**软件工程**24-25：

考前总结

1. 名词解释

用例图：文档9.3

一幅用例图包含的模型元素有**系统**、**行为者**、**用例**及**用例之间的关系**。

**工具：**UML提供的用例图是进行**需求分析**和**建立功能模型**的强有力工具。（1分）

**用例模型**：用例图建立起系统模型称为用例模型，用例模型描述的是外部行为者

（actor）所理解的系统功能（1分）

**组成部分：**系统、行为者、用例及用例之间的关系（2分）

* **系统：**描述用例的黑盒
* **行为者：**与系统交互的人或其他系统，代表外部实体通过用例与系统交互
* **用例：**被行为者感受的系统的一个完整功能
* **用例关系：**泛化关系的两种不同形式——扩展关系，使用关系

数据字典：文档2.4

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合。

**元素组成：**数据流、数据流分量、数据存储、处理（1分）

**组成数据的方法：**顺序、选择、重复、关系算符

**用途：**（2分）

1. 分析阶段的工具
2. 包含的每个数据元素的控制信息很有价值
3. 开发数据库的有价值的第一步

耦合：

模块之间的互相连接的紧密程度的度量

种类：耦合从弱到强，独立性从强到弱，分为（2分）

非直接耦合

`数据耦合

标记耦合

控制耦合

外部耦合

公共耦合

内容耦合

状态转化图：文档3.4

状态转换图（简称为状态图）通过描绘系统的状态及引起系统状态转换的事件，**来表示系统的行为**。此外，状态图还指明了作为特定事件的结果**系统将做哪些动作**。

集成测试：文档7.4

集成测试是**测试**和**组装软件**的系统化技术。将模块组装起来，检查其接口是否存在问

题，以及组装后的整体功能和性能表现

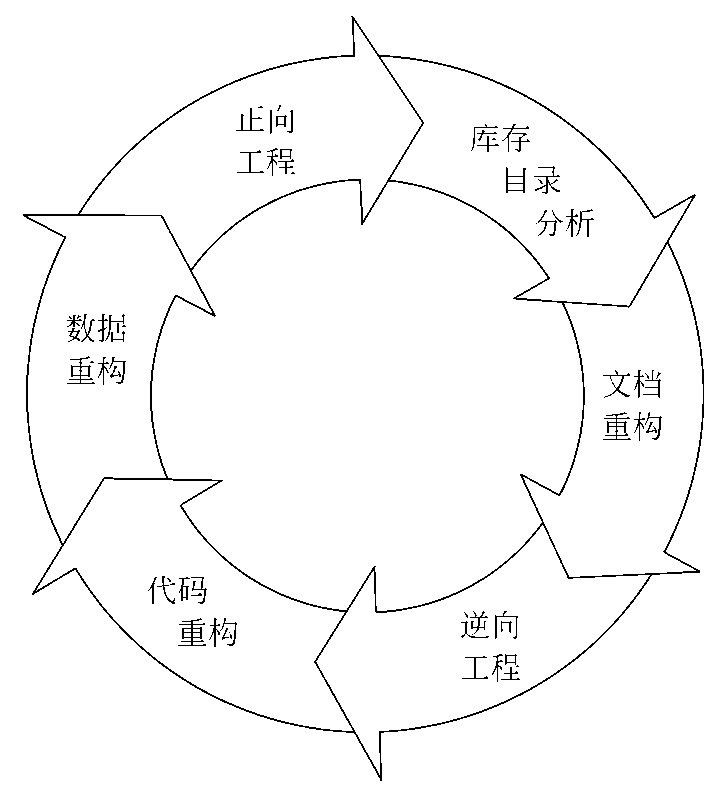
集成测试普遍采用渐增式测试方法，有自顶向下和自底向上两种集成策略

五、简答

rust核心机制：文档7.1

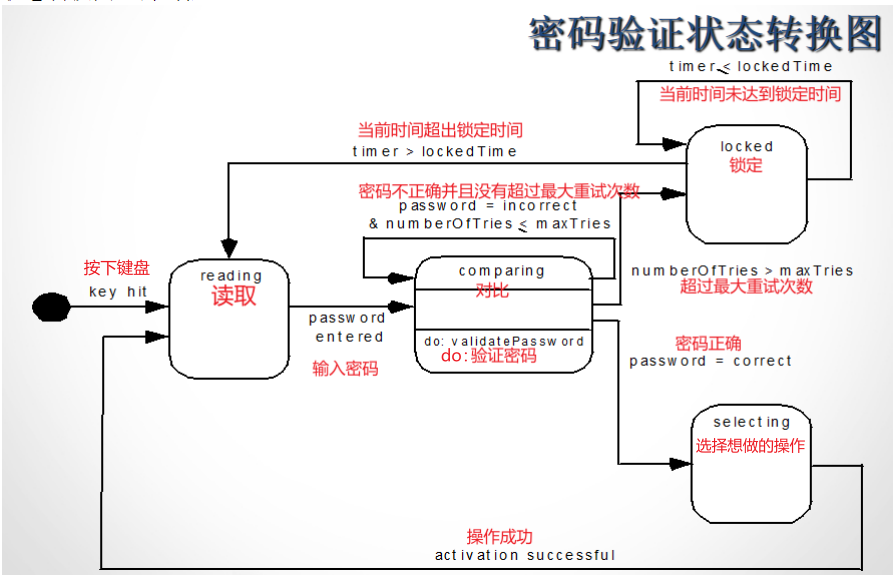
软件再工程定义/步骤：文档8.2

**定义：**软件再工程实质上是预防性维护，即改进未来的**可维护性或可靠性**，为了给未来的改进奠定更好的基础，而进行软件修改。

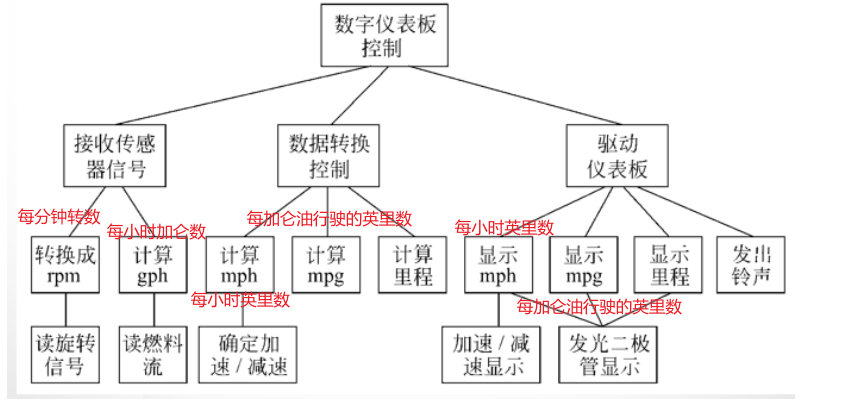


六、综合题

密码验证状态转化图：密码验证：PPT3.6



汽车仪表板：PPT5.5



1. **软件工程学概述**

1.1 **软件定义=软件包括三个组成部分**

软件是多种术语和对象的集合，并将这些术语和对象有效地配置在一起。一般包括程序、文档和数据。

1.2 **结对编程**

* 两名开发者共享一台计算机。一个键入代码，另一个帮他指路。

优点：

1. 所有的设计决定都用了至少两个人的智慧
2. 至少有两个人熟悉系统的每一个部分。
3. 两个人都忽略测试或其他任务的可能性更小。
4. 改变各对的组合可以在团队范围内传播知识。
5. 代码通常至少被一个人复查。

1.3 **软件危机定义**

在计算机软件的开发和维护过程中所遇到的一系列严重问题。

这些问题不是在解决具体问题时遇到的，而是软件开发过程所面临的具有普适性的问题。

* 表现: 开发周期长、成本高、质量差、适应性差、难维护

产生原因：

1. 与软件本身特点有关
2. 软件开发与维护的方法不正确有关

1.4 **软件工程定义和三要素**

|  |
| --- |
| 软件工程是 指导计算机软件 **开发和维护** 的一门工程学科。  采用 **工程的 概念、原理、技术和方法** 来开发和维护软件，  把经过时间考验而证明正确的 **管理技术** 和当前最好的 **技术方法** 结合起来，  以 **经济地** **开发出高质量软件** 并 **有效维护**。 |

