软件工程基础 —— 第18章 测试传统的应用软件

计算机学院 孟宇龙

目录

关键概念

- 基本路径测试
- 黑盒测试
- 边界值分析
- 控制结构测试
- 环路复杂性
- 等价类划分
- 流图
- 方法
- 基于模型的测试
- 白盒测试

测试程序的输入输出域，以发现程序功能、行为和性能方面的错误

测试要求开发者抛弃：“刚开发的软件是正确的”这一先入为主的概念

18.1 软件测试基础

软件工程师在设计与实现基于计算机的系统或产品时，应该想着可测试性。

软件的可测试性是能够被测试的容易程度。可测试的软件应该具有下述特征：

- 可操作性——有效地操作
- 可观察性——每个测试用例的结果都是易观察的
- 可控制性——测试能被自动化执行和优化的程度
- 可分解性——有针对的测试
- 简单性——减少复杂的体系结构和逻辑以简化测试
- 稳定性——测试过程需求变更不经常发生
- 易理解性——对设计的较好理解

什么是“好”的测试

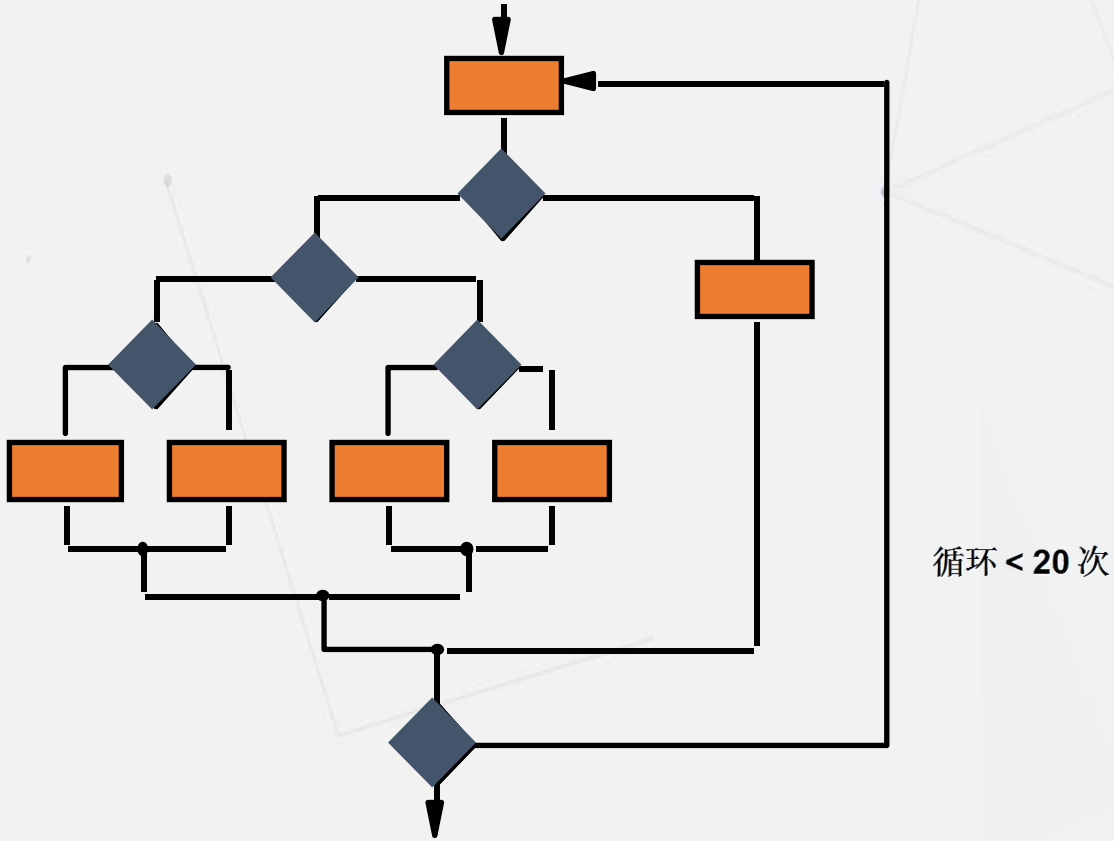
- 好的测试有较高的发现错误的可能性
- 好的测试是不冗余的
- 好的测试应该是“最佳品种”
- 好的测试应该既不简单也不太复杂

