

第1章 软件工程学概述

1.1 软件危机

1.1.1 软件危机的介绍

软件危机(软件萧条、软件困扰)：是指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。

软件危机包含下述两方面的问题：

如何开发软件，满足对软件日益增长的需求；

如何维护数量不断膨胀的已有软件。

软件危机的典型表现：

- (1) 对软件开发成本和进度的估计常常很不准确；
- (2) 用户对“已完成的”软件系统不满意的现象经常发生；
- (3) 软件产品的质量往往靠不住；
- (4) 软件常常是不可维护的；
- (5) 软件通常没有适当的文档资料；
- (6) 软件成本在计算机系统总成本中所占的比例逐年上升；
- (7) 软件开发生产率提高的速度，远远跟不上计算机应用迅速普及深入的趋势。

1.1.2 产生软件危机的原因

- (1) 与软件本身的特点有关
- (2) 与软件开发与维护的方法不正确有关

1.1.3 消除软件危机的途径

对计算机软件有正确的认识。

认识到软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。

应该推广使用在实践中总结出来的开发软件的成功技术和方法，并继续研究探索。

应该开发和使用更好的软件工具。

总之，为了解决软件危机，既要有技术措施(方法和工具)，又要有必要的组织管理措施。

1.2

1.2.1 软件工程的介绍

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、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它，这就是软件工程。(期中考)

软件工程的本质特性：

软件工程关注于大型程序的构造

软件工程的中心课题是控制复杂性

软件经常变化

开发软件的效率非常重要

和谐地合作是开发软件的关键

软件必须有效地支持它的用户

在软件工程领域中是由具有一种文化背景的人替具有另一种文化背景的人创造产品

1.2.2 软件工程的基本原理

用分阶段的生命周期计划严格管理

坚持进行阶段评审
实行严格的产品控制
采用现代程序设计技术
结果应能清楚地审查
开发小组的人员应该少而精
承认不断改进软件工程实践的必要性

1.2.3 软件工程方法学

软件工程包括技术和管理两方面的内容。

软件工程方法学 3 要素：方法、工具、过程

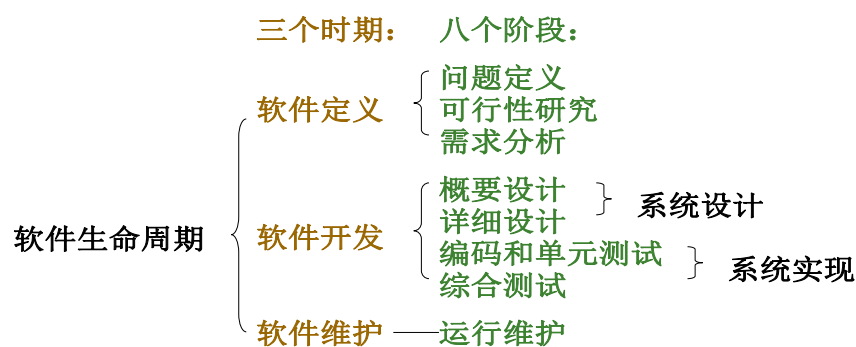
1. 传统方法学(生命周期方法学或结构化范型)——强调自顶向下

2. 面向对象方法学——强调主动地多次反复迭代

面向对象方法学 4 个要点：对象、类、继承、消息

1.3 软件生命周期（必考）

三个时期八个阶段：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)三个时期组成，每个时期又进一步划分成若干个阶段。



1.4 软件过程

1.4.1 瀑布模型

1.4.2 快速原型模型

1.4.3 增量模型

1.4.4 螺旋模型

1.4.5 喷泉模型

第 2 章 可行性研究

2.1 可行性研究的任务

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。

可行性研究的实质：

进行一次大大压缩简化了的系统分析和设计的过程，也就是在较高层次上以较抽象的方式进行的系统分析和设计的过程。

可行性研究的内容：

首先进一步分析和澄清问题定义，导出系统的逻辑模型；

然后从系统逻辑模型出发，探索若干种可供选择的主要解法(即系统实现方案)；
对每种解法都研究它的可行性，至少应该从三方面研究每种解法的可行性。

主要方面：

技术可行性，经济可行性，操作可行性，

其他方面：

运行可行性，法律可行性，

2.2 可行性研究过程

1. 复查系统规模和目标
2. 研究目前正在使用的系统
3. 导出新系统的高层逻辑模型
4. 进一步定义问题
5. 导出和评价供选择的解法
6. 推荐行动方针
7. 草拟开发计划
8. 书写文档提交审查

