# 第 6 章　软件实现

# 本章内容

# 6.1 结构化程序设计

结构化程序（Structured Program）是由基本的控制结构构造而成的程序。每个控制结构只有一个入口点和一个出口点。控制结构集包括指令序列、指令或指令序列的条件选择以及一个指令或指令序列的重复执行。

结构化程序设计（Structured Programming，SP）是一种良好的软件开发技术。它采用自顶向下设计和实现的方法，严格地使用结构化程序构造软件。此技术可降低程序设计的复杂性，提高清晰度，便于排除隐含的错误，有利于程序的修改。

# 本章内容

# 6.2 选择程序设计语言

## 01 OPTION 程序设计语言的发展

### 机器语言 01

由二进制0、1 代码构成指令，不同的CPU 具有不同的指令系统

### 汇编语言 02

机器指令的符号化，与机器指令有着直接的对应关系

### 高级程序设计语言 03

使用的概念和符号与人们通常使用的概念和符号比较接近，一条语句往往对应若干条机器指令

### 非过程化语言 04

编码时只需说明“做什么”，不需描述算法细节

# 6.2 选择程序设计语言

## 02 OPTION 选用汇编语言的情况

一般在设计应用软件时，应当优先选用高级语言，只有下列3 种情况才选用汇编语言。

软件系统对程序执行时间和使用空间都有严格限制

系统硬件是特殊的微处理机，不能使用高级语言

大型系统中的某一部分，其执行时间非常关键，或直接依赖于硬件，这部分用汇编语言编写，其余部分用高级语言编写

# 6.2 选择程序设计语言

03 OPTION

## 目前常用的程序设计语言

### C 语言 01

- 生成的目标代码质量好，程序执行效率高
- 描述问题比汇编语言迅速，工作量小，可读性好，易于调试、修改和移植

### C++ 02

- 一种面向对象的程序设计语言
- 可以定义类、多态性、异常处理，支持数据封装、继承、多继承和数据隐藏

### Java 03

- 一种面向对象的、用于网络环境的程序设计语言，需要 Java 虚拟机解释执行

### Python 04

- 一种面向对象的解释型高级语言，简单而功能强大，易于学习、编辑周期短

# 6.2 选择程序设计语言

## 04 OPTION 选用高级语言的实用标准

### 项目的应用领域

科学工程计算、数据处理与数据库应用、实时处理、系统软件、人工智能

### 软件开发的方法

编程语言的选择依赖于开发的方法

### 软件开发环境

不同的软件开发环境，所使用的编程语言不同

### 移动互联应用系统

移动互联网有几种不同的操作系统，所使用的编程语言有所不同

### 根据系统用户的要求来选择

选择用户所熟悉的编程语言书写程序

### 软件开发人员的知识

选择一种软件开发人员熟悉的编程语言，使开发速度更快，质量更易保证

# 本章内容

# 6.3 程序设计风格

## 01 OPTION 源程序文档编写规则

在编写源程序文档时，标识符名称、注解、程序布局等要合理。

选用含义鲜明的标识符

注解是程序员和程序读者之间通信的重要工具，通常在每个模块开始处用注解简述模块的功能、主要算法、接口特点、调用方式、开发简史以及重要数据的含义、用途、限制、约束等

程序布局。适当利用阶梯形式，使程序的层次结构清晰、明显。

# 6.3 程序设计风格

## 02 OPTION 数据说明

数据说明的顺序应规范


说明同一语句的多个变量时
应按英文字母的顺序排列

对于复杂的数据结构
要加注释语句

变量说明不要遗漏
变量的类型、长度、
存储及初始化要正确

# 6.3 程序设计风格

## 03 OPTION 语句构造要简单直接

- 不要为了节省空间把多个语句写在同一行。
- 尽量避免复杂的条件测试。
- 尽量减少对“非”条件的测试。
- 对于多分支语句，应尽量把出现可能性大的情况放在前面，这样可以缩短运算时间。
- 避免大量使用循环嵌套语句和条件嵌套语句。
- 利用括号使逻辑表达式或算术表达式的运算次序清晰、直观。
- 每个循环要有终止条件，不要出现死循环，也要避免出现不可能被执行的循环。