软件工程的三个基本要素是

- **过程**：规定了完成各项任务的过程；

- **方法**：完成软件开发的各项任务的技术方法；

- **工具**：软件工程的支撑环境；

1.5 **软件生命周期 各阶段定义**

三个阶段，七个环节

软件生命周期由**软件定义、软件开发和运行维护**（也称为**软件维护**）3个阶段组成，每个时期又进一步划分成若干个环节。

* **软件定义：**确定软件开发工程必须完成的总目标；确定工程的可行性；导出实现工程目标应该采用的策略及系统必须完成的功能；估计完成该项工程需要的资源和成本，并且制定工程进度表。这个时期的工作通常又称为系统分析，由系统分析员负责完成。
* 问题定义
* 可行性研究
* 需求分析
* **软件开发**：系统设计+系统实现。具体设计和实现在软件定义阶段的软件
* 总体设计
* 详细设计
* 编码和单元测试
* 综合测试

前两个阶段又称为系统设计，后两个阶段又称为系统实现。

* **软件维护：**修改软件，使软件持久满足用户需要

1.6 **瀑布模型**

典型的软件过程模型：包括两类软件过程模型，**说明性过程模型**和**敏捷过程模型**

|  |  |
| --- | --- |
| * **阶段间具有 顺序性 和 依赖性** * 前阶段完成后开启后阶段 * 前阶段输出文档为后阶段输入文档 * **推迟实现，**系统设计阶段考虑逻辑模型，不考虑物理实现 * **质量保证** * 每个阶段需完成规定的文档 * 每个阶段结束前需要评审文档   **优点：**   * 强迫开发人员采用规范方法 * 严格规定每个阶段必须提交的文档 * 每个阶段提交需经过质量保证小组的仔细验证。 | 瀑布模型（带反馈） |

1.7 **敏捷模型**

为了使软件开发团队具有**高效工作和快速响应变化**的能力

1.以客户描述的需求内容为驱动;

2.充分认识到一切计划都是短暂的；

3.使用迭代方法重点强调结构行为；

4.采用多样的软件增量行为；

5.适应变化的发生；

1.8 **Scrum 三大特点、三大工件、三种角色**



三大特点

* 可能性的艺术（关注当下）
* 团队自组织、自管理（放权，OKR）
* 面对面沟通

三大工件

* 产品待办列表
* Sprint待办列表
* 潜在可交付产品增量

三种角色

* 产品负责人 (确定功能、标准、DDL、交付内容、通过门禁)
* 流程管理员（管理流程、清楚客户-RD沟通阻碍）
* 开发团队

2. **可行性研究**

2.1 **可行性分析定义**

对需求内容进行初步分析，确定解决问题的可能性，分析不同解决方案的优劣，并提出建议

可行性研究的**目的**，就是用最小的代价在尽可能短的时间内确定问题是否能够解决。

可行性研究的目的不是解决问题，而是确定问题是否值得去解决。

* 本质：粗略的系统分析和设计过程，通常由客户完成。

2.2 **可行性分析三个方面**

* 技术可行性 (现有技术可实现)
* 经济可行性 (经济效益＞开发成本)
* 操作可行性 (用户组织内可用)

技术可行性:使用现有的技术能实现这个系统吗?

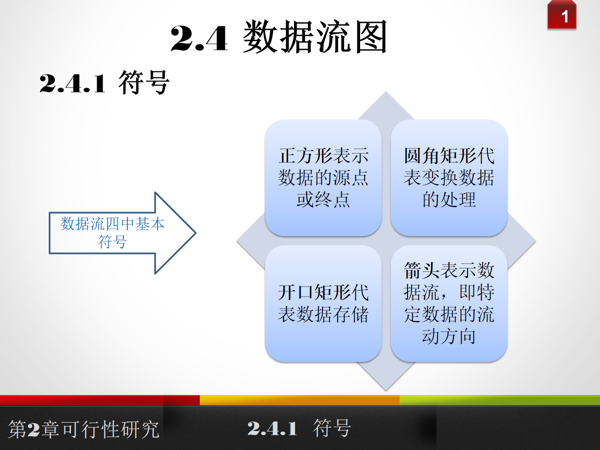
经济可行性:这个系统的经济效益能超过它的开发成本吗?

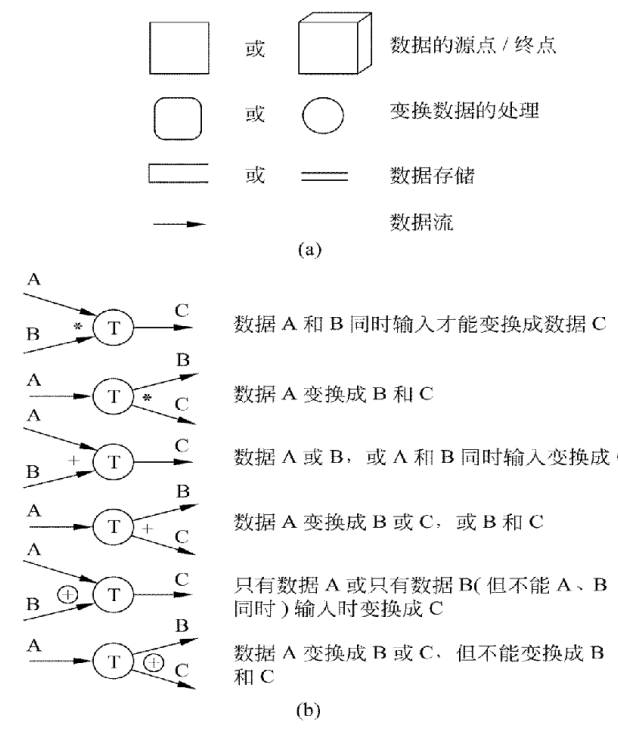
操作可行性:系统的操作方式在这个用户组织内行得通吗?