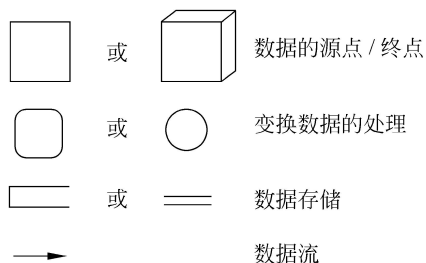
2.3 系统流程图

系统流程图：是概括地描绘物理系统的传统工具。表达的是数据在系统各部件之间流动的情况，而不是对数据进行加工处理的控制过程。

2.4 数据流图

2.4.1 符号

基本符号：



数据存储：数据存储是处于静止状态的数据；

数据流：数据流是处于运动中的数据。

附加符号：

星号 (*)：表示“与”关系

加号 (+)：表示“或”关系

异或 (\oplus)：表示互斥关系

2.5 数据字典

数据流图和数据字典共同构成系统的逻辑模型。

2.5.1 数据字典的内容

数据字典的组成：数据流 数据流分量(即数据元素) 数据存储 处理

2.5.2 定义数据的方法

方法：对数据自顶向下分解。

数据组成方式(三种基本类型)：顺序 选择 重复 附加类型：可选
符号：

=意思是等价于(或定义为)；

+意思是和(即，连接两个分量)；

[] 意思是或(即，从方括弧内列出的若干个分量中选择一个)，通常用“|”号隔开供选择的分量；

{ } 意思是重复(即，重复花括弧内的分量)；常常使用上限和下限进一步注释表示重复的花括弧。

() 意思是可选(即，圆括弧里的分量可有可无)。

2.5.3 数据字典的实现

计算机实现 人工实现

2.6 成本/效益分析

2.6.1 成本估计：1. 代码行技术 2. 任务分解技术 3. 自动估计成本技术

2.6.2 成本/效益分析的方法

成本/效益分析涉及的4个概念：

1. 货币的时间价值

2. 投资回收期

3. 纯收入

4. 投资回收期： $P = F1/(1 + j) + F2/(1 + j)^2 + \dots + Fn/(1 + j)^n$

第3章 需求分析

需求分析的任务：

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。

确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。

系统分析员应该写出软件需求规格说明书，以书面形式准确地描述软件需求

3.1 需求分析的任务

确定对系统的综合要求

分析系统的数据要求

导出系统的逻辑模型

修正系统开发计划

3.1.1 确定对系统的综合要求

1. 功能需求

2. 性能需求

3. 可靠性和可用性需求

4. 出错处理需求

5. 接口需求

6. 约束

7. 逆向需求

8. 将来可能提出的要求

3.1.2 分析系统的数据要求

建立数据模型——ER图

描绘数据结构——层次方框图和 Warnier 图
数据结构规范化

3.2 与用户沟通获取需求的方法

访谈：1. 正式访谈 2. 非正式访谈 3. 调查表 4. 情景分析技术

面向数据流自顶向下求精

简易的应用规格说明技术

快速建立软件原型：(1) 第四代技术 (4GL) (2) 可重用的软件构件 (3) 形式化规格说明和原型环境

3.3 分析建模与规格说明

3.3.1 分析建模

需求分析过程应该建立 3 种模型：数据模型 功能模型 行为模型

数据字典是分析模型的核心

实体-联系图用于建立数据模型的图形

数据流图是建立功能模型的基础

状态转换图是行为建模的基础

3.4 实体-联系图

数据模型中包含 3 种相互关联的信息：数据对象、数据对象的属性、数据对象彼此间相互连接的关系

3.4 状态转换图

3.6.1 状态

状态图分类：

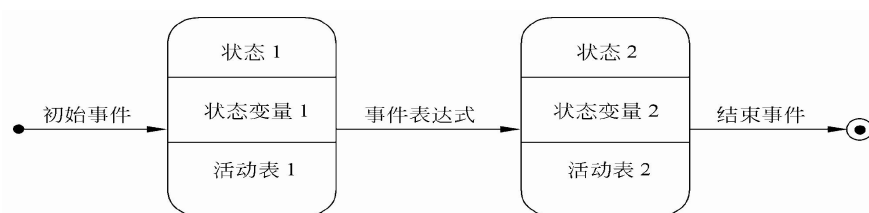
表示系统循环运行过程，通常不关心循环是怎样启动的。

表示系统单程生命期，需要标明初始状态和最终状态。

3.6.2 事件

事件就是引起系统做动作或(和)转换状态的控制信息。

3.6.3 符号



3.7 其他图形工具

3.7.1 层次方框图

3.7.2 Warnier 图

3.7.3 IPO 图

3.8 验证软件需求（重点）

3.8.1 从哪些方面验证软件需求的正确性

一致性 完整性 现实性 有效性

第五章 总体设计

5.1 设计过程

由两个主要阶段组成：

系统设计阶段，确定系统的具体实现方案：设想供选择的方案 选取合理的方案 推荐最佳方案

结构设计阶段，确定软件结构：功能分解 设计软件结构 设计数据库 制定测试文档 书写文档 审查和复查

5.2 设计原理

5.2.1 模块化

模块化的作用：

采用模块化原理可以使软件结构清晰，不仅容易设计也容易阅读和理解。

模块化使软件容易测试和调试，因而有助于提高软件的可靠性。

模块化能够提高软件的可修改性。

