第一章 软件工程概述

**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。

* 是逻辑实体非物理实体
* 是智力产品，生产过程主要集中在研发上
* 永不会磨损，但会退化，直至被弃用
* 开发远未达到软件工程目标提及的产业化生产
* 越来越复杂，今后会更加复杂。

**分类**：系统软件，支撑软件，实时软件，嵌入式软件，人工智能软件，应用软件

**生命周期**：1.可行性分析与计划阶段；2.需求分析阶段；3.设计阶段；4.实现阶段；5.测试阶段；6.运行与维护阶段；

**软件过程模型：**

**瀑布模型**：线性模型，自顶向下的过程；简单、严格、顺序、一次性、质量保证；难以适应需求不明或出现变动，只有等到软件开发结束才能拿到结果。

**原型模型**：开发一个早起使用的简单系统，不确定用户需求，快速开发出原型，用户根据原型再确定需求；快速，符合用户预期；不适宜大软件项目，决定于原型，需求不明可能导致质量问题。

**增量模型**：对功能以一系列增量的方式开发，非整体开发的模型，逐渐交付系统可用部分；灵活，根据需要优先开发，降低风险，适合需求不明确，开发功能多，开发时间长。在逐步增加系统功能的过程中要保证不破坏先前的部分。

**螺旋模型**：迭代式开发过程，每迭代一周就向前推进一层次，每次都要求需求定义、风险评估、过程实现和评审；风险分析：降低管理、技术和成本风险，特别适应大型复杂系统的开发，能及时避开风险；

**喷泉模型**：过程相互重叠，没有明显界限，自底向上；开发阶段重叠，各阶段平滑过渡，支持重用，不严格的阶段划分，增量式开发，对象驱动（分析设计实现并行）；需要较多的软件人员参与，给管理者带来困难。

**敏捷过程模型**：个体和交互胜过过程和工具，可以工作的软件胜过文档，客户合作胜过合同谈判，响应变化胜过遵循计划，简单、变化、有目的的建模、快速反馈；综合瀑布模型和原型模型的优点，快速开发和建模；

**软件开发方法：**

**结构化开发方法**：结构化方法提出的开发过程步骤明确，SA（结构化分析）、SD（结构化设计）和SP（结构化编程）三个阶段彼此衔接，前后照应。提供了一整套可行的、便于管理的方法学。

**面向对象的开发方法**：面向对象=对象+类+继承+消息

第二章 软件需求工程

**基本概念**：确定系统将要实现的各项内容，数据分析，定义逻辑模型，适应需求变更

原则：软件人员要从用户角度考虑软件需求；以流程为主线；尽量重用软部件；划分需求的优先级；需求变更要及时反馈；

**内容**：功能需求、性能需求（满足时间和空间上的约束）、领域需求和其他需求；

**管理**：贯穿整个过程，两大难题：需求确认困难，需求不断变更。

**任务**：确定系统将要实现的各项要求；数据分析；定义逻辑模型；适应需求变更；

**过程**：可行性研究（技术、经济、操作、法律）；需求获取（个别会谈小组会谈，问卷调查，面向用例的场景分析，快速原型技术）；需求分析与建模；需求评审；

结构化的需求分析与建模：

**结构化需求分析**：核心是数据，数据包括在需求分析、设计和实现中涉及的概念、术语、属性等所有内容，并把这些内容定义在数据字典中。

|  |  |  |
| --- | --- | --- |
| **面向数据的建模：**实体关系模型（ER图）描述数据对象（实体）、对象属性和对象间关系； | | |
| **数据对象**：系统中涉及的概念术语属性等都可作为候选数据对象；（结构化数据建模：数据对象是属性的结合；面向对象：数据对象是是属性和行为的集合） | **属性**：对象的静态特征  ER模型的模型元素中，矩阵表示实体，圆角矩阵表示属性，用无向边将对象与属性相连接。 | **关系和基数**：关系表明了数据对象间的关联关系，基数则表明数据对象在关系上的数量约束。（1:1，1：N，M:N）  ER模型：关系用菱形表示，关系也可以具有属性 |
| **面向数据流的功能建模**： 数据流图（DFD图）对功能，操作流程进行抽象和分解，完成功能建模 | | |
| 第0层：语境模型，反应系统交互的外部系统或用户 | * 父图与子图的平衡关系 * DFD图的编号 * 平衡规则 | 1. 确定系统的外部信息源 2. 画出顶层DFD图 3. 第一次精化，划分子系统 4. 逐层精化，对各子系统进行进一步精化 |
| **面向状态转换的行为建模：**是系统的行为建模，通过外部事件触发，导致系统采取相应的操作STD | | |
| 状态：  初态，终态，中间态 符合状态 | 状态装换  用有向箭头表示，是由事件或条件出发的。因而箭头上说明时间的名称或触发状态。 | 事件  指某一时刻发生的事情，是出发状态转换的条件或者一系列动作。 |

