

ISTQB初级认证

第4章 测试设计技术

作者：郑文强

Email: zwqwwuy@163.com

博客: http://blog.csdn.net/Wenqiang_Zheng

声明

→ 本课件的开发基于**ISTQB Foundation Level Syllabus (Version 2007)**。

→ 感谢**ISTQB**和大纲作者的努力，对应的大纲可以从
www.istqb.org下载获得。

→ 本课件为个人开发，只能用于个人学习目的，不能用于任何商业活动。

→ 更多**ISTQB**初级认证资料，参考：
http://blog.csdn.net/Wenqiang_Zheng/archive/2011/04/09/6311523.aspx

课程内容

- 1. 测试开发过程**
- 2. 测试设计技术类型**
- 3. 黑盒测试技术**
- 4. 白盒测试技术**
- 5. 基于经验的技术**
- 6. 测试技术的选择**

测试开发过程

ISTQB考试知识点

- ★ 区别：测试设计规格说明、测试用例规格说明和测试规程规格说明（**K2**）；
- ★ 比较测试术语：测试条件、测试用例和测试规程的不同含义（**K2**）；
- ★ 评估测试用例的质量（**K3**），它们是否满足：
 - ✦ 显示明确的与需求的可追溯性；
 - ✦ 包含预期的结果；
- ★ 根据测试人员的理解水平，将测试用例转换为结构合理的测试规程规格说明（**K3**）；

测试开发过程

容易混淆的术语

- ★ 测试基础(**Test Basis**): 能够从中推断出组件/系统需求的所有文档。测试用例是基于这些文档的;
- ★ 测试条件(**Test Condition**): 组件/系统中能被一个或多个测试用例验证的条目或事件。例如, 功能、事务、特性、质量属性或者结构化元素;
- ★ 测试用例(**Test Case**): 为特定目标或测试条件(例如, 执行特定的程序路径, 或是验证与特定需求的一致性)而制定的一组输入值、执行入口条件、预期结果和执行出口条件[**IEEE 610**];

测试开发过程

容易混淆的术语（续）

- ★ 测试设计规格说明(**Test Design Specification**): 为一个测试条目指定测试条件（覆盖项）、具体测试方法并识别相关高层测试用例的文档；
- ★ 测试用例规格说明(**Test Case Specification**): 为测试项指定一套测试用例（目标、输入、测试动作、期望结果、执行预置条件）的文档[**IEEE829**];
- ★ 测试规程规格说明(**Test Procedure Specification**): 规定了执行测试的一系列行为的文档。也称为测试脚本或手工测试脚本[**IEEE 829**];

测试开发过程

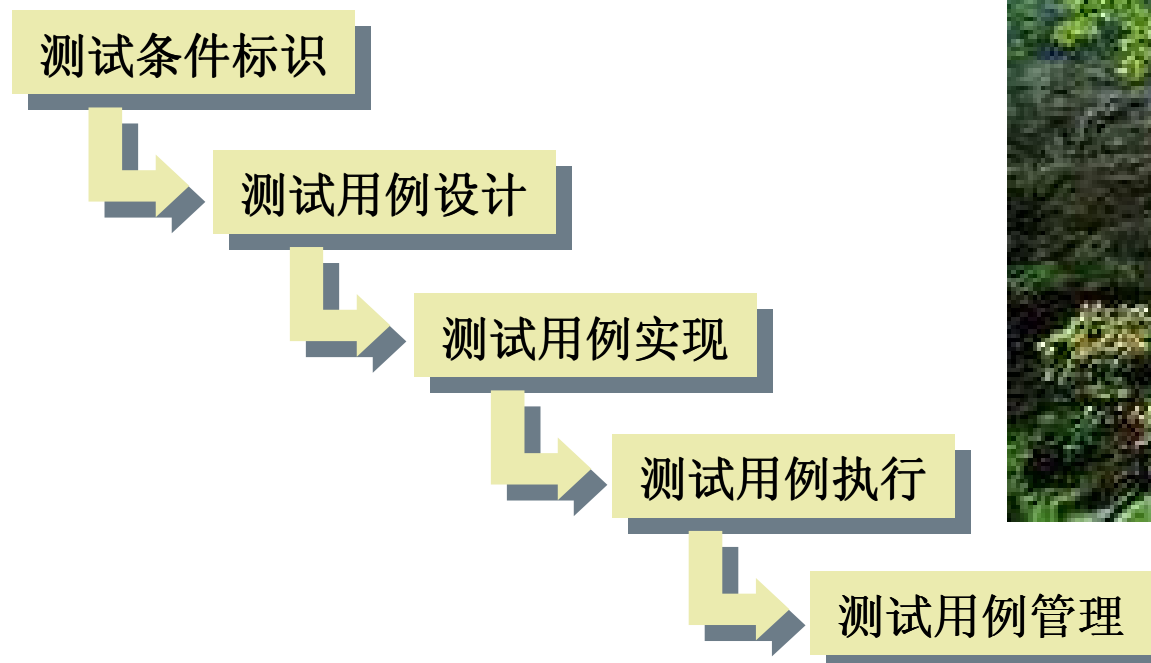
术语例子

员工圣诞节奖金计算的需求描述：

- ★ 员工在公司的工作时间超过**3**年，可以得到相当于其月收入的**50%**的圣诞节奖金；
- ★ 在公司的工作时间超过**5**年，可以得到相当于其月收入的**75%**的圣诞节奖金；
- ★ 工作时间超过**8**年，可以得到相当于其月收入的**100%**的圣诞节奖金。

测试开发过程

测试用例生命周期



测试开发过程

测试用例过程例子

员工圣诞节奖金计算的需求描述：

- ★ 员工在公司的工作时间超过**3**年，可以得到相当于其月收入的**50%**的圣诞节奖金；
- ★ 在公司的工作时间超过**5**年，可以得到相当于其月收入的**75%**的圣诞节奖金；
- ★ 工作时间超过**8**年，可以得到相当于其月收入的**100%**的圣诞节奖金。

测试开发过程

步骤1：测试条件标识

测试条件标识，简单的说，就是确定测试什么？需求文档中的需求条目，我们可以认为是测试条件！



我们测试什么：圣诞节我的奖金是多少？

测试开发过程

步骤2：测试用例设计

白盒测试设计技术

- ★ 语句覆盖
- ★ 分支覆盖
- ★ 条件覆盖
- ★ 路径覆盖
- ★

黑盒测试设计技术

- ★ 等价类划分
- ★ 边界值技术
- ★ 因果图和决策表
- ★ 状态转换测试
- ★

测试用例设计说明如何来识别测试什么？即如何详细识别需求中的测试条件！

测试开发过程

步骤3：测试用例实现

测试用例实现得到的是详细的测试用例步骤！

测试用例详细步骤包括：

- ★测试前置条件
- ★测试输入
- ★测试数据
- ★测试预期结果
- ★测试后置条件
- ★.....