# 6.3 程序设计风格

## 04 OPTION 输入/ 输出语句

- 检验输入数据的合法性、有效性。
- 检查输入项的重要组合的合理性。
- 提示输入的请求，并简明地说明可用的选择或边界数值。
- 输入格式简单，方便用户使用，尽量保持格式的一致性。
- 批量输入数据时，使用结束标志。
- 输出信息中不要有文字错误，要保证输出结果的正确性。
- 输出数据表格化、图形化。
- 给所有输出数据加标志。

# 6.3 程序设计风格

## 05 OPTION 程序效率

程序效率主要指处理机工作时间、内存容量这两方面的利用率，以及系统输入、输出的效率。人机交互界面如果设计得清晰、合理，会减少用户脑力劳动的时间，提高人机通信的效率。

在目前计算机硬件设备运算速度大大提高、内存容量增加的情况下，提高效率不是最重要的，程序设计主要应考虑的是程序的正确性、可理解性、可测试性和可维护性。

# 本章内容

# 6.4 程序设计质量评价

- 正确性：通过对算法的精心设计和详尽的检查实现程序的正确性。
- 清晰的结构：程序的结构必须与数据相适应，采用结构化设计方法，模块的输入和输出过程精确定义。
- 易使用性：操作简便，使用户学习使用软件花费的时间减少。
- 易维护性：程序易读、易理解就容易测试，也容易修改和扩充。修改模块化结构的程序，对程序的总体结构不会产生影响。
- 简单性：简单的程序结构容易理解、容易修改。要把复杂的问题简单化，需要具有一定的程序设计经验和娴熟的技巧，还要有一定的耐性。
- 易移植性：程序从某一环境移植到另一环境的能力。