模块化也有助于软件开发工程的组织管理。

5.2.2 抽象

5.2.3 逐步求精

5.2.4 信息隐藏和局部化

5.2.5 模块独立

尽量使用数据耦合，
少用控制耦合和特征耦合，
限制公共环境耦合的范围，
完全不用内容耦合。

七种内聚的优劣评分结果：

高内聚：功能内聚

顺序内聚

中内聚：通信内聚

过程内聚

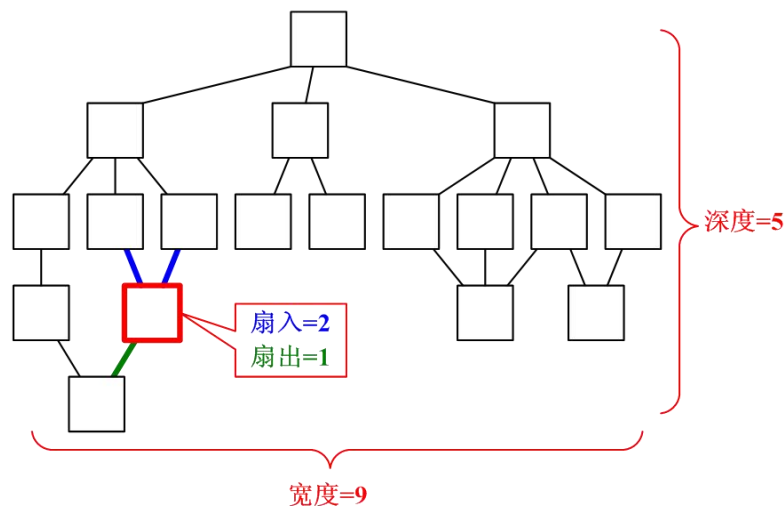
低内聚：时间内聚

逻辑内聚

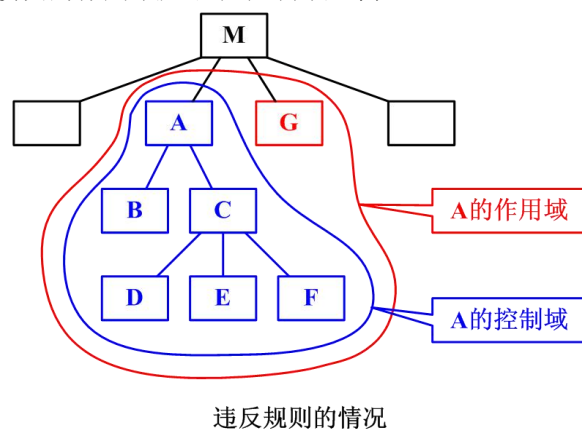
偶然内聚

5.3 启发规则

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、扇出和扇入都应适当



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

5.4 描绘软件结构的图形工具

5.4.1 层次图和HIPO图

1. 层次图(H图)

层次图用来描绘软件的层次结构。很适于在自顶向下设计软件的过程中使用。

2. HIPO图

5.4.2 结构图

5.5 面向数据流的设计方法

结构化设计方法(简称SD方法)，也就是基于数据流的设计方法。

5.5.1 概念

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。

信息流有两种类型：变换流 事务流

第6章 详细设计

6.1 结构程序设计

经典的结构程序设计：

只允许使用顺序、IF-THEN-ELSE 型分支和 DO-WHILE 型循环这 3 种基本控制结构；

扩展的结构程序设计：

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；

修正的结构程序设计：

再加上允许使用 LEAVE (或 BREAK) 结构。

6.2 人机界面设计

6.2.1 设计问题

设计人机界面过程中会遇到的 4 个问题：

系统响应时间：长度 易变性

用户帮助设施：集成的帮助设施附加的帮助设施

出错信息处理

命令交互

6.2.3 人机界面设计指南

一般交互指南

信息显示指南

数据输入指南

6.3 过程设计的工具

6.3.1 程序流程图（程序框图）

程序流程图的主要缺点：

程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制。

程序流程图不易表示数据结构。

6.3.2 盒图 (N-S 图)

盒图具有下述特点：

功能域明确。

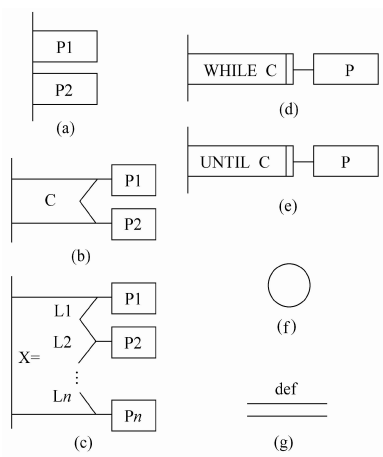
不可能任意转移控制。

很容易确定局部和全程数据的作用域。

很容易表现嵌套关系，也可以表示模块的层次结构。

6.3.3 PAD 图

它用二维树形结构的图来表示程序的控制流，将这种图翻译成程序代码比较容易。



PAD 图的主要优点如下：

使用表示结构化控制结构的 PAD 符号设计出来的程序必然是结构化程序。

PAD 图所描绘的程序结构十分清晰。

PAD 图表现程序逻辑易读、易懂、易记。

容易将 PAD 图转换成高级语言源程序，这种转换可用软件工具自动完成。

即可表示程序逻辑，也可描绘数据结构。

PAD 图的符号支持自顶向下、逐步求精方法的使用。

6.3.4 判定表

判定表却能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

所有条件	条件组合矩阵
所有动作	条件组合 对应的动作

判定表的缺点：

判定表的含义不是一眼就能看出来的，初次接触这种工具的人理解它需要有一个简短的学习过程。

当数据元素的值多于两个时，判定表的简洁程度也将下降。

6.3.5 判定树

判定树的优点：

它的形式简单，一眼就可以看出其含义，因此易于掌握和使用。

判定树的缺点：

简洁性不如判定表，数据元素的同一个值往往要重复写多遍，而且越接近树的叶端重复次数越多。

画判定树时分枝的次序可能对最终画出的判定树的简洁程度有较大影响。

6.3.6 过程设计语言(伪码)

伪代码的基本控制结构：

简单陈述句结构：避免复合语句。

判定结构：IF_THEN_ELSE 或 CASE_OF 结构。

选择结构：WHILE_DO 或 REPEAT_UNTIL 结构。

PDL 的优点：

可以作为注释直接插在源程序中。有助于保持文档和程序的一致性，提高了文档的质量。
可以使用普通的正文编辑程序或文字处理系统，很方便地完成 PDL 的书写和编辑工作。
已经有自动处理程序存在，而且可以自动由 PDL 生成程序代码。