测试开发过程

步骤4：测试用例执行

测试用例执行是通过运行测试用例来对系统进行测试！

自动化测试

手工测试

测试环境搭建

输入测试数据

检查测试输出

比较测试结果

提交缺陷报告

测试开发过程

步骤5：测试用例管理

测试用例管理是如何来组织、跟踪和维护测试用例过程！

测试用例组织

测试用例跟踪

测试用例维护

测试开发过程

测试用例的质量评估

- ★ 测试用例与系统需求（测试条件）之间进行关联，保证需求的可追溯性；
 - ✦ 确定测试覆盖率
 - ✦ 需求变更对测试设计和测试执行的影响
- ★ 测试用例包含明确的测试输出预期结果；
- ★ 测试用例在发现缺陷方面的有效性；
- ★

测试开发过程

测试用例的组织

- ★ 按照软件功能模块组织;
- ★ 按照测试用例的测试类型组织;
- ★ 按照测试用例的优先级组织;
- ★

课程内容

1. 测试开发过程
2. 测试设计技术类型
3. 黑盒测试技术
4. 白盒测试技术
5. 基于经验的技术
6. 测试技术的选择

测试设计技术类型

ISTQB考试知识点

- ★ 复述在测试用例设计中，为什么需要采用基于规格说明的测试（黑盒测试）和基于结构的测试（白盒测试）的方法？列举出各自比较常用的技术（**K1**）；
- ★ 解释基于规格说明的测试、基于结构的测试和基于经验的测试三者的特征和区别（**K2**）；

