《软件工程基础》期末复习

# 一绪论

## **1**、软件工程的来源和研究内容

软件工程的来源和软件产业的发展密不可分，同软件开发的实际工作密不可分。软件工程这门学科，是从实践中碰到的各种问题中产生的，又反过来指导软件开发工作的实践。

软件工程这门学科的主要研究内容包括：

1. 软件开发的理论。任何一门学科发展到一定程度，都会形成一定的理论。软件开发的理论当然是整个软件开发实践的一些经验总结，并且是从这些经验中抽象出来的关于软件开发的一些基本概念、原则、思想等等。
2. 软件开发的方法。在一些公认的理论的指导下，不同的软件开发人员和研究人员，都提出了各种各样的具体方法，这些方法各有优势，也各有不足。但是大多数都已经在实际运用中取得了比较好的效果。
3. 软件开发的标准。任何一个产业，要能够实现工业化的、大批量的生产，必需有一定的大家共同承认和共同遵守的标准。在技术领域，标准的来源主要有两种，一种是由各种标准化组织制定的（包括国际的、国家的、行业的），另一种是由在这个产业里居主导地位的生产厂商事实上形成的，当然，这种标准也会根据市场情况逐渐地被标准化组织所采纳。

软件的本质特性：复杂性、一致性、可变性、不可见性

一致性：软件必须运行在相应的设备上

必须按照已有的标准设计，要与已有的设计、硬件、软件相适应

随着时间、发展、需求产生变化

## **2**、软件的定义和特点 **（要理解 可能变换语义出选择）**

软件是指计算机系统中与硬件相互依存的另一部分，包括程序、数据及相关文档的完整集合。

软件的主要特点为：

1. 软件不是物理实体，而是一种特殊的逻辑实体；
2. 软件的开发和设计过程是以人为主的，是通过设计将人的智力成果转化成软件；
3. 软件设计的结果是没有误差概念；
4. 软件一旦开发成功，它不会随着时间的流逝老化；
5. 软件的开发过程和生产过程是统一在一起的，开发过程的结束也就是生产过程的结束；
6. 软件完成后有一个较长时期的维护工作。

## **3**、软件工程的产生背景：

在本世纪 60 年代末由于缺乏对软件开发工作的研究，缺乏正确的理论和方法导致的“软件危机”，表现在：

1. 软件的质量难以达到用户的要求，并且极易发生故障，可靠性差；
2. 开发一个软件的成本和进度难以预测，也难以控制；
3. 开发出来的软件维护困难；
4. 软件开发工作难以管理。
5. 软件的成本逐年上升。

造成“软件危机”的原因首先是由于软件开发产业的发展时间不长，积累的经验还没有上升成为能够指导实践的理论；但是，从最早的程序员开发软件的方式中遗留下来的很多错误观念，也是使软件开发工作难以跟上整个计算机技术发展水平的重要原因，这些错误观念包括：

1. 开发软件就是编程。
2. 只要向用户提交了程序，并且可以运行，软件开发工作即告结束。
3. 软件是灵活的，可以很容易修改。
4. 增加人员可以加快进度。

正是因为面对着软件开发中的各种各样的问题，在上世纪 60 年代末提出了

“软件工程”这一概念。

软件危机的出现原因：

1. 对软件开发缺乏正确的理论指导
2. 软件人员与用户缺乏充分交流
3. 对软件开发过程缺乏整体认识
4. 对软件产品缺乏有效一致的质量评价标准

## **4**、软件工程的定义和基本目标： **（转变概念）**

“软件工程”的定义（**IEEE**）：将系统的、规范的可度量的工程化方法应用于软件开发、运行和维护的全过程及上述方法的研究。

软件工程研究的目标：在有限的时间、人力、物力、财力的基础上，生产出符合用户需求的软件产品，提高软件生产率。

软件工程的组成：方法、工具和过程。

软件工程的目标 9点

可理解性、功能性、安全性、可靠性、有效性、可扩充性、可维护性、可重用性、可移植性

软件的概念

软件：是计算机系统中与硬件相互依存的另一部分，包括程序、数据及相关文档的完整集合。

## **5**、软件工程的七条基本原理：**（知道这些概念）**

任何一门学科，例如数学、物理等学科都有一些基本的原理，实际上就是一些公理。软件工程的 7 条基本原理是：

1. 用分阶段的生命周期计划严格管理；
2. 坚持进行阶段评审；
3. 实行严格的产品控制；
4. 采用现代程序设计技术；
5. 结果应能清楚地审查；；
6. 开发小组的人员应该少而精；
7. 承认不断改进软件工程实践的必要性。

# 二软件工程的生命周期方法学

## **1**、软件生命周期定义和阶段划分：

软件工程认为：任何一个软件，都有产生、发展、消失（灭亡）的过程。这个过程就叫做软件的“生命周期”。它是软件工程的基本概念，软件工程的各种理论和方法都是以软件的生命周期为基础。

软件生命周期的阶段划分为四大阶段：需求分析（问题定义和可行性分析）、设计（概要设计、详细设计、实现与测试）、应用和维护（灭亡）。

可行性分析包括：技术、操作、经济、法律可行性

## **2**、软件工程生命周期方法学的定义和特点

软件工程的生命周期方法学是指严格按照软件的生命周期，采用分阶段、有计划和控制，以及顺序实施的步骤，以结构化分析与设计（Structured Analysis，

**SA**； Structured Design，**SD**）或面向对象分析与设计（**Object-Oriented Analysis**，

**OOA**；**Object-Oriented Design**，**OOD**）为主的软件开发过程。

生命周期方法学的特点是：

1. 分阶段计划；
2. 瀑布式开发模式；
3. 阶段性的技术审查和管理复审——基线；
4. 在各个阶段采用结构化技术 **/** 面向对象技术。

结构化技术是指采取自上而下，逐步求精，单入口，单出口的模块化的分析与设计的模型。

面向对象技术是指在客观世界映射到信息世界时，采用人们认识客观世界的过程中普遍运用的思维方法，直观、自然地描述客观世界中的有关事物——类与对象，并通过抽象性、封装性、继承性、多态性和消息机制等设计类与对象，并建立他们之间关系的分析与设计的模型。

## **3**、生命周期方法学的阶段划分：（要记住 会对应）

1. 问题定义
2. 可行性研究
3. 需求分析
4. 概要设计
5. 详细设计
6. 编码
7. 测试
8. 应用与维护

按照软件工程思想，采用生命周期方法学，则上述的某些阶段，可以简化和合并，但是不能够完全取消，或者破坏瀑布模型的从上至下的流程。

软件生命周期 6个阶段

可行性与计划阶段、需求分析阶段、设计阶段、实现阶段、测试阶段、运行和维护阶段

1. 软件过程模型
2. 瀑布模型（线性顺序模型）自顶向下
3. 原型模型 压缩的瀑布模型 在每阶段任务不明确时能转入下一阶段
4. 增量模型 是一种非整体开发模型
5. 敏捷过程模型 迭代和增量 敏捷过程方法：测试驱动
6. 渐进交互迭代模型 对整个软件系统不断演进的循环往复
7. 变换模型 “形式化”
8. 软件开发方法
9. 结构化开发

结构化设计=数据结构+算法

1. 面向对象开发

面向对象=对象+类+继承+消息

# 三问题定义和可行性分析

## **1**、问题定义阶段的任务、步骤和文档

问题定义阶段的主要任务是：明确要解决的问题究竟是什么。

问题定义的步骤可以分为三步：

1. 问题定义：系统分析员与用户方的项目负责人交谈，并对现场情况进行调研，然后根据所收集到的资料，写出软件的开发目标报告书草稿。
2. 需求描述：系统分析员以开发目标报告书草稿为基础，一方面交用户审阅，一方面与用户的项目负责人、使用负责人、其他有关人员进行详细的讨论，澄清双方认识上不一致的地方和报告中描述模糊的地方，最后对开发目标报告书进行修改，写出正式的开发目标报告书。
3. 需求评审：系统分析员将正式的开发目标报告书提交用户审阅，经用户方负责人签字认可后问题定义阶段结束。

问题定义阶段结束提交的文档为软件开发目标报告书，内容包括：

1. 软件系统名称；
2. 软件系统目标；
3. 软件开发和运行环境；
4. 软件开发背景；
5. 软件开发的成本和进度限制。

## **2**、描绘系统逻辑模型的图形工具

2.1 数据流图

数据流图（**DFD**）纯粹是从系统数据流动的角度来描绘系统的逻辑模型。 数据流图的基本符号只有数据的源点/终点、变换数据的处理、数据存储和数据流线四种。强调一点：DFD 中带箭头的线段表示数据的流动，而不是控制的转移。

2.2 数据字典 **（概念）**

数据字典是表达系统逻辑模型中所有数据元素详细情况的工具，它严格定义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包括：表格、范式（定义式）、Wornior 图等形式。

在编辑数据字典的时候，要注意这样几个问题：

1. 完整性：在 DFD 中的数据流名称在 DD 中必定有相应的名称。
2. 一致性：在 DFD 中的数据流名与在 DD 中定义的数据语义必须一致。

（3）正确性：数据定义真实描述用户领域的数据、结构、文件等信息。

# 四需求分析

## **1**、需求分析的主要任务 **（选择可能涉及）**

需求分析的主要任务是分析出软件开发目标中的所有具体要求，主要包括：

1. 功能需求：系统实现什么功能？
2. 性能需求：包括速度、存储、数据量、安全性和可靠性等指标。
3. 领域需求：包括与领域相关的保密性、可扩展性等。
4. 其它需求：如预防性维护需求、版权、政策等。