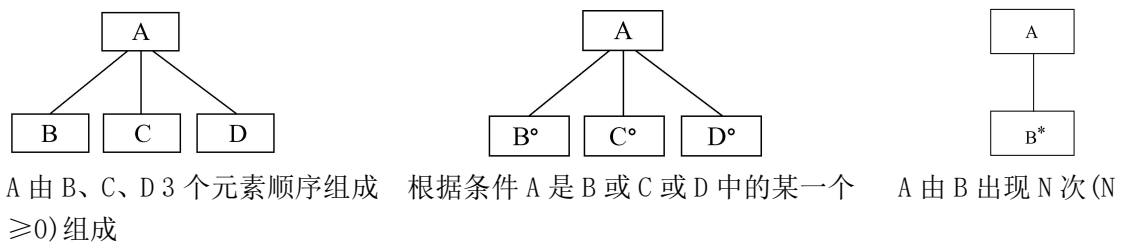
PDL 的缺点：

不如图形工具形象直观，描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单。

6.4 面向数据结构的设计方法

面向数据结构的设计方法的最终目标是得出对程序处理过程的描述。

6.4.1 Jackson



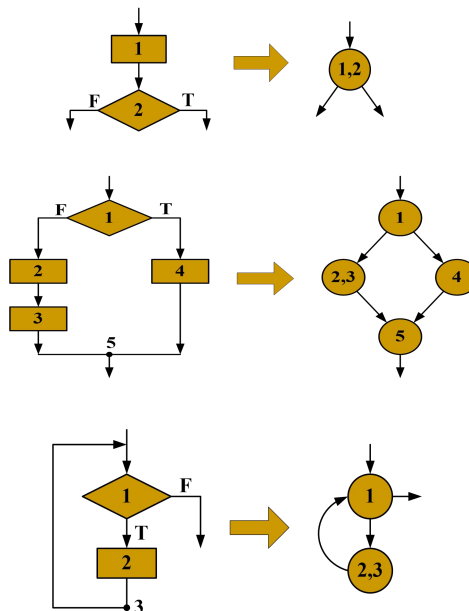
6.4.2 改进的 Jackson 图

6.4.3 Jackson 方法

6.5 程序复杂程度的定量度量

6.5.1 McCabe 方法

1. 流图（程序图）



2. 计算环形复杂度的方法

$V(G)$ = 流图中的区域数

$V(G) = E - N + 2$

其中 E 是流图中的边数，N 是结点数

$V(G) = P + 1$

其中 P 是流图中判定结点的数目

6.5.2 Halstead 方法

令 N_1 为程序中运算符出现的总次数, N_2 为操作数出现的总次数, 程序长度 N 定义为:

$$N = N_1 + N_2$$

程序中使用的不同运算符(包括关键字)的个数 n_1 , 以及不同操作数(变量和常数)的个数 n_2 。

预测程序长度的公式如下:

$$H = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

预测程序中包含错误的个数的公式如下:

$$E = N \log_2 (n_1 + n_2) / 3000$$

第 7 章 实现

编码和测试统称为实现。

7.1 编码

7.1.1 选择程序设计语言

主要的实用标准:

系统用户的要求
可以使用的编译程序
可以得到的软件工具
工程规模
程序员的知识
软件可移植性要求
软件的应用领域

7.1.2 编码风格

1. 程序内部的文档: 恰当的标识符 适当的注解 程序的视觉组织
2. 数据说明
3. 语句构造
4. 输入输出
5. 效率: 程序运行时间 存储器效率 输入输出的效率

7.2 软件测试基础

7.2.1 软件测试的目标

测试是为了发现程序中的错误而执行程序的过程;

好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案;

成功的测试是发现了至今为止尚未发现的错误的测试。

7.2.3 测试方法

黑盒测试(功能测试):

把程序看作一个黑盒子;

完全不考虑程序的内部结构和处理过程;

是在程序接口进行的测试。

白盒测试(结构测试):

把程序看成装在一个透明的盒子里;

测试者完全知道程序的结构和处理算法;

按照程序内部的逻辑测试程序, 检测程序中的主要执行通路是否都能按预定要求正确工作。

7.2.4 测试步骤

1. 模块测试(单元测试)

保证每个模块作为一个单元能正确运行；
发现的往往是编码和详细设计的错误。

2. 子系统测试

把经过单元测试的模块放在一起形成一个子系统来测试；
着重测试模块的接口。

3. 系统测试

把经过测试的子系统装配成一个完整的系统来测试；
发现的往往是软件设计中的错误，也可能发现需求说明中的错误；
不论是子系统测试还是系统测试，都兼有检测和组装两重含义，通常称为集成测试。

4. 验收测试(确认测试)