2.3 **数据流图定义【绘制：课本 p40、PPT 第 35 页】**

数据流图（DFD）是一种图形化技术，它描绘信息流和数据从输入移动到输出的过程中所经受的变换。

* 数据流：箭头
* 加工处理：圆形
* 数据存储：开口矩形
* 外部实体：矩形框





2.4 **数据字典**

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合。

一般说来，数据字典应该由对下列4类元素的定义组成。

1. 数据流
2. 数据流分量
3. 数据存储
4. 处理

3. **需求分析**

3.1 **需求分析的定义**

需求分析是软件定义时期的最后一个阶段，其基本任务是准确回答系统必须做什么，及深入理解用户需求，确定系统的功能、性能、约束等要求，并建立系统的逻辑模型。

目标：开发满足用户真实需求的软件，避免设计与用户期望脱节

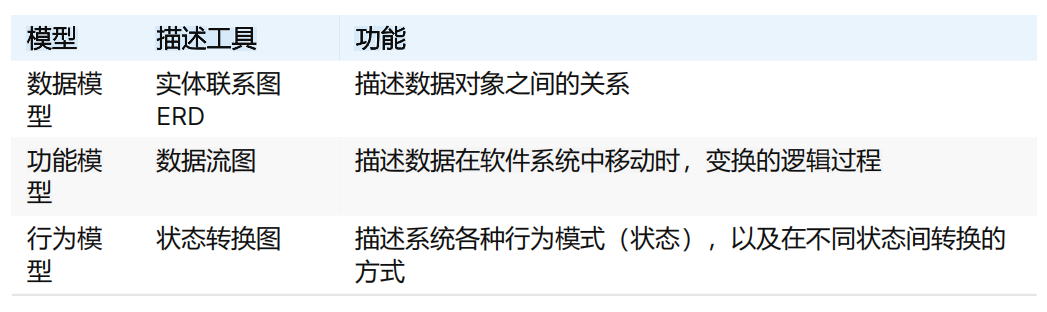
4个任务：分析系统综合需求、数据要求，导出逻辑模型，修正开发计划

3.2 **需求分析（面向过程）需要建立的三类模型**

**数据模型：**主要采用**ERD图**描述，描绘数据对象及数据对象之间的关系。

**功能模型：**主要采用**数据流图**描述，描述当数据在软件系统中移动时被变换的逻辑过程，指明系统具有的变换数据的功能。

**行为模型：**采用**状态转换图**描述，指明了**作为**外部事件结果的系统行为。为此，状态转换图描述了**系统各种行为模式（状态）和在不同状态间转换的方式**。



3.3 **实体联系图定义**

一种面向问题的数据模型，包含 3 种相互关联的信息：**数据对象**、**数据对象的属性**及**数**

**据对象间的关系**。

3.4 **状态转换图【定义，画图】**

状态转换图（简称为状态图）通过描绘系统的状态及引起系统状态转换的事件，**来表示系统的行为**。此外，状态图还指明了作为特定事件的结果**系统将做哪些动作**。

状态：

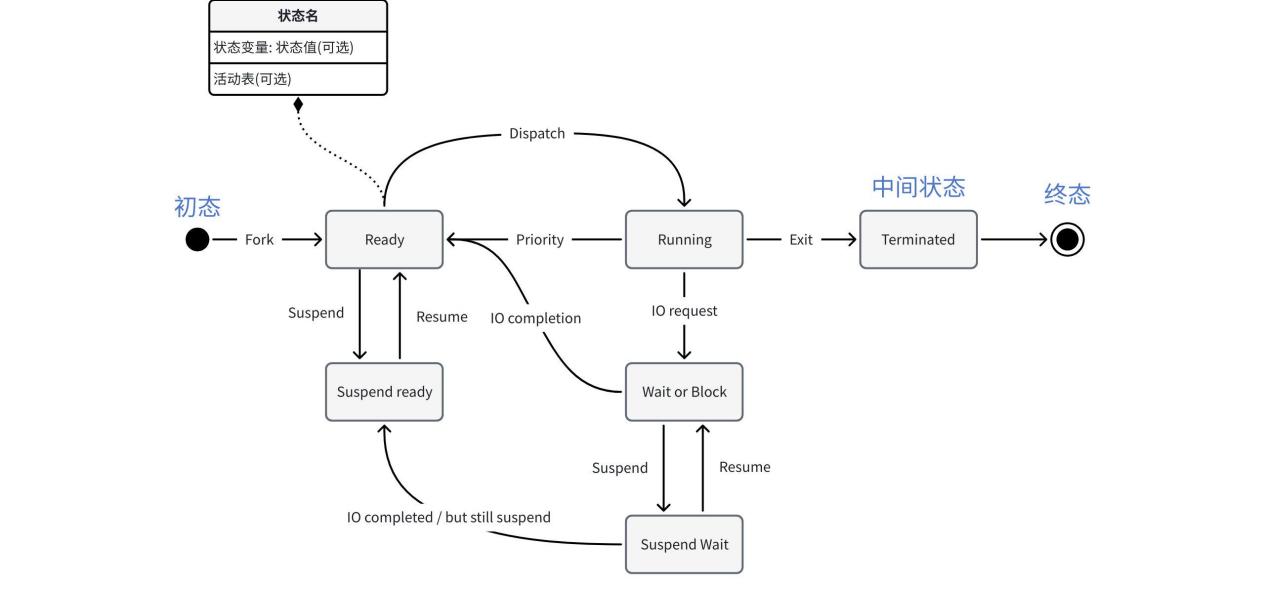
* 可被观察的系统行为模式，初态1、终态\*、中间状态
* 状态：是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。状态规定了系统对事件的响应方式。

在状态图中定义的状态主要有：初态（即初始状态）、终态（即最终状态）和中间状态。在一张状态图中只能有一个初态，而终态则可以有0至多个。

事件：

事件就是引起系统做动作或（和）转换状态的控制信息。

* 事件是在某个特定时刻发生的事情，它是对引起系统做动作或（和）从一个状态转换到另一个状态的外界事件的抽象。



符号：

在状态图中，初态用实心圆表示，终态用一对同心圆（内圆为实心圆）表示。

中间状态用圆角矩形表示，可以用两条水平横线把它分成上、中、下3个部分。**上面部分**为**状态的名称**，这部分是**必须有的**；**中间部分为状态变量的名字和值**，这部分是可选的；下面部分是活动表，这部分也是可选的

* 活动表: 事件名(参数表) / 动作表达式
* 事件名: 例如 entry, exit, do。事件名”可以是任何事件的名称。

entry事件指定进入该状态的动作，exit事件指定退出该状态的动作，而do事件则指定在该状态下的动作。需要时可以为事件指定参数表。活动表中的动作表达式描述应做的具体动作。

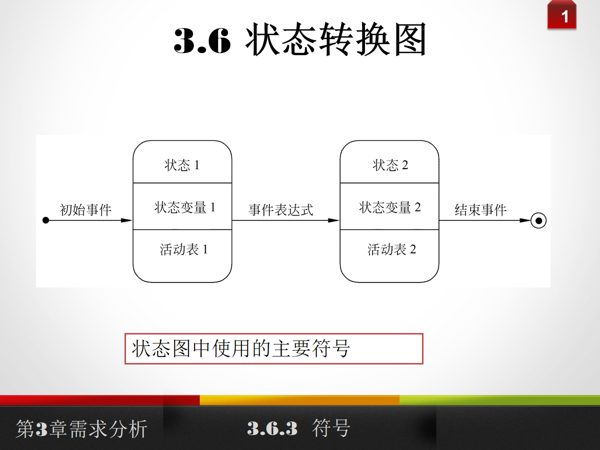
* 状态转换

状态图中两个状态之间带箭头的连线称为状态转换，箭头指明了转换方向。

如果在箭头线上未标明事件，则表示在源状态的内部活动执行完之后自动触发转换。

**事件表达式的语法如下：**

事件说明［守卫条件］／动作表达式。其中，**事件说明的语法为**：事件名（参数表）。



5. **总体设计**

5.1 **总体设计定义**

回答“概括地说，系统如何实现”，又称概要设计或初步设计，核心任务包含**实现系统设计方案**、**设计软件结构**，**明确模块组成和相互关系**。

从工程管理的角度来看，软件设计分两步完成。

* 概要设计： 软件需求 转化为 数据结构和软件系统结构
* 详细设计:即过程设计。通过对结构表示进行细化，得到软件的详细的数据结构和算法。

5.2 **模块定义，三个基本要素**

**模块：** 模块是由边界元素限定的相邻程序元素（例如，数据说明、可执行的语句）的序列，而且有一个总体标识符代表它。模块是构成程序的基本构件。

**模块化：**就是把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。

**三要素：**

* **功能：**描述该模块实现什么功能
* **逻辑：**描述模块内部怎么做
* **状态：**该模块使用时的环境和条件

5.3 **信息隐藏**

**信息隐藏原理：**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，是不能访问的。

**局部化**是指把一些关系密切的软件元素物理地放得彼此靠近。

* 优势：在测试维护期间修改软件，利用信息隐藏原理可以减少各模块之间的相互影响，提高各模块的可维护性。

5.4 **模块独立性，耦合，内聚**

独立性：

软件系统每个模块只涉及软件要求的具体子功能，而和软件系统中其它的模块的接口是简单的。

* 一个模块只具有单一功能且和其他模块没有太多联系，则具有模块独立性。一般采用两个准则度量模块独立性。即模块间耦合和模块内聚

模块的**独立程度**可以由两个定性标准度量，这两个标准分别称为**内聚**和**耦合**。

耦合：

模块间相互连接紧密程度的度量

内聚：

标志一个模块内各个元素彼此结合的紧密程度。

5.5 **作用域、控制域**

作用域：受该模块内一个判定影响的所有模块的集合

控制域：模块本身以及所有直接或间接从属于它的模块的集合。

作用域应在控制域之内

如果一个判定的作用范围包含在这个判定所在模块的控制范围之内，则这种结构是简单的，否则，它的结构是不简单的。

* 模块判定应只影响自身和子模块

5.6 **总体设计原则**

* 模块化：划分独立模块，每个模块完成子功能，模块集成实现系统功能
* 抽象：抽出事物本质特性，忽略细节
* 逐步求精：自顶向下逐层细化软件结构，先处理宏观问题，再逐步完善细节
* 信息隐藏和局部化：模块信息对无关模块隐藏，关系密切的软件元素物理位置靠近
* 模块独立：高内聚低耦合，模块功能单一，接口简单

