

| College of Computer Science, Chongqing University |

Software Engineering

A Practitioner's Approach Seventh Edition

18 测试传统的应用软件

zmqmail@cqu.edu.cn 13708390417



测试各个单元!



Testability

(可测试性)
计算机程序能够被测试的容易程度。

可测试性 (Testability)-计算机程序能够被测试的容易程度。

- 可操作性(Operability)—it operates cleanly
 - 能测试吗?
- 可观察性 (Observability)—the results of each test case are readily observed
- 可控制性 (Controllability)—the degree to which testing can be automated and optimized
- 可分解性 (Decomposability)—testing can be targeted
- 简单性(Simplicity)—reduce complex architecture and logic to simplify tests
- 稳定性(Stability)—few changes are requested during testing
 - 有时候会出现?
- 易理解性 (Understandability)—of the design



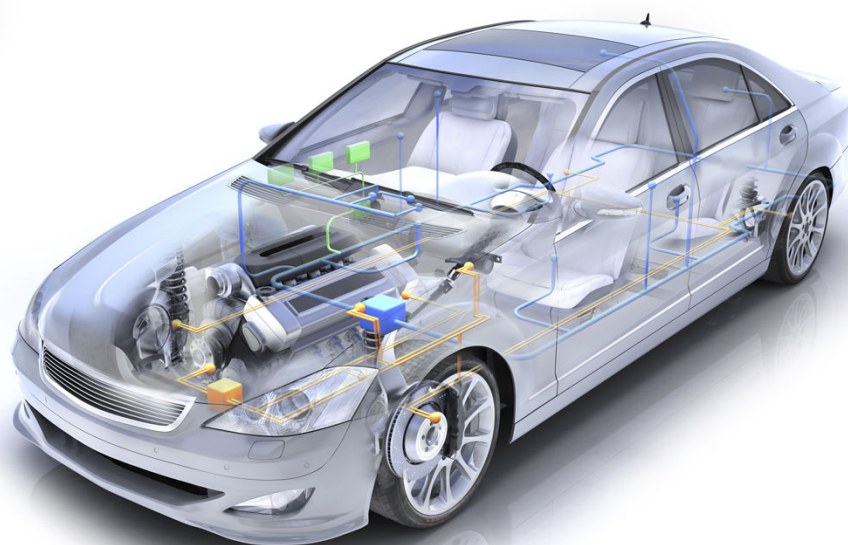
能测，结果能否复现？

一个好的测试 (What is a “Good” Test?)

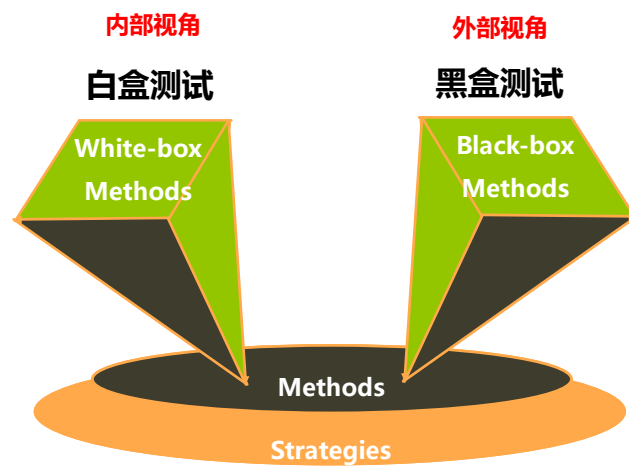
- 好的测试具有较高的**发现错误**的可能性
 - A good test has a high probability of finding an error
- 好的测试是**不冗余**的
 - A good test is not redundant.
- 好的测试应该是 “**最佳品种**”
 - A good test should be “best of breed”
- 好的测试应该即**不简单也不太复杂**
 - A good test should be neither too simple nor too complex



如何设计用例?



软件测试(Software Testing)



Tips

白盒测试

White-Box Testing(without source code)

玻璃盒测试。
一种测试用例设计方法。



黑盒测试

Black-Box Testing(without source code)