## **2**、需求分析与建模

1. 问题定义：功能和性能（约束条件），注意问题说明和定义的一致性和完整性；
2. 结构化分析方法按照结构化的思想，采取从总到分，逐步求精的过程。

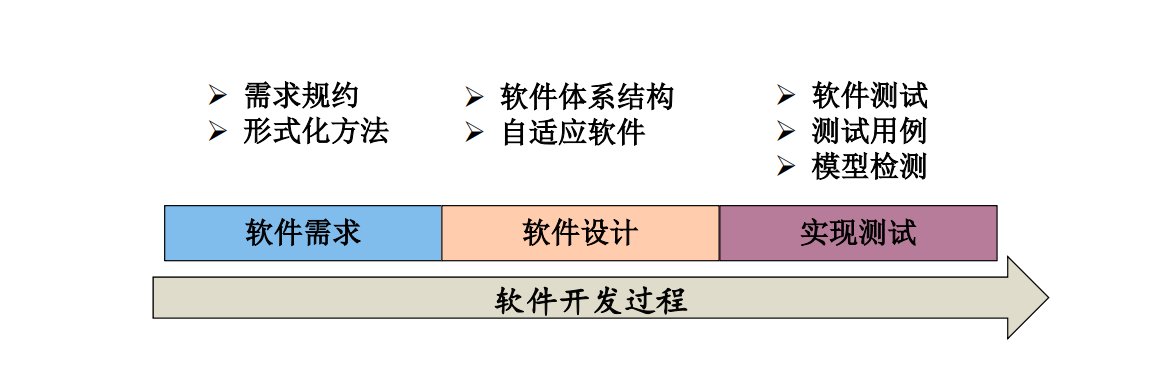
按照面向对象的思想，采取类-对象分析、属性、方法、服务、主题层分析。

1. 需求建模：数据建模、功能建模、行为建模
2. 修正开发计划

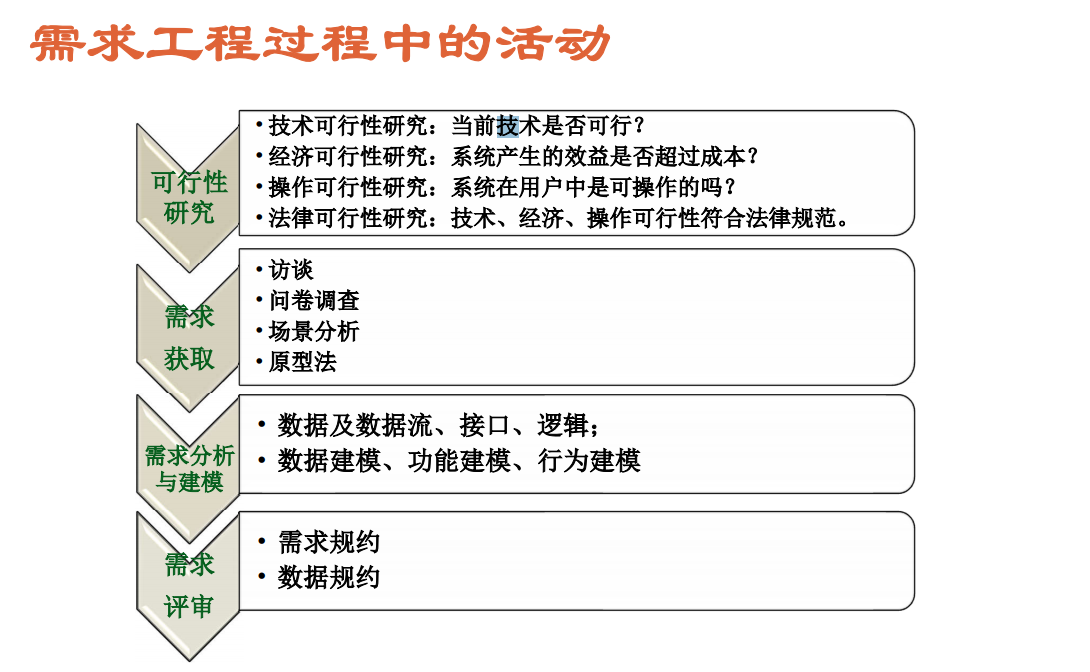
对在可行性研究阶段制定的开发计划进行仔细的修订，对人员的配备、设备的要求、进度安排、费用分配都进行进一步的研究。

软件体系结构设计是系统需求分析阶段的主要任务。

1. 软件开发过程



1. 需求工程过程



1. 结构化需求分析与建模

三个层次的工作：

需求分析：以访谈、问答等形式，用自然语言描述用户的需求

需求建模：选择适当模型一致地描述需求；

需求规格说明(文档)：有效记录建模的过程和内容。

1. 结构化需求建模过程
2. 基于ER模型的数据建模
3. 基于DFD图的功能建模
4. 基于STD图的行为建模

## **3**、需求分析阶段使用的图形工具

* ER 图； （需要知道描述什么）数据对象的属性关系
* DFD：要注意分解前后数据流图的一致性。

###  STD： （需要知道描述什么）

描述系统 状态及引起状态转换的事件来表示系统行为。STD图同时也反映了事 件执行的行为。STD图主要由状态、转换和事件的图形符号构成

* 判定树、判定表； （知道树 表 咋来的）
* PDL 描述：算法描述工具有结构化分析语言、判断树、判定表等。

（2）面向数据流的功能建模（DFD）

适用性：大多数的软件系统

核心方法：按照软件内部数据传递、处理、变换关系，用自顶向下，逐步求精的方法。

三个重要属性： 流向（从加工出发或流向加工）、数据组成、数据流名字

数据流图的四个基本组成部分：

数据加工（转换） 数据流 数据存储 外部实体（源）

（3）面向状态转换的行为建模

状态转换 用状态的跃迁找出条件/事件

1. 需求评审

需求验证的内容:  
（1）有效性检查：功能需求是否符合用户所提出的需求

（2） 一致性检查：系统功能描述、约束是否一致。

（3）完备性检查：是否已包含用户重要的需求、约束，是否已发现 所有的领域性质。

（4）可检验性检查：是否能设计出验证方法，并确定检验的标准， 至少针对主需求进行检查

# 五软件设计基础 **（选择 填空 判断）**

1. 软件设计

设计任务：将需求阶段获得的需求说明（模型）转换为可实现的逻辑系统

设计目标：高内聚 低耦合 的软件模型

1. 几类软件体系结构

①以数据为中心的数据仓库模型

②客户端/服务器模式的分布式结构 发请求 得结果

③层次模型

④控制模型：集中式控制

1. 模块化设计

思想：将一个系统或复杂问题 按照功能拆分为一系列小的系统或简单问题

模块化设计准则：

①软件模块化与分解 最小成本区

②信息隐藏

③模块独立性

④启发式规则

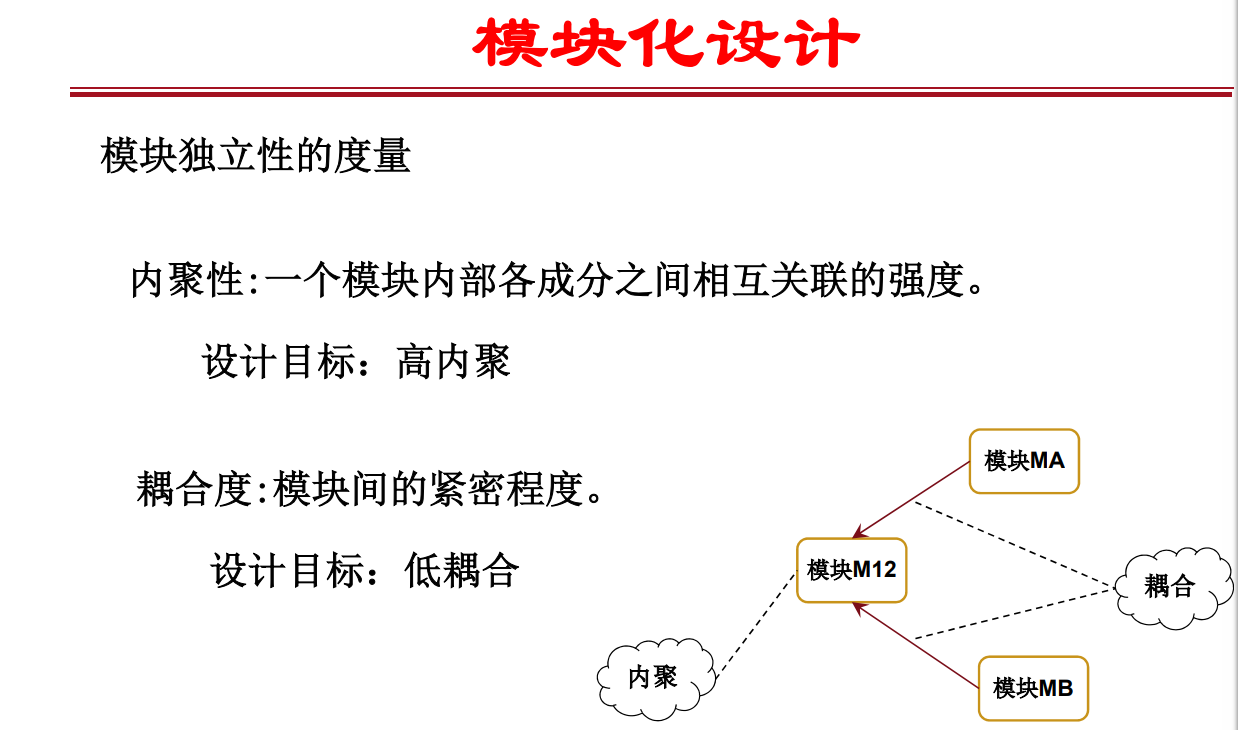
模块化设计流程：

①软件模块化与分解 最小成本区——有一个最优区间

②抽象 实体抽象 接口抽象 设计模式抽象

③信息隐藏 把数据结构与实现过程放在一起

④模块独立性 之软件系统中划分的模块完成一个相对独立的功能 与其它模块的关联尽量只发生在接口上。



## **1**、软件设计的概念和原理

模块的独立性由模块化、抽象和信息隐藏共同体现。

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。模块化的目的就是为了降低软件自身的复杂度，减少开发所需的工作量。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。