**数据字典**：以结构化的方式定义了在数据建模、功能建模和行为建模过程中设计的所有数据信息、控制信息。

|  |  |  |
| --- | --- | --- |
| 词条描述：  词条描述详细说明数据和控制信息在系统内传播的途径，分为数据流词条，数据元素词条，加工词条和储存文件词条 | 定义式（BNF） | Warnier图  描述信息层次结构的一种工具，类似于一种结构式语言 |

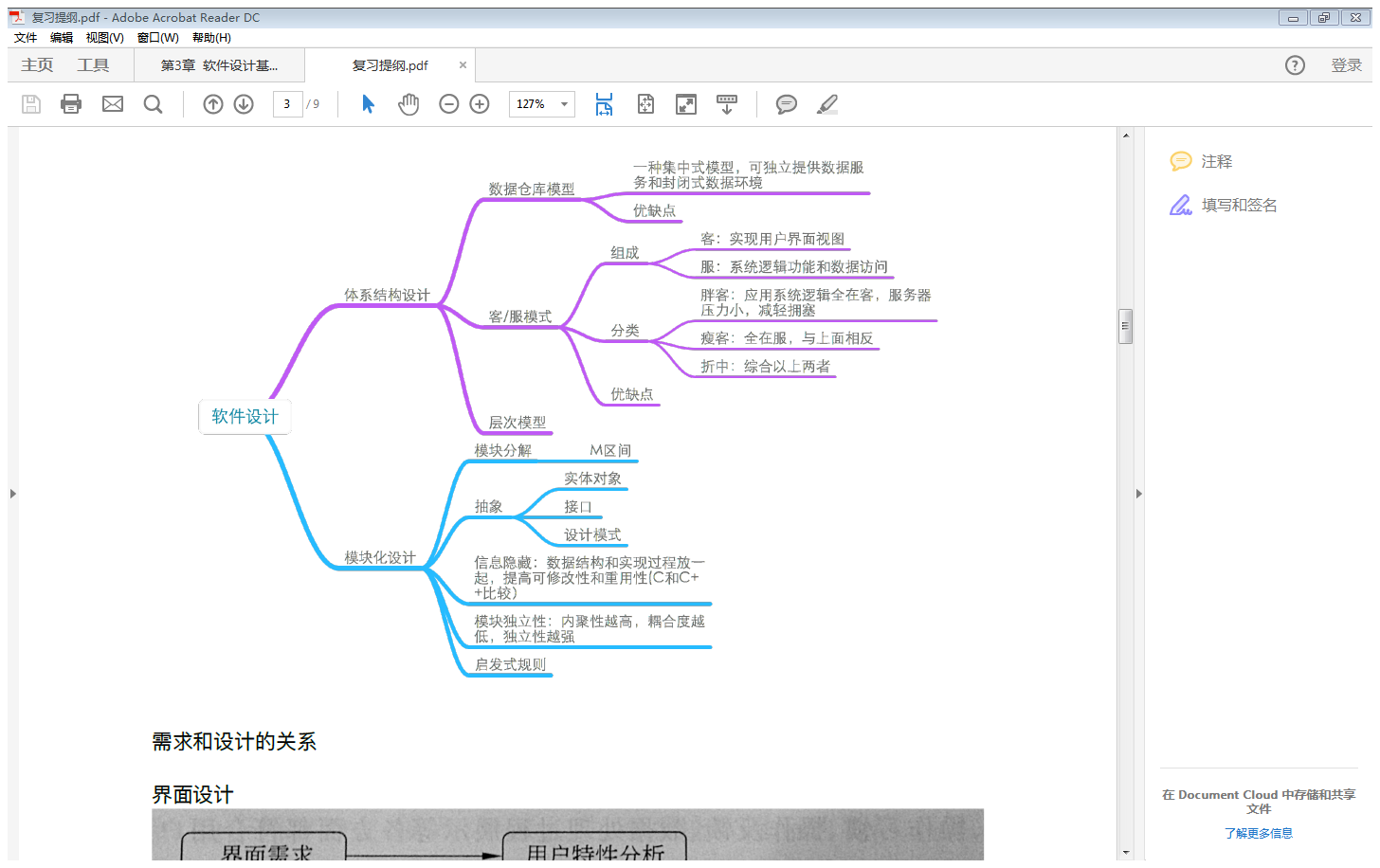
**加工逻辑**：用于描述DFD图中加工部分的流程或算法；加工逻辑的形式主要有过程描述语言，判定树，判定表等。