18.2 测试的内部视角和外部视角

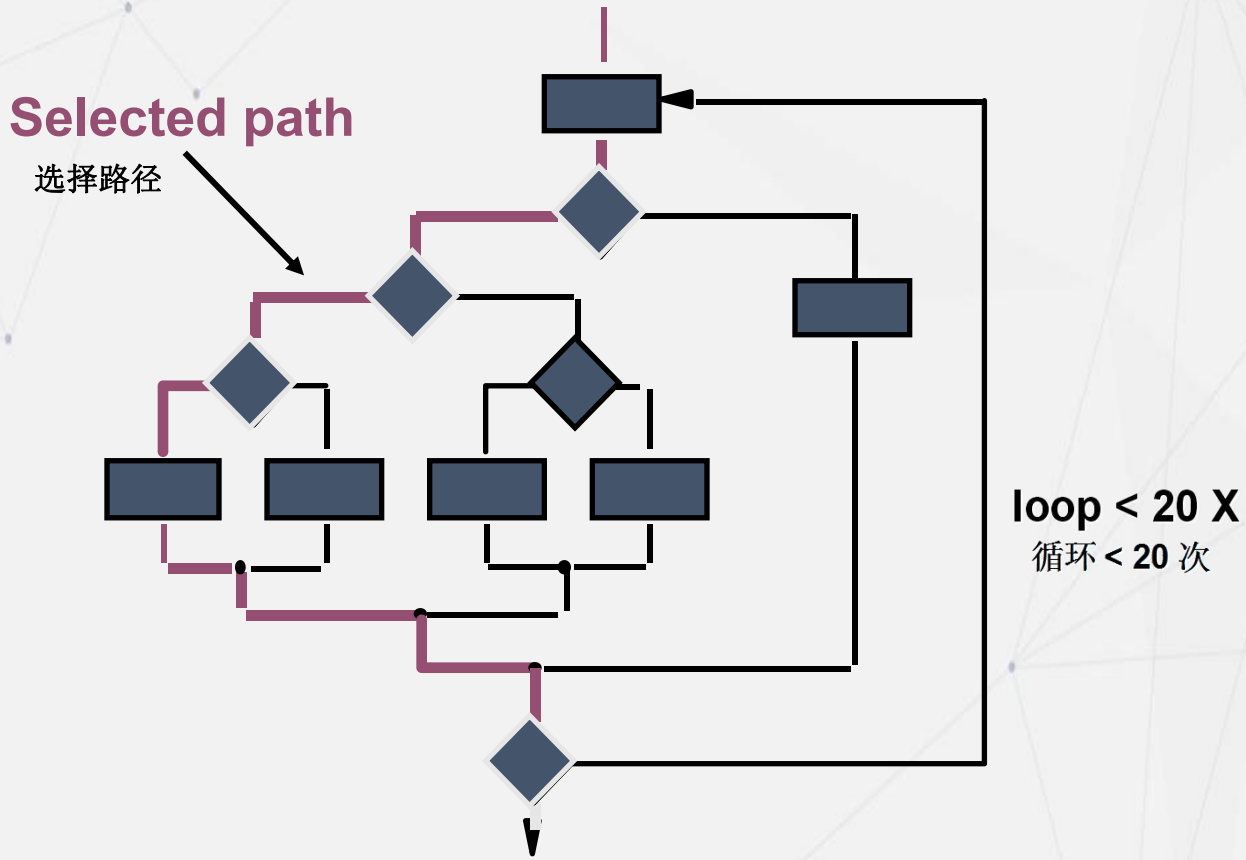
任何工程化的产品（以及大多数其他东西）都可以采用以下两种方式之一进行测试：

1. 内部视角：了解产品的内部工作情况，可以执行测试以确保“所有的齿轮吻合”——即内部操作依据规格说明执行，而且对所有的内部结构已进行了充分测试。（白盒测试）
2. 外部视角：了解已设计的产品要完成的指定功能，可以执行测试以显示每个功能是可操作的，同时，查找在每个功能中的错误；（黑盒测试）