把软件系统作为单一的实体进行测试；
它是在用户积极参与下进行的，而且可能主要使用实际数据(系统将来要处理的信息)进行测试；
发现的往往是系统需求说明书中的错误。

5. 平行运行

7.2.5 测试阶段的信息流

输入信息有两类：

软件配置，包括需求说明书、设计说明书和源程序清单等；
测试配置，包括测试计划和测试方案。

7.3 单元测试

单元测试集中检测模块；

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；

单元测试主要使用白盒测试技术，对多个模块的测试可以并行地进行。

7.3.1 测试重点

模块接口

局部数据结构

重要的执行通路

出错处理通路

边界条件

7.3.2 代码审查

由审查小组正式进行测试称为代码审查；

一次审查会上可以发现许多错误，可以减少系统验证的总工作量。

7.3.3 计算机测试

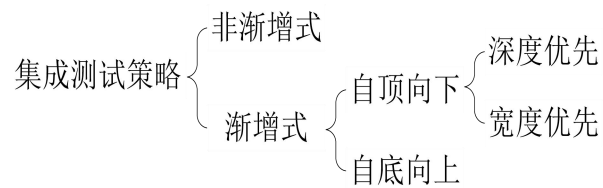
驱动程序是一个“主程序”，它接收测试数据，传送给被测试的模块，并且印出有关的结果。
存根程序代替被测试的模块所调用的模块。它使用被它代替的模块的接口，可能做最少量的数据操作，印出对入口的检验或操作结果，并且把控制归还给调用它的模块。

7.4 集成测试

集成测试是测试和组装软件的系统化技术，主要目标是发现与接口有关的问题。

由模块组装成程序时有两种方法：

7.4.3 不同集成测试策略的比较



混合策略：

改进的自顶向下测试方法

混合法

7.4.4 回归测试

7.5 确认测试

确认测试也称为验收测试，它的目标是验证软件的有效性。

7.5.3 Alpha 和 Beta 测试

Alpha 测试是在受控的环境中进行的。

Beta 测试是软件在开发者不能控制的环境中的“真实”应用。

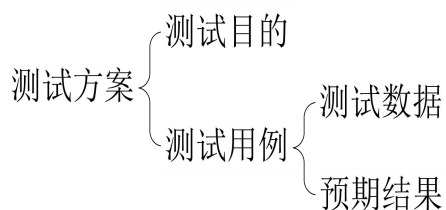
1. 接口测试
 2. 路径测试
 3. 功能测试
 4. 健壮性测试
 5. 性能测试
 6. 用户界面测试
 7. 信息安全测试
 8. 压力测试
 9. 可靠性测试
 10. 安装/反安装测试
- 确认测试也称为验收测试，它的目标是验证软件的有效性。

Alpha 测试是在受控的环境中进行的。

Beta 测试是软件在开发者不能控制的环境中的“真实”应用。

4. 接口测试
5. 路径测试
6. 功能测试
4. 健壮性测试
5. 性能测试
6. 用户界面测试
7. 信息安全测试
8. 压力测试
9. 可靠性测试
10. 安装/反安装测试

7.6 白盒测试技术



7.6.1 逻辑覆盖

语句覆盖

判定覆盖：比语句覆盖强，但对程序逻辑的覆盖程度仍不高。

条件覆盖：判定覆盖不一定包含条件覆盖，条件覆盖也不一定包含判定覆盖。

判定/条件覆盖：有时判定/条件覆盖也并不比条件覆盖更强。

条件组合覆盖：条件组合覆盖标准的测试数据并不一定能使程序中的每条路径都执行到。

6. 点覆盖（语句覆盖标准相同）

7. 边覆盖（判定覆盖一致）

8. 路径覆盖

7.6.2 控制结构测试覆盖

1. 基本路径测试

基本路径测试是 Tom McCabe 提出的一种白盒测试技术。

首先计算程序的环形复杂度；

以该复杂度为指南定义执行路径的基本集合；

2. 条件测试

从该基本集合导出的测试用例可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、假两种植值。

3. 循环测试

循环测试是一种白盒测试技术，它专注于测试循环结构的有效性。

在结构化的程序中通常只有 3 种循环，即简单循环、串接循环和嵌套循环。

7.7 黑盒测试技术

7.7.1 等价划分

7.7.2 边界值分析

7.7.3 错误推测

7.9 软降可靠性

7.9.1 基本概念

软件可靠性：

程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。

软件的可用性：

程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

第 8 章 维护

软件工程的目的是要提高软件的可维护性，减少软件维护所需要的工作量，降低软件系统的总成本。

8.1 软件维护的定义

软件维护：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。可分为 4 项活动：

改正性维护
适应性维护
完善性维护
预防性维护

8.2 软件维护的特点

8.2.1 结构化维护与非结构化维护差别巨大

8.2.2 维护的代价高昂

8.2.3 维护的问题很多

8.3 软件维护过程

1. 维护组织 2. 维护报告 3. 维护的事件流 4. 保存维护记录 5. 评价维护活动

8.4 软件的可维护性

决定软件可维护性的因素主要有 7 个：

可理解性
可测试性
可修改性
可靠性
可移植性
可使用性
效率

第 9 章 面向对象方法学引论

9.1 面向对象方法学概述

9.1.1 面向对象方法学要点

- (1) 认为客观世界是由各种对象组成的，任何事物都是对象
- (2) 把所有对象都划分成各种类对象，每个对象类都定义了一组数据和一组方法
- (3) 按照子类和父类的关系，把若干个对象类组成一个层次结构的系统
- (4) 对象彼此之间仅能通过传递消息相互联系