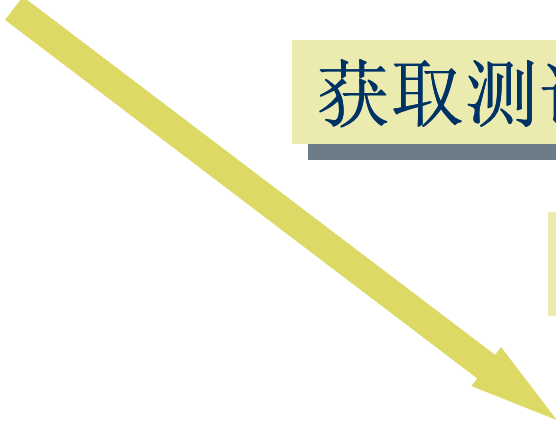
测试设计技术类型

测试设计技术的目的

识别测试条件！

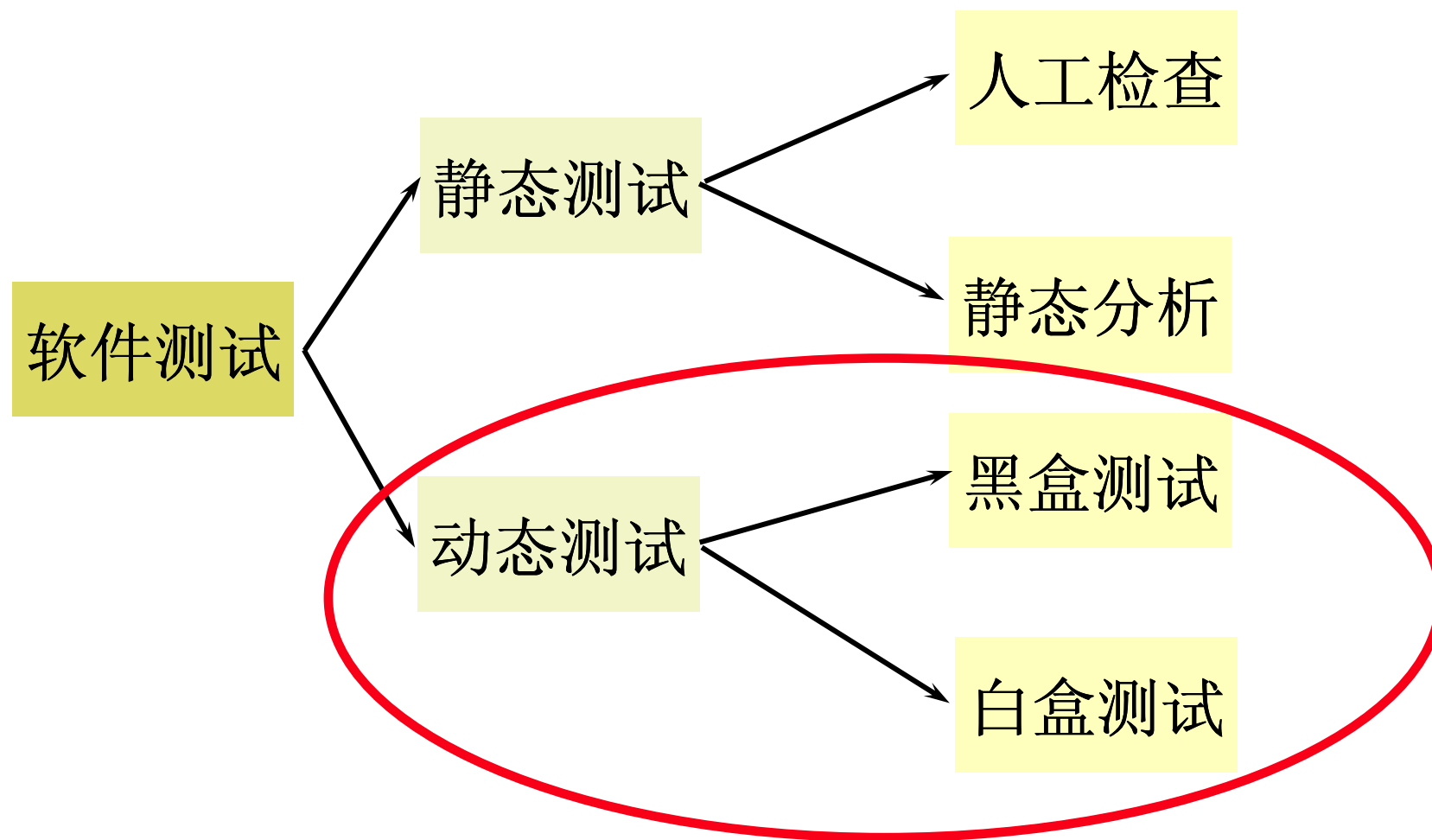
获取测试数据！

设计测试用例！



测试的重要目的是尽早发现尽量多的缺陷，而穷尽测试是不可能的。用尽量少的测试用例，发现尽量多的缺陷→测试设计的主要目的！

测试设计技术类型



测试设计技术类型

黑盒测试技术的定义

- ★ 黑盒测试技术是基于系统功能或非功能规格说明来设计或者选择测试用例的技术，不涉及软件内部结构；
- ★ 黑盒测试技术包括基于规格说明的测试技术和基于经验的测试技术；
- ★ 有的时候，黑盒测试技术又被称为基于规格说明的测试技术；

测试设计技术类型

基于规格说明测试技术的特征

- ★ 使用正式或非正式的模型来描述需要解决的问题、软件或其组件等；
- ★ 根据这些模型，可以系统地导出测试用例；

基于经验的测试技术的特征

- ★ 测试用例根据参与人员的经验和知识来编写；
 - ✦ 测试人员、开发人员、用户和其他的利益相关者对软件、软件使用和环境等方面所掌握的知识；
 - ✦ 对可能存在的缺陷及其分布情况的了解；

测试设计技术类型

黑盒测试技术的分类

- ★ 等价类技术
- ★ 边界值分析
- ★ 决策表测试
- ★ 状态转换测试
- ★ 基于经验的测试

测试设计技术类型

白盒测试技术的定义

- ★ 白盒测试技术是通过分析组件/系统的内部结构来产生和/或选择测试用例的技术；
- ★ 白盒测试技术，有时候又叫做结构化测试技术或者基于结构的测试技术，或者基于代码的测试技术；
- ★ 白盒测试技术需要熟悉源代码和详细的设计文档，并且以它们作为测试用例设计的输入；

测试设计技术类型

白盒测试技术的环境

- ★ 桩模块：一个软件组件框架的实现或特殊目的实现，用于开发和测试另一个调用或依赖于该组件的组件。它代替了被调用的组件[**IEEE 610**];
- ★ 驱动模块：代替某个软件组件来模拟控制和/或调用其他组件或系统的软件或测试工具;

测试设计技术类型

白盒测试技术的特征

- ★ 根据软件的结构信息设计测试用例，比如软件代码和软件设计；
- ★ 可以通过已有的测试用例来测量软件的测试覆盖率，并通过白盒测试技术来系统化的导出设计用例，从而提高覆盖率；

测试设计技术类型

白盒测试技术的分类

- ★ 语句覆盖
- ★ 判定覆盖
- ★ 条件覆盖
- ★ 多重条件覆盖
- ★ 路径覆盖

测试设计技术类型

为什么黑盒测试和白盒测试

- ★ 黑盒测试技术只是观察程序的输入/输出行为。测试对象的功能是我们关注的重点。黑盒测试技术经常应用在级别比较高的测试中，尽管也应用在单元测试中，比如测试优先编程、测试驱动开发等；
- ★ 白盒测试技术考虑了测试对象的内部结构（单元层次、控制流和数据流等）。白盒测试技术一般应用在低级别的测试，比如单元测试和集成测试；

课程内容

1. 测试开发过程
2. 测试设计技术类型
3. 黑盒测试技术
4. 白盒测试技术
5. 基于经验的技术
6. 测试技术的选择

黑盒测试技术

ISTQB考试知识点

- ★ 使用下列技术对软件模块编写测试用例（**K3**）：
 - ✦ □ 等价类划分(**equivalence partitioning**)
 - ✦ □ 边界值分析(**boundary value analysis**)
 - ✦ □ 决策表测试(**decision table testing**)
 - ✦ □ 状态转换测试(**state transition testing**)
- ★ 理解这四种测试设计技术各自的主要目的，这些技术可以应用于什么测试级别和 测试类型，以及如何测量测试覆盖(**test coverage**)（**K2**）；
- ★ 理解用例测试（**use case testing**）的概念和应用这种技术的优点（**K2**）；