1. 抽象

抽象是一个方法论的名词，抽象是指从客观事物中抽取它们共同特征，并对其进行概括和总结；对于它们在其他细节上的差异暂时忽略。在软件设计中，采用抽象的方法也就是采用结构化的思想。

1. 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，其他模块都没有办法访问；每个模块只完成一个相对对立的功能；模块之间交换的仅是为了实现系统功能而必须交换的信息。

局部化：是指应该把可能互相作用的过程和数据尽量放在一个模块内。

信息隐藏和局部化的意义都在于降低模块之间接口的复杂度，提高模块的独立性。

1. 模块独立

模块之间的独立程度的度量方式是模块的耦合和内聚。

1. 耦合 **：（问你xxx属于什么耦合）**

耦合是对一个软件结构内不同模块之间互连程度的度量。按照由弱至强的顺序，模块间的耦合关系有这样几种：

* + 非直接耦合：模块之间没有直接的调用关系；
  + 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合的数据结构；
  + 控制耦合：交换的信息是控制信息；
  + 公共耦合：交换的公共的数据环境；
  + 内容耦合：模块之间的叠加。

对于良好的结构化设计，应该是：尽量使用数据耦合、少用控制耦合，限制公共耦合范围，坚决不用内容耦合。

1. 内聚： **（问你xxx属于什么内聚）**

内聚是指一个模块内的处理和数据之间的关联程度。一个模块的内聚方式按照内聚程度的由低到高排列，有以下几种：

* + 偶然内聚：模块内部是松散的关系（如节约内存等）；
  + 逻辑内聚：逻辑上相关，例如都是输入或者输出操作，用参数区别；
  + 时间内聚：在同一时间内上运行，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，如“打开 / 关系”
  + 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、按顺序执行；
  + 功能内聚：模块是一个整体，完成一个独立的功能。

## **2**、启发式规则

1. 改进软件结构提高模块独立性；
2. 模块规模应该适中；
3. 深度、宽度、扇出和扇入都应该适中；
4. 模块的作用域应该在控制域之内；
5. 力争降低模块接口的复杂程度；
6. 设计单入口、单出口的模块；
7. 模块功能应该可以预测；
8. 内聚高的指标不能绝对；
9. 根据关键任务优化模块划分。

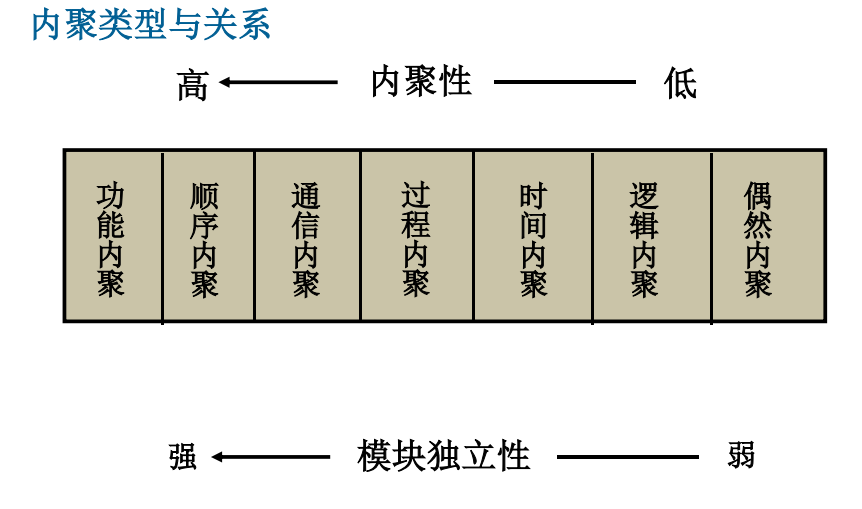
“先让系统正确运行起来，再让系统好起来！”

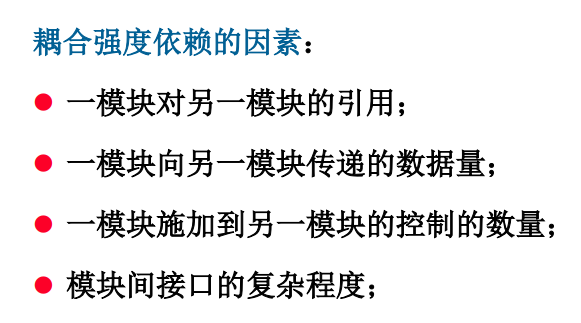
内聚过低，相关功能分散在不同模块中，需要 增加额外的耦合使这些功能聚合在一起，发生变 更时影响多个模块。

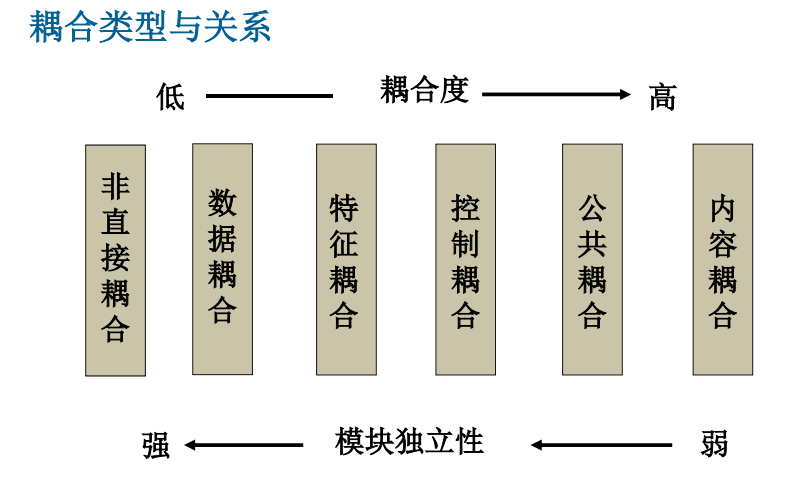
内聚过高，不相关的功能聚集在一个模块中， 耦合度高，发生变更时会产生意想不到的影响。

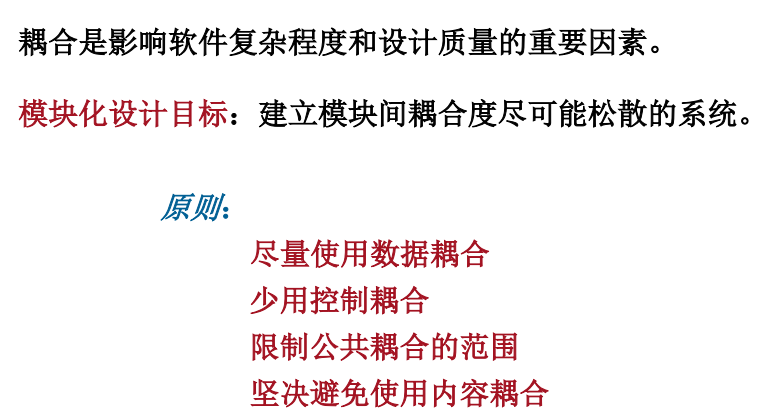
降低耦合度主要因为：

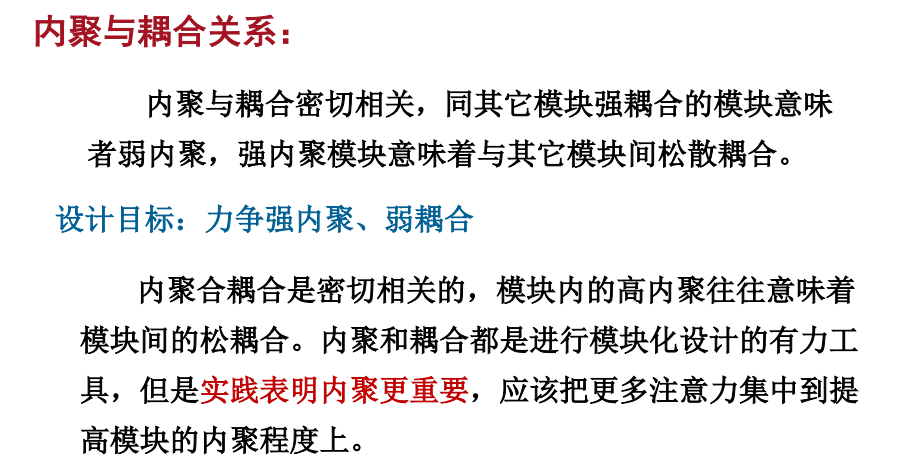
模块间的耦合错综复杂，难以理解，更难以维护。 与不易发生变化的模块之间的耦合关系，很容易受到变化的影响（通常难以避免）。 因此，一般对系统模块间的耦合关系进行评估：双向依赖、循环依赖、单向依赖（期望）。