侧重于软件的**功能**需求。
黑盒测试倾向于用在测试的**后期阶段**。



Tips

White-Box Testing

白盒测试

Basis Path Testing

基本路径测试

Control Structure Testing

控制结构测试

Condition Testing

条件测试

Data Flow Testing

数据流测试

Loop Testing

循环测试

Black-Box Testing

黑盒测试

Equivalence Partitioning

等价类

Boundary Value Analysis

边界值分析

Cause Elimination

错误推测法

Graph-Based Methods

基于图的方法

因果图/功能图

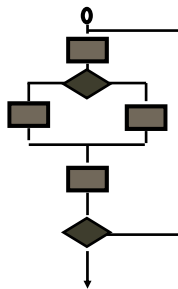
Model-Based Testing

基于模型的测试

白盒测试

■ 把测试对象看做一个**玻璃盒子**，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。



所有路径

玻璃盒子

白盒测试

■ 软件人员使用白盒测试方法，主要想对程序模块进行如下的检查：

- 对程序模块的所有独立的执行路径至少测试一次 — **路径覆盖测试**；
- 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次 — **逻辑覆盖测试**；
- 在循环的边界和运行界限内执行循环体 — **控制流测试**；
- 测试内部数据结构的有效性 — **数据流测试、领域测试**等。

100%

白盒测试将获得100%正确的程序？

白前提：覆盖所有路径

全部路径

100%?



基本路径测试

部分代表性的路径?



基本路径测试

Basis Path Testing

Basis Path Testing (基本路径测试)

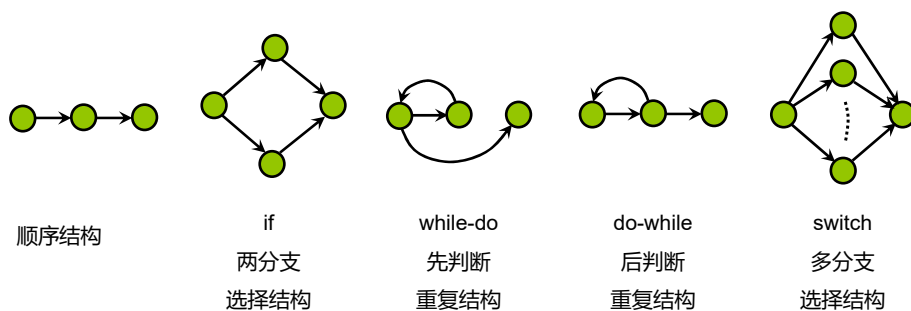
- 基本路径测试方法把覆盖的路径数压缩到**一定限度内**，程序中的循环体**最多只执行一次**。
- 在程序控制流图的基础上：
 - 分析控制构造的环路复杂性
 - 导出基本可执行路径集合
 - 针对每条路径设计测试用例
- 测试用例要保证在测试中程序的**每一个可执行语句至少要执行一次**。

至少一次

所有路径

1. 程序的控制流图

- **符号 ●**：为控制流图的一个结点，表示**一个或多个无分支的PDL语句或源程序语句**。
- **箭头 →**：为边，表示控制流的方向。
 - 在选择或多分支结构中，分支的汇聚处应有一个汇聚结点。



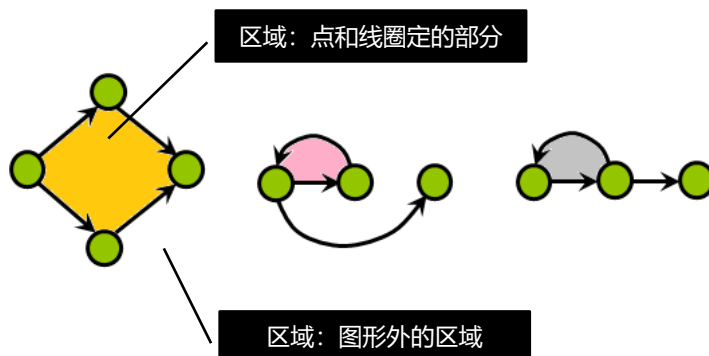
1. 程序的控制流图



图形外的区域也是一个!

