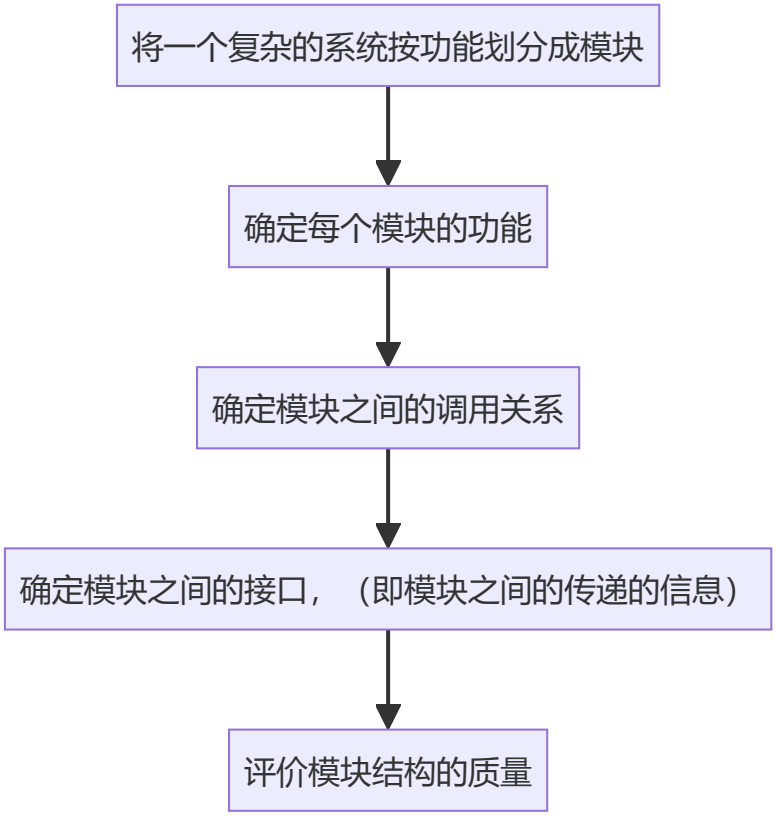
系统设计

系统设计的主要目的就是为系统制定蓝图，在各种技术和实施方法中权衡利弊，精心设计，合理使用各种资源，最终勾画出新系统的详细设计方案。

系统设计分为：概要设计、详细设计。

1、概要设计的基本任务

1. 设计软件系统的总体结构



2. 数据结构及数据库设计
3. 编写概要设计文档

> 概要设计文档主要有：
> 概要设计说明书
> 数据库设计说明书
> 用户手册以及修订测试计划

4. 评审

> 对设计不对是否完成地实现了需求中规定的功能、性能等要求进行评审

2、详细设计的基本任务

> 详细设计的阶段的根本目的是确定应该怎样具体的实现所要求的系统。

1. 对每个模块进行详细的算法设计
2. 对模块内的数据结构进行设计
3. 对数据库进行物理设计，（即确定数据库的物理结构）
4. 其他设计
5. 编写详细的设计说明书
6. 评审，对处理过程的算法和数据库的物理结构都有评审

3、系统设计的基本原理

1. 抽象

> 抽象是一种技术，重点说明一个实体的本质方面，而忽略或掩盖不是很重要或非本质的方面

2. 模块化

> 在软件的体系结构中，模块是可组合、分解和更换的单元。模块化是指将一个待开发的软件分解成若干个小的简单的部分 某某模块。
>
> 每个模块可独立的开发、测试，最后组装成完整的程序。模块化的目的是使程序的结构清晰，容易阅读、理解、测试和修改。

3. 信息屏蔽

> 信息屏蔽是开发整体程序结构时使用的法则，即将每个程序的成分隐蔽或封闭在一个单一的设计模块中，定义每一个模块时尽可能少的显露其内部的处理。信息隐蔽原则对提高软件的可修改性、可测试性和可移植性都有重要的作用

4、模块独立

> 模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系简单。衡量模块独立程度的标准有两个：
>
> 1、耦合性
>
> 2、内聚性

1. 耦合

耦合性是指模块之间联系的紧密程度。

耦合性越高，则模块的独立性越差。

1)、无直接耦合：指两个模块间没有直接的关系，它们分别从属于不同模块的控制与调用，它们之间不传递信息

2）、数据耦合：两个模块间有调用关系，传递简单的数据值，相当于高级语言中的值传递

3）、标记耦合：两个模块间传递的是数据结构，如高级语言中的数据组名、记录名、文件名等这些名字即为标记

4）、控制耦合：一个模块调用另一个模块时，传递的是控制变量，被调用模块通过控制变量的值有选择的执行模块内的某一功能

5）、公共耦合：通过一个公共数据环境相互作用的那些模块之间的耦合

6）、内容耦合：是耦合性最高的。当一个模块直接使用另一个模块内部数据或通过非正常入口而转入另一个模块内部，这种模块之间的耦合为内容耦合

2. 内聚

内聚是指模块内元素之间的联系紧密程度，如一个完成多个功能的模块内聚度就比完成单一功能的米宽的内聚度低。

内聚度越低，模块的独立性越差

1）、偶然内聚：一个模块内的各个处理元素之间没有任何联系.