# 本章内容

# 6.5 程序设计文档

在编码结束前，应对每个程序模块的源程序进行静态分析和模块测试，做好测试记录。静态分析和模块测试时，应检查下述内容。

程序与详细设计是否相符合，模块的运行是否正确

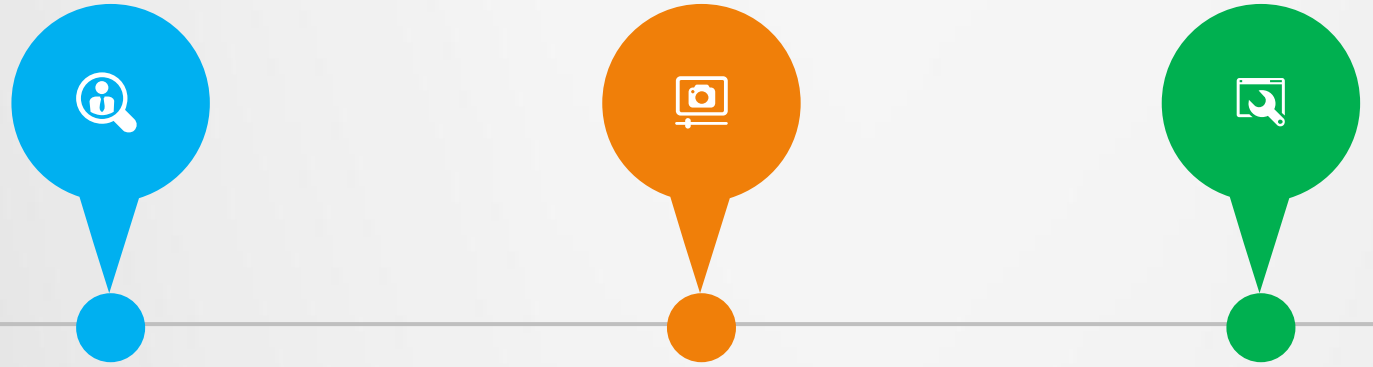
内部文件和程序的可读性如何

坚持结构化程序设计标准，语言使用是否得当

# 本章内容

# 6.6.1 软件测试目标

G.J.Myers 在《软件测试的艺术》一书中对测试提出的如下规则，不妨可看作软件测试的目标。



软件测试是为了发现程序中的错误而执行程序的过程

好的测试方案能够发现尚未发现的错误

成功的测试是发现了尚未发现的错误的测试

# 6.6.2 软件测试原则

- 在测试开始时，不要认为程序中没有错误。如果没有测试错误的愿望，是不太可能找出错误的。
- 要避免测试自己编写的程序，由别人来测试会更客观、更有效。但在发现错误之后，要找出错误的根源并纠正错误时，则应由程序编写者来进行。
- 测试用例要有输入数据和对应的预期结果。要对合理的输入数据和不合理的输入数据都进行测试，这样才可测试出程序的排错能力。
- 不仅要检查程序功能是否完备，还应检查程序是否做了多余的工作。
- 要精心设计测试方案，尽量把软件中的错误测试出来。
- 对错误较多的程序段应进行更深入的测试，因为错误较多的程序段质量较差，修改错误时容易引入新的错误。
- 应长期保存所有测试用例，直至该程序被废弃。

# 本章内容

# 6.7.1 静态分析与动态测试

**静态分析**

静态分析不执行被测试软件，通过对需求分析说明书、软件设计说明书及源程序做结构检查、流程图分析、编码分析等来找出软件错误。

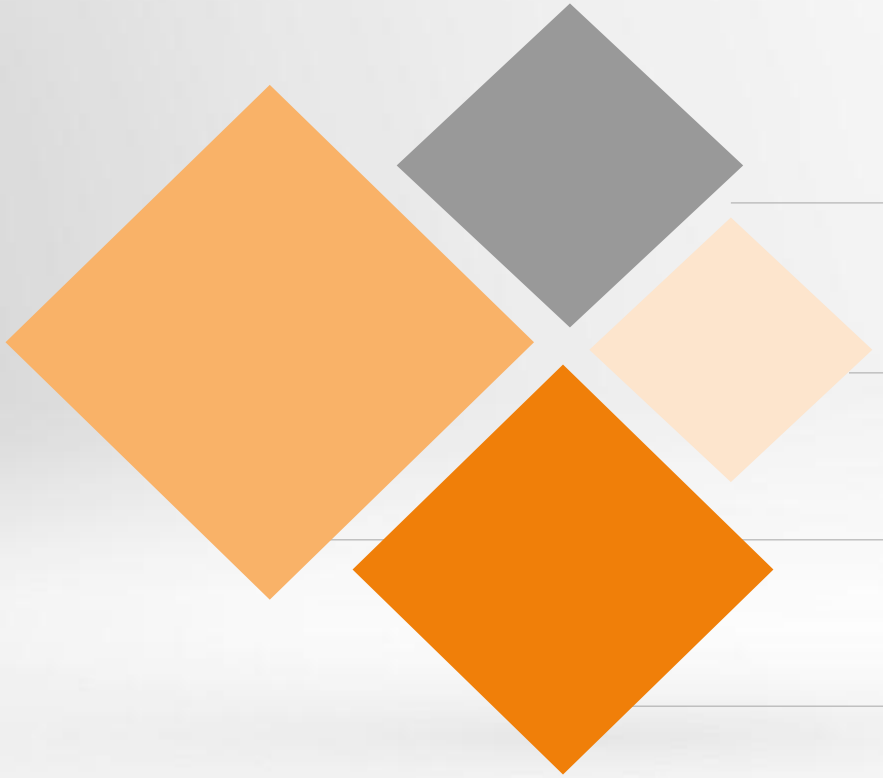
**动态测试**

- ✓ 动态测试通过执行程序并分析程序来查错。
- ✓ 为了进行软件测试，需要预先准备好两种数据，一是输入数据，二是预期的输出结果。
- ✓ 以发现错误为目标的用于软件测试的输入数据及与之对应的预期输出结果叫测试用例。设计测试用例是动态测试的关键。