9.1.2 面向对象开发方法

面向对象=对象+类 +继承+通信

9.1.4 面向对象方法组成

面向对象的分析

面向对象的设计

面向对象的程序设计

9.1.6 面向对象方法的优点

1. 与人类习惯的思维方式一致
2. 稳定性好
3. 可重用性好
4. 可维护性好
5. 较易开发大型软件产品

9.2 面向对象的概念

9.2.1 对象

是客观事物或概念的抽象表述，即对客观存在的事物的描述统称为对象，对象可以是事、物、或抽象概念，是将一组数据和使用该数据的一组基本操作或过程封装在一起的实体。

对象的特点

- (1) 以数据为中心。
- (2) 对象是主动的。
- (3) 实现了数据封装。
- (4) 本质上具有并行性。
- (5) 模块独立性好。

9.2.2 类

是一组具有相同属性和相同操作的对象的集合。

9.2.3 实例

由某个特定的类所描述的一个具体的对象。

9.2.4 消息

向对象发出的服务请求（互相联系、协同工作等）。一个消息包含 3 个部分：接收消息的对象，消息名，消息变元

9.2.5 方法

方法就是对象所能执行的操作，也就是类中所定义的服务。

9.2.6 属性

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。

9.2.7 封装

对象封装了对象的数据以及对这些数据的操作。

9.2.8 继承(I)

继承是子类自动地共享基类中定义的数据和方法的机制。

单重继承：子类仅从一个父类继承属性和方法

多重继承：子类可从多个父类继承属性和方法

9.2.9 多态性

9.2.10 重载

9.3 面向对象建模(II)

面向对象开发软件，需要建立 3 种形式的模型。

对象模型。描述系统数据结构—数据结构。

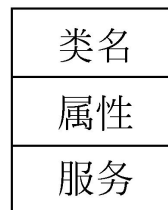
动态模型。描述系统控制结构—执行操作。

功能模型。描述系统功能—数值变化。

9.4 对象模型

9.4.1 类图的基本符号(I)

1. 定义类



2. 定义属性

可见性 属性名 : 类型 = 缺省值 {性质串}

可见性(visibility)表示该属性对类外的元素是否可见。

分为：

public (+) 公有的，即模型中的任何类都可以访问该属性。

private (-) 私有的，表示不能被别的类访问。

protected (#) 受保护的，表示该属性只能被该类及其子类访问。

如果可见性未申明，表示其可见性不确定。

3. 定义操作

可见性 操作名 (参数表): 返回类型 {性质串}

9.4.2 表示关系的符号(I)

9.4.2.1 关联(I)

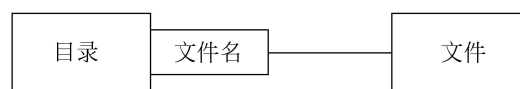
关联表示两个类的对象之间存在某种语义上的联系。

(1) 普通关联

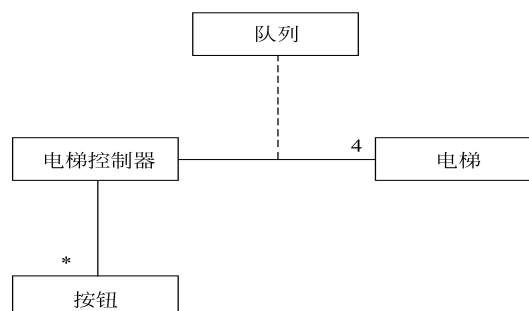


递归关联：一个类与本身有关联关系

(3) 限定关联



(4) 关联类



9.4.2.2 聚集(I)

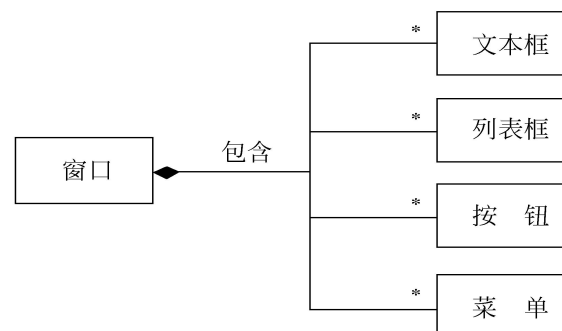
(1) 共享聚集

如果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。



(2) 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,则该聚集称为组合聚集。

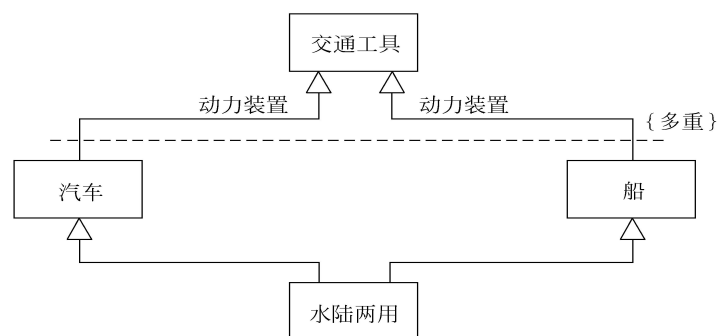


9.4.2.3 泛化(I)

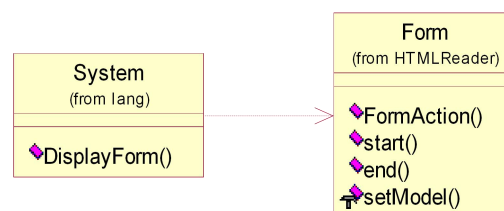
(1) 普通泛化

(2) 受限泛化

预定义的约束有 4 种: 多重、不相交、完全和不完全。



9.4.2.4 依赖



9.4.2.5 细化



9.5 动态模型

9.6 功能模型

9.6.1 用例图

模型元素：系统、行为者、用例及用例之间的关系(扩展关系、使用关系)