5.7 **模块扇入，扇出定义**

扇入：直接调用模块的上层模块数。扇入增加，复用性增加，被多个上级共享

扇出：模块直接调用的下层模块数。扇出增加，复杂度增加，应增加中间控制模块

5.8 **变换流，事务流定义**

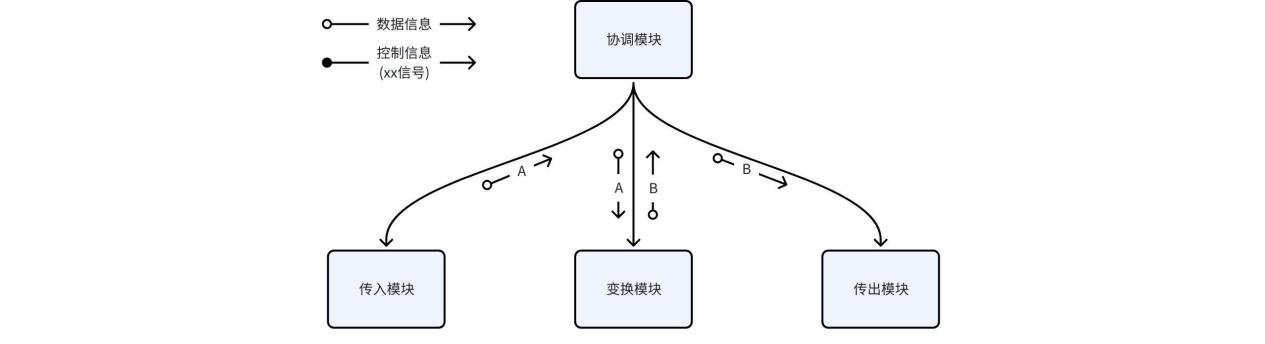
**变换流：**信息沿输入通路**进入系统**，由外部形式变换成内部形式，进入系统的信息通过**变换中心**，经加工处理以后再沿**输出通路**变换成**外部形式**离开软件系统。当**数据流图**具有这些特征时，这种**信息流**就叫作**变换流。**

过程可分为取得数据→变换数据→输出数据

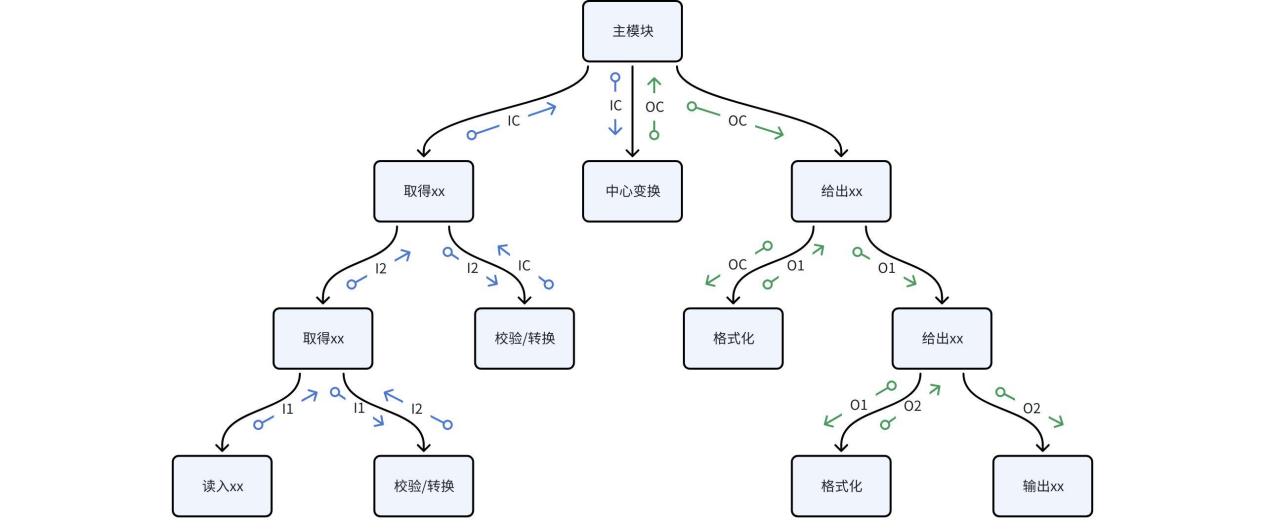
**事务流：**数据沿**输入通路**到达一个**处理T**，这个**处理**根据输入数据的类型在若干个动作序列中选出一个来执行。这类数据流应该划为一类特殊的数据流，称为事务流。**处理T称为事务中心**

过程可分为接收输入数据（事务）→分析事物，确定类型→根据事务类型选取活动通路。

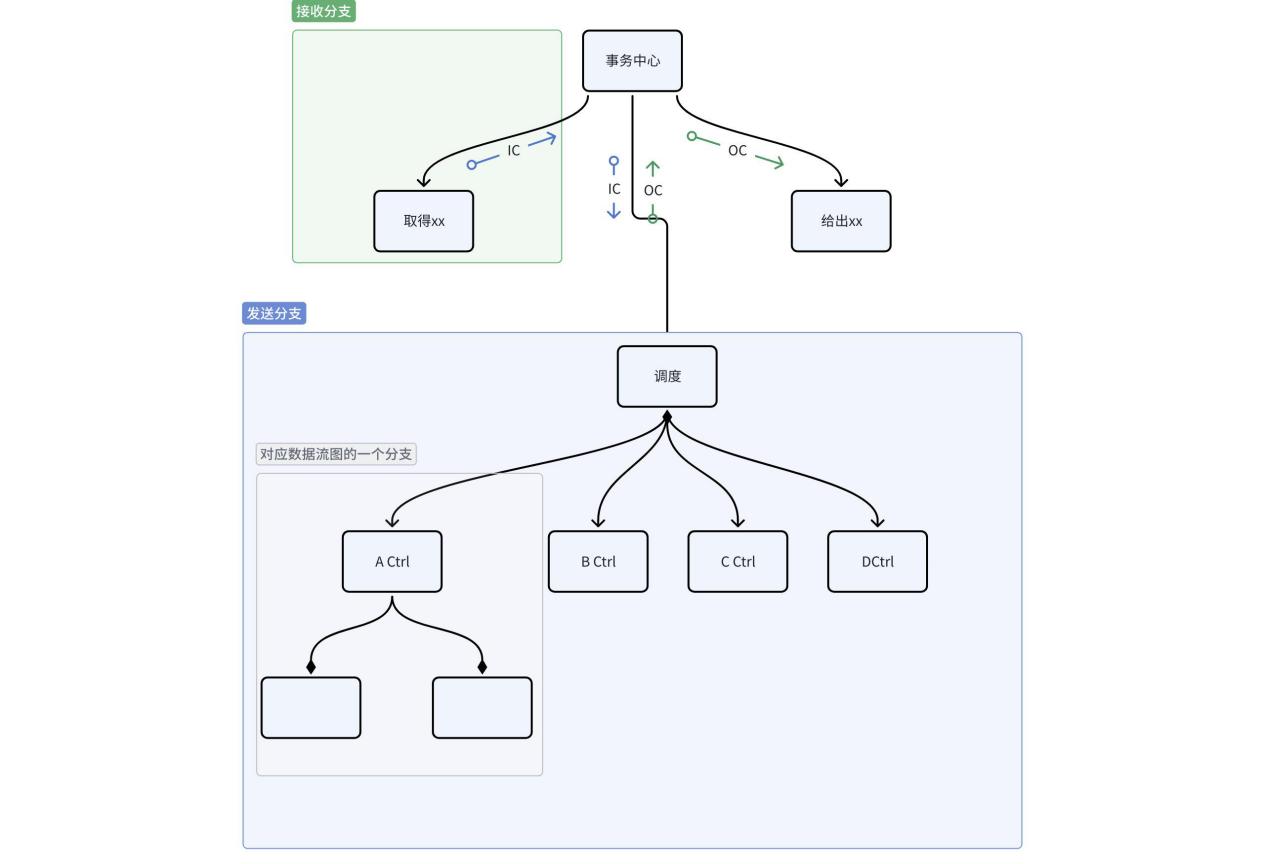
5.9 **软件结构图【画图】**



* 变换流→软件结构图
* 区分有效输入、有效输出、中心变换
* 进行一级分解，设计上层模块
* 进行二级分解，设计输入、输出、中心变换的中下层模块
* 终点：不能细分、用户模块/外部子程序、IO、不宜细分