# 6.7.2 黑盒法与白盒法

## 01 OPTION 黑盒法

黑盒法（Black Box Testing）又称功能测试，其测试用例完全是根据程序的功能说明来设计的。黑盒法是最基本的测试法之一，其主要目的是发现以下错误。



- 是否有不正确的或遗漏的功能
- 检查输入能否被正确地接收，软件能否正确地输出结果
- 访问外部信息是否有错
- 性能上能否满足要求

# 6.7.2 黑盒法与白盒法

## 02 OPTION 白盒法

白盒法（White Box Testing）又称结构测试，其测试用例是根据程序内部的逻辑结构和执行路径来设计的。

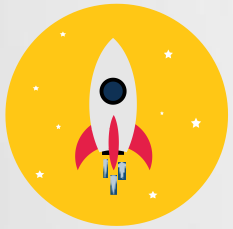
对于一些简单的程序，穷尽测试无法实现，而穷尽路径测试有可能实现。用白盒法测试时，从检查程序的逻辑着手，检验程序中的每条通路是否都能按预定要求正确工作。

在软件测试时，常把黑盒法和白盒法联合起来进行，这也称为灰盒法。

# 本章内容

# 6.8.1 模块测试

测试模块主要有驱动程序（也称驱动模块）和存根程序（也称桩模块）两种。



驱动程序代替主程序，用来测试子程序。它主要用在接收测试数据后，调用被测试模块，输出测试结果，由此来检验子程序的处理是否正确。

存根程序也称“虚拟子程序”，代替被测试模块所调用的模块，利用存根程序测试主模块。

# 6.8.2 集成测试

## 01 OPTION 集成测试分类

**子系统测试**

子系统测试是把通过模块测试运行正确的模块放在一起形成子系统后再测试。这个步骤着重测试模块的接口，测试模块之间能否相互协调及通信时有没有问题。

**系统测试**

- ✓ 把经过测试运行正确的子系统组装成完整的系统后再进行测试。
- ✓ 系统测试的目的是测试整个硬件和软件系统，验证系统是否满足规定的需求。

# 6.8.2 集成测试

02 OPTION

## 渐增式集成策略

**自顶向下集成**

自顶向下集成即从主控模块开始，把附属的模块组装到软件结构中，可使用深度优先的策略或宽度优先的策略。

**自底向上集成**

自底向上集成策略从软件结构最底层的模块开始组装和测试，不需要存根程序，需要驱动程序，底层模块的错误发现早，但是总体结构的合理性及上层模块的接口错误发现较晚。

# 6.8.3 程序审查会和人工运行

## 01 OPTION 程序审查会

- 程序员逐句讲述程序的逻辑结构，由测试专家提问、研究，判断是否有错误存在。
- 程序审查会成员根据常见程序错误分析程序。为了确保会议的效率，应使参加者集中精力查找错误，而不是改正错误。会后再由程序员自己来改正错误。程序审查会的时间每次最好控制在90 到120 分钟之间，时间太长了效率不高。