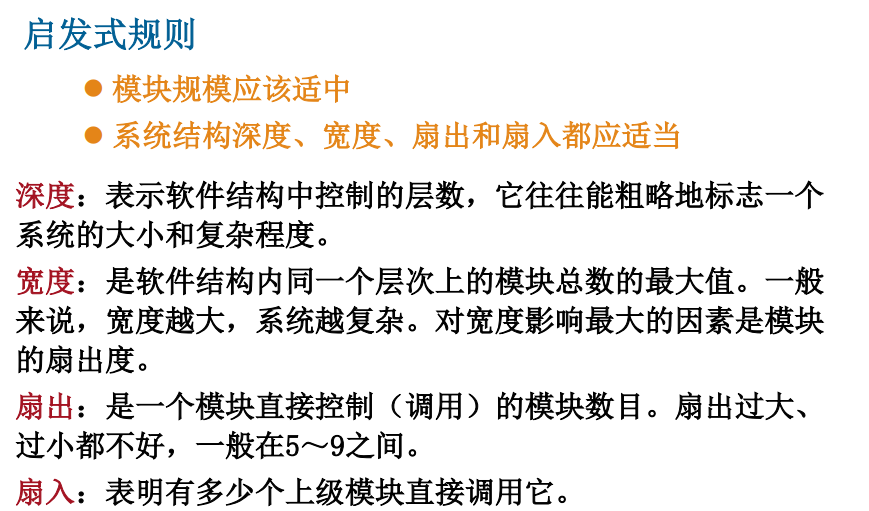


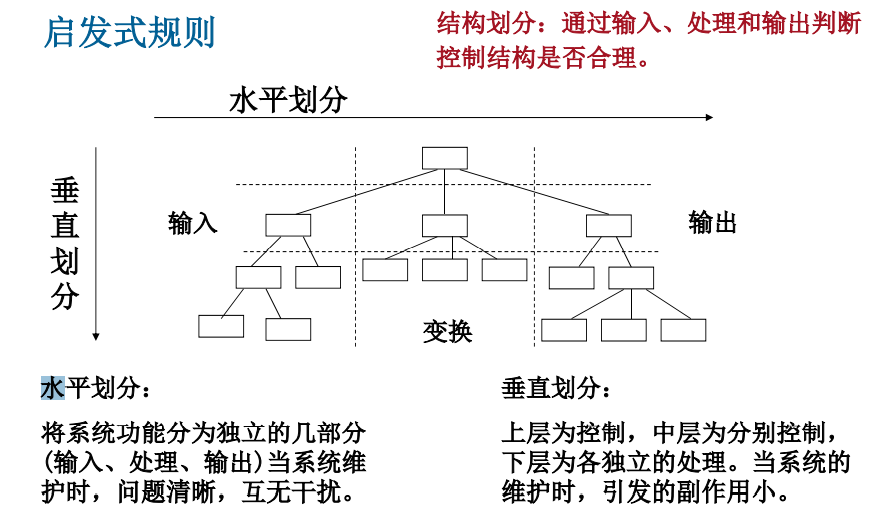












启发式规则：

改进软件结构提高模块独立性

模块规模应该适中

系统结构深度、宽度、扇出和扇入都应适当

模块的作用域应该在控制域之内

力争降低模块接口的复杂程度

模块功能可以预测

1. 界面设计

**界面设计的任务：**

用户特性分析、用户工作分析、界面任务分析、界面类型确定、界面原型评估

**界面设计的原则：**

①系统所有界面操作的统一

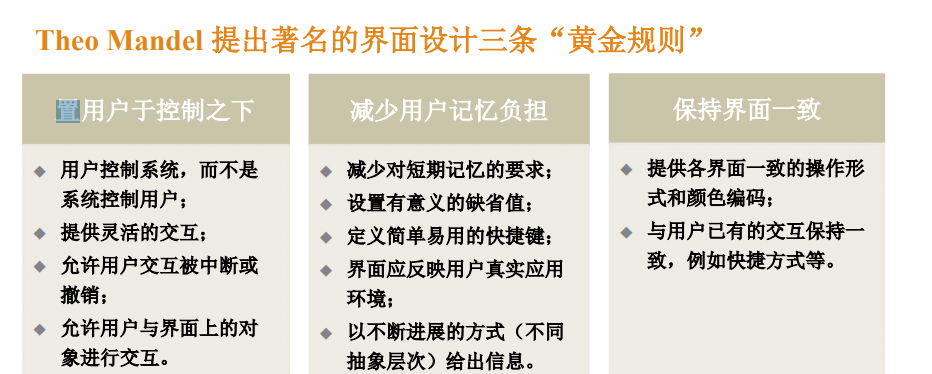
②提供系统运行过程中必要的反馈信息

③提供快捷方式和回滚操作

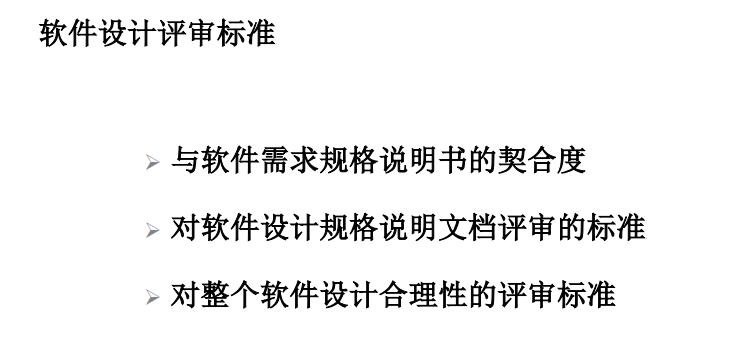
**界面设计的特性：**

简易性、帮助性、容错性、灵活性、个性化

导航方式：线性、层次、网络式、混合式



1. 软件设计评审



六总体设计

## **1**、总体设计的目标

总体设计的目标从根本上来说，就是要确定软件的总体实现方案和总体结构。

总体设计分为：结构设计、系统数据设计和系统接口**/**界面设计。

## **2**、总体设计的步骤 **（可能考设计步骤）**

1. 设想供选择的方案；
2. 选取合理的方案，要选择至少三种方案：低成本、中等成本和高成本的。成本是综合因素的考虑，包括：技术、资金、人力、物力、时间、空间、管理等。
3. 推荐最佳方案
4. 功能分解
5. 设计软件结构
6. 数据设计
7. 制定测试计划
8. 书写文档

## **3**、图形工具

1. 层次图（逻辑结构）

表达软件结构的层次图也使用很多分层的矩形框和分支线条。层次图中每一个矩形框表示软件结构中的一个模块，矩形框之间的线条表示调用关系。

1. 结构图（表示模块间数据的传递）

结构图是层次图的一种改进。它的线条直接标示在模块之间，而且在调用线旁边使用带箭头的线段来表示模块之间传递的信息。

## **4**、面向数据流图的设计方法**(**事务流，变换流**)**

1. 事务流和变换流的形态；
2. 变换流的映射方法：自动化边界的划分；
3. 事务流的映射方法：事务中心的确认； （尽可能少入多出）

# 七详细设计

## **1**、详细设计的任务、步骤和文档

详细设计阶段的主要任务是对每一个模块的内部细节进行具体的设计，实现所有软件的功能。

按照什么顺序去完成每一个模块内部细节的设计，从系统整体结构实现上来说，有两种不同的开发方式：自顶向下和自底向上。

## **2**、结构化程序设计

1. 基本概念

经典的结构化程序设计仅使用顺序、选择、循环这三种基本控制结构的组合来构成整个软件。

1. 结构化程序设计原则

结构化程序设计的原则有如下几条：

* + 使用基本控制结构完成所有软件控制结构的构造。
  + 设计软件控制结构的过程应该采用逐步求精技术。
  + 所有结构都应该是单入口单出口的。
  + 没有无限循环和不可及结构部分。

## **3**、详细设计的工具

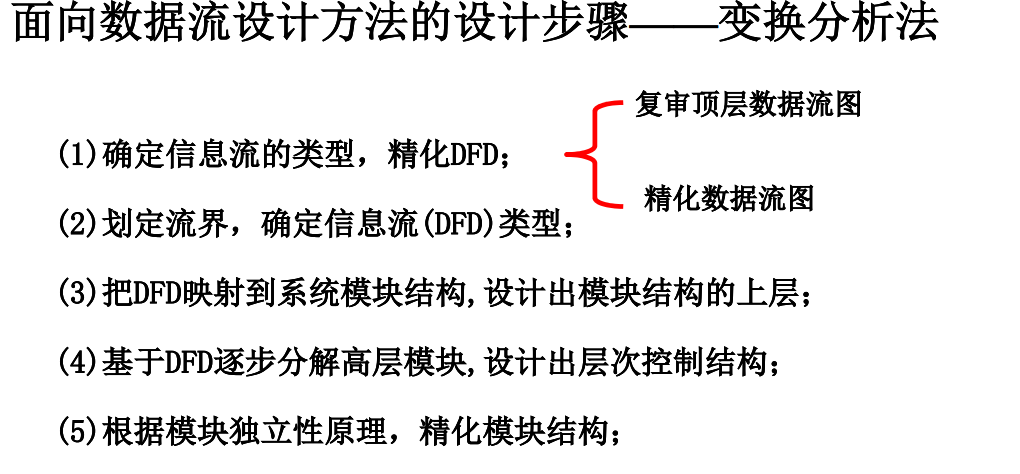
1. 程序流程图；
2. 盒图（NS 图）：要能够用盒图描写简单的处理过程；
3. PAD 图；
4. 过程设计语言（PDL）：要能够阅读简单的 PDL 描述。