用例的实例是脚本

第 10 章 面向对象分析

10.1 面向对象分析的基本过程

面向对象分析：抽取和整理用户需求并建立问题域精确模型的过程。

理解——用户、分析员和领域专家

表达——需求规格说明书（对象模型、动态模型、功能模型）

验证——二义性，完善性

对象模型最基本、最重要、最核心。

3 个子模型

- 静态结构（对象模型）
- 交互次序（动态模型）
- 数据变换（功能模型）

—— 主题层

—— 类与对象层

—— 结构层

—— 属性层

—— 服务层

复杂问题的对象模型的 5 个层次

面向对象分析的过程

寻找类与对象

识别结构

识别主题

定义属性

建立动态模型

建立功能模型

定义服务

10.2 需求陈述

需求陈述是阐明“做什么”，而不是“怎样做”

问题范围

功能需求
性能需求
应用环境
假设条件

第 11 章 面向对象设计

11.1 面向对象设计的准则

1. 模块化
2. 抽象
3. 信息隐藏
4. 弱耦合

耦合指不同对象之间相互关联的紧密程度。

对象之间的耦合分两类：

交互耦合

如果对象之间的耦合通过消息连接来实现，则这种耦合就是交互耦合。交互耦合应尽可能松散。

继承耦合

与交互耦合相反，应该提高继承耦合程度。

5. 强内聚

在面向对象设计中存在下述 3 种内聚：

服务内聚：一个服务应该完成一个且仅完成一个功能。

类内聚：一个类应该只有一个用途，它的属性和服务应该是高内聚的。

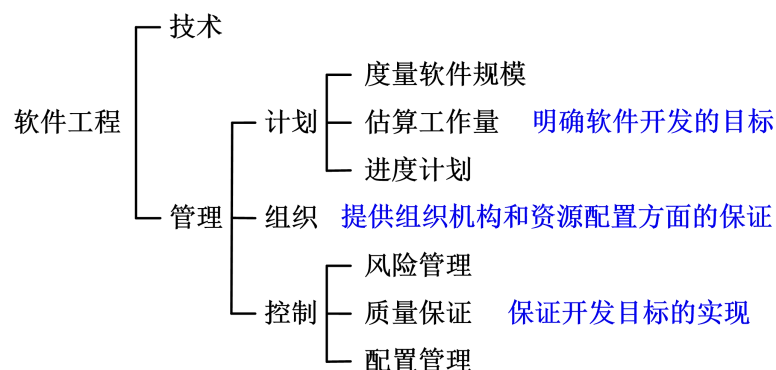
一般-特殊内聚：设计出的一般-特殊结构，应该符合多数人的概念

6. 可重用

11.2 启发规则

1. 设计结果应该清晰易懂
2. 一般-特殊结构的深度应适当
3. 设计简单的类
4. 使用简单的协议
5. 使用简单的服务
6. 把设计变动减至最小

第 13 章 软件项目管理



13.1 估算软件规模

13.1.1 代码行技术

估算方法：

由多名有经验的软件工程师分别做出估计。

每个人都估计程序的最小规模(a)、最大规模(b)和最可能的规模(m)，分别算出这3种规模的平均值之后，再用下式计算程序规模的估计值：

$$L = \frac{\bar{a} + 4\bar{m} + \bar{b}}{6}$$

单位：

LOC 或 KLOC。

代码行技术的优点：

代码是所有软件开发项目都有的“产品”，而且很容易计算代码行数；

有大量参考文献和数据。

代码行技术的缺点：

源程序仅是软件配置的一个成分，由源程序度量软件规模不太合理；

用不同语言实现同一个软件所需要的代码行数并不相同；

不适用于非过程性语言。

13.1.2 功能点技术

功能点技术依据对软件信息域特性和软件复杂性的评估结果，估算软件规模。

这种方法用功能点(FP)为单位度量软件规模。

1. 信息域特性

输入项数(Inp)、输出项数(Out)、查询数(Inq)、主文件数(Maf)、外部接口数(Inf)

每个特征根据其复杂程度分配一个功能点数，即信息域特征系数 a1, a2, a3, a4, a5

2. 估算功能点的步骤

(1) 计算未调整的功能点数 UFP

$$UFP = a1 \times Inp + a2 \times Out + a3 \times Inq + a4 \times Maf + a5 \times Inf$$

(2) 计算技术复杂性因子 TCF

技术因素对软件规模的综合影响程度 DI：

$$DI = \sum_{i=1}^{14} F_i$$

技术复杂性因子 TCF 由下式计算：

$$TCF = 0.65 + 0.01 \times DI$$

因为 DI 的值在 0~70 之间，所以 TCF 的值在 0.65~1.35 之间。

(3) 计算功能点数 FP

$$FP = UFP \times TCF$$

功能点技术优点：与所用的编程语言无关，比代码行技术更合理。

功能点技术缺点：在判断信息域特性复杂级别和技术因素的影响程度时主观因素较大，对经验依赖性较强。

13.2 工作量估算

13.2.1 静态单变量模型

$$E = a \times \text{KLOC}^b \times \prod_{i=1}^{17} f_i$$

ev 是估算变量 (KLOC 或 FP)