测试用例设计

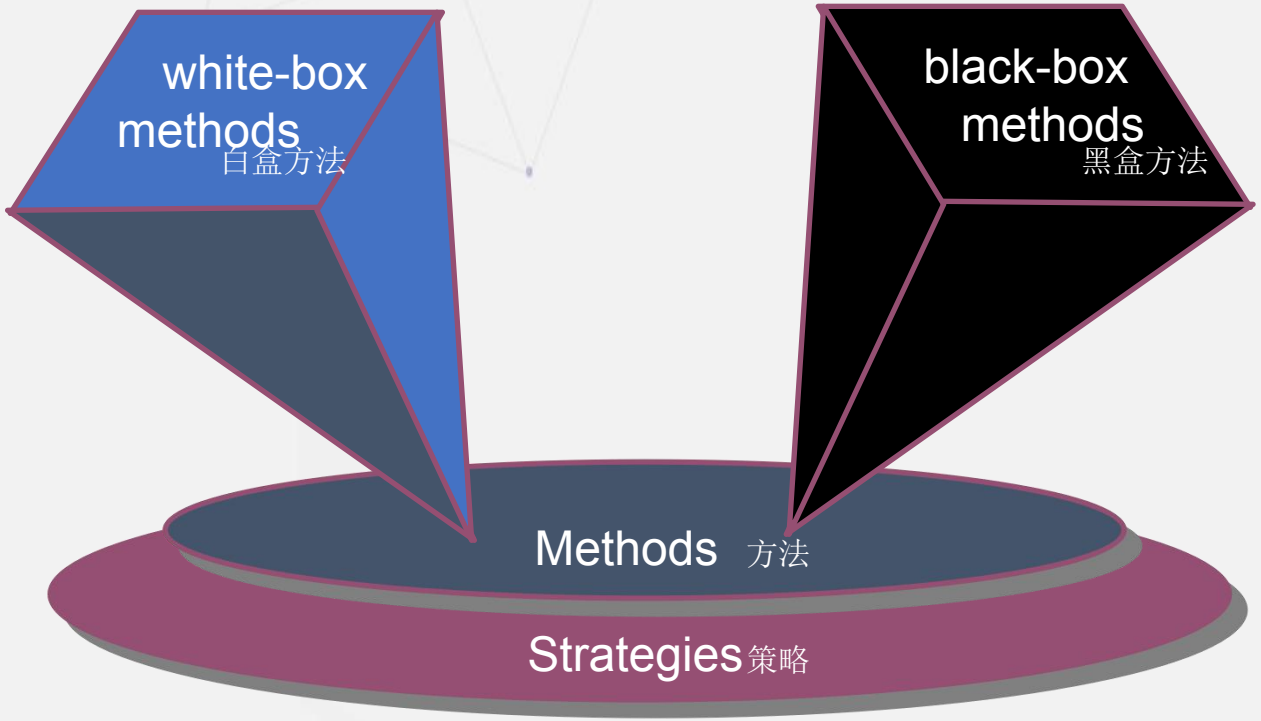


穷举设计

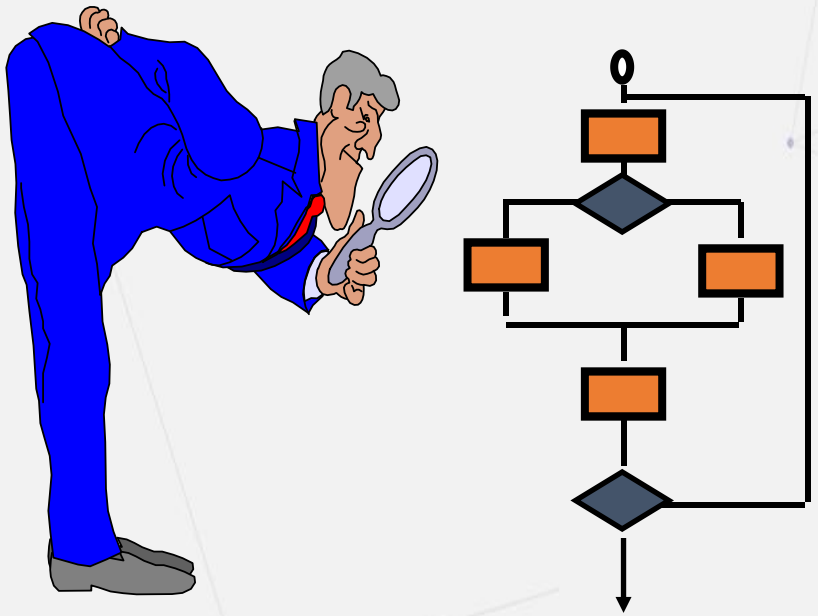


选择测试

软件测试


white-box
methods
白盒方法
black-box
methods
黑盒方法
Methods 方法
Strategies 策略

18.3 白盒测试



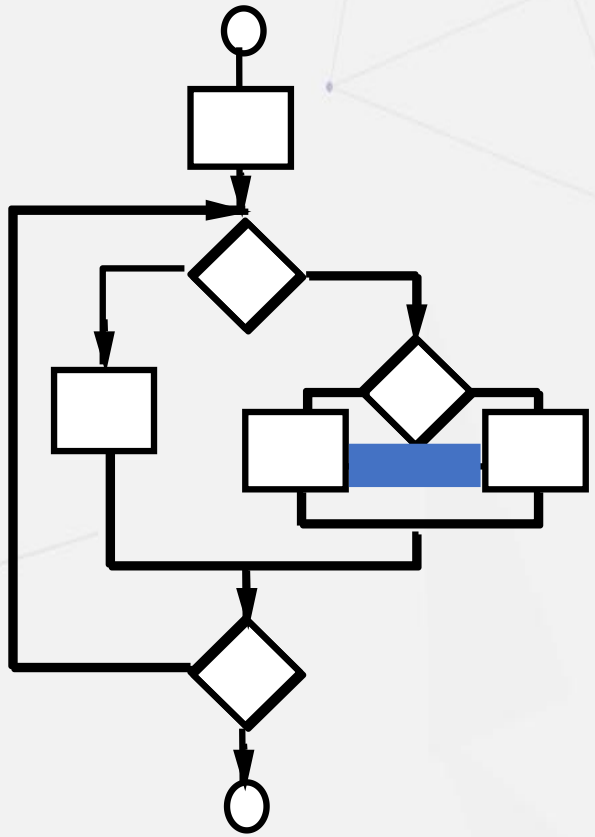
我们的目标是

1. 确保程序中的每一条语句和条件都至少被执行一次...
2. 对所有逻辑判定取真假判定
3. 在上下边界范围内执行所有循环
4. 检验内部数据结构以确保其有效性

为什么要覆盖？

- 逻辑错误和错误的假定与路径的可能执行是成反比的
- 我们总是认为那条路径不可能被执行；事实上，现实总是与直觉相反
- 印刷错误是随机的；很可能包含一些未经测试的路径