■ 边和结点圈定的部分叫做**区域**

- 区域计数时，图形外的区域也应记为一个区域。



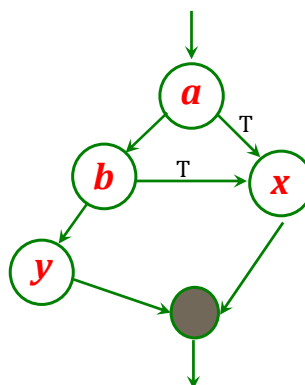
1. 程序的控制流图

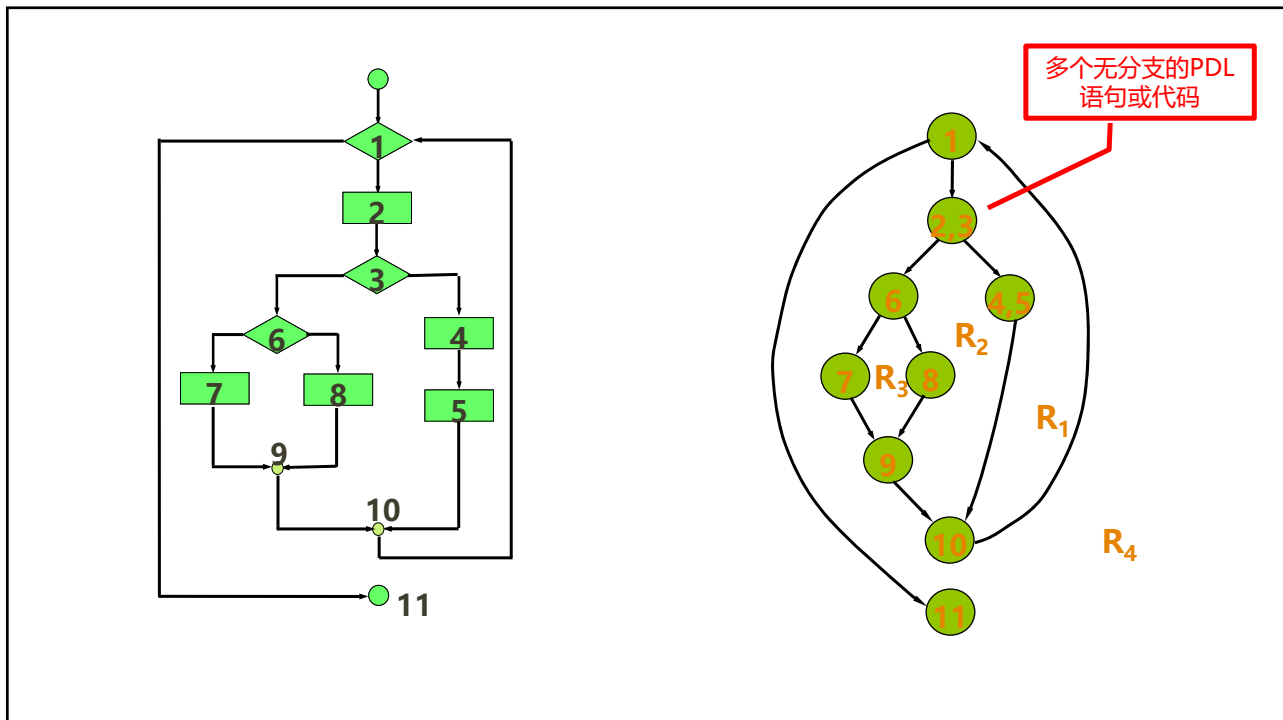
复合要转换!

■ 条件表达式:

- 如果判断中的条件表达式是由一个或多个逻辑运算符(or, and, ...)连接的复合条件表达式，则需改为一组只有单个条件的嵌套的判断。

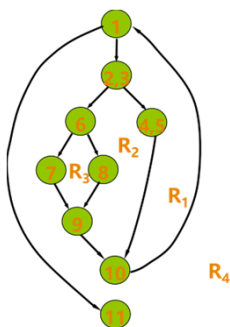
if (a or b)
 then do x
 else do y;





2. 程序环路复杂性

- 程序**环路复杂性**给出了程序基本路径集中的**独立路径条数**，程序中每个可执行语句至少执行一次所必需的**测试用例数目的上界**。



R_4

用例上限

作用

2. 程序环路复杂性

■ 程序**环路复杂性**给出了程序基本路径集中的**独立路径条数**，程序中每个可执行语句至少执行一次所必需的**测试用例数目的上界**。

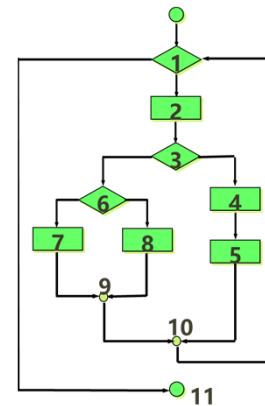
- 一条独立路径是至少包含有一条在其他独立路径中从未有过的边的路径。
- 例如，在图示的控制流图中，一组独立的路径是