* 事务流→软件结构图
* 识别事务源
* 规定事务结构
* 识别事务及其操作
* 提取公共模块
* 每一个/组事务→事务处理模块
* 规定事务处理模块下层操作模块
* 规定操作模块下层细节模块



6. **详细设计**

6.1 **详细设计定义，任务要求**

详细设计是软件工程的一个关键阶段，根本目标是确定如何具体地实现所要求的系统，此阶段不是编写具体的程序，而是设计程序的蓝图，结果基本决定了最终程序代码的质量。

要求：

* 确定算法与组织：明确各组成部分的算法和内部数据组织方式
* 选择过程表达形式：描述算法
* 开展详细的设计评审

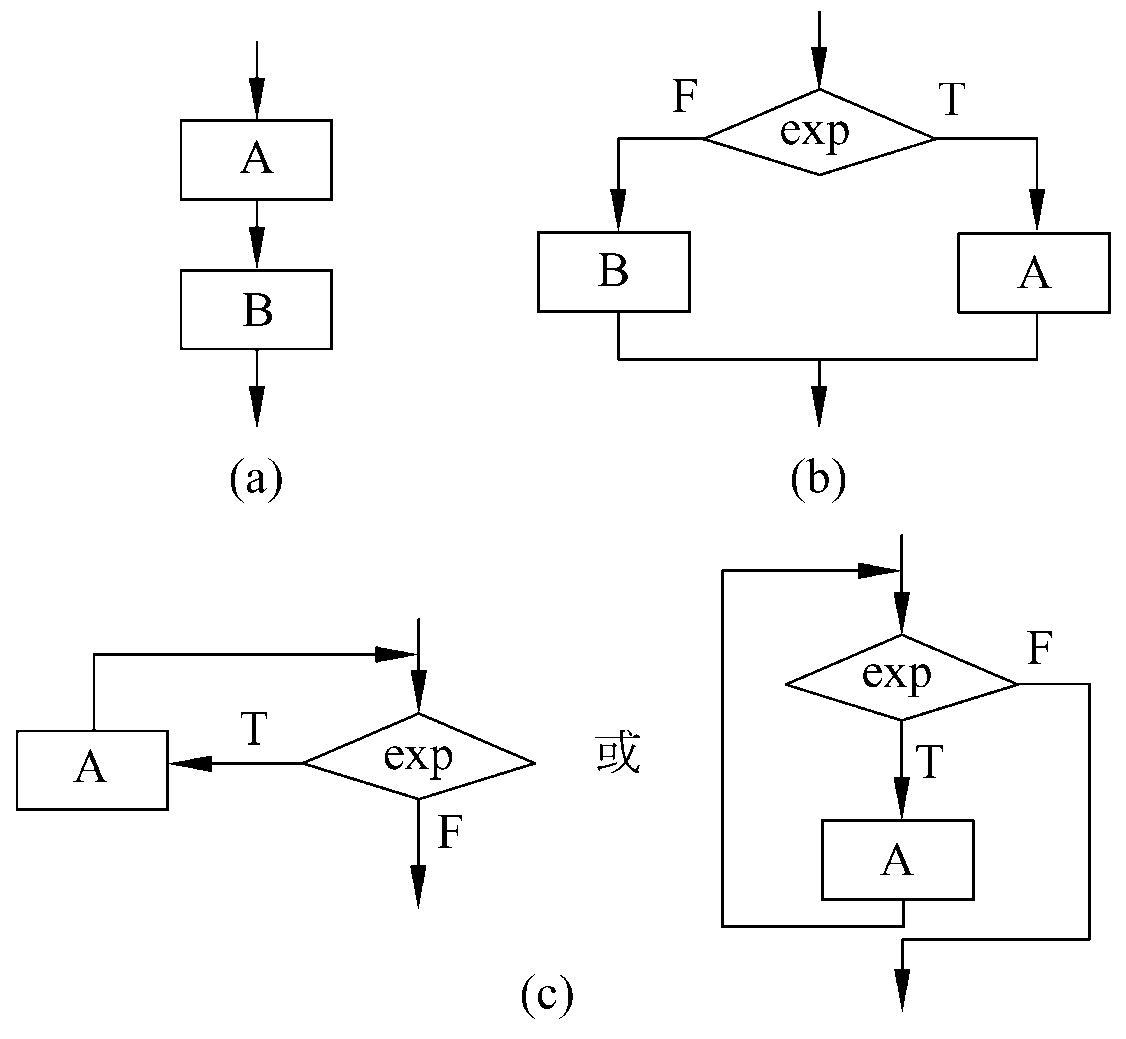
6.2 **结构化程序的三种基本控制结构**

代码块仅通过顺序、选择和循环这 3 种基本控制结构进行连接，并且每个代码块只有一

个入口和一个出口的程序称为**结构化程序**

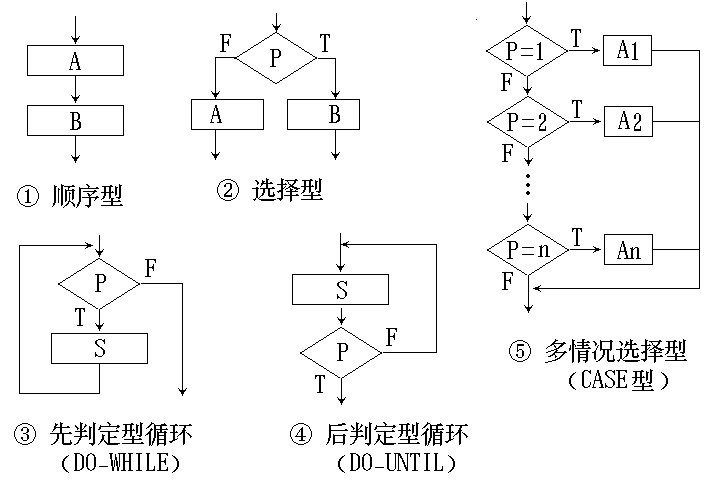
实际上用顺序结构和循环结构（又称DO-WHILE结构）完全可以实现选择结构（又称IF-THEN-ELSE结构），因此，理论上最基本的控制结构只有两种。

* 顺序（）
* 选择（IF-THEN-ELSE）
* 循环（DO-WHILE）



* + do...while & switch case: 扩展的结构程序设计
* + break: 修正的结构程序设计

6.3程序流程图



6.4判定表和判定树**【课本 p127、第六章 33 页】**

**6.5流图绘制【课本 p137、PPT 第六章 57 页】**

7. **实现**

7.1 **Rust 语言核心机制（所有权机制，借用与引用，生命周期机制）**

* **所有权机制**：每个值都有唯一的所有者，值的生命周期和所有者（变量）的生命周期相同，值被赋给另一个所有者时会移动，所有者离开作用域时，所有的值将被丢弃。Rust 通过所有权机制实现无垃圾回收保障内存安全。
* **借用与引用机制**：借用是指通过引用**使用值的行为**。当你创建一个引用时，实际上就是在借用这个值。借用允许你在不转移所有权的情况下访问和操作数据。每个值可以有一个可变引用&mut T 或者 多个不可变引用 &T。
* **生命周期机制**：Rust 中的每个引用都具有生命周期，编译器通过确保数据的生命周期不短于其引用的生命周期，来保障引用始终有效，防止悬垂引用的发生。大多数情况下，Rust 能够自动推断生命周期，并允许通过生命周期注解来指定多个引用之间的生命周期约束关系。

7.2 **软件测试的目的，原则**

**目的**

* 发现错误：系统地找出软件中潜在的错误和缺陷
* 验证功能：证明软件功能性能符合需求说明
* 实施测试收集到的测试结果数据为**可靠性分析**提供依据。
* 测试不能表明软件中不存在错误，它只能说明软件中存在错误。

