## 第1章 软件工程学概述

### 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\*\*

- 具体的定义：

1. 把\*\*系统的、规范的、可度量的\*\*方法应用于软件开发、运行和维护过程，也就是把工程应用于软件

2. 将第一点提到的方法作为对象的研究活动

### 2、软件特点

- 软件是被工程化的逻辑系统

- 软件一般没有磨损

- 软件具有不同于一般实物系统的复杂性

### 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：概括说，开发周期长、成本高、质量差、适应性差和难维护等四大难题

### 4、软件工程方法学三要素

- \*\*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

### 5、软件生命周期/各阶段定义

> 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个时期组成，每个时期又进一步划分成若干个阶段

- \*\*软件定义\*\*时期的任务是：

- 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

- \*\*软件定义\*\*时期通常进一步划分成3个阶段：\*\*问题定义、可行性研究和需求分析\*\*

- \*\*软件开发\*\*时期具体设计和实现在前一个时期定义的软件，它通常由下述4个阶段组成：

- \*\*总体设计，详细设计，编码和单元测试，综合测试\*\*

- 其中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\*\*时期的主要任务是使软件持久地满足用户的需要

### 6、Scrum 定义

- Scrum 通常被认为是一个敏捷项目管理框架，它描述了一组帮助协同工作的事件、工件和角色，旨在帮助团队更高效的组织和管理其工作

![image-20240512212009975](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240512212009975.png)

### 7、Scrum 三大特点

- “可能性的”艺术

- 团队自组织，自管理

- 面对面沟通

### 8、Scrum 团队模型的三种角色

#### （1）Scrum Master

- 保证Scrum团队可以遵守Scrum的价值，实践和规范

- 帮助Scrum团队和组织采用Scrum模式进行项目流程组织

- 指导并带领团队变得更加高效，实现更高质量

- 保护团队不要受到外界因素的干扰

- 保证各个不同角色之间的良好写作，消除障碍

- 帮助PO更好地利用团队的能力

- \*\*不要管理\*\*团队

#### （2）Product Owner

- PO是一个人并只能由一个人来担任

- 负责管理产品待办事项表( Product Backlog )并保证其对于客户和团队保持透明度

- 对产品代办事项表进行优先级排序

- 与团队一起来进行工作量估算

- 对于项目的成功负责并保证投资回报率(ROI )

#### （3）团队

- 最佳团队大小：5-9人

- 多功能团队：程序员，测试人员，设计师，数据库管理员和架构师

- 保证团队成员全职参与开发

- 自我管理，没有头衔之分，不组建子团队

- 成员更替只能在迭代之间进行，最佳方式是在发布之间进行

## 第2章 可行性研究

### 1、可行性分析定义

- 对需求内容进行初步的分析，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

- 其本质实际上就是一个粗略的系统分析和设计过程，通常由客户完成

### 2、可行性分析三个方面

- \*\*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？

### 3、数据流图定义

- \*\*数据流图\*\*是一种图形化技术，它描述\*\*信息流和数据\*\*从输入移动到输出的过程中所经受的\*\*变换\*\*

- 为了表达数据处理过程的数据加工情况，需要采用\*\*层次结构\*\*的数据流图。按照系统的层次结构进行\*\*逐步分解\*\*，并以分层的数据流图反映这种结构关系，能清楚地表达和容易理解整个系统

![image-20240512213226446](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240512213226446.png)

### 4、数据字典定义

- \*\*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，由对下列4类元素的定义组成：\*\*数据流、数据流分量、数据存储、处理\*\*

![image-20240512213810078](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240512213810078.png)

### 5、数据字典【ppt 北京电话号码】

![image-20240512213558978](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240512213558978.png)

![image-20240512213614053](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240512213614053.png)