path1: 1 - 11

path2: 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11

path3: 1 - 2 - 3 - 6 - 8 - 9 - 10 - 1 - 11

path4: 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11



环路复杂性的计算 三种方法

- 1、流图中的**域的数量**于环复杂性相对应。

- 2、对于流图 G ，环复杂性 $V(G)$ 定义如下：

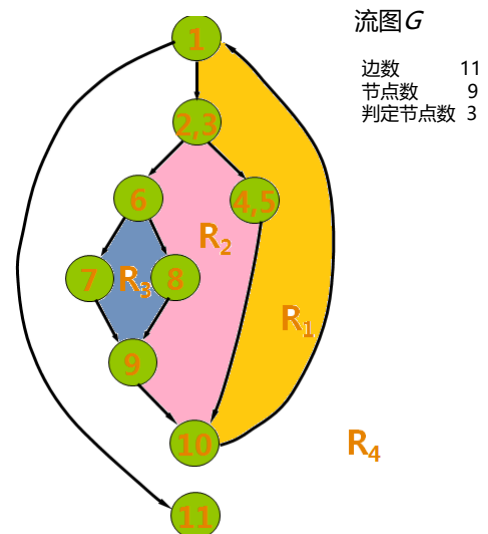
$$V(G) = E - N + 2$$

其中 E 为流图的边数， N 为流图的节点数。

- 3、对于流图 G ，环复杂性 $V(G)$ 定义如下：

$$V(G) = P + 1$$

其中 P 为包含在流图 G 的**判定节点数**。



3. 导出测试用例

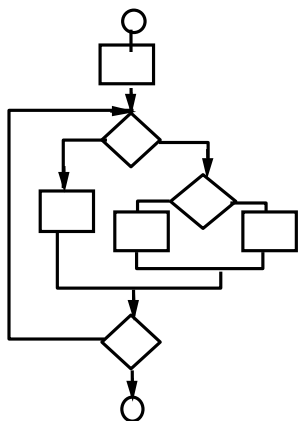
- 路径 path1, path2, path3, path4组成了控制流图的一个**基本路径集**。
 - 导出测试用例，确保基本路径集中的每一条路径的执行。
 - **选择适当的数据**以保证某一条路径可以被测试到 — 用逻辑覆盖的方法。
- 每个测试用例执行之后，与预期结果进行比较。
 - 可以确信程序中所有的可执行语句至少被执行了一次。
 - 一些独立的路径(如例中的path1)，往往不是完全孤立的，有时它是程序正常的控制流的一部分，这时，这些路径的测试可以是另一条路径测试的一部分。

- 例如，在图示的控制流图中，一组独立的路径是
 - path1: 1 - 11
 - path2: 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11
 - path3: 1 - 2 - 3 - 6 - 8 - 9 - 10 - 1 - 11
 - path4: 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11

设计测试用例(Deriving Test Cases)

- Summarizing:
 - Using the design or code as a foundation, draw a corresponding flow graph.
以设计或代码为基础，画出相应的流图
 - Determine the cyclomatic complexity of the resultant flow graph.
确定所得流图的环复杂性
 - Determine a basis set of linearly independent paths.
确定线性独立路径的集合
 - Prepare test cases that will force execution of each path in the basis set.
准备测试用例，强制执行基本集合中的每条路径

Basis Path Testing Notes



- 图不是必须的，但图有利于分析。
- You don't need a flow chart, but the picture will help when you trace program paths.
- 简单的逻辑测试复杂度为1，复杂的逻辑测试复杂度为2或者更多。
 - Count each simple logical test, compound tests count as 2 or more.
- 关键模块应该完成基本路径测试。
 - Basis path testing should be applied to critical(关键) modules.

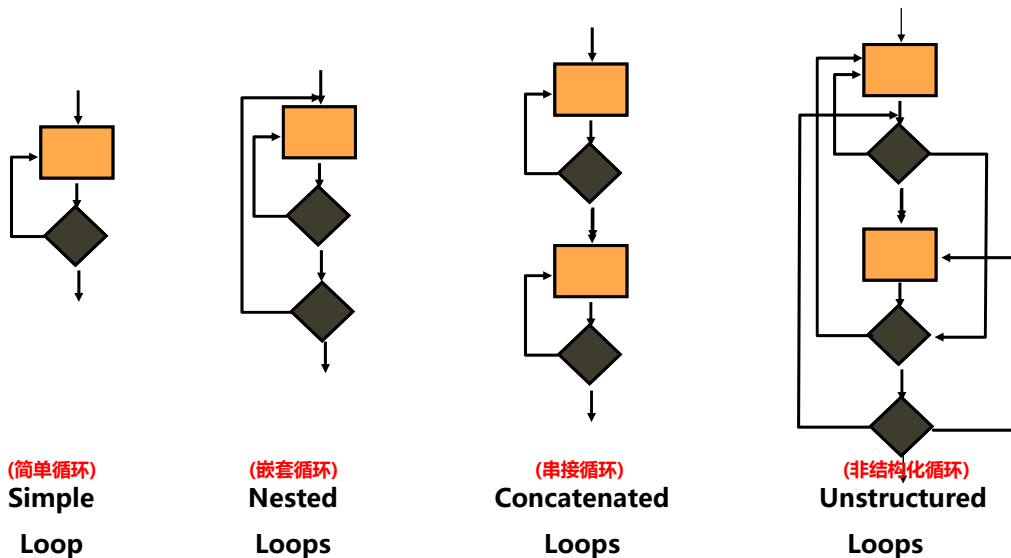
控制结构测试(Control Structure Testing)

- **基本路径测试**是控制结构测试技术之一。

The basis path testing technique described in Section 18.4 is one of a number of techniques for control structure testing.