**原则**

1. 追溯需求
2. 提早制定计划
3. 群集现象：80% 的错误可能集中在 20% 的模块中，应重点测试。
4. 小规模到大规模：单测、集成、确认、系统
5. 不穷举
6. 独立QA
7. 尽早测试
8. 用例=测试输入+预期输出
9. 合理输入+不合理输入
10. 全面检查结果
11. 保存计划、用例、统计、分析报告

（1）     所有测试都应该能追溯到用户需求；

（2）     应该远在测试开始之前就制定出测试计划；

（3）     把Pareto原理应用到软件测试中（测试存在群集现象）；

（4）  该从“小规模”测试开始，并逐步进行“大规模”的测试；

（5）     穷举测试是不可能的；

（6）   为了达到最佳测试效果，应该由独立的第三方从事测试工作；

（7）   应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭；

（8）   测试用例应由测试输入数据和对应的预期输出结果这两部分组成；

（9）   在设计测试用例时，应当包括合理的输入条件和不合理的输入条件；

（10）  应当对每一个测试结果做全面检查；

（11） 妥善保存测试计划，测试用例，出错统计和最终分析报告，为维护提供方便；

7.3 **单元测试定义**

**单元测试**又称**模块测试**，是针对软件设计的最小单位 ─ 程序模块，进行正确性检验的测试工作。

单元测试需要从程序的**内部结构**出发设计测试用例。多个模块可以**平行地独立进行**单元测试。

**目的：**在于发现各模块内部可能存在的各种差错。(采用的方法**主要是白盒测试**方法)

重点：

* 模块接口，全局变量
* 局部数据结构
* 重要执行通路
* 出错处理通路
* 边界条件

7.4 **集成测试**

集成测试是**测试**和**组装软件**的**系统化技术**。将模块组装起来，检查其接口是否存在问

题，以及组装后的整体功能和性能表现

由模块组装成程序时有两种方法。

一种方法是先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，这种方法称为**非渐增式测试**；

另一种方法是把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。这种每次增加一个模块的方法称为**渐增式测试**，这种方法实际上**同时完成单元测试和集成测试。**

对象：组装后的模块集合

* 非渐增式测试方式：测每个模块，测结合后程序
* 渐增式测试方式：
* 自顶向下：主控制模块开始，逐步向下集成，使用存根程序模拟待测下层模块，需要回归
* 自底向上：从原子模块开始组装和测试，使用驱动程序调用模块

优缺点：

自顶向下测试方法的主要优点是**不需要测试驱动程序**，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。

主要缺点是**需要存根程序**，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力。

自底向上测试方法的优缺点与上述自顶向下测试方法的优缺点刚好相反。

7.5 **确认测试（有效性测试）**

确认测试又称有效性测试。

任务是验证**软件的功能和性能**及**其它特性**是否与用户的要求一致。

**主要采用黑盒测试。**

对象：完整的软件系统，基于用户需求规格说明书测试

验证软件是否满足用户需求，性能测试，安全性测试

方法：

* 黑盒测试
* 软件配置复查
* Alpha/Beta测试

7.6 **白盒测试定义/类型/步骤**

此方法把测试对象看做一个透明的盒子，它允许测试人员**利用程序内部的逻辑结构及有关信息**，设计或选择测试用例，**对程序所有逻辑路径进行测试**。

通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。因此白盒测试又称为**结构测试**或**逻辑驱动测试**。

类型：

1. 逻辑覆盖：至少执行xxx一次
2. 语句覆盖：每个语句
3. 判定覆盖：每个判定的真分支和假分支
4. 条件覆盖：每个条件的真结果和假结果
5. 判定-条件覆盖：每个判定的真分支和假分支 + 每个条件的真结果和假结果
6. 条件组合覆盖：每个判定的条件的所有组合
7. 路径覆盖
8. 控制结构测试
9. 基本路径测试：基于控制流图、计算环形复杂度，设计覆盖所有独立路径的测试用例

使用基本路径测试设计测试用例时，首先计算程序的**环形复杂度**，并用该复杂度为指南定义执行路径的基本集合，从该基本集合导出的测试用例可以保证程序中的每条语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值。

使用第6.5.1小节讲述的3种方法之一计算环形复杂度。

1. 条件测试
2. 循环测试

步骤：

1. 画控制流图
2. 计算环形复杂度：边-节点+2
3. 确定线性独立路径
4. 为每条独立路径设计测试用例

7.7 **黑盒测试方法**

这种方法是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据**程序的需求规格说明书**，检查程序的功能是否符合它的功能说明。黑盒测试又叫做功能测试或数据驱动测试。

目的是发现**需求侧错误**（功能遗漏，IO错误，性能缺陷）

方法：

* **等价划分：等价划分把程序的输入域划分成若干个数据类，据此导出测试用例。**

等价划分法力图设计出能发现若干类程序错误的测试用例，从而减少必须设计的测试用例的数目。划分为**有效等价类**和**无效等价类，每个类选一个代表**

* **边界值分析：**使用边界值分析方法设计测试方案首先应该**确定边界情况**，**通常输入等价类和输出等价类的边界。**选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。

通常设计测试方案时总是联合使用等价划分和边界值分析两种技术。

* **错误推测：**错误推测法在很大程度上**靠直觉和经验进行。**它的基本想法是列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试方案。

直觉+经验，推测可能错误的场景

7.8 **软件调试定义/步骤**

**调试**（也称为**纠错**）作为成功测试的后果出现，即调试是在**测试发现错误之后排除错误的过程。**

步骤：

1. 分析错误
2. 假设原因
3. 验证原因（断点、新增测试用例）
4. 改正错误

方法：

* 蛮干法
* 回溯法（分析调用栈）
* 原因排错法
* 对分查找法
* 归纳法：总结具体错误推导原因
* 演绎法：从可能原因出发排除

8. **维护**

8.1 **软件维护定义/四大类型**

**定义**：

所谓软件维护就是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。

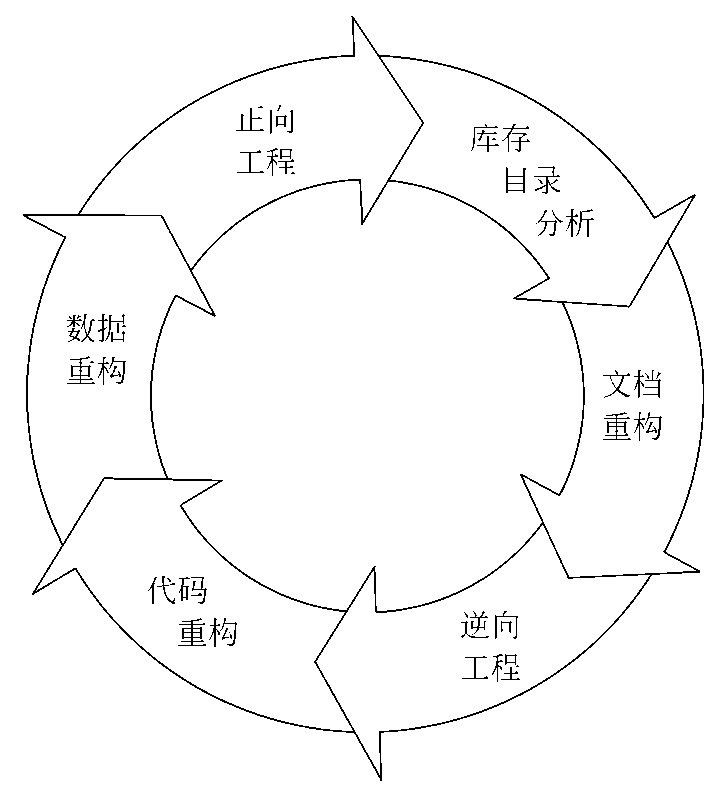
* **改正性维护：**把诊断和改正错误的过程称为改正性维护。
* **适应性维护**：为了和变化了的环境适当地配合而进行的修改软件的活动，是既必要又经常的维护活动。
* **完善性维护：**在使用软件的过程中用户往往提出**增加新功能或修改已有功能**的建议，还可能提出一般性的改进意见。为了满足这类要求，需要进行**完善性维护。**这项维护活动通常占软件维护**工作的大部分。**
* **预防性维护**：当为了**改进未来的可维护性或可靠性**，或为了给未来的改进奠定更好的基础而修改软件时，出现了第四项维护活动。这项维护活动通常称为预防性维护，目前这项维护活动相对比较少。