18.4 基本路径测试



首先，我们计算出环复杂性：

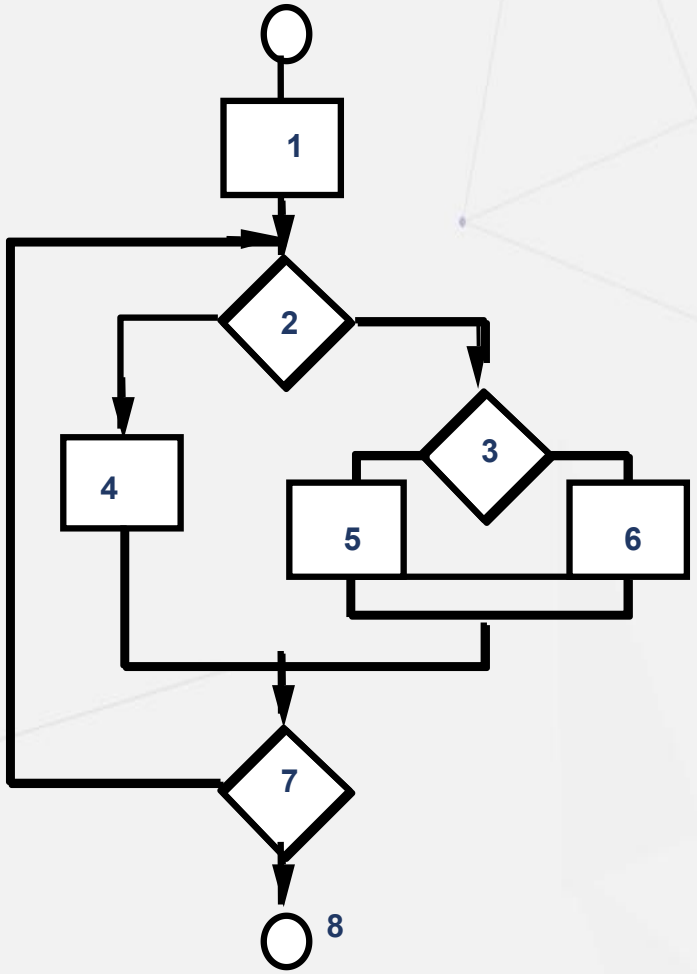
简单决策数+ 1

或

封闭区域数+ 1

在这个例子中， V(G) = 4

18.4 基本路径测试



下一步，我们导出独立路径

因为 **V(G) = 4,**

该值定义了程序基本集合中的独立路径数，并,提供了保证所有语句至少执行一次所需的测试数量的上限。

有4条路径

路径1: 1,2,3,6,7,8

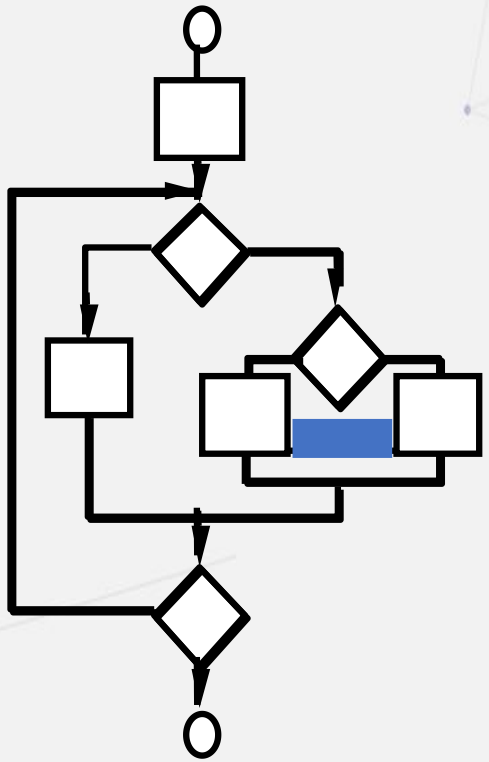
路径2: 1,2,3,5,7,8

路径3: 1,2,4,7,8

路径4: 1,2,4,7,2,4,...7,8

最后，我们导出测试用例执行这些路径

18.4 基本路径测试



- 我们不需要画流图，但是，当追踪程序路径时这幅图能帮助我们
- 计算每一个简单逻辑测试，当有两个或更多时，复合测试计算
- 基本路径测试应应用到关键模型

导出测试用例

- 总结
  - 以设计或源码为基础，画出相应的流图。
  - 确定所得流图的环复杂性。
  - 确定线性独立路径的基本集合。
  - 准备测试用例，强制执行基本集合中的每条路径。