- **条件测试: Condition Testing**
 - A test case design method that exercises the logical conditions contained in a program module.
- **数据流测试: Data Flow Testing**
- **循环测试: Loop Testing**

循环测试 Loop Testing



循环测试——简单循环 (Loop Testing: Simple Loops)

简单循环 Minimum Conditions— Simple Loops

- 1. Skip the loop entirely
零次循环: 从循环入口到出口(直接跳过)
- 2. Only one pass through the loop
一次循环: 检查循环初始值
- 3. Two passes through the loop
二次循环: 检查多次循环
- 4. m passes through the loop $m < n$
m次循环: 检查在多次循环
- 5. $(n-1)$, n , and $(n+1)$ passes through the loop
最大次数循环、比最大次数**多一次、少一次**的循环。
where n is the maximum number of allowable passes

循环测试——嵌套循环 (Loop Testing: Nested Loops)

■ 嵌套循环 Nested Loops

- Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.
对最内层循环做**简单循环**的全部测试。所有其他层的循环变量置为最小值；
- Test the min+1, typical, max-1 and max for the innermost loop, while holding the outer loops at their minimum values.
逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环的循环变量取最小值，所有其他嵌套内层循环的循环变量取“典型”值。
反复进行，直到所有各层循环测试完毕。
- Move out one loop and set it up as in step 2, holding all other loops at typical values.
Continue this step until the outermost loop has been tested.
对全部各层循环同时取最小循环次数，或者同时取最大循环次数。

Tips

White-Box Testing

白盒测试

Basis Path Testing

基本路径测试

Control Structure Testing

控制结构测试

Condition Testing

条件测试

Data Flow Testing

数据流测试

Loop Testing

循环测试

Black-Box Testing

黑盒测试

Equivalence Partitioning

等价类

Boundary Value Analysis

边界值分析

Cause Elimination

错误推测法

Graph-Based Methods

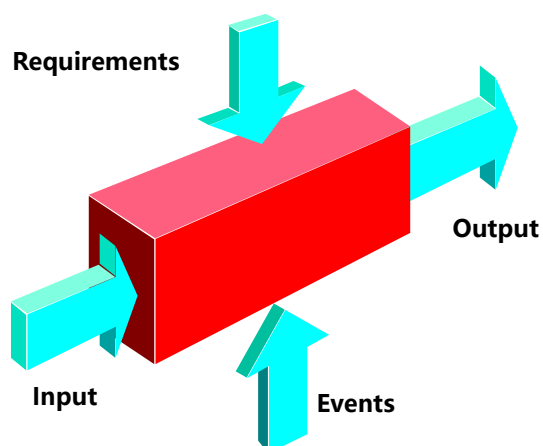
基于图的方法

因果图/功能图

Model-Based Testing

基于模型的测试

黑盒测试 (Black-Box Testing)

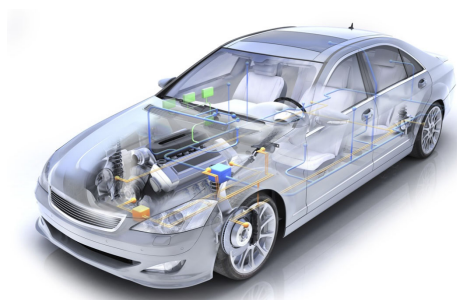


黑盒测试 (Black-Box Testing)

- 测试对象看做一个**黑盒**：
 - 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**需求和功能规格说明**，检查程序的功能是否符合它的功能说明。



依据：需求和功能规格说明



黑盒测试的主要作用

- 黑盒测试方法是在程序接口上进行测试。
 - 又称功能测试或数据驱动测试。
 - 一种特殊的黑盒测试叫做接口测试。
 - 不管程序的需求和实现细节，仅依据程序与其外部环境的接口来选择测试数据。
- 主要是为了发现以下错误：
 - 是否有不正确或遗漏了的**功能**？
 - 在接口上，输入能否正确地接受？能否输出正确的**结果**？
 - 是否有数据结构**错误**或外部信息(例如数据文件)访问错误？
 - **性能**上是否能够满足要求？
 - 是否有初始化或终止性错误？

Think

所有的可能输入数据？

用黑盒测试发现程序错误，必须在所有可能的输入条件和输出条件中确定测试数据，检查程序能否产生正确的输出。



选取少数代表性的用例

每一部分中选取少数代表性的数据做为测试用例

等价类

一种典型的黑盒测试方法

等价类



依据程序的规格说明!