## 第3章 需求分析

### 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，它的基本任务是准确地回答“\*\*系统必须做什么\*\*”这个问题

### 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、需求分析的八个方面

- \*\*功能需求\*\*：

- 这方面的需求指定系统必须提供的\*\*服务\*\*

- 通过需求分析应该划分出系统必须完成的所有功能

- \*\*性能需求\*\*：

- 指定系统必须满足的\*\*定时约束\*\*或\*\*容量约束\*\*

- 通常包括速度（响应时间）、信息量速率、主存容量、磁盘容量、安全性等方面的需求

- \*\*可靠性和可用性需求\*\*：

- 可靠性需求定量地指定系统的可靠性

- 可用性与可靠性密切相关，它量化了用户可以使用系统的程度

- \*\*出错处理需求\*\*：

- 这类需求说明\*\*系统对环境错误应该怎样响应\*\*

- 例如，如果它接收到从另一个系统发来的违反协议格式的消息，应该做什么？

- 注意，上述这类错误并不是由该应用系统本身造成的

- \*\*接口需求\*\*：

- 接口需求描述应用\*\*系统与它的环境通信\*\*的格式

- 常见的接口需求有：用户接口需求；硬件接口需求；软件接口需求；通信接口需求

- \*\*约束\*\*：

- 设计约束或实现约束描述在设计或实现应用系统时应遵守的\*\*限制条件\*\*

- 常见的约束有：精度；工具和语言约束；设计约束；应该使用的标准；应该使用的硬件平台

- \*\*逆向需求\*\*：

- 说明\*\*软件系统不应该做什么\*\*

- 理论上有无限多个逆向需求，人们应该仅选取能澄清真实需求且可消除可能发生的误解的那些逆向需求

- \*\*将来可能提出的需求\*\*：

- 应该明确地列出那些虽然不属于当前系统开发范畴，但是据分析将来很可能会提出来的要求

- 这样做的目的是，在设计过程中对系统将来可能的\*\*扩充和修改\*\*预做准备，以便一旦确实需要时能比较容易地进行这种扩充和修改

### 4、状态转换图定义

- \*\*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

### 5、状态转换图【ppt 复印机】

![image-20240513202928707](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513202928707.png)

![image-20240513202950742](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513202950742.png)

## 第5章 总体设计

### 1、总体设计定义

- 总体设计又称为\*\*概要设计\*\*或\*\*初步设计\*\*

- 总体设计的基本目的就是回答“\*\*概括地说，系统应该如何实现\*\*”这个问题

- 总体设计阶段的另一项重要任务是\*\*设计软件的结构\*\*，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系

> 软件设计的任务：

>

> - 【技术观点】根据用信息域表示的软件需求，以及功能和性能需求，进行：

> - \*\*数据设计\*\*：侧重于数据结构的定义

> - \*\*系统结构设计\*\*：定义软件系统各主要成份之间的关系

> - \*\*过程设计\*\*：是把结构成份转换成软件的过程性描述。在编码步骤，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件

> - 【管理观点】从工程管理的角度来看，软件设计分两步完成：

> - \*\*概要设计\*\*：将软件需求转化为数据结构和软件的系统结构。（概要设计对应了技术观点中的系统结构设计）

> - \*\*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。（详细设计对应了技术观点中的数据设计和过程设计）

### 2、模块的三个基本要素

“模块”，又称“组件”。它一般具有如下三个基本属性：

- \*\*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

### 3、模块化

- \*\*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

- 软件系统的模块化是指整个软件被划分成若干单独命名和可编址的部分，称之为模块。这些模块可以被组装起来以满足整个问题的需求

- \*\*模块化\*\*是为了使一个复杂的大型程序能被人的智力所管理，\*\*是软件应该具备的唯一属性\*\*

### 4、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

- 如果在测试期间和以后的软件维护期间需要修改软件，使用信息隐藏原理作为模块化系统设计的标准就会带来极大好处

### 5、模块独立性，耦合，内聚

- 模块的\*\*独立性\*\*很重要，因为：

- 有效的模块化（即具有独立的模块）的软件比较容易开发出来

- 独立的模块比较容易测试和维护

- 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\*\*

- \*\*模块独立性\*\*：是指软件系统中每个模块只涉及软件要求的具体的子功能，而和软件系统中其它的模块的接口是简单的

- \*\*耦合\*\*：是模块之间的互相连接的紧密程度的度量

- ![image-20240513204324981](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513204324981.png)

- \*\*内聚\*\*：标志一个模块内各个元素彼此结合的紧密程度

- ![image-20240513204431299](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513204431299.png)

### 6、作用域、控制域（扇入、扇出）

- \*\*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

![image-20240513204643269](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513204643269.png)