**（要知道什么是事务中心 自动化的边界 判断填空）**

第四章 结构化设计方法

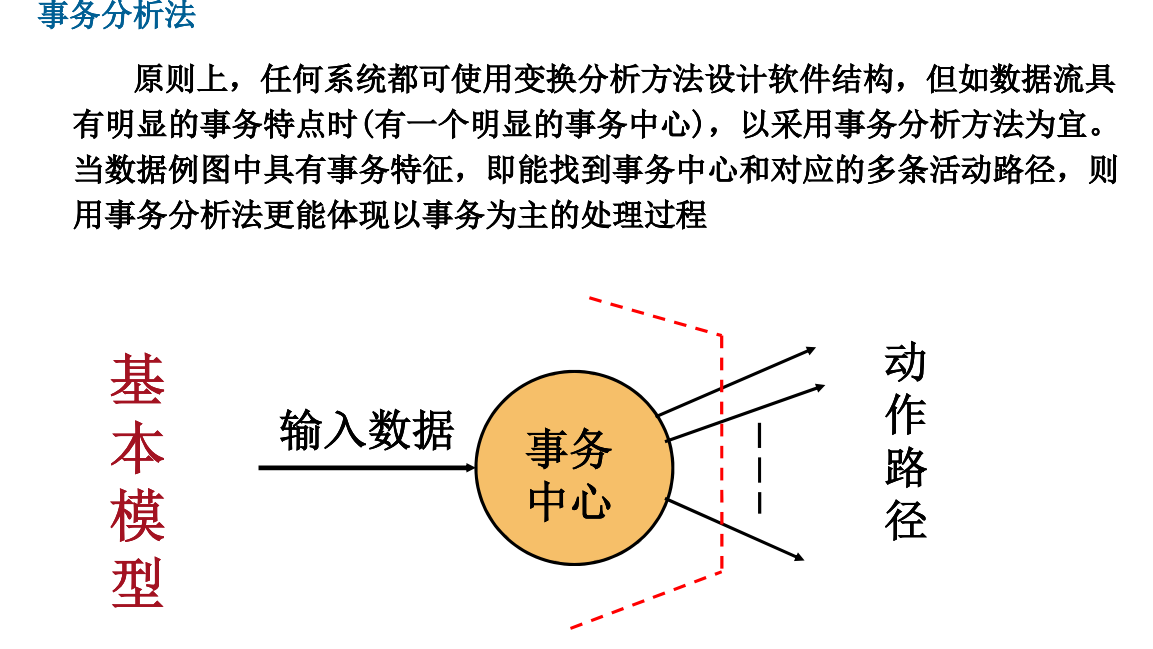
1. 面向数据流的设计方法 —— 变换分析法

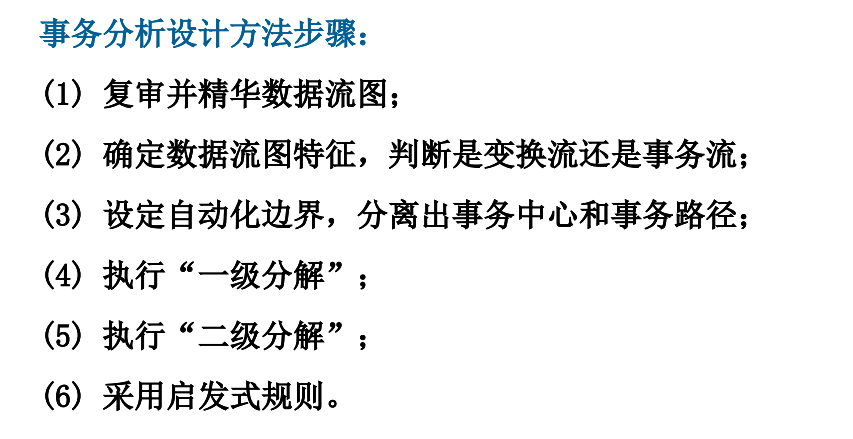
在数据流图中，基本的系统数据流经过输入 基本的系统数据流经过输入、系统变换、输出， 完成对数据的分析处理



1. 面向数据流的设计方法 —— 事务分析法

事务分析法是特殊的变换分析法，它的典型特征在于数据流图中有一 个“事务中心” ，它处理从多条变换输出路径中选择一条活动路径，具有 这种选择处理能力的加工逻辑 这种选择处理能力的加工逻辑（模块）就称为事务中心 就称为事务中心。





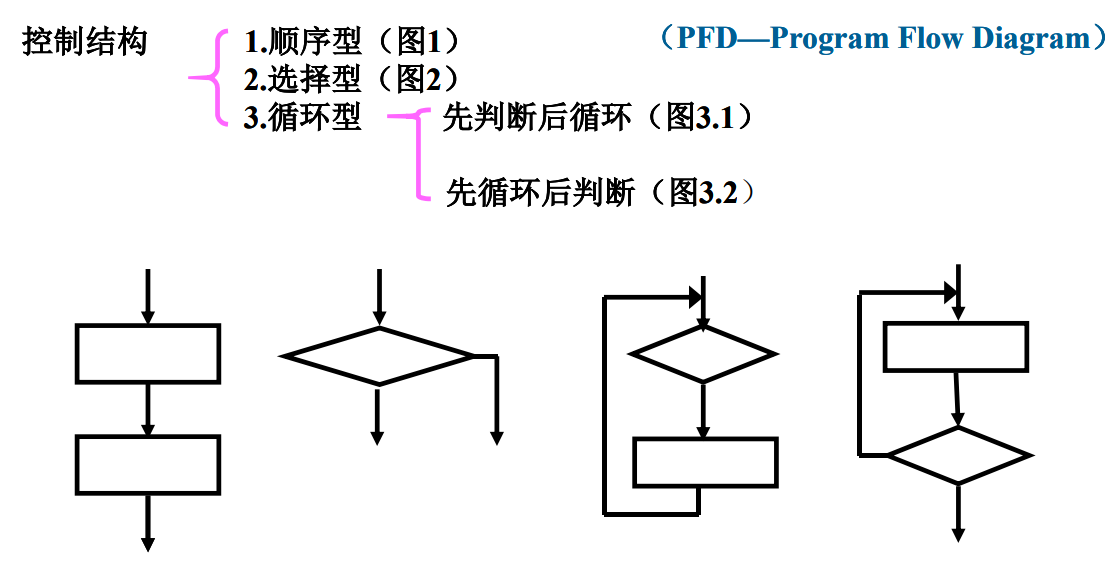
1. 面向数据流的设计方法 —— 混合分析法

在一个大型系统的数据流图中，变换流和事务流会同时出现。按照结构化设计 中分解的思想，上层数据流图整体反映一个主题 上层数据流图整体反映一个主题：变换流或事务中心 变换流或事务中心。在对数据流 图分解的下层图或各条路径活动中 图分解的下层图或各条路径活动中，再确定变换流或事务中心 再确定变换流或事务中心。如此往复迭代，形成混合分析法。