第三章：软件设计基础

目标：构造高内聚，低耦合。高可靠性、高可维护性、高可理解性且高效

主要任务：概要设计（体系结构设计、界面设计、数据设计）和详细设计（过程设计）

原则：分而治之、重用设计模式、可跟踪性、灵活性、一致性；



|  |  |
| --- | --- |
| 界面设计原则（总） | The Mandel原则： |
| 以人为本，用户体验为标准，美学与功能统一 | 简易、帮助、容错、灵活、个性化 |
| * 系统所有界面的操作的同一 * 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 | * 置用户于控制之下 * 减少用户的记忆负担 * 保持界面一致 |
| MVC模型：模型-视图-控制器 | |
| 优点：  一个模型对应多个视图  模型的自包含性  易修改，支持多种请求  利于过程化 | 缺点：  复杂  导致修改的连锁反应  数据访问效率低 |

第四章 结构化的设计方法

|  |  |  |
| --- | --- | --- |
| 面向**数据流**的设计方法 | 层次图：只反映上下调用关系，不反应组成关系，不反应系统执行过程 | 结构图：与层次图相似，用方框表示模块，连线表示调用关系 |
| 变换分析法：（符合软件理解）  确定信息流类型，精化DFD  划分自动化边界，确定DFD类型  划分数据输入、输出边界、分离处理部分 | 一级分解：DFD映射到系统模块设计模块结构上层  二级分解：DFD逐步分解出高层模块，设计出层次控制结构  根据模块独立性，精化模块结构 |
| 事务分析法：（符合用户的操作过程，活动路径独立）  复审并精化数据流图  确定数据流图特征，判断是变换流还是数据流  设定自动化边界，分离出事务中心和事务路径  执行“一级分解”，“二级分解  采用启发式规则” | |
| 面向**数据**的设计方法 | Jackson图：描述JSD方法的图形工具。按照输入输出和内部信息的数据结构进行软件设计，将数据结构的描述映射成程序结构的描述，设计出反应数据结构的程序 | |
| 结构化详细设计工具 | 程序流程图（同流程图PFD） | |
| 盒图（N-S图） | |
| 问题分析图（PAD图） | |
| 判定树： | |
| 判定表 | |

第五章 软件实现

程序设计语言：是机器按照人的指令完成相应任务的工具

分类：机器语言，汇编语言，高级语言，4GL;

特点：

|  |  |  |
| --- | --- | --- |
| 一致性 | 二义性 | 局部性 |
| 易编码性 | 可移植性 | 可维护性 |

选择准则：

|  |  |  |
| --- | --- | --- |
| 工程项目规模 | 用户需求 | 开发维护成本 |
| 项目的领域背景 | 编程人员对编程语言的熟悉程度 | |

程序设计风格：编程习惯、编程特点、编程逻辑

|  |  |
| --- | --- |
| 程序编排和组织准则 | 程序设计的效率 |
| * 源程序文件：包含标识符命名，注释以及排版格式 * 数据说明 * 语句结构处理 * 输入输出设计准则 | 设计逻辑结构清晰，高效的算法  储存效率  输入输出 |

|  |  |
| --- | --- |
| 代码复用：提高开发效率，提高质量 | 代码复用  源代码形式的复用；库文件形式的复用；面向对象机制下的复用 |
| 设计模式复用 |
| 软件开发过程 |