> - 模块的\*\*作用域应该在控制域之内\*\*

> - 模块的控制范围包括它本身及其所有的从属模块

> - 模块的作用范围是指模块内一个判定的作用范围，凡是受这个判定影响的所有模块都属于这个判定的作用范围

> - 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

> - 尽可能\*\*减少高扇出\*\*结构，随着\*\*深度增大扇入\*\*

> - 如果一个模块的扇出数过大，就意味着该模块过分复杂，需要协调和控制过多的下属模块

> - 应当适当增加中间层次的控制模块

### 7、总体设计原则

#### （1）模块化

- \*\*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

#### （2）抽象

- 抽象就是抽出事物的本质特性而暂时不考虑它们的细节

- 软件工程过程的每一步都是对软件解法的抽象层次的一次精化

#### （3）逐步求精

- 逐步求精定义为为了能集中精力解决主要问题而尽量推迟对问题细节的考虑

- 将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构

#### （4）信息隐藏和局部化

- \*\*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

- 如果在测试期间和以后的软件维护期间需要修改软件，使用信息隐藏原理作为模块化系统设计的标准就会带来极大好处

#### （5）模块独立

- \*\*模块独立性\*\*：是指软件系统中每个模块只涉及软件要求的具体的子功能，而和软件系统中其它的模块的接口是简单的

### 8、变换流，事务流定义

> - 面向数据流的设计方法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

> - 信息流有下述两种类型

- \*\*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

![image-20240513205411403](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513205411403.png)

- \*\*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。图中的处理T称为事务中心，它完成下述任务：

- 接收输入数据（输入数据又称为事务）

- 分析每个事务以确定它的类型

- 根据事务类型选取一条活动通路

![image-20240513205317699](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513205317699.png)

## 第6章 详细设计

### 1、详细设计定义

> - \*\*根本目标\*\*：确定应该怎样\*\*具体\*\*地实现所要求的系统

> - 详细设计阶段的任务不是具体地编写程序，而是要\*\*设计出程序的“蓝图”\*\*

> - 详细设计的结果基本上决定了最终的程序代码的质量

- 确定\*\*软件各个组成部分内的算法\*\*以及\*\*各部分的内部数据组织\*\*

- 选定某种过程的表达形式来描述各种算法

- 进行详细设计的评审

### 2、结构化程序的三种基本控制结构

> - 结构程序设计经典定义：如果一个程序的代码块仅仅通过顺序、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，则称这个程序是结构化的

> - 结构程序设计更全面的定义：结构程序设计是尽可能少用GO TO语句的程序设计方法。最好仅在检测出错误时才使用GO TO语句，而且应该总是使用前向GO TO语句

>

> - 三种基本控制结构为：\*\*顺序、选择和循环\*\*

- 如果只允许使用\*\*顺序\*\*、I\*\*F-THEN-ELSE\*\*型分支和\*\*DO-WHILE\*\*型循环这3种基本控制结构，则称为\*\*经典的结构程序设计\*\*

- 如果除了上述3种基本控制结构之外，还允许使用\*\*DO-CASE\*\*型多分支结构和\*\*DO-UNTIL\*\*型循环结构，则称为\*\*扩展的结构程序设计\*\*

- 如果再允许使用LEAVE（或BREAK )结构，则称为\*\*修正的结构程序设计\*\*

### 3、判定树/判定表【ppt 交易所手续费】

![image-20240513211400080](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513211400080.png)

![image-20240513211618821](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513211618821.png)

![image-20240513211645818](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513211645818.png)

![image-20240513211702092](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513211702092.png)

![image-20240513211720122](https://qingwu-oss.oss-cn-heyuan.aliyuncs.com/lian/img/image-20240513211720122.png)

## 第7章 实现

### 软件测试的原则

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；

### 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。\*\*目的在于发现各模块内部可能存在的各种差错\*\*。

单元测试和编码属于软件过程的同一个阶段。

可以应用\*\*人工测试\*\*和\*\*计算机测试\*\*这样两种不同类型的测试方法，完成单元测试工作。

单元测试主要使用\*\*白盒测试技术\*\*，而且对多个模块的测试可以\*\*并行\*\*地进行。

### 集成测试

集成测试是测试和组装软件的系统化技术。

模块组装成程序的两种方法：

- 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\*\*非渐增式测试方法\*\*

- 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\*\*。

#### 渐增式测试和非渐增式测试

- 非渐增式测试把所有模块放在一起，作为一个整体来测试。测试时会遇到许多的错误，改正错误非常困难，因为在\*\*庞大的程序中想要诊断定位一个错误非常困难，而且改正一个错误之后，马上又会遇到新的错误，这个过程会继续下去，没有尽头\*\*。