2）、逻辑内聚：模块内执行几个逻辑上相似的功能，通过参数确定该模块完成哪一个功能

3）、时间内聚：需要同时执行的几个动作组合在一起形成的模块为时间内聚模块

4）、通信内聚：模块内所有处理元素都在同一个数据结构上操作，或各处理使用相同的输入数据或产生相同的输出数据

5）、顺序内聚：一个模块中各处理元素都密切相关于一个功能且必须顺序执行，前一功能元素的输出就是下一功能元素的输入

6）、功能内聚：最强的内聚，模块内所有元素共同完成一个功能，缺一不可。

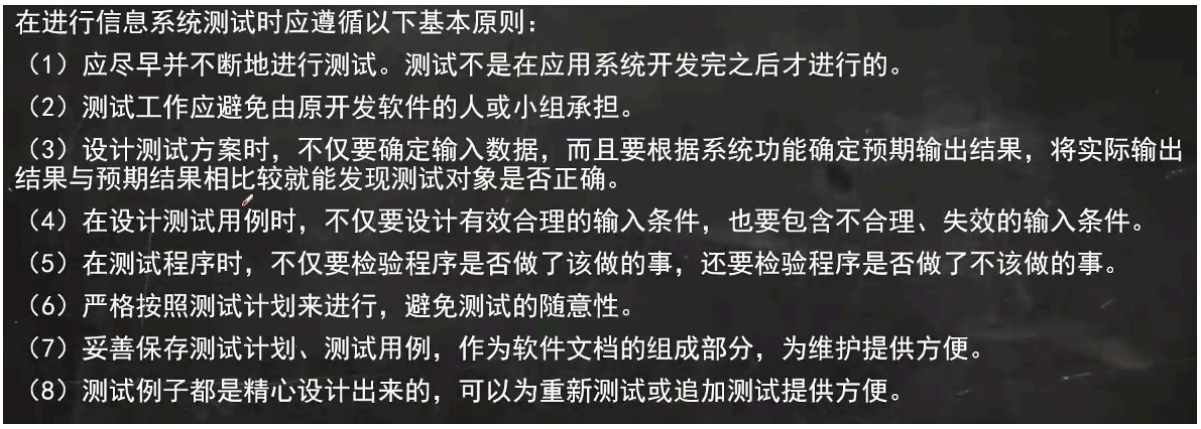
软件系统划分模块时，尽量做到高内聚、低耦合，提高模块的独立性

5、系统测试基础知识（可以不看）

系统测试的目的是为了发现错误而执行程序的过程

成功的测试是发现了至今尚存未发现的错误的测试

测试的目的就是希望能以最少的人力和时间发现潜在的各种错误和缺陷



测试类型：

软件测试可分为：

1. 单元测试：

> 单元测试也称模块测试，侧重测试模块中的内部处理逻辑结构和数据结构

2. 集成测试：

> 目的是检查模块之间，及模块和已集成的软件之间的接口关系。并验证已集成的软件是否符合设计要求

3. 确认测试：

> 用于验证软件的功能、性能和其他特性是否与用户需求一致

4. 系统测试：

> 系统测试的对象是完整的、集成的计算机系统。
> 目的是在真实系统工作环境下，验证完整的软件配置项能否和系统正确连接，并满足系统或子系统设计文档和软件开发合同规定的要求。
> 技术依据是用户需求或开发合同

5. 回归测试：

> 目的是测试软件变更之后的变更部分的正确性和对变更需求的符合性，以及软件原有功能、性能和其他规定的要求的不损害性

6、软件测试方法

1. 黑盒测试：

> 黑盒测试也称为功能测试；在不考虑软件的内部结构和特性的情况下，测试软件的外部特性。
> 主要用于集成测试，确认测试与系统测试中。
> 常用黑盒测试技术有：等价类划分、边界值分析、错误推测、因果图

2. 白盒测试：

> 白盒测试也称为结构测试，将程序看作一个透明盒子，清楚知道程序的结构和处理算法，安装程序内部逻辑结构设计测试用例。
> 主要用于软件单元测试。
> 常用白盒测试技术有：逻辑覆盖、循环覆盖、基本路径测试。
> 主要的逻辑覆盖标准：语句覆盖、判定覆盖、条件/判定覆盖、条件组合覆盖、路径覆盖等