等价类划分

等价类划分技术的定义

等价类划分技术把所有可能的输入数据，即软件或者系统的输入域划分成若干部分，然后从每一部分中选取少数有代表性的数据做为测试用例的输入数据！

等价类划分技术可以应用在所有测试级别中！

等价类划分

等价类的含义

等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。测试某等价类的代表值就等价于对这一类其它值的测试！

从测试人员的角度，**等价类**对揭露软件或者系统中的缺陷来说，集合中的每个输入条件是等效的！

等价类划分

等价类的类型

有效等价类

是指对于软件或者系统的规格说明来说，是合理的，有意义的输入数据构成的集合！

无效等价类

是指对于软件或者系统的规格说明来说，是不合理的，无意义的输入数据构成的集合！

在设计测试用例时，要同时考虑有效等价类和无效等价类的设计！

等价类划分

等价类划分的原则1

如果输入条件规定了连续的取值范围，则可以创建一个有效等价类和两个无效等价类；

等价类划分的例子

月工资参数 x 的取值范围是**1000**到**20000**；

★ 有效等价类： **$1000 \leq x \leq 20000$** ；

★ 无效等价类： **$x < 1000$ 或者 $x > 20000$** ；

等价类划分

等价类划分的原则2

假如描述的是允许数值的范围（离散的数字），则可以创建一个有效等价类（所有可能的有效值），和两个无效等价类（少于和多于有效的个数）；

等价类划分的例子

注册的用户名的组成应该是**6至12**位的字符串：

- ★ 有效等价类： **$6 \leq \text{用户名长度} \leq 12$** ；
- ★ 无效等价类：用户名长度大于**12**或者小于**6**；

等价类划分

等价类划分的原则3

假如输入数据规定的是一组值（假定 n 个），并且软件对每一个输入值分别处理的情况下，可确立 n 个有效等价类和一个无效等价类；

等价类划分的例子

根据学历来定岗位工资，学历可为：专科、本科、硕士、博士四种之一。

- ★ 有效等价类：专科、本科、硕士、博士四个等价类；
- ★ 无效等价类：其他的学历；

等价类划分

等价类划分的原则4

假如输入数据规定的是一组值（假定 n 个），并且软件对每个输入值都一样的处理，可确立一个有效等价类和一个无效等价类；

等价类划分的例子

预订飞机票的人可以是：小孩、青少年、成人、学生、残疾人或者退休人员，软件处理方式一样；

★ 有效等价类：列表中的任何一个；

★ 无效等价类：不包含在列表中的其他类型；

等价类划分

等价类划分的原则5

在规定了输入数据必须遵守的规则的情况下,可确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）；

等价类划分的例子

函数变量名必须以字母开头，并且长度为**8**；

★ 有效等价类：字母开头，长度为**8**；

★ 无效等价类：不是以字母开头，或长度不是**8**；

等价类划分

等价类划分的原则6

在输入条件是一个布尔量的情况下,可确定一个有效等价类和一个无效等价类;

等价类划分的例子

在论坛中注册用户的时候, 需要判断是否接受用户协议;

- ★ 有效等价类: 接受协议;
- ★ 无效等价类: 不接受协议;

等价类划分

等价类划分的原则7

在输入条件规定了输入值的集合或者规定了"必须如何"的条件的情況下,可确立一个有效等价类和一个无效等价类;

等价类划分的例子

变量标识必须是以字母开头的字符串;

- ★ 有效等价类: 以字母开头的字符串;
- ★ 无效等价类: 不是以字母开头的其他类型;

等价类划分

等价类划分的原则8

在确知已划分的等价类中各元素在程序处理中的方式不同的情况下,则应再将该等价类进一步的划分为更小的等价类;

等价类划分的例子

比如：在**ATM**取款时，首先依据卡是否有效划分等价类，对于卡有效的情况再进一步划分子等价类，如依据提款金额是否在允许范围内，**ATM**机余额是否充足等再划分等价类；

等价类划分

等价类划分的测试用例

测试对象通常有不只一个输入参数，如何对这些参数等价类进行组合测试，来保证等价类的覆盖率，是我们测试用例设计首先需要考虑的问题！

等价类划分

等价类组合的原则

- ★ 所有有效等价类的代表值都需要集成到测试用例中，即覆盖有效等价类的所有组合。任何一个组合都将设计成一个有效的测试用例，或者叫正面测试用例；
- ★ 无效等价类的代表值只能和其他有效等价类的代表值（随意）进行组合。因此，每个无效等价类将产生一个额外的无效测试用例，或者叫负面测试用例；

等价类划分

等价类组合的数量

- ★ 正面测试用例：即有效等价类的组合数量，是每个参数等价类数目的乘积；
- ★ 负面测试用例：即无效等价类的组合数量，是每个无效等价类数目之和；

等价类的组合，将产生数以百计的测试用例，如何有效的减少测试用例的数目？

等价类划分

等价类测试用例的选择规则

- ★ 由所有代表值组合而成的测试用例按使用频率（典型的使用特征）进行排序，并按照这个序列设置优先级。这样就能仅对相关的测试用例（典型的组合）进行测试；
- ★ 优先考虑包含边界值或者边界值组合的测试用例；
- ★ 将一个等价类的每个代表值和其他等价类的每个代表值进行组合来设计测试用例（即双向组合代替完全组合）；
- ★ 保证满足最小原则：一个等价类的每个代表值至少在一个测试用例中出现；
- ★ 无效等价类的代表值不与其他无效等价类的代表值进行组合；

等价类划分

等价类划分出口准则

测试对象计划测试得越彻底，需要的测试覆盖率就会越高。

在测试执行之前，事先定义的覆盖率作为决定测试活动是否足够的一个标准；在测试执行之后，它又作为判断测试强度是否达到要求的一个指标；

等价类划分

等价类划分的价值

- ★ 等价类划分技术可以显著的减少测试用例的数量。因为它的假设条件是等价类范围的任何一个输入，都结果而言都是一样的；
- ★ 等价类划分技术不仅仅用来确定方法和函数的输入和输出，也可以用来准备中间值、状态、与时间相关的值（如事件之前或之后的值）以及接口参数等；
- ★ 等价类划分技术可以应用于任何测试级别：单元测试、集成测试、系统测试和验收测试；

等价类划分：练习

练习的GUI

| | |
|-----------------------|-------------------------------------|
| Index | <input type="text"/> |
| Actor Admin Key | <input type="text"/> |
| Aggregator Size | <input type="text"/> |
| Aggregator Name | <input type="text"/> |
| LACP State | <input type="text" value="LACP"/> |
| Interface Client Type | <input type="text" value="ETS"/> |
| Admin State | <input type="text" value="Enable"/> |

参数取值范围

- ★ **Index:** [10001,10006];
- ★ **ActorKey:** [1,65535];
- ★ **Aggregator Size:** [1,8];
- ★ **Aggregator Name:** 必须是字母，并且长度不能超过8个；

练习输出

- ★ 输出练习中每个参数的等价类；
- ★ 输出练习的概要测试用例，来覆盖有效等价类和无效等价类；

边界值分析

边界值分析的定义

边界值分析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！

边界值分析方法需要测试边界值以及离边界值最近的上下两个值（等价类里面的值和等价类外面的值）。因此，通常情况下，对于边界值，一般可以得到**3**个测试用例！

边界值分析

边界值分析的理由

- ★ 长期的测试工作经验告诉我们，大量的错误是发生在输入或输出范围的边界值上，而不是发生在输入输出范围的内部。出现这种现象的原因有：
 - ✦ 需求文档或者规格说明中经常没有明确的定义边界值；
 - ✦ 或者编程人员对边界值产生误解；

边界值分析

常见的边界值类型

- ★ 对**16-bit** 的整数而言 **32767** 和 **-32768** 是边界值；
- ★ 屏幕上光标在最左上、最右下位置；
- ★ 报表的第一行和最后一行；

★ 数据元素的第一、二个和最后一个

对于列表形式输入，一般情况下没有边界值。比如前面等价类划分中的“旅游者”的例子中，就不存在输入域的边界值，输入数据类型是不连续的，即**6**个元素组（儿童、青少年、成人学生、残疾人和退休人员）！