13.2.2 动态多变量模型

动态多变量模型也称为软件方程式，该模型把工作量看作是软件规模和开发时间这两个变量的函数。

$$E = (\text{LOC} \times B0.333/P)^3 \times (1/t)^4$$

13.2.3 COCOMO2 模型（构造性成本模型）

3 个层次的估算模型：

应用系统组成模型：这个模型主要用于估算构建原型的工作量，模型名字暗示在构建原型时大量使用已有的构件。

早期设计模型：这个模型适用于体系结构设计阶段。

后期体系结构模型：这个模型适用于完成体系结构设计之后的软件开发阶段。

COCOMO2 使用的 5 个分级因素：项目先例性、开发灵活性、风险排除度、项目组凝聚力、过程成熟度

13.3 进度计划

13.3.1 估算开发时间

Brooks 规律：向一个已经延期的项目增加人力，只会使得它更加延期。

13.3.2 Gantt 图

Gantt 图的主要优点：

Gantt 图能很形象地描绘任务分解情况，以及每个子任务(作业)的开始和结束时间。

具有直观简明和容易掌握、容易绘制的优点。

Gantt 图的 3 个主要缺点：

不能显式地描绘各项作业彼此间的依赖关系；

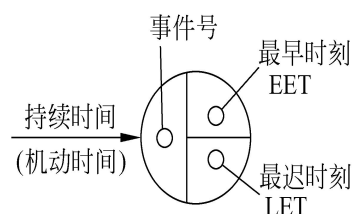
进度计划的关键部分不明确，难于判定哪些部分应当是主攻和主控的对象；

计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费。

13.3.3 工程网络

工程网络是系统分析和系统设计的强有力的工具。

13.3.4 估算工程进度



计算最早时刻 EET 使用下述 3 条简单规则：

考虑进入该事件的所有作业；

对于每个作业都计算它的持续时间与起始事件的 EET 之和；

选取上述和数中的最大值作为该事件的最早时刻 EET。

计算最迟时刻 LET 使用下述 3 条规则：

考虑离开该事件的所有作业；

从每个作业的结束事件的最迟时刻中减去该作业的持续时间；

选取上述差数中的最小值作为该事件的最迟时刻 LET。

13.3.5 关键路径

关键事件：EET=LET

13.3.5 机动时间=(LET)结束-(EET)开始-持续时间

=右下角-左上角-持续时间

13.4 人员组织

13.4.1 民主制程序员组

如果小组内有 n 个成员，则可能的通信信道共有 $n(n-1)/2$ 条。

13.4.2 主程序员组

主程序员组的两个重要特性：专业化、层次性

13.4.3 现代程序员组

13.5 质量保证

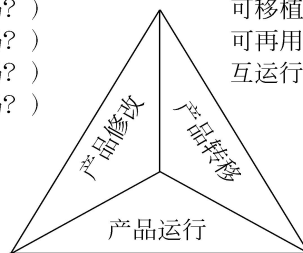
13.5.1 软件质量

可理解性（我能理解它吗？）

可维修性（我能修复它吗？）

灵活性（我能改变它吗？）

可测试性（我能测试它吗？）



可移植性（我能在另一台机器上使用它吗？）

可再用性（我能再用它的某些部分吗？）

互运行性（我能把它和另一个系统结合吗？）

正确性（它按我的需要工作吗？）

健壮性（对意外环境它能适当地响应吗？）

效率（完成预定功能时它需要的计算机资源多吗？）

完整性（它是安全的吗？）

可用性（我能使用它吗？）

风险（能按预定计划完成它吗？）

13.5.2 软件质量保证措施

13.6 软件配置管理

基于非执行的测试（复审或评审），主要用来保证在编码之前各阶段产生的文档的质量；

基于执行的测试（软件测试），需要在程序编写出来之后进行，它是保证软件质量的最后一道防线；

程序正确性证明，使用数学方法严格验证程序是否与对它的说明完全一致。

1. 技术复审的必要性

2. 走查：参与者驱动法、文档驱动法

3. 审查：综述 准备 审查 返工 跟踪

4. 程序正确性证明

软件配置管理是在软件的整个生命期内管理变化的一组活动。

具体地说，这组活动用来：①标识变化；②控制变化；③确保适当地实现了变化；④向需要知道这类信息的人报告变化。

软件配置管理的目标：使变化更正确且更容易被适应，在必须变化时减少所需花费的工作量。

13.6.1 软件配置

1. 软件过程的输出信息（软件配置项）：
计算机程序（源代码和可执行程序）；
描述计算机程序的文档（供技术人员或用户使用）；
数据（程序内包含的或在程序外的）

2. 基线

基线就是通过了正式复审的软件配置项。

13.6.2 软件配置管理过程

软件配置管理主要有 5 项任务：

1. 标识软件配置中的对象：基本对象、聚集对象
2. 版本控制
3. 变化控制
4. 配置审计
5. 状态报告

13.7 能力成熟度模型

1. 初始级

软件过程的特征是无序的，有时甚至是混乱的。

2. 可重复级

软件机构建立了基本的项目管理过程(过程模型)，可跟踪成本、进度、功能和质量。

3. 已定义级

软件机构已经定义了完整的软件过程（过程模型），软件过程已经文档化和标准化。

4. 已管理级

软件机构对软件过程（过程模型和过程实例）和软件产品都建立了定量的质量目标，所有项目的重要的过程活动都是可度量的。

5. 优化级

软件机构集中精力持续不断地改进软件过程。这一级的软件机构是一个以防止出现缺陷为目标的机构，它有能力识别软件过程要素的薄弱环节，并有足够的手段改进它们。