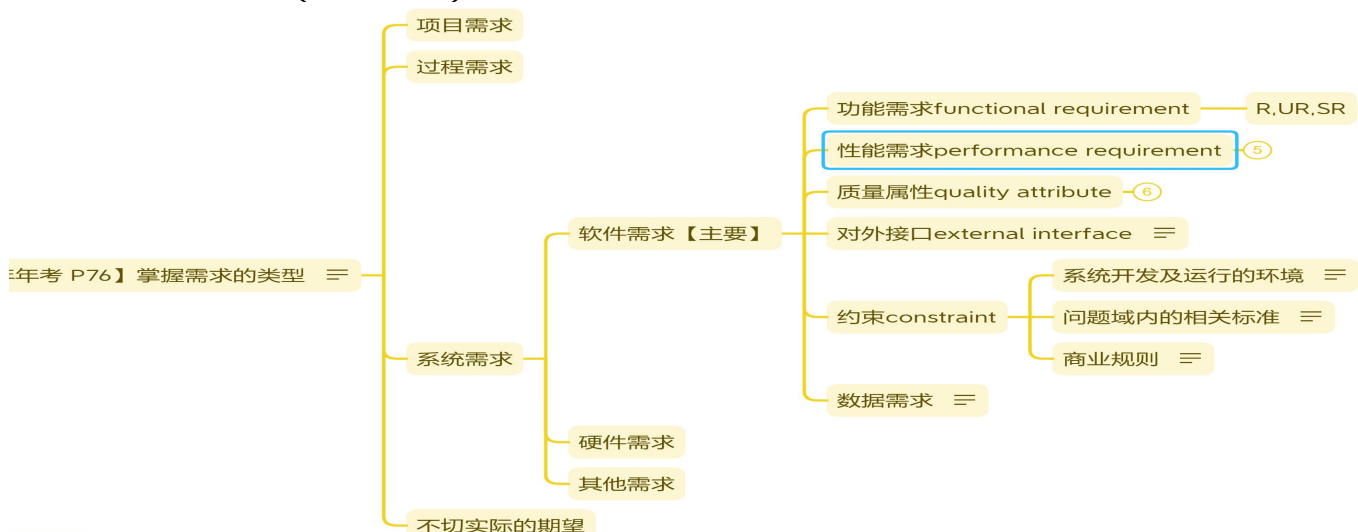


传统软件工程

- 1.1 软件工程的定义（名词解释）：应用系统的、规范的、可量化的方法来开发、运行和维护软件，即将工程应用到软件。
 - 什么是需求工程：所有需求处理活动的总和。它收集信息、分析问题、整合观点、记录需求并验证其正确性，最终描述出软件被应用后与其环境互动形成的期望效应
- 1.2
 - 为什么要需求分析：通过建模来整合各种信息，以使得人们更好的理解问题；为问题定义出一个需求集合，这个集合能够为问题界定一个有效的解决方案
 - 需求分析的任务：建立分析模型，达成开发者和用户对需求信息的共同理解；依据共同的理解，发挥创造性，创建软件系统解决方案
- 1.3 需求的层次性：
 - 业务需求（Business Requirement）（为什么要开发系统）
业务需求是高层次的解决方案和系统特性、系统开发的战略出发点、高层次的需求，**描述为什么要开发系统**。
Eg. 在系统使用 3 个月后，销售额度应该提高 20%（期望，没有从软件角度进行描述，业务需求）
 - 用户需求（User Requirement）
执行具体任务的用户对系统所能完成任务的期望，描述了**系统能帮用户做什么**（直接用户、间接用户）
Eg. 在系统要帮助收银员完成销售处理
 - 系统级需求（System Requirement）
需求分析模型：用户对系统行为的期望，每个系统级需求反映了一次外界与系统的交互行为，或者**系统的一个实现细节**（和用户需求有着很大的区别）
Eg. 在接到客户经理的请求后，系统应该为客户经理提供所有会员的个人信息。
- 1.4 区分需求，问题域和规格说明：
 - 需求
 - **是一种期望**
 - 源自现实又高于现实
 - 需求是多变和可调整的，项目可以依据实际情况调整需求的实现程度。
 - 问题域
 - **现实世界运行规律的一种反映**
 - 需求的产生地，也是需求的解决地。
 - 最终的软件产品要在现实部署，它能够部分影响问题域，但不能任意改变现实
 - 软件开发必须尊重问题域，不能因为技术原因擅自修改现实世界的实际情况
 - 规格说明
 - **软件产品的方案描述**，它以软件产品的运行机制为主要内容。
 - 它不是需求但实现需求，不是问题域但需要与问题域互动。
 - 规格说明要以关注对外交互的方式描述软件解决方案，它既需要从软件产品的角度而不是用户的角度进行描述，又不能太多地涉及软件产品的内部构造机制
- 超市销售系统
 - 问题 超市的成本太高
 - 需求 超市的成本应该降低
 - 问题域 超市的成本主要由人力成本和库存成本组成
 - 规格说明 系统的成本计算为成本=人力成本+库存成本