■ 一种**典型的黑盒测试方法**

- 完全不考虑程序的内部结构，只**依据程序的规格说明**来设计测试用例。

■ 代表性的数据

- 即程序的输入域划分成若干部分，然后从**每一部分中选取少数代表性的数据做为测试用例**。
- 设计测试用例要经历划分等价类(列出等价类表)和选取测试用例两步。

等价类划分 (Equivalence Partitioning)

■ 等价类的划分有两种不同的情况：

- ① **有效等价类**：
对于程序规格说明来说，是合理的，有意义的输入数据构成的集合。
- ② **无效等价类**：
对于程序规格说明来说，是不合理的，无意义的输入数据构成的集合。

划分等价类等价类的原则

原则

- **有效等价类 + 无效等价类**
- 如果输入条件规定了取值范围，或值的个数，则可确立一个有效等价类和两个无效等价类。
 - 如，在程序规格说明中对输入条件有一句话：“..... 项数可以从1到999”
 - 则有效等价类是 “ $1 \leq \text{项数} \leq 999$ ”，两个无效等价类是 “项数 < 1” 或 “项数 > 999”。



日期检查进行等价类划分



给出测试用例才算结束!

1) 等价类划分

- 假设日期限定在1990年1月有~2049年12月，并规定日期由6位数字字符组成，前4位表示年，后2位表示月。

输入等价类	有效等价类	无效等价类
日期的类型及长度	①6 位数字字符 19900101	②有非数字字符 ③少于 6 位数字字符 ④多于 6 位数字字符
年份范围	⑤在 1990~2049 之间 1991	⑥小于 1990 ⑦大于 2049 2050
月份范围	⑧在 01~12 之间 01	⑨等于 00 ⑩大于 12 13

日期检查进行等价类划分

2) 设计测试用例

- 以便覆盖所有等价类在表中列出了3个有效等价类，编号分别为①⑤⑧，设计的测试用例如下

● 测试数据	期望结果	覆盖的有效等价类
200211	输入有效	①⑤⑧
● 测试数据	期望结果	覆盖的无效等价类
95June	无效输入	②
20036	无效输入	③
2001006	无效输入	④
198912	无效输入	⑥
205001	无效输入	⑦
200100	无效输入	⑨
200113	无效输入	⑩

用例！

对等价类划分方法的补充

边界值分析

边界值分析 (Boundary Value Analysis)

- 一种黑盒测试方法，是对等价类划分方法的补充。

- 大量的错误是发生在输入或输出范围的边界上，而不是在输入范围的内部。
- 针对各种边界情况设计测试用例，可以查出更多的错误。

- 例如，有一段用C编写的小程序：

```
int A[20]; int i;  
for ( i = 1; i <= 10; i++ )  
    A[i] = -1;
```

- 因为C语言中数组下标从 0 开始，而本程序中从1开始赋值，如果以后用户不了解，可能从0开始使用，就会出错。
- 所以边界值可能查出更多的问题来。

边界值域 选取原则

- 通常的边界检查原则：

- 类型：数字、字符、位置、质量、大小、速度、方位、尺寸、空间等。
- 边界值：最大/最小、首位/末位、上/下、最大/最小、最快/最慢、最高/最低、最短/最长、空/满等。

- 确定正确的边界值域。

- 对于输入/输出等价类，选取正好等于、刚刚大于和刚刚小于边界值的数据作为测试数据。

选取测试用例的原则

原则

- (1) 条件规定了**值的范围**，则应取刚刚到达这个范围边界的值，以及刚刚超过这个范围边界的值作为测试输入数据。
 - 例如，某数据的取值范围为-1.0 ~ 1.0，测试数据可取-1.0、1.0，以及-1.1、1.1。
- (2) 条件规定了**值的个数**，则应取最大个数、最小个数、比最大个数多1，比最小个数少1的数作为测试输入数据。
 - 例如，某文件有255个记录，测试数据可取1、255，以及0、256。
- (3) 根据规格说明和每个输出**条件**，使用原则(1)。
 - 例如，研究生录取分数范围84 ~ 150，测试数据可取84、150，以及83、151。
- (4) 根据规格说明和每个输出**条件**，使用原则(2)。
 - 例如，研究生录取人数34人，测试数据可取1、34、以及0、35。
- (5) 给出的输入域或输出域是**有序集合**(如有序表)，则选取集合的第一个元素和最后一个元素作为测试用例。
 - 例如，学生文件的学生记录按学号存放，班上总共30人，测试数据可取第1、第30个学生。
- 程序中使用了**一个内部数据结构**，则应选择此数据结构的边界上的值作为测试用例。

Summary

White-Box Testing

白盒测试

Basis Path Testing

基本路径测试

Control Structure Testing

控制结构测试

Condition Testing

条件测试

Data Flow Testing

数据流测试

Loop Testing

循环测试

Black-Box Testing

黑盒测试

Equivalence Partitioning

等价类

Boundary Value Analysis

边界值分析

Cause Elimination

错误推测法

Graph-Based Methods

基于图的方法

因果图/功能图

Model-Based Testing

基于模型的测试

提醒：测试用例！

各种方法最终要得到一个中间结果——用例！

Question?