第六章 软件测试：保证软件质量，提高软件可靠性 最后检查和复审

软件测试过程模型

|  |  |  |
| --- | --- | --- |
| V模型（结束以后测） | W模型（完一块测一块） | H模型：随时测，只要有必要 |
|  |  |  |

软件测试技术分类

1.静态和动态；2.白盒与黑盒；3.测试策略与过程划分（单元、集成、确认、系统）

|  |  |  |  |
| --- | --- | --- | --- |
| 白盒测试（模块内部逻辑） | | 黑盒测试（把模块当做整体） | |
| 逻辑覆盖  语句、判定、条件、判定/条件、条件组合、点覆盖（语句覆盖），边覆盖（判定覆盖） | | 等价类的划分  划分为有效等价类和无效等价类 | |
| 循环测试（从内层开始，由里向外）  简单循环，嵌套循环，并列循环，非结构循环 | | 边界值分析法：选择等价类中的边界值 | |
| 错误推测法：通过经验和直觉 | |
| 路径测试（流图） | | 因果图法：找到对应输入输出关系 | |
| 优点 | | 容易入手，容易生成测试用例  测试系统功能强，效率高  用户和非技术人员参与  适用于测试的各阶段  能发现系统结构问题 | |
| 充分性度量手段  发现大部分错误  较多工具支持  测试用例自动生 | |
| 人工生成，难度大，易出错  测试用例耗时长；对全局数据结构和数据测试缺乏全面了解和掌控  仅适用于单元测试  不能验证体统整体规格说明的正确性 | | 缺点 | |
| 某些代码可能没有测试  隐藏了未测试代码的错误  充分性测试不足  测试的重复性 | |
| 单元测试  （可编译的最小单位，个人开发） | 集成测试  单元测试后，模块组装的过程 | 确认测试  模拟实际操作 | 系统测试  功能测试、性能测试、压力测试、容量测试、安全测试、文档测试、恢复性测试、备份测试 |
| 错误处理，模块接口，局部数据结构，边界条件，重要执行路径 | 非渐增式集成：测试各模块合格；  渐增式集成：逐渐集成到系统； | 确认：保证满足用户需求而进行一列活动 |
| 驱动模块：主模块，调用被测模块  桩模块：被测模块调用子模块 | 自顶向下结合：先主模块，逐渐加  自底向上：从子模块开始，向上 | 验证：保证软件正确的实现用户的某个需求 | α测试：用户开发共同测试  β测试：开放公测 |

软件调试方法

试探法，归纳法、演绎法、回溯法

软件测试报告：软件测试说明和软件测试报告

第7章 UML统一建模语言

UML:统一标准、可视化、易用易用性、面向对象特征、可编程性

|  |  |
| --- | --- |
| 对象 | 描述客观存在实体，具有逻辑性，数据是基础，对数据封装 |
| 类 | 具有相同属性和方法的对象集合，类中的方法体现该对象的具体行为 |
| 属性 | 类中定义的遗嘱数据特征的集合，是客观世界性质的抽象，类的静态特征 |
| 方法 | 方法是类提供的一组操作，类的动态特征，最小设计单元 |
| 封装性 | 类的定义，提供外部接口访问类的机制（私有、公有、受保护） |
| 继承性 | 体现重用性，类间的组织关系、面向对象的多项特征 |
| 多态性 | 类的一个接口对应多种实现的机制 |
| 重载 | 同名的方法，通过接口定义不同有不同实现的功能，减少记忆负担 |
| 消息 | 对象执行某个类定义的方法时所传递的数据规格说明 |

