求，完善性维护包括软件功能的扩充，功能的新增，性能的提升，效率的增加。
完善性维护是所有软件维护活动中占比最大的过程，约占50%
2.纠错性维护
3.适应性维护
工作量约占25%
4.预防型维护
不同于上述三种被动的维护方式，预防性维护是主动性维护类型。
采取的策略是提前定义将来系统的框架和接口
10.4逆向工程（1*）
逆向工程恢复信息的级别
代码级
结构级
功能级
领域级
软件重构
指的是从源代码重构开始，将无结构的代码转化为结构化的代码，并加以注释。
文档重构
文档重构需要代码的支持
采取逐步递进的方式进行重构文档
风险分析
风险程度较高
分类
过程风险
技术风险
人员风险
法律风险