Q&A?



| College of Computer Science, Chongqing University |

Software Engineering

A Practitioner's Approach Seventh Edition

18 测试传统的应用软件

zmqmail@cqu.edu.cn 13708390417

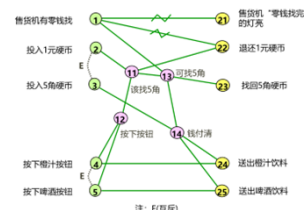


其它的一些测试方法

错误推测/基于图/判定图/etc.

状态图(功能图)测试

- 一种黑盒测试方法
- 基于**状态迁移图**和**判定表**来设计测试用例的。



- **状态迁移图**是一种动态说明
 - 由输入条件和当前状态决定输出数据和后续状态。
- **判定表**则是一种静态说明
 - 用于表示在状态中输入与输出的对应关系。
 - 在这种环境下，测试用例是由测试中经过的一系列状态和在每个状态中必须依靠输入/输出数据满足的一对条件组成。

[illegible]

错误推测法

- 基本想法：利用不同测试阶段的**经验**和对软件系统的**认识**来设计测试用例。**列举**出程序中所有可能有的错误和容易发生错误的特殊情况选择测试用例。

- 靠经验和直觉推测程序中可能存在的各种错误，有针对性地编写检查这些错误的例子。
- 例如：
 - 在单元测试中某程序模块已经遇到错误，在系统测试中可以在这些可能出现问题的地方再组织测试用例。
 - 在前一个版本中发现的常见错误在下一个版本的测试中有针对性地设计测试用例。

- 根据以上想法，测试用例的设计原则：

- 客观因素：产品以前版本已出现的问题。
- 已经因素：语言、操作系统、浏览器的限制可能带来的问题。
- 经验：由模块之间关联所联想到的测试；由修复软件的错误可能会带来的问题。

经验值

基于图的测试方法(本科教学版未涉及)

■ 创建描述重要的程序对象(模块或语句集)及其相互关系的图, 然后导出测试序列以检查对象及其关系并发现错误。

- 图中用**结点**表示**对象**, 用**边**(或连接)表示对象间的**关系**, 用结点权值表示结点的属性, 用边上的权值表示连接的特征。

- 基本路径测试方法就有这样的图。

■ 使用图进行行为测试的方法有:

- **事务流建模**: 结点是事务的每一步骤, 边是步骤之间的逻辑连接。
- **有限状态建模**: 结点是用户可见的软件的不同状态, 边是状态之间的转换。
 - 可利用状态迁移图辅助建立这种图
- **数据流建模**: 结点是数据对象, 边是将数据对象转换为其他数据对象时发生的变换。
- **时序建模**: 结点是程序对象, 边是对象间的顺序连接。边上权值用于表示执行时间。

基于图的测试方法(本科教学版未涉及)

■ 仿照基本路径测试的方法设计测试用例。

- 图中可能有环, 可能要考虑循环测试。
- 对于图中的传递关系、对称(双向的有向边)关系、自反关系也需要进行检查。

- 第一个目标是**结点的覆盖度**。
 - 必须确保不遗漏某个结点, 而且结点的权值(对象属性)是正确的。

- 第二个目标是**边的覆盖度**。
 - 要设计测试以证实权值是否有效, 最后加入循环测试。

因果图

因果图->判定表

■ 适用范围

- 测试时必须考虑输入条件的各种组合。
- **因果图**方法最终**生成**的就是**判定表**。
- 适合于检查程序输入条件的各种组合情况

因果图

■ 用因果图生成测试用例的基本步骤

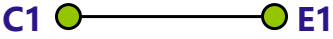

- 哪些是原因(即输入条件或输入条件的等价类), 哪些是结果(即输出条件), 并给每个原因和结果赋予一个标识符。
- 找出原因与结果之间对应的关系? 根据这些关系, 画出因果图。
 - 有些原因与原因之间, 结果与结果之间的组合情况不可能出现, 图上用一些记号标明约束或限制条件。
- **因果图转换成判定表**。
 - 把判定表的每一列拿出来作为依据, 设计测试用例。