边界值分析

边界值的提示1

如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据；

边界值的例子

如果软件的规格说明中规定：“重量在**10**公斤至**50**公斤范围内的邮件，其邮费计算公式为.....”。假如设计基于边界值的测试用例，我们应取**10**及**50**，还应取**10.01**，**49.99**，**9.99**及**50.01**等；

边界值分析

边界值的提示2

如果输入条件规定了值的个数,则用最大个数,最小个数,比最小个数少**1**,比最大个数多**1**的数作为测试数据;

边界值的例子

一个输入文件应包括**1~255**个记录,则测试用例可取**1**和**255**,还应取**0**及**256**等;

边界值分析

边界值的提示3

尽量选择非常庞大的数据结构、列表和表格等作为边界值分析的数据，比如，那些能使内存溢出、文件和数据存储到达边界的数据，来检查测试对象在这种极端情形下的行为；

边界值的提示4

对于列表和表格，空列表和满列表以及列表的第一个元素和最后一个元素都是应该作为分析的对象，因为测试它们常常可以发现由于编程错误而导致的失效；

边界值分析

边界值分析步骤

- ★ 边界值分析使用与等价类划分法相同的划分，只是边界值分析假定错误更多地存在于划分的边界上，因此在等价类的边界上以及两侧的情况设计测试用例；
- ★ 将软件的输入或者输出参数进行等价类划分；
- ★ 在等价类的基础之上进行边界值分析。一般情况下，假如边界值已经由等价类划分覆盖，则可以不予考虑；
- ★ 将边界值进行组合，作为测试用例的输入数据；

边界值分析

边界值分析覆盖率

= (执行的边界值数量 / 总的边界值数量)
×100%

边界值分析

边界值分析的价值

- ★ 边界值分析技术可以显著的减少测试用例数量。边界值上是最容易发现缺陷的地方；
- ★ 边界值分析技术应该和等价类划分技术紧密结合使用；
- ★ 边界值分析技术可以应用于任何测试级别：单元测试、集成测试、系统测试和验收测试；

边界值分析：练习

练习的GUI

Index

Actor Admin Key

Aggregator Size

Aggregator Name

LACP State

Interface Client Type

Admin State

Add Delete Apply

参数取值范围

- ★ **Index:** [10001,10006];
- ★ **ActorKey:** [1,65535];
- ★ **Aggregator Size:** [1,8];
- ★ **Aggregator Name:** 必须是字母，并且长度不能超过8个;

练习输出

- ★ 输出练习中每个参数的边界值;
- ★ 输出练习的概要测试用例，来覆盖有效边界值和无效边界值;

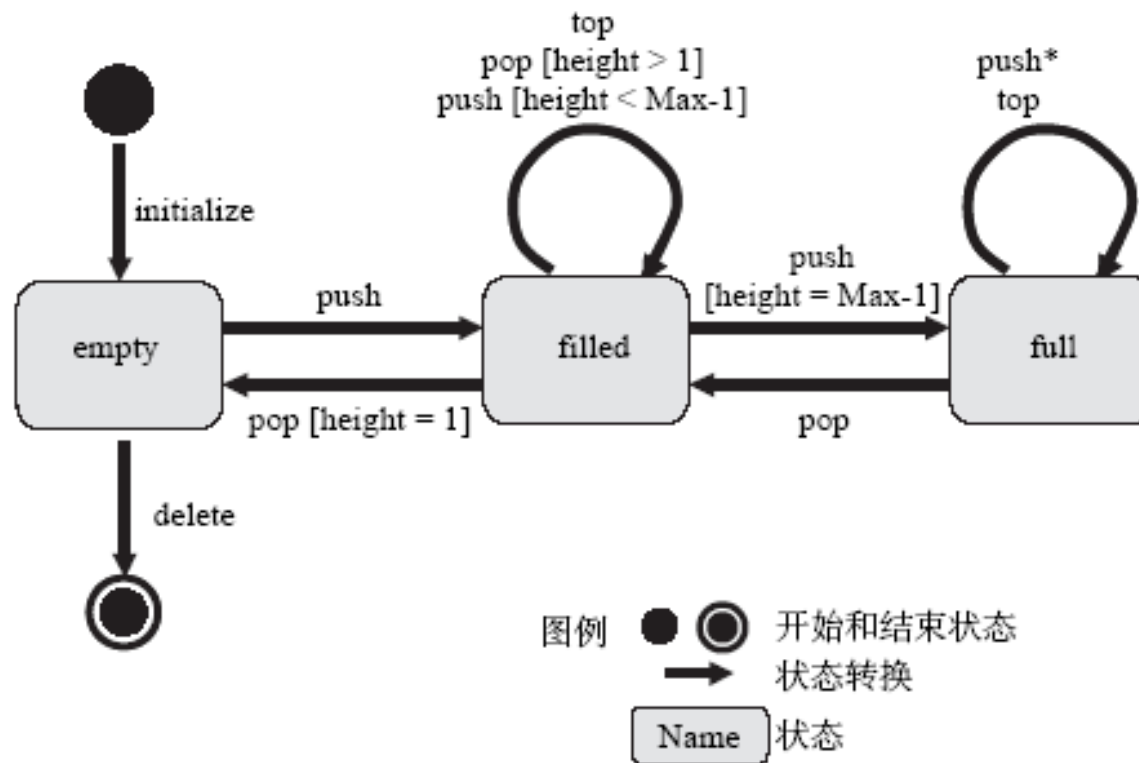
状态转换测试

状态转换测试定义

- ★ 测试对象的输出和行为方式不仅受当前输入数据的影响，同时还与测试对象之前的执行情况，或者之前的事件或以前的输入数据等有关；
- ★ 通过引入状态图（**state diagram**）来描述测试对象和测试数据、对象状态之间的关系；
- ★ 状态图中的各个状态是通过不同的事件驱动的，比如函数的调用；
- ★ 基于状态图开展的测试称之为状态转换测试；