1.5 需求谱系（大的分类）与软件需求的分类



· 判断需求类型 见<https://quas-modo.github.io/2023/06/21/NJUSE/软件工程与计算二期末复习/>

- 约束：Eg. 已过保质期的食品不能销售
Eg. 顾客可以使用美元付款

1.6：结构化分析方法：

- 数据流程图
- 实体关系图

1.7：分层数据结构表示法 · 分层框图

- Warnier图

· 以上四种图要亲手画才行，不多的动手题了

二. 软件设计

2.1 什么是软件设计：

· 软件设计是指关于软件对象的设计，是一种设计活动。软件设计既指软件对象实现的规格说明，又指这个规格说明产生的过程。软件设计的核心思想是抽象和分解。

2.2 软件设计分层：

- 根基抽象程度的不同，可将软件设计分为三个层次：高层设计，中层设计和低层设计

· 高层设计：基于反映软件高层抽象的构建层次，描述系统的高层结构、关注点和设计决策

· 中层设计：更加关注组成构件的模块的划分、导入 / 导出、过程之间的调用关系或者类之间的协作

· 低层设计：深入模块和类的内部，关注具体的数据结构、算法、类型、语句和控制结构等体系结构

2.3 设计过程的主要活动：分析设计出发点、建立候选方案、生成最终方案、评价

2.4 设计方法与模型：

· 设计方法：结构化设计、面向对象设计、数据结构为中心设计、基于构件的设计、形式化方法设计

· 设计模型：描述软件设计的模型通常可以分为两类：静态模型和动态模型。

· 结构化设计：实体关系图描述静态模型，数据流图和结构图描述动态模型。

· 面向对象设计：静态模型通常使用类图、对象图、构件图、部署图；动态模型通常使用交互图（顺序图和通信图）、状态图、活动图等。

2.5 软件设计描述：

· 设计视图和设计图：软件设计描述由一个或多个软件设计视图组成，每一个设计视图由多个设计图组成。设计图中主要表现的是设计元素及其之间的关系和限制条件。设计图是软件设计视图的一个逻辑片段，通常用图表的方式来表示。

· 设计视角和设计关注：每一个设计视图都是从一个设计视角出发的。

三. 概要设计

· 软件设计是把软件需求变为软件的具体方案

· 软件设计包括两个阶段：概要设计和详细设计

3.1 软件概要设计的基本任务（目标）：

概要设计根据软件需求所确定的信息流程或信息结构，导出软件的总体表示----软件结构或程序过程

3.2 软件概要设计方法：

- 结构化程序设计 (Dijkstra) (结构化程序设计的基础建立在三种能够构成结构化程序的逻辑构造 (顺序, 选择, 重复) 上。)
- 面向数据的设计方法
 - 面向数据流的设计 (把信息流映射成软件结构)
 - 信息流的类型决定了映射的方法
 - 信息流有两种类型: 变换流、事务流
 - 面向数据结构的设计 (Jackson方法)
- 上面的三种方法要找题目动笔画

3.3 软件设计的基本原理：

3.3.1 模块化：

- 软件被划分成独立命名和可独立访问的被称作模块的构件，每个模块完成一个子功能，它们集成到一起满足问题需求
- 实现模块化的手段：抽象和信息隐蔽。抽象就是接口，隐藏就是实现上的设计决策
- **模块独立性**：模块独立是指开发具有独立功能而且和其它模块之间没有过多的相互作用的模块。
- 模块独立的意义：
 - (1) 功能分割，简化接口，易于多人合作开发同一软件；
 - (2) 独立的模块易于测试和维护。
- 模块独立程度的衡量标准：
 - **耦合性**：对一个软件结构内不同模块间互连程度的度量
 - **内聚性**：标志一个模块内各个处理元素彼此结合的紧密程度，理想的内聚模块只做一件事情