UML图

|  |  |
| --- | --- |
| 用例图：参与者、用例、关系，画出系统边界；从系统外部描述系统的功能及相关关系，不描述性能约束 | 用例不描述功能实现的各项处理，仅描述外部可视的交互操作 |
| 类图：用于描述类的属性，方法和类间关系，用于定义静态模型；  包括类名、可见性、属性、方法 |  |
| 包图：对用例图，类图，UML关系等模型元素的封装，描述具有相似功能的模型元素的组合 | 用矩形框表示包，名称准确，包之间具有UML的依赖和泛化关系 |
| 状态图：用于描述一个对象在生存周期的所有可能状态，以及引起状态改变的事件或条件 | 有一系列的状态、事件、条件和状态间装换共同构成 |
| 活动图：  用于描述用例或场景活动的顺序；类内部处理流程或整个系统的操作流程；  反应系统功能逻辑中参与的对象和对象格子的活动行为 |  |
| 顺序图：描述对象间的动态协作关系，并表现在时间顺序上；  参与者，对象，对象的生命线、消息 |  |
| 协作图：描述类和类之间的关系，反应的事一组类的共同协作。对象：用下划线修饰；链接：用于表达对象间的关系； | 消息：简单，同步，异步 |

UML关系

|  |  |
| --- | --- |
| 关联关系：描述类与类之间的关系构成，对象是类的集合，对象对象之间的关系是链，是关联的实例；  关联主要用来组织系统模型 | 普通关联：二元关系、多元关系  限定关联：描述一对多或多对多  关联类：关联属性和行为；  递归关联：和自身有关联 |
| 泛化关系(继承)：用于描述一个类具有另一个类的属性和方法 | 普通泛化：单继承与多继承，抽象类  受限泛化：对泛化增加约束条件，强化语义信息； |
| 依赖关系：非独立，必须依赖另一个事物存在，并且随之改变 | 绑定、许可、使用、调用、派生、实例化、有元、发送 |
| 实现关系：用于描述同一模型的的不同细化过程；一类通过继承体现，一类通过对模型精化体现 | 继承  对模型不断精化 |

第8章

OOA：面向对象分析是以类和对象为基础，面对对象方法学为指导，分析用户需求，并最终建立问题域模型的过程；确定系统将要实现的各项要求、进行重要数据分析、用例分析、控制分析、确定软件和其他成分的接口和通信；建立功能模型、静态模型和动态模型，确定需求规格说明书

|  |  |  |
| --- | --- | --- |
| 功能模型：通过用户陈述进行分析，建立功能模型以体现用户将要实现的功能 | 识别参与者：标识与系统相关的所有外部事物，包括人、外部系统；  识别用例和用例间的关系：  用例描述：详细用力的内容 | |
| 静态模型（最基本、最重要、最耗时）：通过提取问题定义中的实体得到的类和对象；静态模型的：5个层次  识别类和对象；划分主题；确定结构；确定属性；确定服务；类图/包图的文档 | 类-对象层 | 类，对象 |
| 结构层 | 泛化，关联关系 |
| 属性层 | 属性，对象连接 |
| 服务层 | 服务，消息连接 |
| 主题层 |  |
| 动态模型：结合功能模型和静态模型，定义类和对象的内部表示和外在联系 | 建立顺序图：遵循系统的某次或通用交互操作过程；  状态图  协作图；  活动图 | |

第9章 面向对象设计OOD

把OOA阶段得到的需求装换为符合用户功能和性能，便于与某种面向对象程序设计语言编程的体统实现方案

6个原则：

* 信息隐藏和模块化：信息隐藏式为了提高模块独立性
* 重用
* 单一原则：一个类只涉及和他相关的服务
* 规划和统一接口，不急于处理细节
* 优先使用聚合
* “开放封闭”原则

**设计类的属性**

1. 复杂属性的分离和描述：单一、复杂、策略（单独定义）
2. 类间重用的属性表示：指针（一对多或多对一），数组（一对多）
3. 对属性的约束：类的封装性约束了类的外部对属性和方法的存取权限
4. 对属性的初始化：属性的初始化设计确保了对象在启动时处于正常初始状态
5. 导出新“属性“：通过计算的出的具有属性特征的结果