- 渐增式测试与“一步到位”的非渐增式测试相反，它\*\*把程序划分成小段来构造和测试，在这个过程中比较容易定位和改正错误\*\*；目前普遍采用渐增式测试方法。

- 当使用渐增方式把模块结合到程序中去时，有\*\*自顶向下\*\*和\*\*自底向上\*\*两种集成策略。

#### 自顶向上集成和自底向上集成

- \*\*自顶向下集成方法\*\*是从\*\*主控制模块开始，沿着程序的控制层次向下移动\*\*，逐渐把各个模块结合起来。在把附属于（及最终附属于）主控制模块的那些模块组装到程序结构中去时，或者\*\*使用深度优先的策略\*\*，或者\*\*使用宽度优先的策略\*\*。

- \*\*自顶向下的结合策略\*\*能够在测试的早期对主要的控制或关键的抉择进行检验。在一个分解得好的软件结构中，关键的抉择位于层次系统的较上层，因此首先碰到。

- \*\*自底向上集成\*\* 从“原子”模块（即在软件结构最低层的模块）开始组装和测试。因为是从底部向上结合模块，总能得到所需的下层模块处理功能，所以不需要存根程序。

- 用下述步骤可以实现自底向上的结合策略：

- 把低层模块组合成实现某个特定的软件子功能的族；

- 写一个\*\*驱动程序（用于测试的控制程序）\*\*，协调测试数据的输入和输出；

- 对由模块组成的\*\*子功能族\*\*进行测试；

- 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族。

### 驱动程序

用于测试的控制程序。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517202758.png)

### 存根程序

存根程序（Stub）是一种临时的程序或代码，用于\*\*模拟正在开发或尚未开发的软件组件的行为\*\*。它们通常在软件开发的早期阶段使用，用于测试和验证软件系统的整体架构和设计。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517202843.png)

### 回归测试

- 在集成测试的范畴中，\*\*回归测试\*\*是指重新执行已经做过的测试的某个子集，以保证上述这些变化没有带来非预期的副作用。

- \*\*回归测试\*\*就是用于保证由于调试或其他原因引起的变化，不会导致非预期的软件行为或额外错误的测试活动。

- \*\*回归测试\*\*可以通过\*\*人工地\*\*进行，也可以使用\*\*自动化\*\*的捕获回放工具自动进行。利用\*\*捕获回放工具\*\*，软件工程师能够捕获测试用例和实际运行结果，然后可以回放（即重新执行测试用例），并且比较软件变化前后所得到的运行结果。

回归测试包括下述\*\*3类不同测试用例\*\*：

- 检测\*\*软件全部功能\*\*的代表性测试用例。

- 专门针对可能\*\*受修改影响的软件功能\*\*的附加测试。

- 针对\*\*被修改过的软件成分\*\*的测试。

### 白盒测试

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。

测试方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、域测试、符号测试、Z路径覆盖、程序变异。

其中运用最为\*\*广泛\*\*的是\*\*基本路径测试法\*\*。

#### 逻辑覆盖

1. 语句覆盖：所有可执行语句至少执行一次

2. 判定覆盖：不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次。

3. 条件覆盖：不仅每个语句至少执行一次，而且使判定表达式中的\*\*每个条件\*\*都取到各种可能的结果。

4. 判断/条件覆盖：\*\*使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。\*\*

5. 路径覆盖：选取足够多测试数据，使程序的每条可能路径都至少执行一次（如果程序图中有环，则要求每个环至少经过一次）。

6. 点覆盖：如果连通图G的子图G′是连通的，而且包含G的所有结点，则称G′是G的点覆盖。满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过流图的每个结点一次，由于流图的每个结点与一条或多条语句相对应，显然，点覆盖标准和语句覆盖标准是相同的。

#### 白盒测试技术

- \*\*控制结构测试\*\*：基本路径测试、条件测试、循环测试。

### 黑盒测试

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\*\*。

\*\*黑盒测试方法\*\*：等价类划分、边界值分析、错误推测等。

## 第8章 维护

### 软件维护定义/类型

\*\*软件维护定义\*\*：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。可以通过描述软件交付使用后可能进行的\*\*4项活动\*\*，具体地定义软件维护。