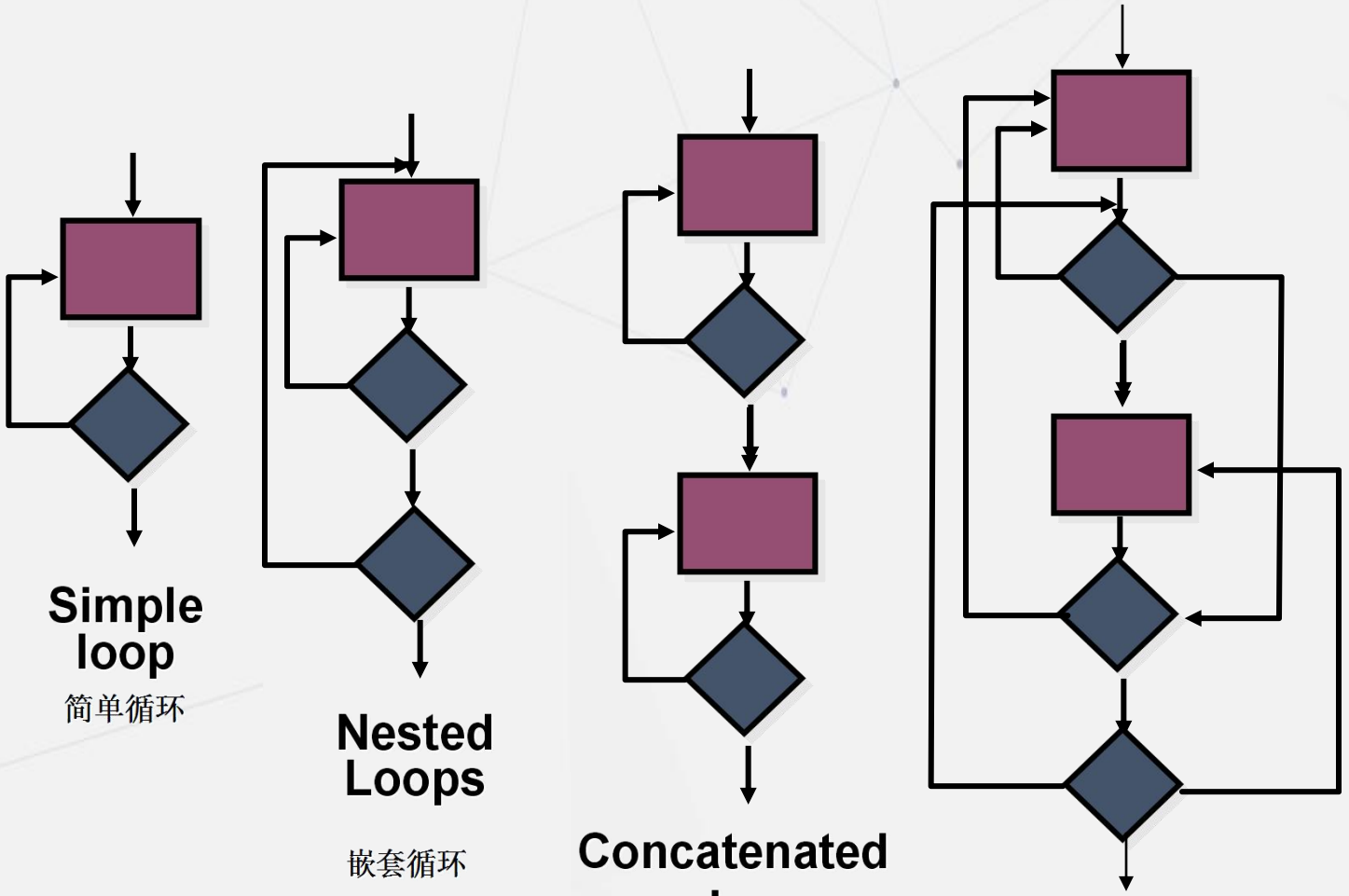
18.5 控制结构测试

- 基本路径测试简单、高效，但其本身并不充分。本节的控制结构测试的变体，这些技术拓宽了测试的覆盖率并提高了白盒测试的质量。
- 条件测试——通过检查程序模块中包含的逻辑每个逻辑条件进行测试用例设计的方法
- 数据流测试——根据变量的定义和使用位置来选择程序测试路径
- 循环测试——完全侧重于循环构成的有效性

数据流测试

- 数据流测试方法[Fra93] 就是根据变量的定义和使用位置来选择程序测试路径。
  - 假设程序的每条语句都赋予了独特的语句号，而且每个函数都不改变其参数或全局参量。对于语句号为S的语句
    - DEF(S) = {X丨语句S 包含X的定义}
    - USE(S) = {X丨语句S 包含X的使用}
  - 变量X的定义-使用链（或称DU链）的形式为[X，S， S']，其中S 和S'为语句号，X 在DEF(S) 和USE(S')中，且在语句S中定义的X在语句S'中有效。

循环测试


Simple loop
简单循环
Nested Loops
嵌套循环
Concatenated Loops
串接循环
Unstructured Loops
非结构化循环