状态转换测试

状态转换测试之状态图



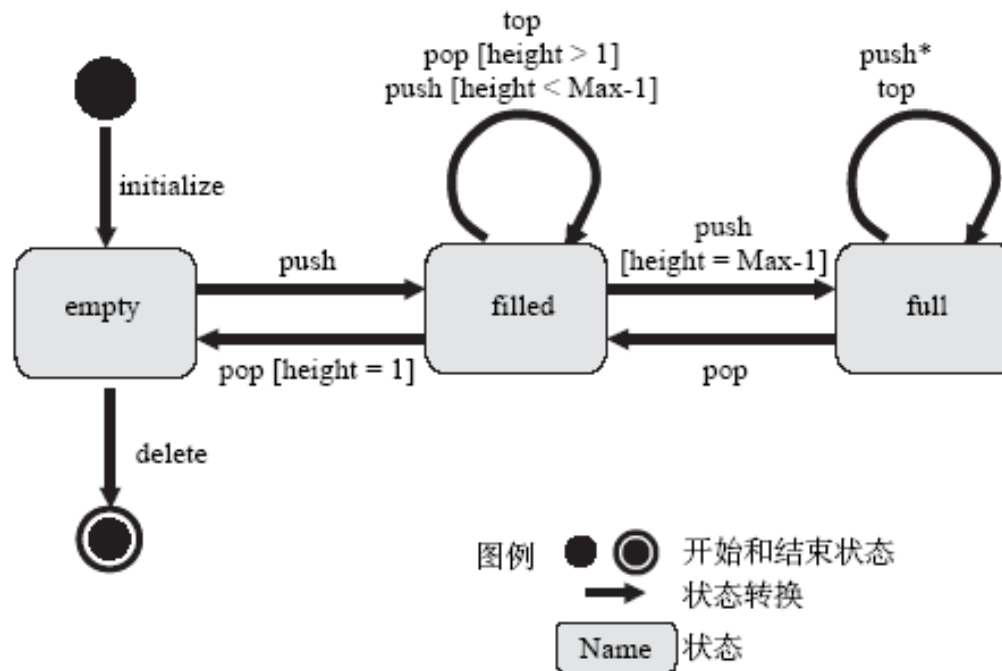
状态转换测试

状态转换测试之状态

★ **empty**

★ **filled**

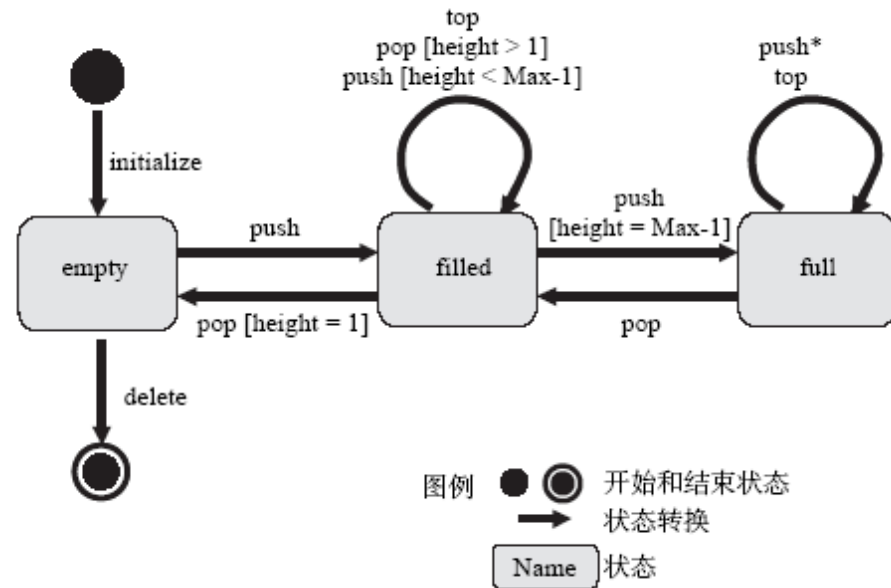
★ **full**



状态转换测试

状态转换测试之事件

- ★ **Push:** 加入一个元素
- ★ **Pop:** 移除一个元素
- ★ **Delete:** 删除堆栈

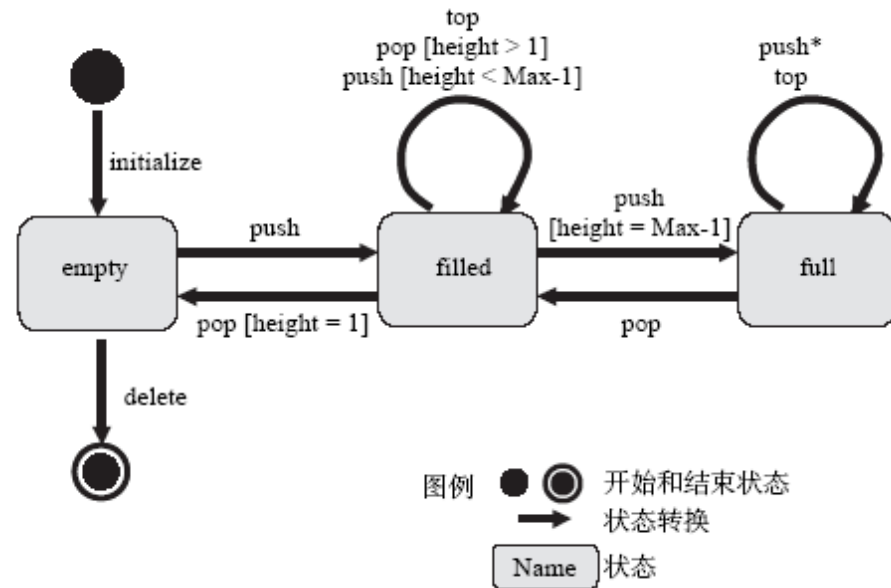


事件驱动着测试对象的状态发生变化，这里指的是调用**push**、**pop**和**delete**函数！

状态转换测试

状态转换测试之信息

- ★ **Top:** 显示最上的元素
- ★ **Max:** 显示栈最大高度
- ★ **Show:** 显示栈的状态



信息中的这些函数调用并不会改变测试对象的状态！

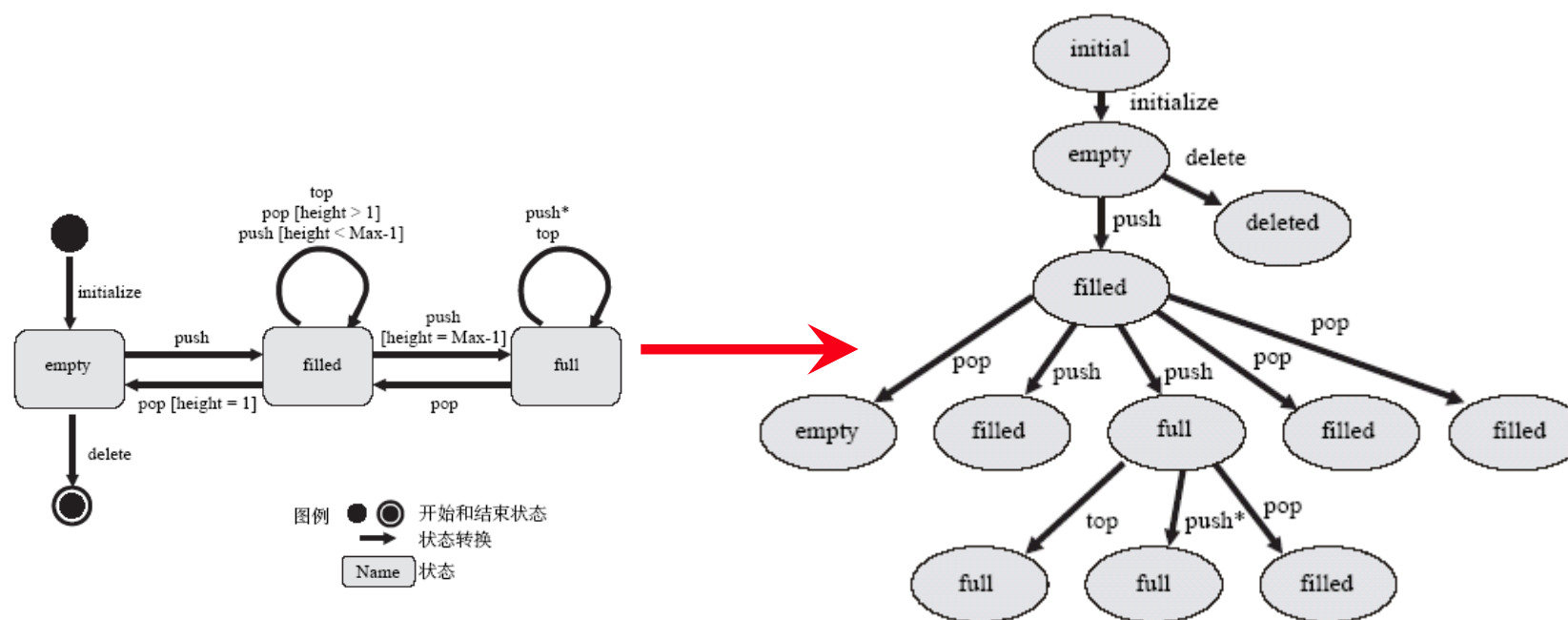
状态转换测试

状态图到状态树的规则

- ★ 初始状态或开始状态是树的根；
- ★ 对于状态图内从开始状态出发到达一个任意可达状态的每个可能的转换，转换树都包含了从根出发到达一个代表此状态的下一后续状态的结点的分支；