结构化设计的详细设计阶段，主要完成系统各模块功能的 过程描述。 详细设计提供了图形、表格和语言等三类不同工具。

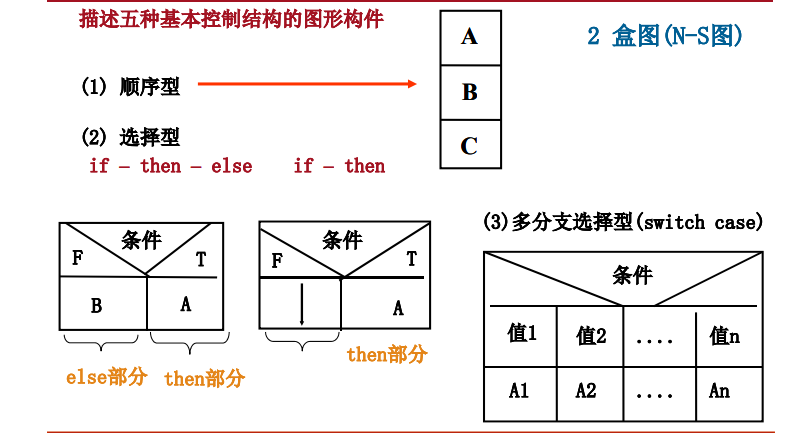
软件详细设计阶段的任务之一是 **过程设计**

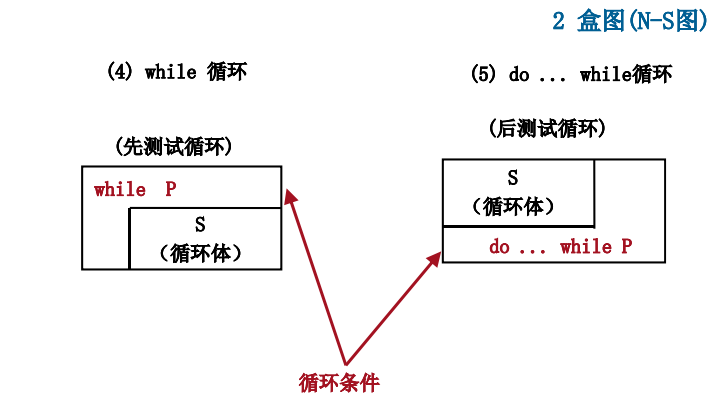
1. 程序流程图



1. 盒图**（引入盒图的目的 和程序流程图的差别）**

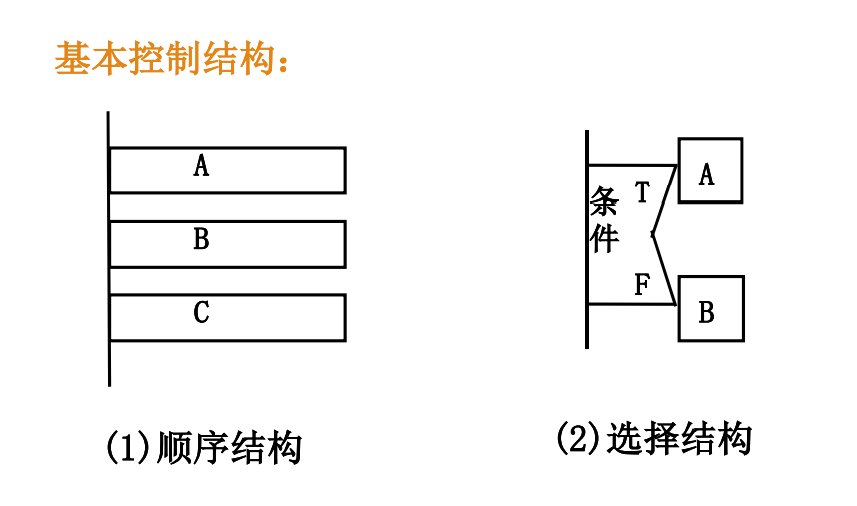
（流程图得控制流随意 盒图可以避免 是嵌套调用）

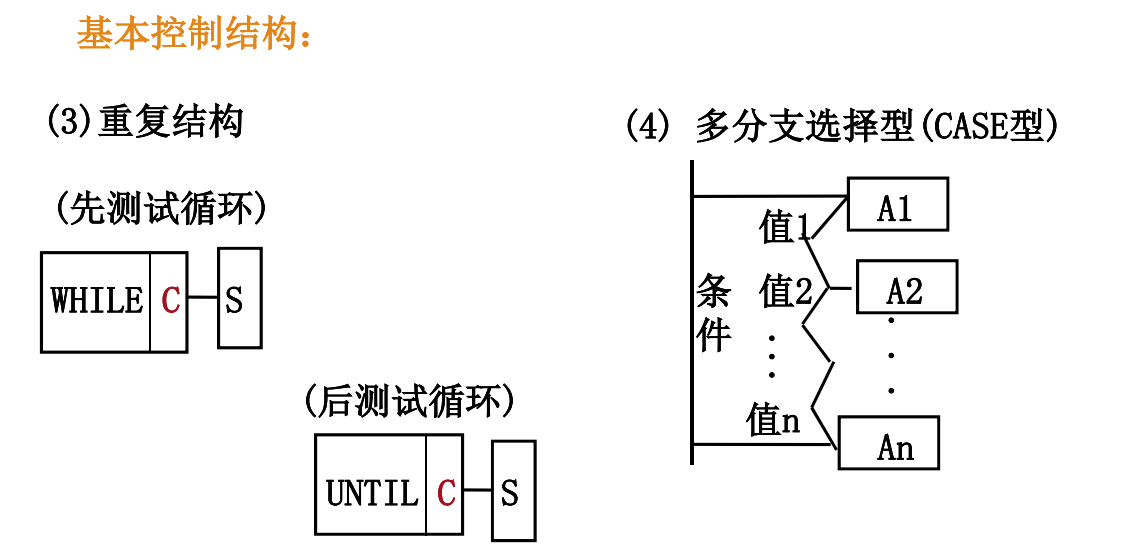




1. 问题分析图**（与程序流程图的关系 有什么优势）**

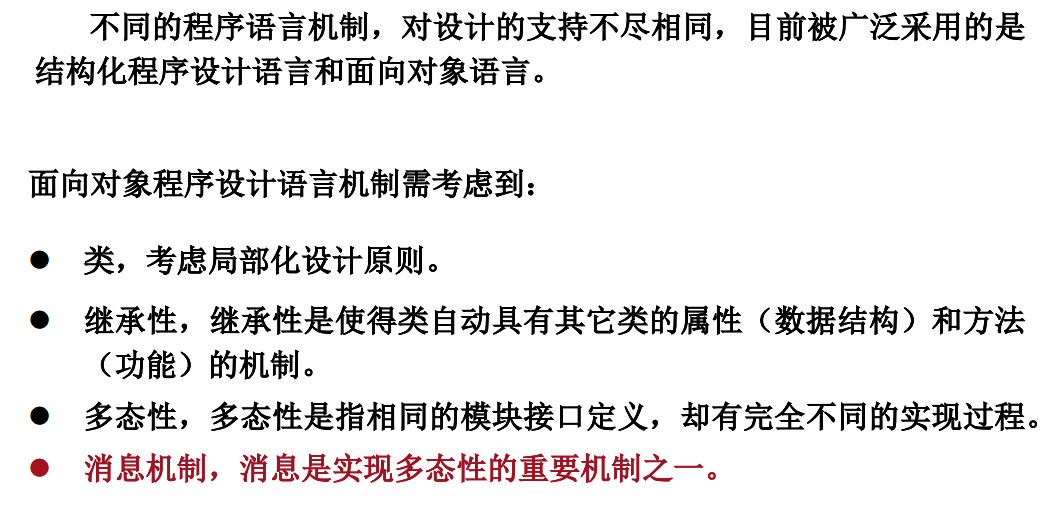
**（没有控制流 避免随意性 自顶向下 自左向右 逐步细化 逐层推进）**





第五章 软件实现

1.软件实现包括编码、测试、调测、优化等一系列工作

2. 

3. 代码评审主要分为正式评审和轻量级评审

**正式评审：**

按照软件需求要求设计代码输入、输出的文档；

代码功能描述和性能描述；

代码编写标准、接口定义规范；

源代码。

**轻量级评审：**

程序员向评审者提供需评审的代码；

程序员可以借助代码自动评审工具；

程序员可以与评审员同步进行软件设计和开发；

定期召开小型的评审会，共同探讨代码编写过程中遇到的各类问题。

八编码

# 九软件测试

## **1**、软件测试的基本概念

1. 测试的定义和目标测试是为了发现程序中的错误而执行程序的过程。

测试的目标，是为了发现错误，而不是为了证明软件是正确的。

软件测试的四个特点：动态、有限、选取、预期

软件测试过程主要包括测试对象和测试方法两部分内容。

测试对象分为程序代码和文档。

文档分为技术文档和用户文档。

**静态测试&动态测试：**

静态：测试对象包括源程序和文档。规格说明的技术审查和管理复审

动态：测试对象针对源程序，运行程序发现存在的错误和问题

1. 测试方法：

**1**、黑盒测试

不考虑程序的内部结构和处理过程，只对它的输入输出进行测试，就是

“黑盒”测试。实际上，黑盒测试完成的是对模块功能的测试。

**2**、白盒测试

如果测试是按照程序的内部结构或者运行的逻辑顺序来进行，这样的测试就是白盒测试。

无论是黑盒法还是白盒法，要想做到穷举测试都是不可能的。

1. 测试策略

**软件测试策略：**

软件测试阶段完成的主要任务有两类：

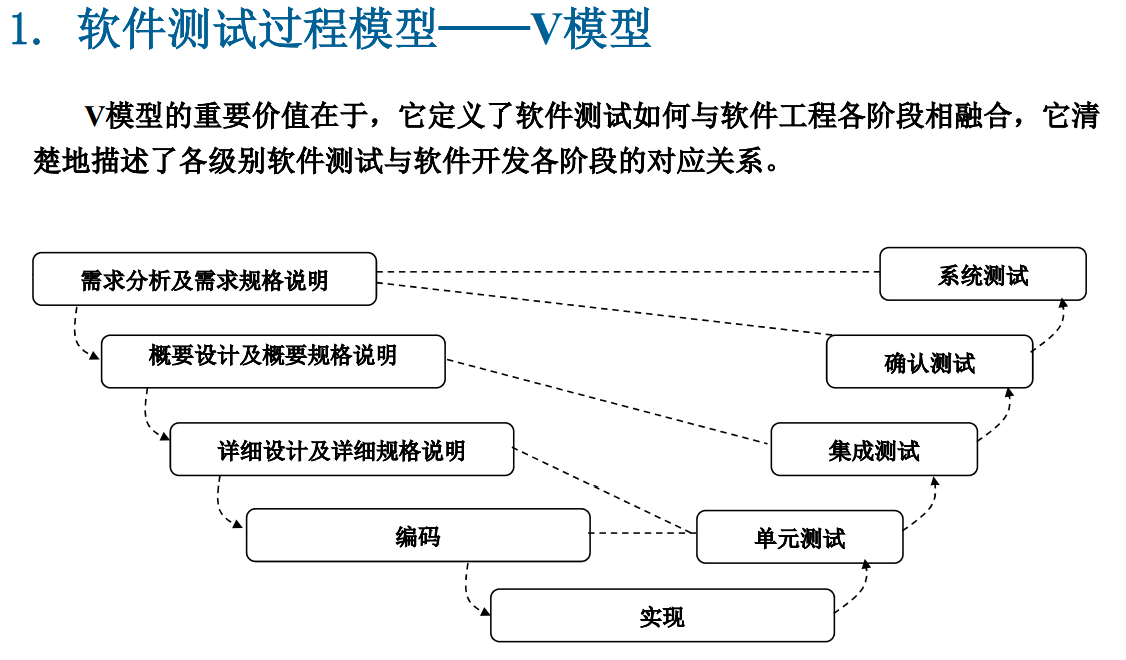
一类是局部模块的测试，它是整个测试阶段的基石。

另一类是软件系统全局结构的测试，它构成整个测试系统的大厦

软件测试的测试策略和基本步骤是：

* 1. 单元测试； **（能用什么技术 以白盒测试为主）**
  2. 集成测试； **（怎么做 跟V模型结合）**
  3. 系统测试；

**4**、验收测试；



如果在测试中，发现这样一些情况：

* 1. 错误非常多，并且很多是需要对软件进行大规模修改的严重错误，那么这个软件的设计一定存在着很大的问题，需要进一步仔细测试或者进行重新审查；
  2. 错误不多，并且多数很容易修正，那么可能说明这个软件的可靠性是不错的，当然也可能说明是测试方案还不够好，还不能够发现其它的错误；
  3. 如果测试的结果一个错误都没有发现，那么说明测试方案一定有严重的缺陷，没有起到有效测试的目的。

1. 测试原则
   1. 一般不测试自己设计的程序。
   2. 要对测试输入有预期输出。
   3. 要对合法输入和非法输入都进行测试。
   4. 多干工作也是错误。
   5. 测试数据必须长期保存。
   6. 程序中的错误率和已经发现的错误的比例成正比。