软件维护绝不仅限于纠正使用中发现的错误，事实上在全部维护活动中一半以上是完善性维护。应该注意，上述4类维护活动都必须应用于整个软件配置，维护软件文档和维护软件的可执行代码是同样重要的。

\*\*4项活动\*\*：

- \*\*改正性维护\*\*：在任何大型程序的使用期间，用户必然会发现程序错误，并且把他们遇到的问题报告给维护人员。把\*\*诊断和改正错误的过程\*\*称为改正性维护。

- \*\*适应性维护\*\*：适应性维护，也就是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\*\*：软件系统顺利地运行时，\*\*常常出现第三项维护活动\*\*：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。

- \*\*预防性维护\*\*：\*\*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\*\*，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。

### 软件再工程定义/步骤

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。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517203533.png)

软件再工程步骤：

- \*\*库存目录分析\*\*： 每个软件组织都应该保存其拥有的所有应用系统的库存目录。该目录包含关于每个应用系统的基本信息。应该仔细分析库存目录，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\*\*：老程序固有的特点是缺乏文档。具体情况不同，处理这个问题的方法也不同。

- 建立文档非常耗费时间，不可能为数百个程序都重新建立文档。如果一个程序是相对稳定的，正在走向其有用生命的终点，而且可能不会再经历什么变化，那么，让它保持现状是一个明智的选择。

- 为了便于今后的维护，必须更新文档，但是由于资源有限，应采用“使用时建文档”的方法。

- 如果某应用系统是完成业务工作的关键，而且必须重构全部文档，则仍然应该设法把文档工作减少到必需的最小量。

- \*\*逆向工程\*\*：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。\*\*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。

- \*\*代码重构\*\*：某些老程序具有比较完整、合理的体系结构，但是，个体模块的编码方式却是难于理解、测试和维护的。在这种情况下，可以重构可疑模块的代码。

- 用重构工具分析源代码，标注出和结构化程序设计概念相违背的部分。

- 重构有问题的代码（此项工作可自动进行）。

- 复审和测试生成的重构代码（以保证没有引入异常）并更新代码文档。

- \*\*数据重构\*\*：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。\*\*在大多数情况下，数据重构始于逆向工程活动\*\*，分解当前使用的数据体系结构，必要时定义数据模型，标识数据对象和属性，并从软件质量的角度复审现存的数据结构。

- \*\*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。

## 第9章 面向对象方法学引论

### 面向对象的统一软件开发过程各阶段/里程碑/

\*\*ODD开发过程各阶段\*\*：维护期、集成和测试阶段、编码阶段、面向对象设计阶段、面向对象分析阶段、需求阶段。

\*\*RUP的各个阶段和里程碑\*\*：初始阶段(生命周期目标里程碑)、细化阶段(生命周期结构里程碑)、构造阶段(初始运行能力)、交付阶段(产品发布里程碑)。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517204745.png)

### 封装

封装是指把数据和实现操作的代码集中起来放在对象内部。

对象具有封装性的条件:

- \*\*有一个清晰的边界\*\*。所有私有数据和实现操作的代码都被封装在这个边界内，从外面看不见更不能直接访问。

- \*\*有确定的接口（即协议）\*\*。这些接口就是对象可以接受的消息，只能通过向对象发送消息来使用它。

- \*\*受保护的内部实现\*\*。实现对象功能的细节（私有数据和代码）不能在定义该对象的类的范围外访问。

### 关联/两种聚集关联定义

关联表示两个类的对象之间存在某种语义上的联系。

\*\*普通关联\*\*

只要在类与类之间存在连接关系就可以用普通关联表示。普通关联的图示符号是连接两个类之间的直线，如下图所示。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205229.png)

在表示关联的直线两端可以写上\*\*重数（multiplicity）\*\*，它表示该类有多少个对象与对方的一个对象连接。重数的表示方法通常有：

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205327.png)

如果图中未明确标出关联的重数，则默认重数是1。

上图表示，一个作家可以使用1到多台计算机，一台计算机可被0至多个作家使用。

\*\*限定关联\*\*

限定关联通常用在一对多或多对多的关联关系中，可以把模型中的重数从\*\*一对多变成一对一，或从多对多简化成多对一\*\*。在类图中把限定词放在\*\*关联关系末端的一个小方框内\*\*。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205416.png)