- ★ 对转换树的每个叶结点（新增加的结点）重复步骤**2**的过程，直到满足下面结束条件中的一个：
 - ✦ 与叶结点相关的状态已经出现过一次从状态树的根到叶结点的连接上，这个结束条件对应于状态图中的一遍循环；
 - ✦ 与叶结点相关的状态是一个结束状态，并且也没有更多的状态转换需要考虑；

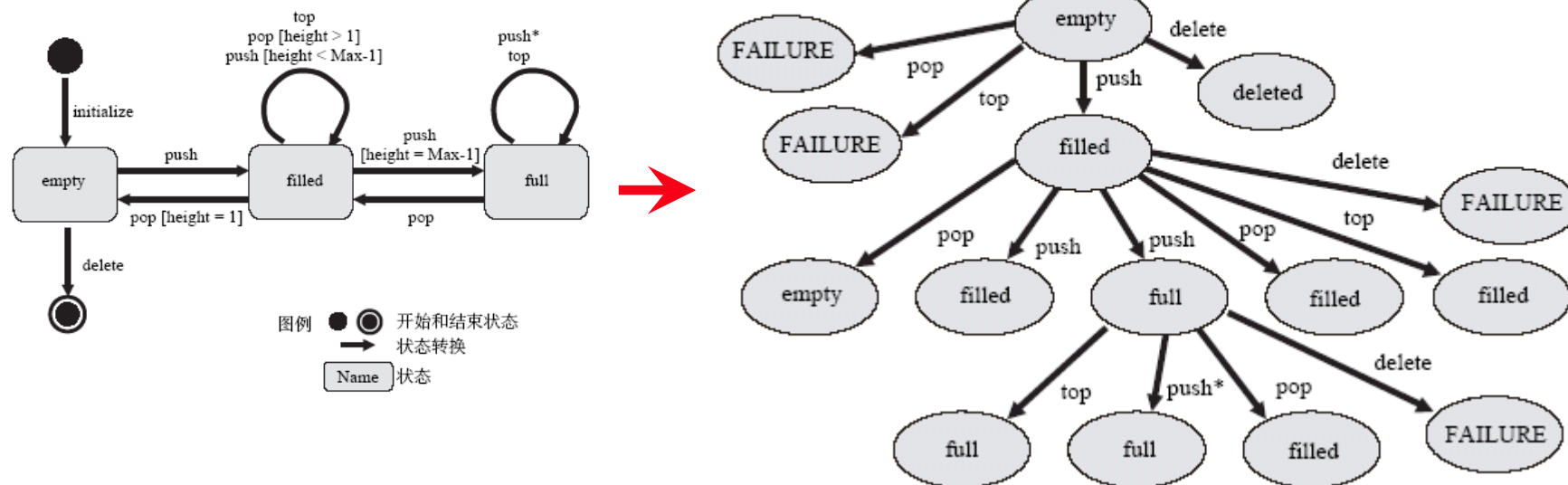
状态转换测试

状态图到状态树



状态转换测试

状态树扩展



错误函数的调用：健壮性测试！

状态转换测试

测试用例包含的信息

- ★ 测试对象的初始状态（组件或系统）；
- ★ 测试对象的输入；
- ★ 期望输出或者期望行为；
- ★ 期望的结束状态；

状态转换测试

状态转换测试的提示

- ★ 开始写规格说明的时候，就从测试的角度来评估测试对象的状态转换图。假如测试对象有很多状态和状态转换，表明需要更多的测试工作量，可能的情况下应该寻找对策简化测试对象；
- ★ 检查规格说明，确保测试对象状态容易识别，且这些状态不是由大量不同变量组合构成的；
- ★ 检查规格说明，保证从外部可以比较容易地访问状态变量。在测试过程中如能使用设置状态/重设状态函数以及读取状态值函数等将会给测试带来很大益处；