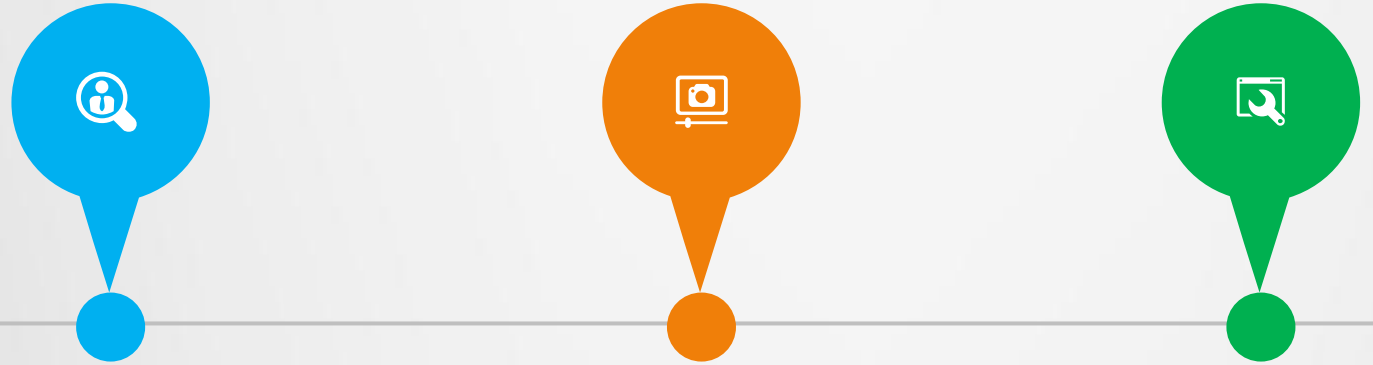
# 6.8.3 程序审查会和人工运行

## 02 OPTION 人工运行

- 人工运行是阅读程序查错的一种方法，人工运行小组的成员由编程人员及其他有丰富经验的程序员、其他项目的参加者等组成。
- 人工运行时，要求与会者模拟计算机运行程序，把各种测试情况沿着程序逻辑走一遍，通过向程序员询问程序的逻辑设计情况来发现错误。

# 6.8.4 确认测试

软件确认是在软件开发过程结束时对软件进行评价，以确认它和软件需求规格说明书是否一致的过程。确认（Validation）测试也称验收（Verification）测试，其目标是验证软件的有效性。



确认测试必须有用户积极参与，以用户为主进行

软件配置复审

Alpha 测试和Beta 测试

# 6.8.5 平行运行

比较重要的软件要有一段试运行时间，此时新开发的系统与原先的旧系统同时运行，称为平行运行。平行运行时要及时与旧系统比较处理结果，这样做有以下几个好处。

- 让用户熟悉系统运行情况，并验证用户手册的正确性。
- 若发现问题可及时对系统进行修改。
- 可对系统的性能指标进行全面测试，以保证系统的质量。

# 本章内容

# 6.9.1 等价类划分法

## 01 OPTION 方法

等价类划分（Equivalence Partitioning）法是黑盒法设计测试方案的一种，它把所有可能的输入数据划分成有限个等价类，用每个等价类中的一个典型值作为测试数据。

在等价类中各个数据的测试作用与这一类中所有其他数据的测试作用相同，因此在每个等价类中只用一组数据作为代表进行测试来发现程序中的错误。等价类可以交错。

# 6.9.1 等价类划分法

## 02 OPTION 等价类划分的规则

如果输入数据有规定范围，则范围内的数据属于有效等价类，小于最小值或大于最大值均属于无效等价类

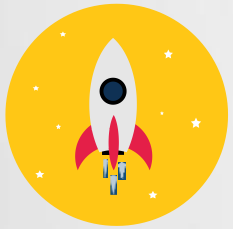

如果输入数据有规定的个数，则符合规定个数的数据属于有效等价类，不输入数据或超出规定个数的数据属于无效等价类

如果输入数据有一个规定的集合，而且程序对不同的输入数据有不同的处理，则集合中的元素属于有效等价类，集合外的元素属于无效等价类

如果输入数据有一定的规则，则符合规则的数据组成有效等价类，各种不符合规则的数据组成不同的无效等价类

# 6.9.1 等价类划分法

## 03 OPTION 等价类划分法的步骤



研究程序的功能说明，以确定输入数据是有效等价类还是无效等价类。

分析输出数据的等价类，以便根据输出数据的等价类导出相应的输入数据等价类。

# 6.9.1 等价类划分法

## 04 OPTION 等价类测试步骤

- 划分等价类，为每个等价类编号。
- 设计测试用例，使它覆盖尽可能多的有效等价类，直到所有有效等价类均被覆盖为止。
- 设计新的测试用例，每个测试用例覆盖一个且仅覆盖一个无效等价类，直到所有无效等价类均被覆盖为止。