## **2**、单元测试的任务、步骤和方法

单元测试的测试对象是模块，主要任务是检查每一个模块是否能够按照预定的设计要求进行工作，完成预定的功能。

单元测试包含于编码阶段之中，一般按照以下的步骤来进行：

1. 代码审查：静态测试、代码走查；
2. 运行测试；
3. 测试软件测试：

单元测试一般以白盒测试为主，之后的其它测试以黑盒测试为主。

软件测试报告主要包括软件测试说明和软件测试报告两个部分 。

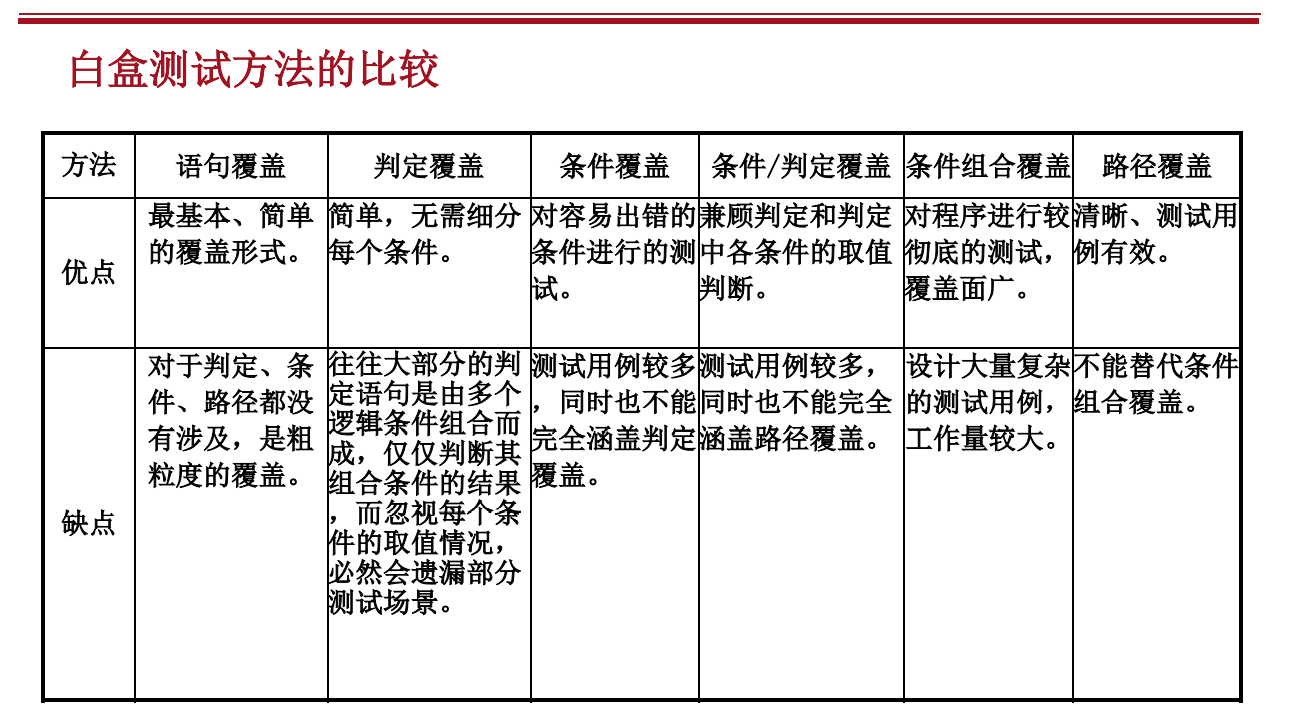
## **3**、综合测试的任务、步骤和方法

综合测试包括集成测试和系统测试，它的任务主要是测试软件结构上的问题，包括模块之间的接口和整体的功能。

综合测试的方法主要区别在于集成子系统的方法，有非渐增式测试和渐增式测试。渐增式测试其实是把单元测试和组装测试或系统测试结合在一起来进行，是一种循序渐进的方法。使用渐增式测试方法，在如何组合模块的过程上，又有两种不同的策略，自顶向下和自底向上，分别需要设计桩模块和驱动模块。

## **4**、测试方案的设计

1. 基于白盒法的测试数据设计（逻辑覆盖） **（要知道覆盖具体是啥）**
   1. 语句覆盖； **每个点都覆盖**
   2. 判定覆盖； **每条边都覆盖**
   3. 条件覆盖； **每个条件判定子表达式结果均有对 错出现**
   4. 判定**/**条件覆盖； **结合判定覆盖+条件覆盖**
   5. 条件组合覆盖； **每个判定的子表达式的取值组合都出现一次**
   6. 路径覆盖。 **每条可能路径都要执行一次 （若出现环 每个环至少经过一次）**



1. 基于黑盒法的测试数据设计

主要考虑等价类划分：至少有两大类，无效等价类和有效等价类。

* 1. 等价类划分； **尽可能一次多测几个有效等价类、但只测一个无效等价类**
  2. 边界值分析； **6点** 有序集集中第一、第二，倒数第二和最后一个元素
  3. 错误推测；

**黑盒白盒比较**

**角度不同：测试技术不同、测试人员不同、文档配置不同**



第七章 UML

（1）

UML通过统一的、标准化的图形符号、元素语法和语义 为OO提供标准

**UML的组成：**

视图 图 模型元素 通用机制

1. 视图 有多个图构成，是在一个抽象层上对系统的抽象表示。视图把建模语言和系统开发时选择的方法或过程连接起来。

常用试图 ：用例视图 设计视图 实现视图 过程视图 配置视图

1. 图 9类：用例图、类图、包图、状态图、活动图、顺序图、协作图、构件图、部署图
2. 模型元素： 代表面向对象中的类，对象，关系和消息等概念

连接关系：关联、泛化、依赖、聚合

1. 通用机制：代表面向对象中的类，对象，关系和消息等概念

用于适应用户需求的扩展机制，包括构造型、标记值和约束

**UML的特点：**

统一标准、实现和过程的独立性、可视化、易学易用性、面向对象特征、可编程性

（1）用例图：由参与者 用例 关系 共同构成

用例图不描述功能实现的细节和性能的约束。

从系统外部描述系统的功能及功能间的关系，用于子系统、包、类等的功能行为描述。

(2)类图 ：类图用于描述类的属性、方法和类间关系

类的内部结构涉及类名、类内部事物的属性、方法及其它们的可见性

（3）包图 是对用例图、类图等的封装，用于描述具有相似功能的模型元素的组合、或组织软件系统结构的层次、或展现物理部署

（4）状态图：用于描述一个对象在生命周期内的所有可能状态，即引起状态改变的事件或条件

（5）活动图：主要图符 活动、状态、判断、同步

描述用例或场景的活动顺序，或描述从一个到另一个活动的控制流

(6)顺序图：顺序图也称为序列图，它用于描述对象间的动态协作关系，并着重表现在时 间先后顺序上，多个对象是如何进行交互的。

（7）协作图：描述类和类间关系，反应一组类共同合作来完成系统功能

（8）构件图：反映了软件组件间的依赖关系，显示了软件系统的逻辑组成结构

（6）配置图 表示系统构件和节点之间的关系 用于描述系统在硬件系统中的部署

**UML的关系：**

1. 关联关系：描述类与类之间的关系构成
2. 泛化关系：描述一个类自动具有另一个类的属性和方法的机制
3. 依赖关系：描述有较强关联的、多个对象间的关系。一个对象对另一个对象数据的访问
4. 实现关系：描述同一模型的不同细化过程，体现的是类间的语义关联。

共享聚合多对多 复合聚合一对多

**UML的通用机制：**

常用的通用机制包括修饰、注释和规格说明

# 十面向对象分析（**OOA**）

## **1**、**OO** 基本概念

1. 面向对象的基本概念，包括：类、对象、属性、方法、消息。
2. 面向对象的三大特征：封装性、继承性、多态性。
3. 类-&-对象之间的关系：关联关系、依赖关系、聚合关系、泛化关系、实现关系、消息连接

# **2**、**OOA** 的过程

建立功能模型——建立静态模型——建立动态模型

1. 建立用例（功能）模型；
2. 确定类－&－对象；
3. 确定类属性；

静态（对象）模型

1. 确定类服务；确定类结构；
2. 确定主题；
3. 建立动态模型。

# **3**、**UML** 的 **OOA** 建模

面向对象的分析方法，就是采用基于UML的图形工具，将现实世界的问题转换为分析模型。

1. UML 的建模图： **（有大题 但前面可能也有其他图！）**

1．（功能）用例建模图形工具：用例图。

2．静态建模图形工具：类图（类对象图）、包图、构建图（组件图）、配置图。

3．动态建模图形工具：顺序图（序列图）、协作图（合作图）、活动图（泳道图）、状态图。

1. UML 的建模过程：

1．用“用例图”、场景描述用户需求；

2．用包图和类图描述系统软件总体框架结构；

3．用顺序图（序列图）、协作图（合作图）、活动图（泳道图）、状态图描述用户操作过程。

## 十一面向对象设计（**OOD**）

### **1**、基本概念

面向对象的设计方法，就是采用基于 UML 的图形工具，将分析模型转换为设计模型。

**面向对象分析与设计的原则：**

**①信息隐藏于模块化 ②单一原则 ③开放封闭原则 ④替换原则 ⑤依赖倒置原则 ⑥LKP原则 （**一个对象应该对其他对象有最少的了解，一个类对自己依赖的类知道得 越少越好）

**⑦基米特法则（**一个 对象对其它对象尽可能少的理解**）**

**⑧聚合优先原则（继承与聚合）**

### **2**、**OOD** 的主要过程

1. 系统设计：分层、分块、抽象；
2. 类-对象设计：精化类的属性和方法、精化类间关系；

**在一般情况下，如果只是使用类B提供的方法，聚合要比继承方式好。**

1. 数据设计：实体类与关系数据库的映射关系；
2. 消息设计：消息的类型和发送；
3. 人机交互设计：界面类、接口类的设计。

结构化设计准则仍然有用：

1）模块化 2）抽象 3）信息隐藏 4）弱耦合 5）强内聚 6）可重用

面向对象的方法可扩展性/可承重性更好

面向对象测试：

1. 确保属性的封装性约束 2.派生类对基类成员函数的测试 3. 对抽象类的测试

十二维护

### **1**、基本概念

1. 维护的定义和分类：

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，称作维护。 **（具体是啥要知道）**

* 1. 纠错性维护； **（2nd）**
  2. 适应性维护； **（3rd）**
  3. 完善性维护； **（主要）**
  4. 预防性维护。 **（主动性维护类型）**