因果图

在因果图中出现的基本符号

- 通常在因果图中用 C_i 表示原因，用 E_i 表示结果，各结点表示状态，可取值“0”或“1”。
 - “0”表示某状态不出现
 - “1”表示某状态出现

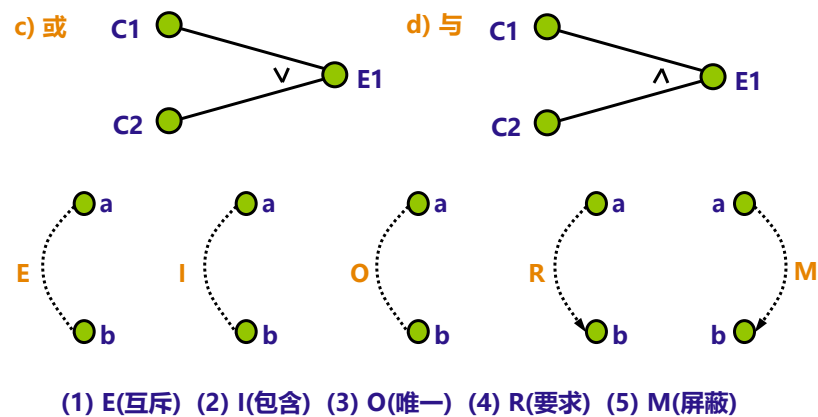
主要的原因和结果之间的关系有：

- 恒等 
- 非 

因果图

表示约束条件的符号

- 表示原因与原因之间，结果与结果之间可能存在的约束条件，图中可附加一些表示约束条件的符号。

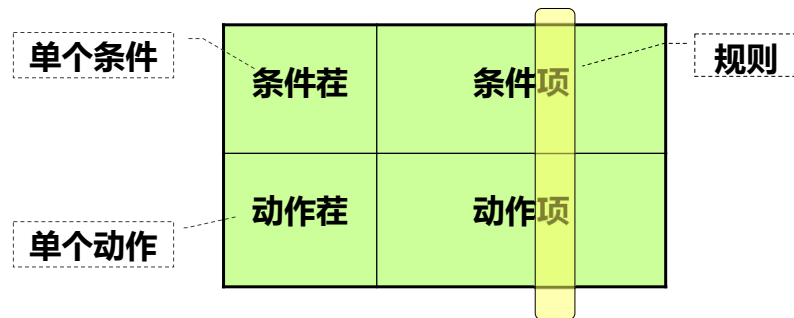


因果图转换成判定表

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
条件	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	2	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	3	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0
	4	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
	5	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
中间条件	11	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
	12	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
	13	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	14	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0
结果	21	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	22	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
	23	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	24	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	25	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
测试用例		√	√	√	√	√	√	√	√		√	√	√	√	√	√	√	√	

判定表

- 如果数据流图的加工需要依赖于多个逻辑条件的取值，使用判定表比较合适。
- DFD(数据流图)中每个加工给予说明的辅助资料。



商店业务处理系统中“检查发货单”

```

if (发货单金额超过$500) then
    if 欠款超过了60天 then
        在偿还欠款前不予批准
    else (欠款未超期)
        发批准书，发货单
else (发货单金额未超过$500)
    if 欠款超过60天 then
        发批准书，发货单及赊欠报告
    else (欠款未超期)
        发批准书，发货单
  
```


		1	2	3	4
条件	发货单金额	> \$500	> \$500	≤ \$500	≤ \$500
	赊欠情况	> 60天	≤60天	> 60天	≤60天
操作	在偿还欠款前不予批准	√			
	发出批准书		√	√	√
	发出发货单		√	√	√
	发出赊欠报告			√	

基于模型的测试(Model-Based Testing)

■ 基于模型的测试(Model-based testing, MBT) 是一种黑盒测试技术

- 1) 分析软件的已有行为模型或者创建一个行为模型。
 - 软件如何响应外部事件或刺激?
- 2) 遍历行为模型, 并标明促使软件在状态之间进行转换的输入。
 - 输入将触发事件, 使转换发生。
- 3) 评估行为模型, 并标注当软件在状态之间转换时所期望的输出。
- 4) 运行测试用例。
 - 手工或者测试脚本、或者测试工具执行测试。
- 5) 比较实际结果和期望结果, 并根据需要进行调整。

行为模型

| College of Computer Science, Chongqing University |

Software Engineering

A Practitioner's Approach Seventh Edition

18 测试传统的应用软件

zmqmail@cqu.edu.cn 13708390417