8.2 **软件再工程定义/步骤**

**定义：**软件再工程实质上是预防性维护，即改进未来的**可维护性或可靠性**，为了给未来的改进奠定更好的基础，而进行软件修改。

典型的软件再工程过程模型如下图所示。在某些情况下这些活动以线性顺序发生，但也并非总是这样。例如，为了理解某个程序的内部工作原理，可能在文档重构开始之前必须先进行逆向工程。

图要会画



* 库存目录分析：仔细分析库存目录，按照业务重要程度、寿命、当前可维护性、预期的修改次数等标准，把库中的应用系统排序，从中选出再工程的候选者，然后明智地分配再工程所需要的资源。
* 文档重构
* 逆向工程：软件的逆向工程是分析程序以便在比源代码更高的抽象层次上创建出程序的**某种表示的过程**，也就是说，逆向工程是一个**恢复设计结果的过程**，逆向工程工具从现存的程序代码中抽取有关数据、体系结构和处理过程的设计信息。

抽取数据、结构、过程，恢复设计结果

* 代码重构
* 数据重构：与代码重构不同，数据重构发生在**相当低的抽象层次上，**它是一种全范围的再工程活动——对数据的修改必然会导致体系结构或代码层的改变。 当数据结构较差时（例如在关系型方法可大大简化处理的情况下却使用平坦文件实现），应该对数据进行再工程。

数据再工程（改数据组织结构）

* 正向工程：正向工程也称为**革新或改造**，这项活动不仅从现有程序中恢复设计信息，而且使用该信息去改变或重构现有系统，以提高其整体质量。

恢复设计信息，改进整体质量

9. **面向对象方法学引论**

9.1 **面向对象核心机制、封装、继承、多态**

面向对象=对象+类+继承+消息

面向对象核心机制通过封装、继承、多态，模拟现实世界的抽象思维，实现软件系统的模块化、可用性和灵活性。

* 封装：将数据和实现操作的代码封装在对象内部，通过接口对外提供服务，隐藏实现细节，具有较强的独立性
* 继承：子类自动共享父类的属性和方法，形成类的层次结构。实现代码重用
* 多态：在类等级的不同层次中可以共享（公用）一个行为（方法）的名字，然而不同层次中的每个类却各自按自己的需要来实现这个行为。