软件维护中出现的问题分为：

一是较少考虑软件的可维护性问题，

二 是软件维护过程中带来的副作用；

三是维护过程的非结构化 （缺乏必要的文档说明..）

完整的软件配置：需求规格说明、设计规格说明、代码、测试文档及相关数据说明

1. 可维护性：

可维护性是指维护人员理解、改正、改动和改进需要进行维护的软件的难易程度。

影响软件可维护性的主要因素有以下这样三种：

* 1. 可理解性；
  2. 可测试性；
  3. 可修改性。（可移植性 可重用性）

### **2**、维护的过程

1. 人员组织： 以维护管理员为核心的人员组织方式。
2. 维护的事件流
3. 维护阶段的文档：

维护阶段自身的文档主要是指**维护记录**。包括软件问题报告和软件修改报告。

软件维护的成本主要由非技术因素和技术因素两大类构成

### **3**、维护的副作用 **（具体副作用是什么 有什么区别 ）**

1. 代码修改的副作用； 引入新的代码错误
2. 数据修改的副作用； 原有的操作与数据不匹配
3. 文档修改的副作用。 文档记录与软件不匹配 引起维护混乱

**提高软件的可维护性：**

1. 软件文档配置
2. 使用能提高维护效率的开发技术和工具
3. 可维护性复审

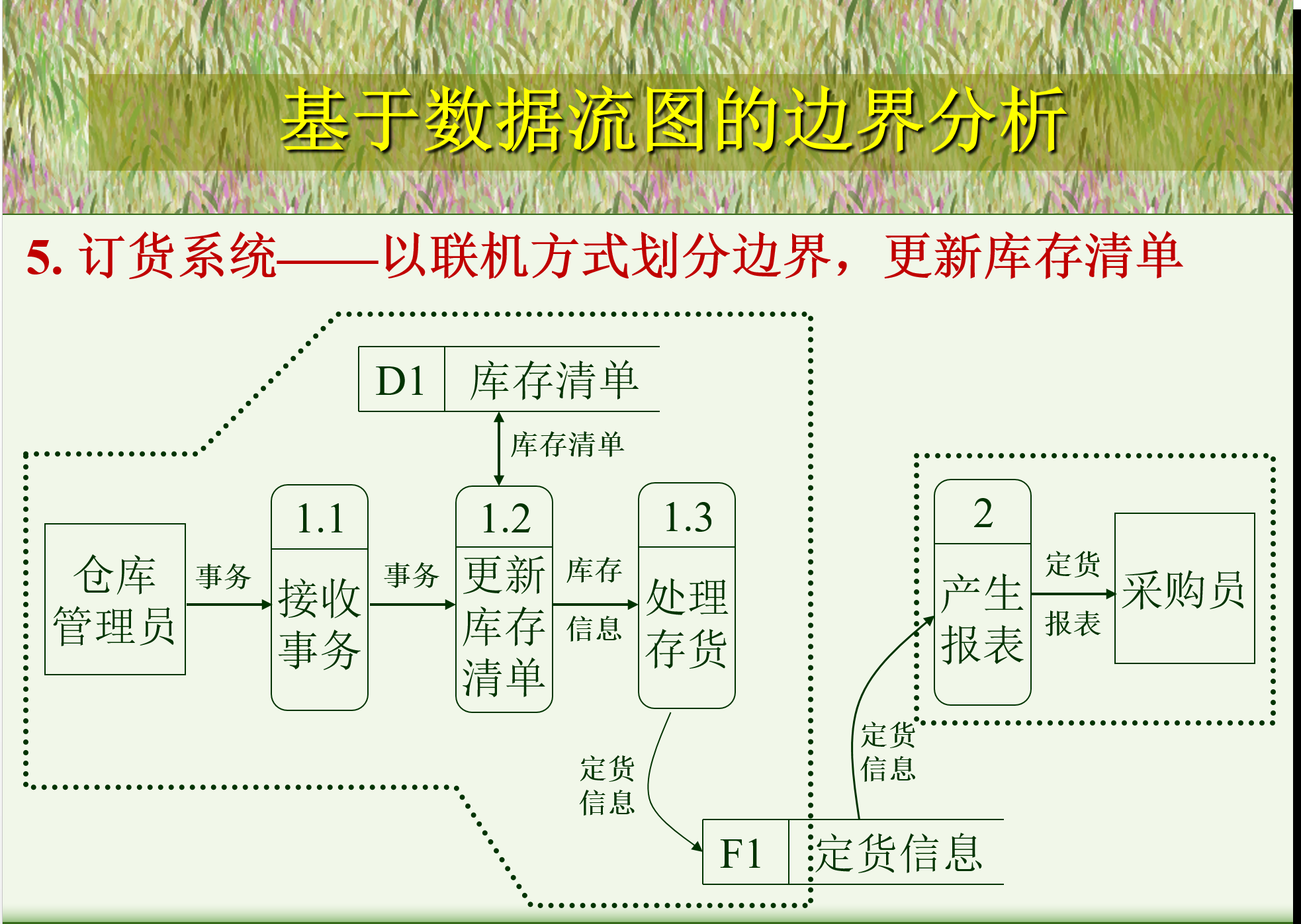
逆向工程

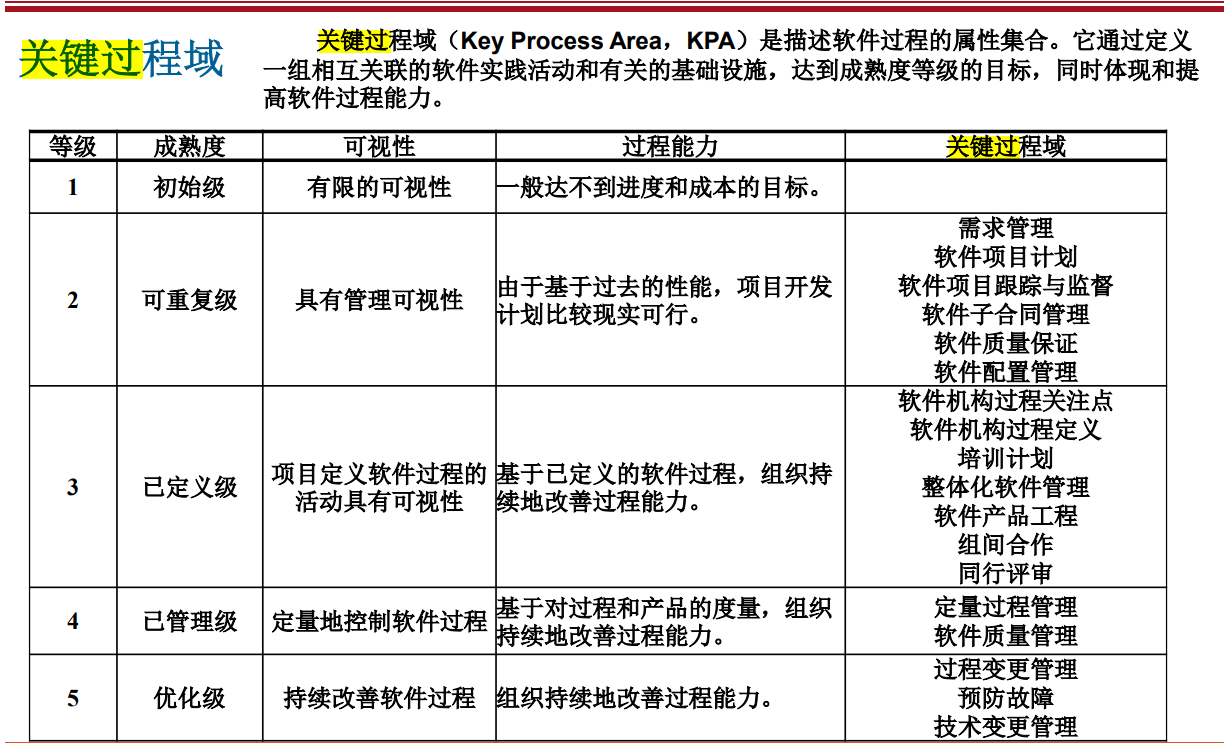
1. 恢复信息的级别：代码级 结构级 功能级 领域级
2. 风险：过程 领域 技术 人员 法律

十三、软件项目管理 **(选择 判断 大概五分)**

1. 项目管理的基本概念：过程、软件过程、软件配置、软件配置项、关键过程、关键过程域。
2. 规模度量：代码行技术；功能点；项目进度管理的甘特图、工程网络图、CMM （软件能力成熟度模型）：

五个等级：初始级、可重复级、已定义级、已管理级、优化级





软件工程的概念 选择 可能会概念变更 要接近书上的语言

软件开发的概念

软件设计的原则

具体的开发的方法 结构化 面向对象

选择 软件的定义和特点 要理解

软件危机错误的概念 要理解

大题：

* 1. DFD一定会考 怎么画 在具体问题里怎么分解

外部实体标少了必定扣分！ 多一点没关系 别少

第一层一定要分出来 关键在于数据流 有没有少 有没有错

分析的过程 1有没有1.1 输入输出对应不对应 （觉得对的都画 画对了加分）一定箭头上面要写东西

* 1. 软件测试应该是路径测试 要画流图！！ 算环形复杂度 path
  2. 如果画类图 要判定类的属性 关联（一对一 一对多 多对多）

关联关系（考察重点！） 泛化关系

用例图 类图 顺序图 活动图 考的可能性大！！！

PDL图不是表达详细设计的图形工具。

软件结构是以\_\_\_\_\_模块\_\_\_为基础而组成的一种控制层次结构。

确认软件的功能是否与需求规格说明书中所要求的功能相符的测试是验收测试。

为了提高模块的独立性，模块内部最好是（功能内聚 ）

软件结构图中，模块框之间若有直线连接，表示它们之间存在（调用关系 ）

具有风险分析的软件生命周期模型是（ 螺旋模型 ）。

需求规格说明书在软件开发中具有重要的作用，它也可以作为软件可行性分析的依据（错）

数据流图建立系统的功能模型，它由数据流、数据处理和数据存贮组成。（N ）

在结构化分析建模中要建立的是对象模型、功能模型和行为模型，它们以数据字典为核心。（ N ）

当A模块调用B模块时，若两个模块之间传递的是数值型参数，则这两个模块的耦合方式是 数值耦合

软件概要设计关键的一步是 确定软件体系结构 \_\_\_\_\_，它直接影响下一阶段详细设计和编码的工作。

数据流图中的箭头表示 数据流 ，椭圆或圆形表示 数据处理 ，矩形表示 数据源点或终点

数据流图仅反映系统必须完成的逻辑功能，所以它是一种 功能 模型。

详细设计与概要设计衔接的图形工具是 SC图

SP 方法中的基本控制结构有 顺序结构 选择结构 重复结构