图利用限定词“文件名”表示了目录与文件之间的关系，可见，利用限定词把一对多关系简化成了一对一关系。

上图一个受限的关联限定提高了语义精确性，增强了查询能力。限定的语法表明，文件名在其目录内是唯一的。

\*\*关联类\*\*

说明关联的性质，引入一个关联类来记录附加信息。\*\*关联中的每个连接与关联类的一个对象相联系\*\*。关联类通过一条虚线与关联连接。关联类与一般的类一样，也有属性、操作和关联。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205614.png)

### 聚集

聚集也称为聚合，是关联的特例。聚集表示类与类之间的关系是整体与部分的关系。使用的“包含”、“组成”、“分为……部分”等字句，意味着存在聚集关系。有\*\*共享聚集\*\*和\*\*组合聚集\*\*两种特殊的聚集关系。

\*\*共享聚集\*\*

在\*\*聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成\*\*，则该聚集称为共享聚集。

一个课题组包含许多成员，每个成员又可以是另一个课题组的成员，则课题组和成员之间是共享聚集关系。一般聚集和共享聚集的关联关系用空心菱形表示。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205727.png)

\*\*组合聚集\*\*

如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失（或失去存在价值了），则该聚集称为组合聚集（简称为组成）。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205805.png)

### 用例图

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\*\*。右图是自动售货机系统的用例图。图中的\*\*方框代表系统\*\*，\*\*椭圆代表用例\*\*（售货、供货和取货款是自动售货机系统的典型用例），\*\*线条人代表行为者，它们之间的连线表示关系\*\*。

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517205919.png)

### 类图

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517212519.png)

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517212536.png)

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517212547.png)

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517212603.png)

![image.png](https://raw.githubusercontent.com/Aoshine999/pigo/main/20240517212634.png)

## 第10章 面向对象分析

### 面向对象需要建立的三类模型

- \*\*对象模型(静态结构)\*\*：表示静态的结构化的系统的“数据”性质。对客观世界实体的对象以及对象之间关系的映射，描述了系统的静态结构。是三种模型中最重要的模型。

- \*\*动态模型(交互结构)\*\*：表示瞬时的行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。

- \*\*功能模型(数据变换)\*\*：表示变化的系统的“功能”性质，它表明了系统应该“做什么”，因此更直接地反映了用户对目标系统的需求。

### 对象模型的5个层次

- 主题层：通过\*\*划分主题把一个大型、复杂的对象模型分解成几个不同的概念范畴\*\*。人类的短期记忆能力一般限于一次记忆5～9个对象，这就是著名的7±2原则。增加了一个主题层，它可以\*\*从一个相当高的层次描述总体模型，并对读者的注意力加以指导\*\*。

- 类与对象层

- \*\*对象\*\*是由描述该对象属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体。

- 对象以\*\*数据为中心、对象是主动的、实现了数据封装、本质上有并行性、模块独立性好\*\*。

- \*\*类\*\*就是对具有相同数据和相同操作的一组\*\*相似对象\*\*的定义，也就是说，类是对具有\*\*相同属性和行为\*\*的一个或多个对象的描述，通常在这种描述中也包括对怎样创建该类的新对象的说明。

- 结构层：使用\*\*类图\*\*描述类及类与类之间的静态关系。

- 属性层：属性的可见性（即可访问性）通常有下述3种：\*\*公有的（ public）\*\*、\*\*私有的（private）和保护的（protected）\*\*，分别用\*\*加号(+)、减号(-)和井号(#)\*\* 表示。注意，没有默认的可见性。

- 服务层：服务就是操作，及一个类的方法调用。明确定义出每个类所提供的服务内容，也就是其提供的操作功能。

### 面向对象分析的基本步骤

1. 寻找类与对象

2. 识别结构

3. 识别主题

4. 定义属性

5. 建立动态模型

6. 建立功能模型

7. 定义服务

### 软件重用定义

软件重用是指在软件开发过程中，\*\*利用已经开发好的软件组件或模块\*\*，将其直接或经过一定修改后应用到新的软件系统中，以\*\*减少软件的开发成本和提高软件的质量\*\*。

软件重用的\*\*核心思想\*\*是将\*\*软件系统分解成可复用的组件或模块\*\*，并建立起一个可供查询和检索的组件库或模块库，以便于在新的软件开发中能够\*\*快速地找到和使用这些组件或模块\*\*。