循环测试：简单循环

1. 跳过整个循环
2. 只有一次通过循环
3. 两次通过循环
4. m次通过循环，其中m < n
5. (n-1), n, 和 (n+1) 次通过循环

其中n是允许通过的最大次数

循环测试：嵌套循环

嵌套循环

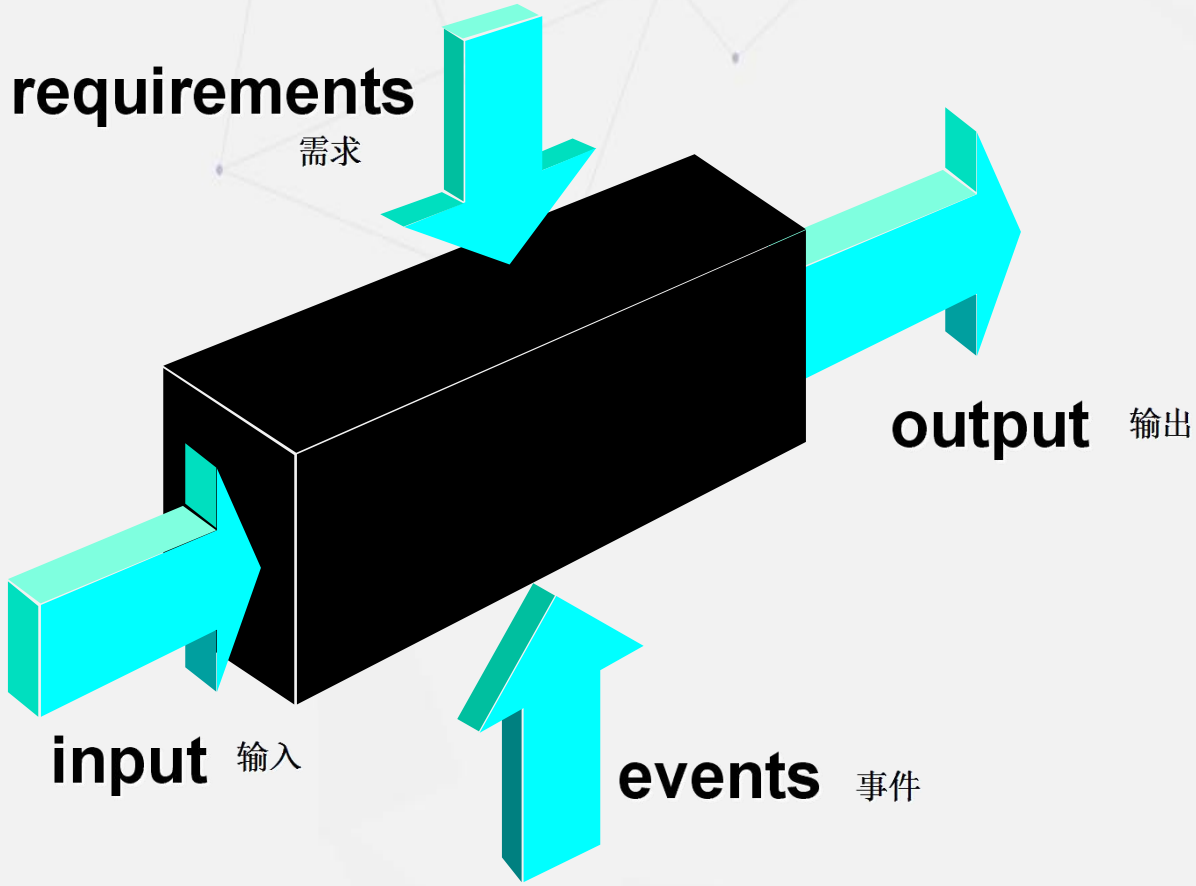
1. 从最内层循环开始，将其他循环设置为最小迭代参量值。
2. 测试min+1、典型的、max1-1和内层循环的最大值，而使外层循环的迭代参数值最小。
3. 由内向外构造下一个循环并按步骤2设置，使所有其他循环为典型值。
4. 继续这一步直至最外层循环被测试