3.3.2 抽象和信息隐蔽：


- 抽象：抽出事物的本质特性而暂时不考虑它们的细节，聚焦本质，降低认知复杂度
- 信息隐蔽：应该这样设计和确定模块，使得一个模块内包含的信息 (过程和数据) 对于不需要这些信息的模块来说，是不可访问的。

3.3.3 内聚和耦合

- 耦合划分

类 型	耦 合 性	解 释	例 子
内容耦合	<div>最高</div> <div>↑</div> <div>↓</div> <div>最低</div>	一个模块直接修改或者依赖于另一个模块的内容	程序跳转 GOTO；某些语言机制支持直接更改另一个模块的代码；改变另一个模块的内部数据
公共耦合		模块之间共享全局的数据	全局变量
重复耦合		模块之间有同样逻辑的重复代码	逻辑代码被复制到两个地方
控制耦合		一个模块给另一个模块传递控制信息	传递“显示星期天”。传递模块和接收模块必须共享一个共同的内部结构和逻辑
印记耦合		共享一个数据结构，但是却只用了其中一部分	传递了整个记录给另一个模块，另一个模块却只需要一个字段
数据耦合		两个模块的所有参数是同类型的数据项	传递一个整数给一个计算平方根的函数

· 内聚划分

类 型	内 聚 性	解 释	例 子
偶然内聚	 最低	模块执行多个完全不相关的操作	把下列方法放在一个模块中：修车、烤面包、遛狗、看电影
逻辑内聚		模块执行一系列相关操作，每个操作的调用由其他模块来决定	把下列方法放在一个模块中：开车去、坐火车去、坐飞机去
时间内聚		模块执行一系列与时间有关的操作	把下列方法放在一个模块中：起床、刷牙、洗脸、吃早餐
过程内聚		模块执行一些与步骤顺序有关的操作	把下列方法放在一个模块中：守门员传球给后卫、后卫传球给中场球员、中场球员传球给前锋、前锋射门
通信内聚		模块执行一系列与步骤有关的操作，并且这些操作在相同的数据上进行	把下列方法放在一个模块中：查书的名字、查书的作者、查书的出版商
功能内聚		模块只执行一个操作或达到一个单一目的	下列内容都作为独立模块：计算平方根、决定最短路径、压缩数据
信息内聚		模块进行许多操作，各个都有各自的入口点，每个操作的代码相对独立，而且所有操作都在相同的数据结构上完成	比如数据结构中的栈，它包含相应的数据和操作。所有的操作都是针对相同的数据结构
	最高		

3.4 模块独立性原则对指导设计的意义：

· ？

3.5 软件结构优化准则

3.5.1

- 模块的作用范围是指该模块中一个判断所影响的所有其它模块；
- 模块的控制范围指该模块本身以及所有直接或间接从属于它的模块。

3.5.2 软件结构设计的优化准则（？）

- 改进软件结构，提高模块独立性
- 模块规模应该适中（最好能写在一页纸上）
- 大模块分解不充分；小模块使用开销大，接口复杂。
- 尽量减少高扇出结构的数目，随着深度的增加争取更多的扇入
- 扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块。一般来说，顶层扇出高，中间扇出少，低层高扇入
- 模块的作用范围保持在该模块的控制范围内
- 力争降低模块接口的复杂程度
- 设计单入口单出口的模块（避免内容耦合，易于理解和维护）
- 模块的功能应该可以预测（相同的输入应该有相同的输出，否则难以理解、测试和维护）

四. 详细设计

4.1 详细设计的目标（基本任务）

- 详细设计是给出软件结构中各模块的内部过程描述（详细设计也既是要导出一种算法设计表示，由此可以直接而简单地导出程序代码）

· （关系）依赖 < 关联 < 聚合 < 组合 < 继承

4.2 图形设计工具（要会画）

- 流程图
- 方块图
- PAD图

五. 代码设计

5.1 设计可靠代码（提高代码可靠性的方法往往会降低代码的易读性和性能，也可能牺牲易维护性，只有针对那些可靠性比较重要的代码，才会使用提高可靠性的设计方法）

- 契约式设计（断言式设计）
 - 基本思想：如果一个函数或方法，在前置条件满足的情况下开始执行，完成后能够满足后置条件，那么这个函数或方法就是正确、可靠的。
 - 契约式设计有两种常见的编程方式：异常与断言。