**设计类的方法（类似于结构设计中的过程或函数）**

1. 具有公共服务性质的方法，应该放在继承结构的高层类中
2. 尽量在已有类中定义新方法，或重用已有代码
3. 反映类间的动态关系，即类间的每个消息都有对应操作

**设计类间泛化关系：**分为**多继承**和**单继承**

在**单继承**的设计中，可以比较聚合方式与单继承对类的组织结构利弊

在**多继承**的设计中，由于多继承带来的二义性，需要考虑将其进行装换

**类间关系定义为泛化还是聚合：**

泛化：派生类直接的到基类的属性和方法，多态性的应用，不足在于基态修改会影响派生类

聚合：定义另一个类的子对象，通过子对象方位类的公有部分

只使用另一个类的方法，定义成聚合好

**多继承的二义性**

转换为单继承：增加一个新的类，采用聚合来实现多继承

转换为聚合：将派生类聚合在基类中

**类间重数关联**：一对多的关联、一对多的关联、多对多的关联

泛化关系的数据设计：1.由于基类和派生类之间的继承关系，使得派生类具有基类的属性和方法。2.由于基类和派生类各自定义对应的表，同时把基类的表中的主见定义为派生类表中的外键，实现基类和派生类的泛化关系；

**人机交互设计的策略：**

1. 对用户分类。根据不同的用户人物和权限，区分不同操作
2. 对控制命令的分类：导航式设计
3. 设计人机交互的见面类：MVC模型中的“V”

**人机交互设计的原则**

1. 保持用户界面的一致性：公共显示特征的类
2. 显示必要信息：应只显示必要信息，避免给用户造成麻烦
3. 提供不同的数据输入方式

**构件图：系统代码文件的组织和构成**

系统有关的构件：

* 面向对象开发过程中产生的代码文件，包括源文件和数据文件，以及通过源文件编译得到的中间代码，链接库，可执行文件等
* 系统中使用的其他构件，如开发环境提供的库文件，可执行程序等
* 与系统有关的文档，这些文档包括除源程序和数据之外的其他文档，这些文件不能编译和执行

建立构件图的一般过程如下：

* 确定构件
* 确定构件的依赖关系
* 确定与构件相关的其他文档

**部署图/配置图：软件在硬件系统上的部署**

一般过程

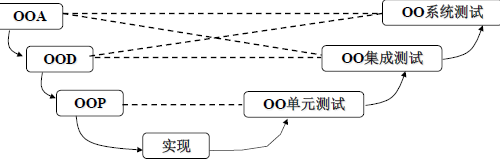
* 确定系统配置的拓扑结构，定义物理节点
* 确定构件图中的构件在拓补图中的位置，这是构件在硬件系统上的配置
* 确定物理节点间的关系，确定部署在同一节点上的构件间关系

**设计模式**：

动态变化：设计模型要适应将来的新需求

静态特征：设计所得的模型要尽可能复用原有类和模型

面对对象的测试概述



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | Singleton模式 | Abstract Factory | mediator | Adapter |
| 目的 | 一个类仅提供一个实例，并且该实例贯穿于整个应用系统的生存期 | 提供一个获得不同类的对象的方法 | 通过定义中介类的方式处理类间复杂的交互关系，也能消除类间多对多的关联关系 | 解决类间接口不匹配的问题 |
| 问题描述 | 只需要对类实例化出一个对象 | 在一个类中能实例化出不同类型的对象 | 降低类间关联的复杂程度，希望能灵活地改变类间的关联 | 将类的接口装换为所希望的另一种接口 |
| 解决方案 | 为了确保一个类只有一个对象，静态成员数据和静态成员函数，以得到控制访问的唯一实例 | 定义类的成员函数，该函数能得到不同类的对象实例 | 定义中介类，依次降低类间耦合度，提高类的内聚性 | 定义adapter类，用其装换类的接口 |
| 参与者 | 包括一个静态成员数据，他是对该类访问的唯一实例；获取该静态成员数据的静态成员函数，他使得能从外部访问类的唯一实例 | 抽象工厂类，得到不同类的实例；需生成对象的类即统一的访问接口 | 发生关联的双方（类）。以及中介类 | 接口不匹配的类，adapter类 |
| 结构 | 9-15 | 9-16 | 9-17 | 9-18 |

第十章 软件维护：是指软件系统在交付使用以后，为了改正错误或满足新的需求而修改软件的过程