子类实例可替代父类对象使用，相同消息可对应不同类的实现。提高代码灵活性和可重用性，减少信息冗余。

9.2 **面向对象和面向过程(结构化方法)的比较**

（1）思维特点

面向对象方法：使用现实世界的概念抽象思考问题从而自然地解决问题，重点针对需要

处理的问题进行分析。

结构化方法：以算法为核心，把数据和过程作为独立的部分，数据为客体，程序代码则

处理数据，这种思维方法与计算机处理问题方法一致。

（2）方法特点

面向对象方法：重点强调反映现实需求的对象业务模型的建立，设计和编码部分工作为

次要。

结构化方法：重点在软件处理过程的设计和实现上。

（3）适用范围比较

面向对象方法：适用于比较大型的应用。

结构化方法：适用于涉及底层处理的应用或直接对硬件系统操作的系统；对一些小的需

要复杂处理流程、强调算法设计的软件系统也适用。



9.2 **面向对象的统一软件开发过程的各阶段 里程碑**

（1）初始阶段（生命周期目标里程碑）：为系统建立业务模型并确定项目的边界；

（2）细化阶段（生命周期结构里程碑）：分析问题的领域，建立完善的架构，淘汰项目中

最高风险的元素；

（3）构造阶段（初始运行能力）：要开发所有剩余的构建和应用程序功能，把这些构建集

成为产品，并进行详细测试；

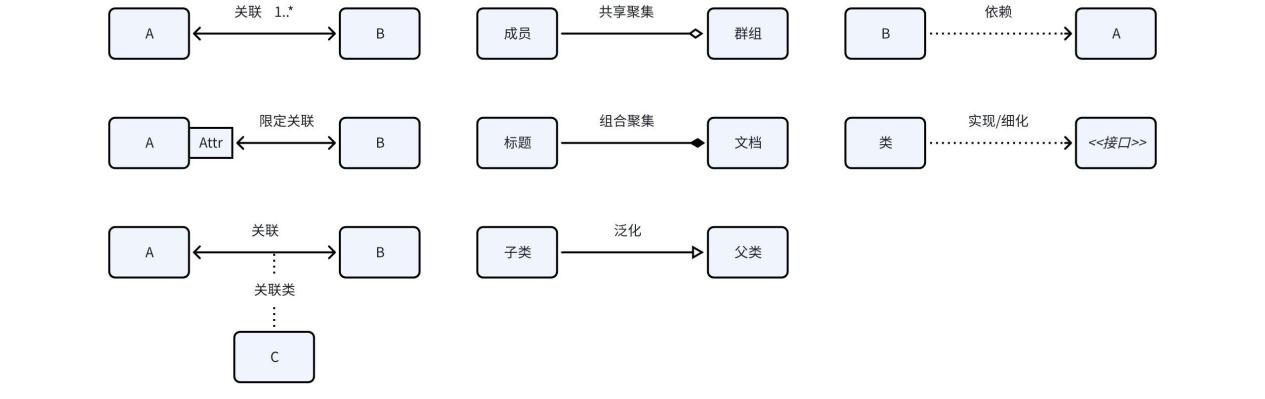
（4）交付阶段（产品发布里程碑）：重点是确保软件对最终用户是可用的

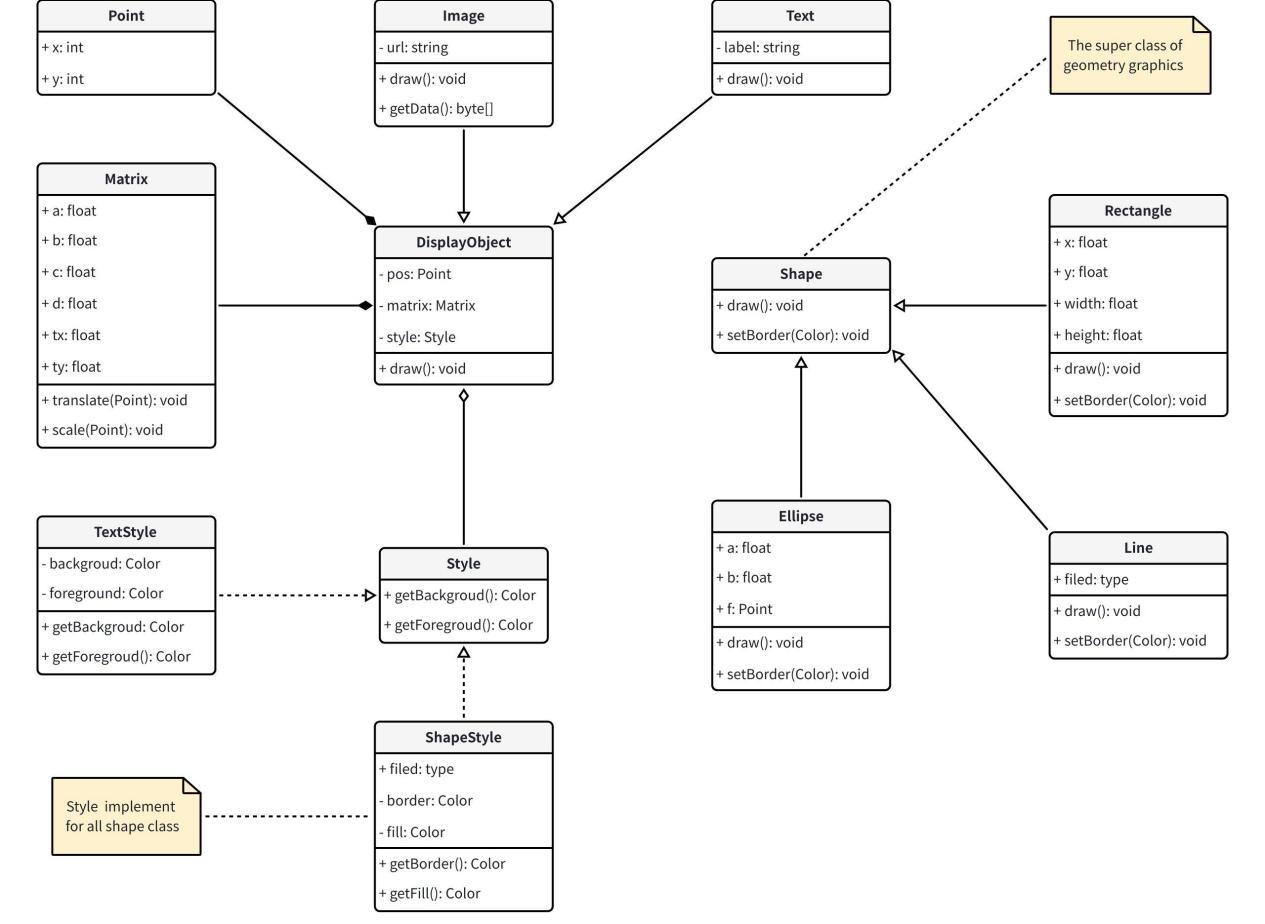
9.2 **类图**

**（课本 p217、PPT 第九章 65 页）**

类图描述类及类与类之间的静态关系。类图是一种静态模型，它是创建其他UML图的基础。一个系统可以由多张类图来描述，一个类也可以出现在几张类图中。

* 类：矩形，类名、属性、方法
* +/-/# attr: type=value {abstract/readonly}
* +/-/# method(args): returnType {abstract/readonly}
* 关系： 类与类之间通常有关联、泛化（继承）、依赖和细化4种关系。
* 关联 Association： 关联表示两个类的对象之间存在某种语义上的联系。
* 聚集 Aggregation：聚集也称为聚合，是关联的特例
* 泛化（继承） Generalization： 在UML中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素。
* 依赖 Dependency：依赖关系描述两个模型元素（类、用例等）之间的语义连接关系： 其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。
* 细化 Realization： 当对同一个事物在不同抽象层次上描述时，这些描述之间具有细化关系。





9.3 **用例图**

**（课本 p224 页、PPT 第九章 88 页）**

一幅用例图包含的模型元素有系统、行为者、用例及用例之间的关系。

描述系统功能，从用户视角定义系统提供的服务和与外部实体的交互

* 系统：系统被看作是一个提供用例的黑盒子，内部如何工作、用例如何实现对于建立用例模型来说都是不重要的。

方框，划定功能范围

* 用例：一个用例是可以被行为者感受到的、系统的一个完整的功能。

椭圆，系统功能

* 行为者： 行为者是指与系统交互的人或其他系统，它代表外部实体。使用用例并且与系统交互的任何人或物都是行为者。

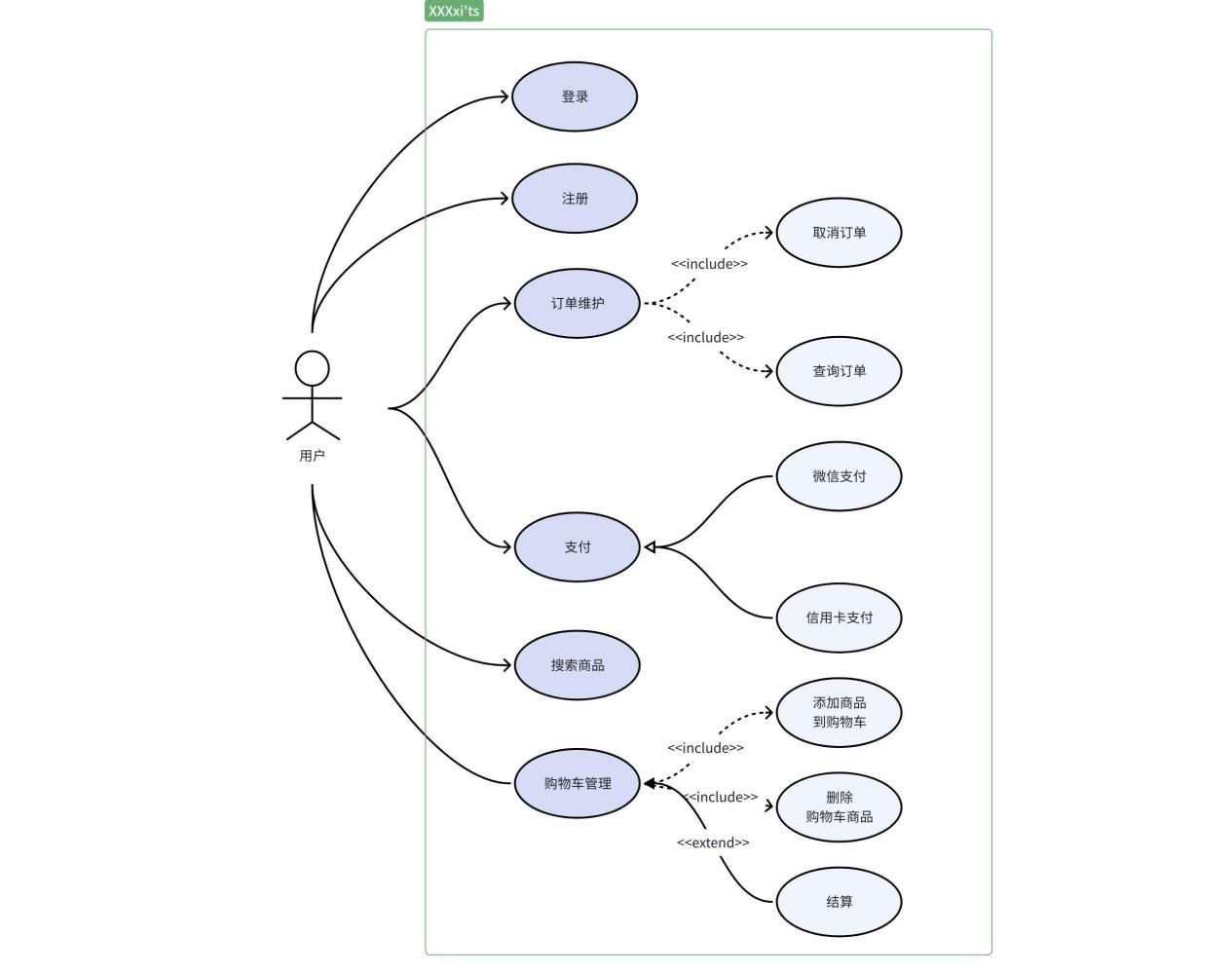
行为者代表一种角色，而不是某个具体的人或物。一个具体的人可以充当多种不同角色。

火柴人，与系统交互的外部实体

* 用例之间的关系：

(1) 扩展关系：向一个用例中添加一些动作后构成了另一个用例

(2) 使用关系：当一个用例使用另一个用例时，这两个用例之间就构成了使用关系



10. **面向对象分析**

10.1 **面向对象需要建立的三类模型**

对象模型是最基本、最重要、最核心的

* 对象模型（静态结构）类图
* 动态模型（交互次序）状态图
* 功能模型（数据变换和功能）用例图 → 数据流图

10.2 **对象模型的5个层次**

复杂问题（大型系统）的对象模型通常由下述5个层次组成：

* 主题层：对大型复杂模型进行概念范畴划分，帮助理解，控制可见性
* 类与对象层：识别问题域中的类与对象
* 结构层：描述类与对象之间的关系（继承、关联、聚集）
* 属性层：类属性，描述对象特征
* 服务层：类方法，描述对象行为

10.3 **面向对象分析的基本过程（七个步骤）**

1. 寻找类与对象：筛选实体
2. 识别结构：确定类间关系
3. 识别主题：划分大型系统主题
4. 定义属性:
5. 建立动态模型：描述系统交互行为
6. 建立功能模型：描述数据处理功能
7. 定义服务：根据5、6确定类的操作

10.4 **软件重用定义**

指在两次或多次不同的软件开发过程中重复使用 **相同或相似软件元素**的过程。

广义地说，软件重用可分为以下 3 个层次：

1. 知识重用
2. 方法和标准重用
3. 软件成分的重用：代码、设计成果、分析结果

在软件成分的重用中，具体划分成了代码重用、设计结果重用和分析结果重用。