# 6.9.2 边界值分析法

- 如果输入值有规定范围，则测试这个范围内及两个边界和刚刚超出范围的情况。
- 如果输入数据有规定的个数，分别对最多个数、最少个数、稍大于最多个数和稍小于最少个数的情况进行测试。
- 在输入值有一定规则或有规定的集合时，要多思考以找出各种边界条件，对它们进行测试，尽量不要遗漏可能产生错误的情况。
- 对输出数据等价类的边界情况，要分析出与其对应的输入数据，对它们也应进行测试。边界值分析法看起来很简单，若能正确地掌握这种分析法，往往会是最有效的测试方法之一。

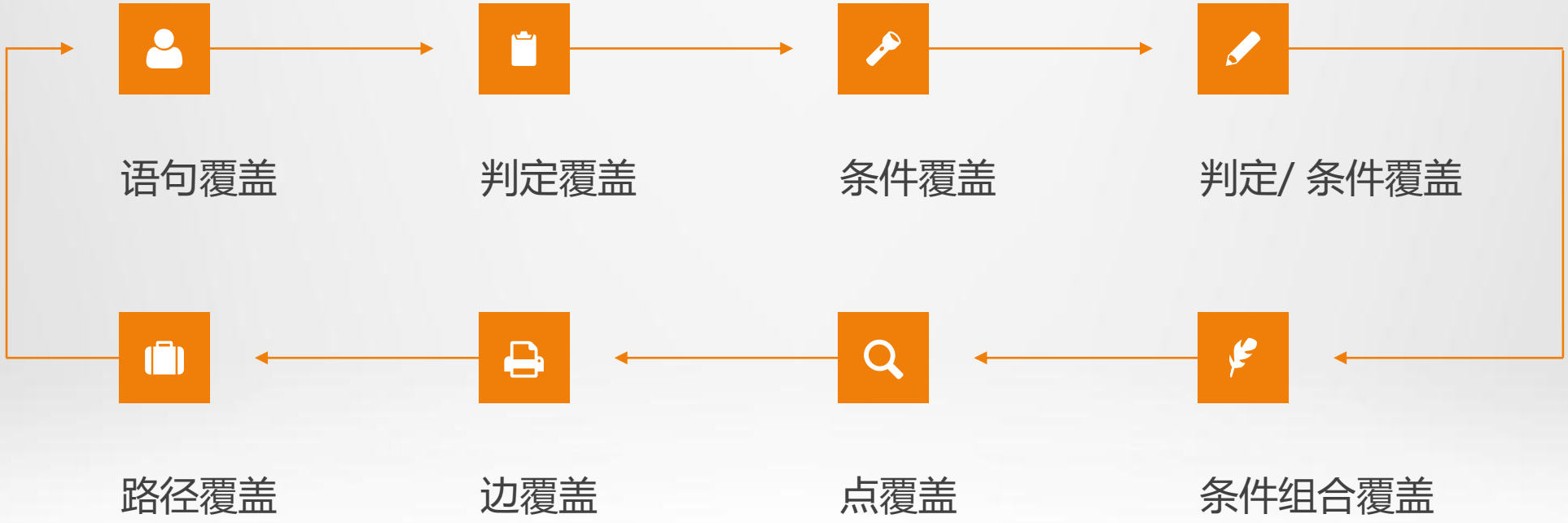
# 6.9.3 错误推测法

错误推测法主要考虑某些容易发生错误的特殊情况来设计测试用例。错误推测法主要靠直觉和经验进行，因而没有确定的步骤。

等价类划分法和边界值分析法都只孤立地考虑单个数据输入后的测试效果，而没有考虑多个数据输入时不同的组合所产生的效果，有时可能会遗漏容易出错的输入数据的组合情况，有效的办法是用判定表或判定树把输入数据的各种组合与对应的处理结果列出来进行测试。