· 防御性编程

- 基本思想：在一个方法与其他方法、操作系统、硬件等外界环境交互时，不能确保外界都是正确的，所以要在外界发生错误时，保护方法内部不受损害。

二者的异同点：

- 共同点：都要检查输入参数的有效性，（都可以用异常和断言来实现）
- 不同点：防御性编程将所有与外界的交互（不仅仅是前置条件包含的）都纳入防御范围；防御性编程不检查输出和后置条件。

5.2 模型辅助设计复杂代码：决策表，伪代码（略），程序流程图（感觉要会画）

表 18-2 决策表的基本结构

条件和行动	规 则
条件声明 (condition statement)	条件选项 (condition entry)
行动声明 (action statement)	行动选项 (action entry)

表 18-3 决策表示例

条件和行动	规 则		
prePoint	<1000	<2000	<5000
postPoint	>=1000	>=2000	>=5000
Gift Event Level 1	√		
Gift Event Level 2		√	
Gift Event Level 3			√

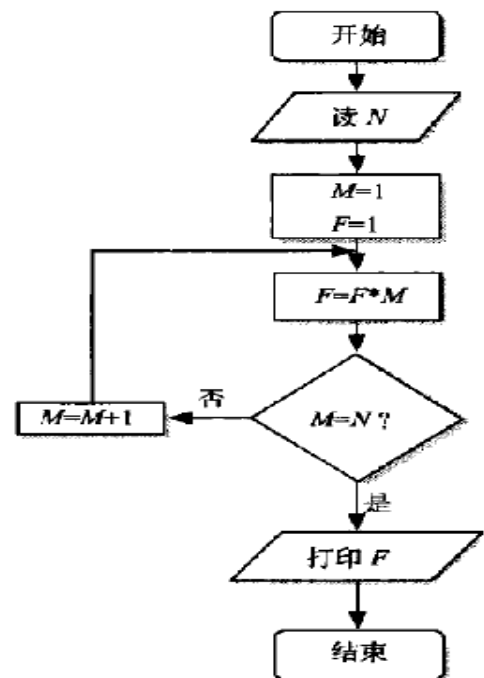


图 18-20 程序流程图示例

5.3 单元测试用例开发

(1) 为方法开发测试用例主要使用两种线索：

- 方法的规格：根据第一种线索，可以使用基于规格的测试技术开发测试用例，等价类划分和边界值分析是开发单元测试用例常用的黑盒测试方法。
- 方法代码的逻辑结构：根据第二种线索，可以使用基于代码的测试技术开发测试用例，对关键、复杂的代码使用路径覆盖，对复杂代码使用分支覆盖，简单情况使用语句覆盖。

(2) 使用 Mock Object测试类方法

- (3) 在复杂类中，常常有着多变的状态，每次一个方法的执行改变了类状态时，都会给其他方法带来影响，也就是说复杂类的多个方法间是互相依赖的。所以，除了测试类的每一个方法之外，还要测试类不同方法之间的互相影响情况（为复杂类开发测试用例可以使用基于状态机的技术）

六. 软件测试

6.1 测试完备性的含义（必考好像）

- ？没找到

6.2 白盒，黑盒以及使用来设计测试用例（必考）白盒部分ppt更全

！要注意的是在写白盒测试用例前要先说明所有的语句或条件组合等

6.3 黑盒测试和白盒测试的优缺点（感觉会成为附带的小题）：

- 白盒优点：覆盖率高；发现的缺陷较多
- 白盒缺点：测试开销大；不能检验需求规格
- 黑盒优点：测试效率高；可以检验需求规格
- 黑盒缺点：覆盖率低；发现的缺陷少

七. 软件维护

7.1 软件维护的类型

- 完善性维护 (Perfective maintenance) : 为了满足用户新的需求、增加软件功能而进行的软件修改活动。
- 适应性维护 (Adaptive maintenance) : 为了使软件能适应新的环境而进行的软件修改活动。
- 修正性维护 (Corrective maintenance) : 为了排除软件产品中遗留缺陷而进行的软件修改活动。
- 预防性维护 (Preventive maintenance) : 为了让软件产品在将来可维护, 提升可维护性的软件修改活动。

7.2 软件维护技术

- 常用的专门技术有: 遗留软件处理、逆向工程和再工程