串接循环

If　串接循环的每个循环彼此独立
　then　可以使用简单循环测试方法
　else　使用嵌套循环测试方法
Endif

例如，第1个循环的最后循环计数值为第2个循环的初始值。

18.6 黑盒测试

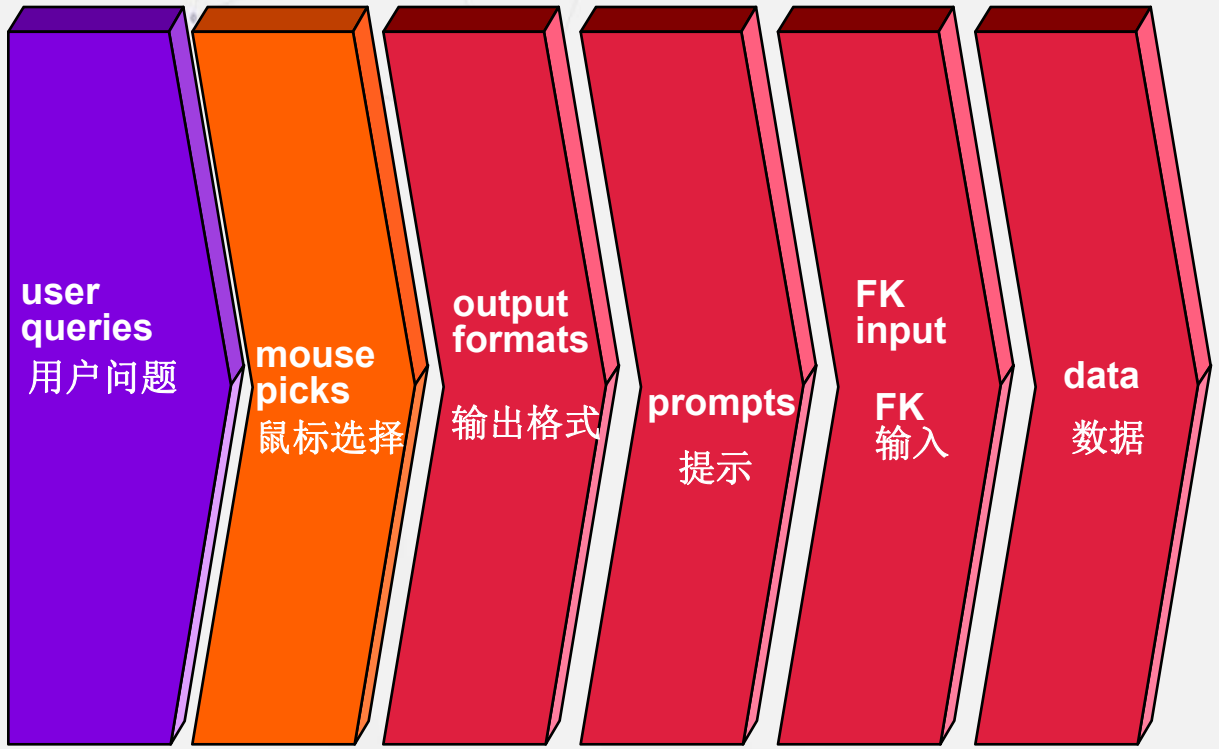


18.6 黑盒测试

- 黑盒测试也称为**行为测试**或**功能测试**，侧重于软件的功能需求。黑盒测试要回答下述问题：

- 如何测试功能的有效性？
- 如何测试系统的行为和性能？
- 哪种类型的输入会产生好的测试用例？
- 系统是否对特定的输入值特别敏感？
- 如何分离数据类的边界？
- 系统能承受什么样的数据速率和数据量？
- 特定类型的数据组合会对系统运行产生什么样的影响？

18.6.1 等价类划分


user
queries
用户问题
mouse
picks
鼠标选择
output
formats
输出格式
prompts
提示
FK
input
FK
输入
data
数据

18.6.1 等价类划分

- **等价类划分法**是一种重要的、常用的黑盒测试方法，它将不能穷举的测试过程进行合理分类，从而保证设计出来的测试用例具有完整性和代表性。
- 等价类划分法是把所有可能的输入数据，即程序的输入域划分为若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。

18.6.1 等价类划分

- 划分等价类可分为两种情况：

（1）有效等价类

——是指对软件规格说明而言，是有意义的、合理的输入数据所组成的集合。利用有效等价类，能够检验程序是否实现了规格说明中预先规定的功能和性能。

（2）无效等价类

——是指对软件规格说明而言，是无意义的、不合理的输入数据所构成的集合。利用无效等价类，可以鉴别程序异常处理的情况，检查被测对象的功能和性能的实现是否有不符合规格说明要求的地方。