# 6.9.4 逻辑覆盖法

白盒法根据程序逻辑结构进行测试，逻辑覆盖（Logic Coverage）法是一系列测试过程的总称，这些测试是逐渐地、越来越完整地进行通路测试。



# 6.9.5 实用测试策略

在对软件系统进行测试时，应联合使用各种测试方法进行综合测试，通常先用黑盒法设计基本测试用例，再用白盒法补充一些必要的测试用例，具体测试策略如下。

- 用等价类划分法设计测试方案。
- 使用边界值分析法，既测试输入数据的边界情况，又检查输出数据的边界情况。
- 如果含有输入条件的组合情况，要分析所有条件组合的执行情况。
- 必要时用错误推测法补充测试方案。
- 用逻辑覆盖法检查现有测试方案，若没有达到逻辑覆盖标准，再补充一些测试用例。

# 本章内容

# 6.10.1 软件调试

## 1 软件调试的目的

尽可能多地发现程序中的错误。软件调试的目的是确定错误的原因和位置、分析和改正程序中的错误。

## 2 软件调试的方法

- 进行软件测试，检查哪个模块、哪段程序有错。
- 纠错，要确定错误发生的确切位置和错误的原因并改正错误。

## 3 软件调试技术

- 对计算机工作过程进行模拟或跟踪，记录中间结果，发现错误立即纠正。
- 设置输出语句
- 逐层分块调试
- 对分查找调试
- 回溯法

# 6.10.2 软件验证

软件验证是确定软件开发周期中一个给定阶段的产品是否满足需求的过程。软件验证的方法如下。

- 确定软件操作正确
- 指示软件操作错误
- 指示软件执行时产生错误的原因
- 把源程序和软件配置的其他组成部分自动输入系统

# 6.10.3 软件确认

软件确认是指在软件开发过程结束后，对所开发的软件进行评价，以确定它是否和软件需求一致的过程。软件确认测试又称有效性测试。需求规格说明书是软件确认测试的基础。

软件确认工作应在用户直接参与下，在最终用户环境中进行，即在事先规定的时期内运行软件的全部功能，考查软件运行有无严重错误。

完成测试计划中的所有要求，分析测试结果，并书写测试分析报告和开发总结。

按用户手册和操作手册进行软件实际运行，验证软件的实用性和有效性，并修正所发现的错误。

# 本章内容

# 6.11 软件测试计划和分析报告

## 01 OPTION 软件测试计划

软件测试计划描述测试活动的范围、方法、资源和进度，规定被测试的项、特性、应完成的测试任务、承担各项工作的人员的职责及与本计划有关的风险等。

每项测试活动都包括测试内容、进度安排、设计考虑、测试数据的整理方法及评价准则等。

# 6.11 软件测试计划和分析报告

## 02 OPTION 软件测试说明

### 1 测试设计说明

每项测试的控制、输入、输出、过程、评价准则、范围、数据整理及评价尺度等。

### 2 测试用例说明

要测试的功能、输入值以及对应的输出结果，在使用具体测试用例时对测试规程的限制

### 3 测试规程说明

规定对于系统执行指定的测试用例来实现测试设计所要求的所有步骤

# 6.11 软件测试计划和分析报告

03 OPTION

## 软件测试报告

**测试项传递报告** 01

测试项的位置、状态等

**测试日志** 02

记录测试执行过程中发生的情况，如活动和事件的条目、描述

**测试事件报告** 03

测试执行期间发生的一切事件的描述和影响

**测试总结报告** 04

总结与测试设计说明有关的测试活动、差异、结果概述、评价、活动总结及批准者等