7、系统维护

1. 系统的可维护性定义：

> 维护人员理解、改正、改动和改进这个软件的难易程度

2. 系统可维护性的评价指标：

> 1）可理解性：别人能理解系统的机构、界面、功能和内部过程的难易程度
> 2）可测试性：诊断和测试的容易程度取决于易理解的程度
> 3）可修改性：模块的耦合、内聚、作用范围与控制范围的关系，都对修改性有影响

3. 软件维护的一般内容：

> 1、正确性维护：改正在系统开发阶段已发生而系统测试阶段尚未发现的错误
> 2、适应性维护：使应用软件适应信息技术变化和管理需求变化而进行的修改
> 3、完善性维护：为扩充功能和改善性能而进行的修改，主要对已有的软件系统增加在系统分析和设计阶段中没有规定的功能与性能特征。这些功能对完善系统功能是非常必要的
> 4、预防性维护：改进应用软件的可靠性和可维护性，为了适应未来的软硬件环境的变化，而主动增加预防性的新功能

8、CMM（软件能力成熟度模型）

> CMM（软件能力成熟度模型）是一种对软件组织在定义、实施、度量、控制和改善其软件过程的实践中各个发展阶段的描述形成的标准

CMM/CMMI将软件过程的成熟度分为5个等级：

1. 初始级：工作无序，项目进行过程中长放弃当初的计划；管理无章法，缺乏健全的管理制度。
2. 可重复级：管理制度化，建立了基本的管理制度和规程，管理工作有章可循。初步管理实现标准化，开发工作比较好的按标准化实施。变更依法进行，做到基线化，稳定可跟踪，新项目的计划和管理基于过去的时间经验，具有重复以前成功项目的环境和条件。
3. 已定义级：开发过程，包括技术工作和管理工作，都已实现标准化、文档化。建立完善的培训制度和专家评审制度，全部技术活动和管理均可控制，对项目进行中的过程、岗位和职责有共同理解
4. 已管理级：产品和过程已建立了立量的质量目标。开发活动中的生产率和质量是可度量的。已建立过程数据库。已实现项目产品和过程的控制。可预测过程和产品质量趋势，如预测偏差，实现及时纠正。
5. 优化级：可以通过采用新技术、新方法、集中精力改进过程。拥有防止出现缺陷、识别薄弱环节与加以改进的手段。可取的有效性的统计数据，并可据此进行分析，从而得出最佳方法

> CMM/CMMI将软件过程的成熟度分为5个等级简单理解：
> 1、初始级：工作无序、无章法、无制度
> 2、可重复级：基本的制度和标准
> 3、已定义级：标准化、文档化、完善的制度
> 4、已管理级：质量定量、可度量的
> 5、优化级：有新技术、新方法，可以避免缺陷，

9、ISO/IEC9126软件质量模型

> ISO/IEC9126软件质量模型是一种评价软件质量的同意模型，包括3个层次：
>
> 1）质量特性、
> 2）质量子特性
> 3）度量指标

其中六个质量特性与二十一个质量子特性的关系如下表：

6个质量特性	21个质量子特性	含义
功能性	适合性	软件产品为指定的任务和用户目标提供一组合适功能的能力
	准确性	软件提供给用户功能的精确度是否符合目标
	互用性/互操作性	软件与其它系统进行交互的能力
	依从性	遵循相关标准
	安全性	软件保护信息和数据的安全能力
可靠性	成熟性	软件产品为避免软件内部的错误扩散而导致系统失效的能力
	容错性	软件防止外部接口错误扩散而导致系统失效的能力
	易恢复性	系统失效后，重新恢复原有的功能和性能的能力
可用性	易理解性	软件提供给用户的信息要清晰，准确，且要易懂，使用户能够快速理解软件
	易学性	软件使用户能学习其应用的能力
	易操作性	软件产品的用户能易于操作和控制它的能力

质量特性及定义	质量子特性	含义
效率	时间特性	软件处理特定的业务请求所需要的响应时间
	资源特性	软件处理特定的业务请求所消耗的系统资源
可维护性	易分析性	软件提供辅助手段帮助开发人员定位缺陷产生的原因，判断出修改的地方
	可修改性（易改变性）	软件产品使得指定的修改容易实现的能力
	稳定性	软件产品避免由于软件修改而造成意外结果的能力
	易测试性	软件提供辅助性手段帮助测试人员实现其测试意图
可移值性	适应性	软件产品无需作相应变动就能适应不同环境的能力
	易安装性	尽可能少地提供选择，方便用户直接安装
	一致性（共存性）	软件产品在公共环境中与其它软件分享公共资源共存的软件
	可替换性	软件产品在同样的环境下，替代另一个相同用途的软件产品的能力