18.6.1 等价类划分

- 划分等价类原则：

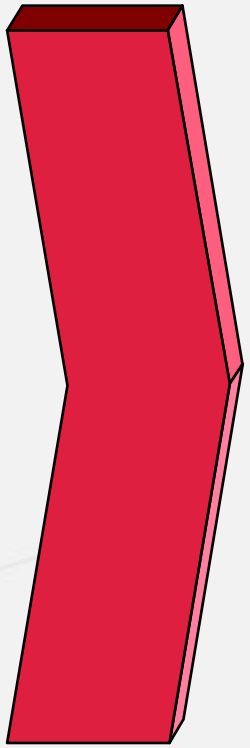
（1）输入条件有范围

（2）输入条件需要特定的值

（3）输入条件指定集合的某个元素

（4）输入条件为布尔值

等价类样本



有效数据

用户提供命令

响应系统提示符
文件名
计算数据

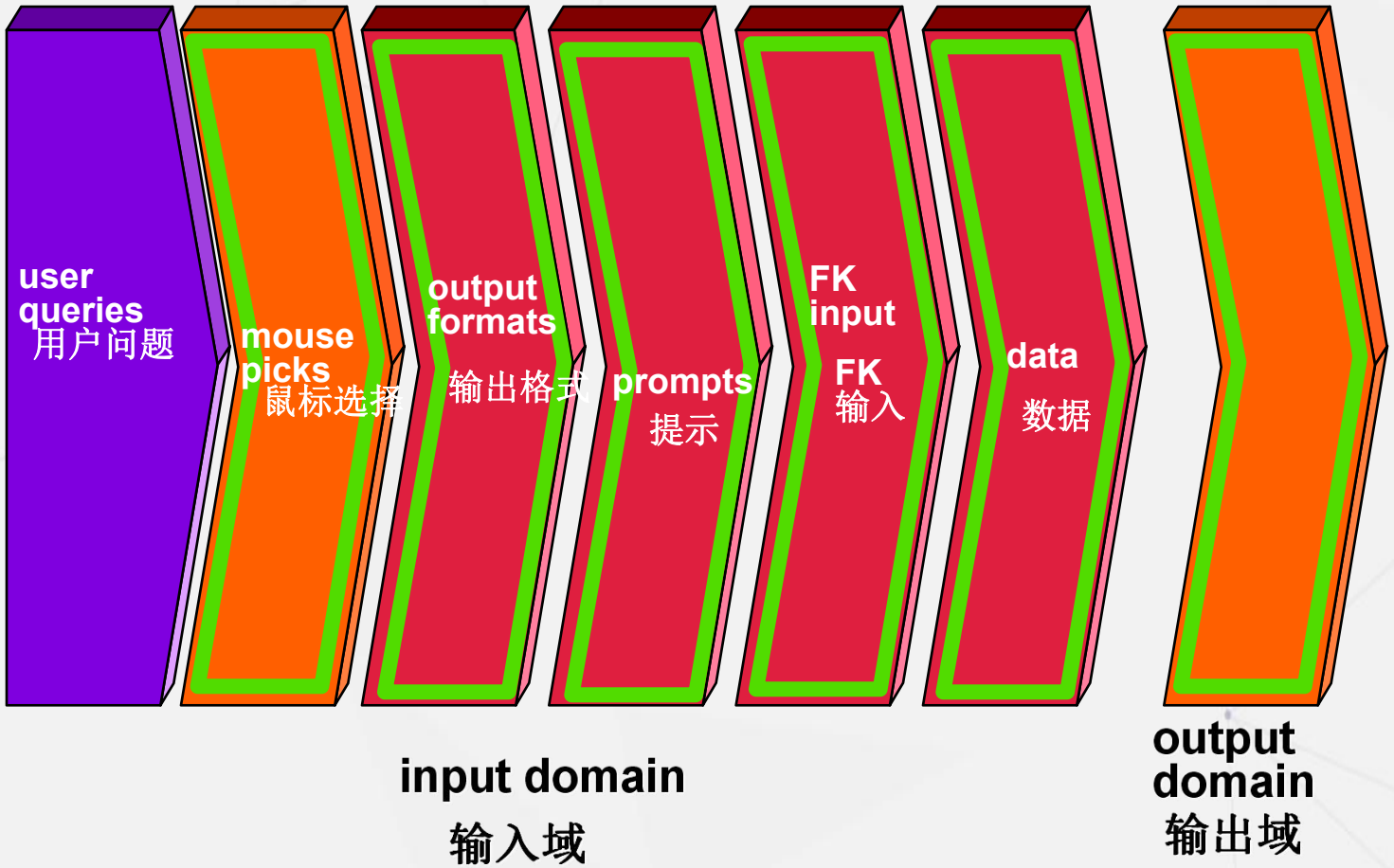
物理参数
边界值
初始值

输出数据格式
响应错误消息
图解数据（例如，鼠标选择）

无效数据

程序边界范围外的数据
物理上不可能的数据
在错误的地点提供适当的值

边界值分析


user
queries
用户问题
mouse
picks
鼠标选择
output
formats
输出格式
prompts
提示
FK
input
FK
输入
data
数据
input domain
输入域
output
domain
输出域

边界值分析

- 边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。**通常边界值分析法是作为对等价类划分法的补充**，这种情况下，其测试用例来自等价类的边界。
- 为什么使用边界值分析法？

  无数的测试实践表明，**大量的故障往往发生在输入定义域或输出值域的边界上**，而不是在其内部。因此，针对各种边界情况设计测试用例，通常会取得很好的测试效果。

边界值分析

- 实例2：NextDate函数的边界值分析测试用例
- 在NextDate函数中，隐含规定了变量mouth和变量day的取值范围为1≤mouth≤12和1≤day≤31 ，并设定变量year的取值范围为1912≤year≤2050 。

边界值分析

测试用例	mouth	day	year	预期输出
Test 1	6	15	1911	1911.6.16
Test2	6	15	1912	1912.6.16
Test3	6	15	1913	1913.6.16
Test4	6	15	1975	1975.6.16
Test5	6	15	2049	2049.6.16
Test6	6	15	2050	2050.6.16
Test7	6	15	2051	2051.6.16
Test8	6	-1	2001	day超出[1...31]
Test9	6	1	2001	2001.6.2
Test10	6	2	2001	2001.6.3
Test11	6	30	2001	2001.7.1
Test12	6	31	2001	输入日期超界
Test13	6	32	2001	day超出[1...31]
Test14	-1	15	2001	Mouth超出[1...12]
Test15	1	15	2001	2001.1.16
Test16	2	15	2001	2001.2.16
Test17	11	15	2001	2001.11.16
Test18	12	15	2001	2001.12.16
Test19	13	15	2001	Mouth超出[1...12]

对比测试

- 仅在这种情况下使用，即软件的可靠性是绝对关键的（例如，与人相关的系统）
  - 独立的软件工程团队使用相同的规格说明独立开发应用系统的版本
  - 可以使用相同的测试数据测试每一个版本，以确保所有的版本有确定的输出

  然后，并行执行所有的版本，实时比较结果，以确保一致性。

基于模型的测试

- 分析软件的已有行为模型或创建一个行为模型。
  - 回忆一下，行为模型指明软件是如何响应外部事件或刺激的。
- 遍历行为模型，并标明促使软件状态之间进行转换的输入。
  - 输入将触发事件，使转换发生。
- 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。
- 运行测试用例。
- 比较实际结果和期望结果，并根据需要进行调整。

软件测试模式

- 测试模式可以采用与设计模式（第12章）同样的描述方式。
- 举例：
  - 模式名称：**场景测试**
  - 摘要：一旦已经执行了单元测试，就需要确定软件是否以让用户满意的方式执行。**场景测试**描述一种从用户的角度测试软件的技术。在这个层次上的失败表明软件不能满足用户的可见需求。[Kan01]