状态转换测试

定义状态需要考虑的信息

- ★ 状态转换之前的状态;
- ★ 触发状态转换的触发事件;
- ★ 在状态转换时触发的期望反应;
- ★ 接下来的期望状态;

状态转换测试

状态转换测试强度定义

测试强度1

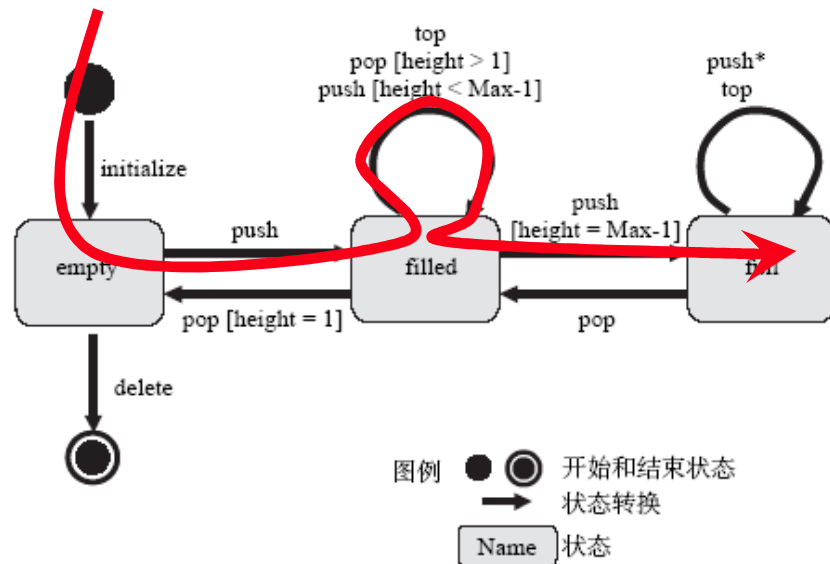
覆盖测试对象的所有状态；

例子

initialize[empty], push[filled],
push, push, push[full];

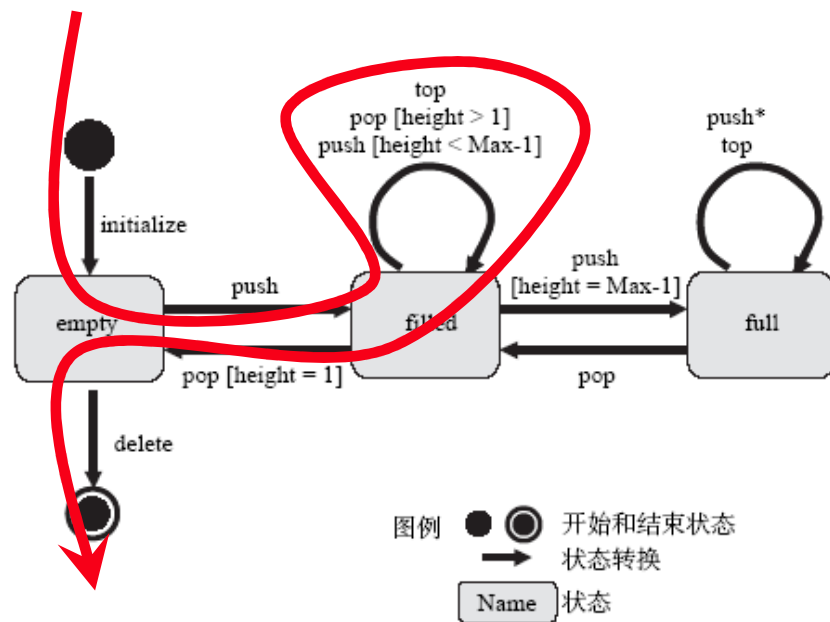
缺点：没有调用所有的函数；

假设：**Max = 4**



状态转换测试

状态转换测试强度定义



测试强度2

覆盖测试对象的所有函数；

例子

`initialize[empty]`, `push[filled]`,
`top`, `pop[empty]`, `delete`;

缺点：没有覆盖所有的状态；

假设： **Max = 4**

状态转换测试

状态转换测试强度定义

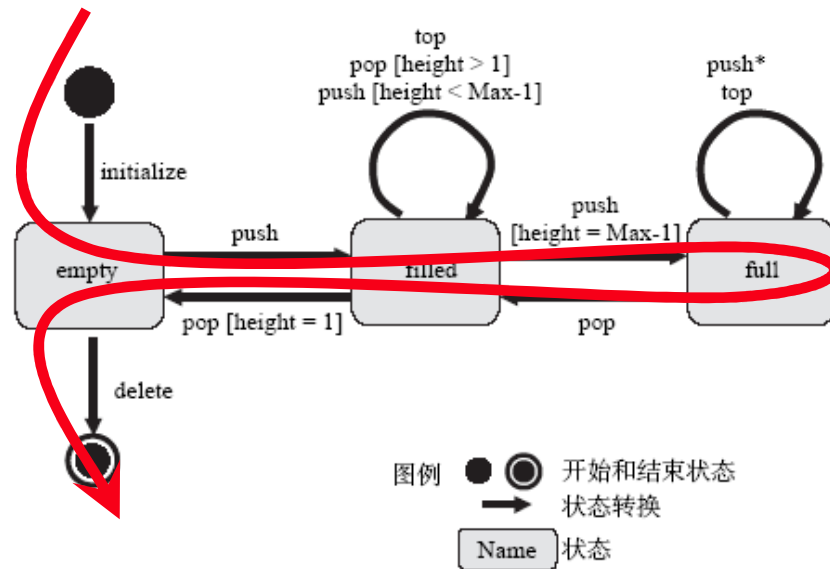
测试强度3

每个状态至少转换一次；

例子

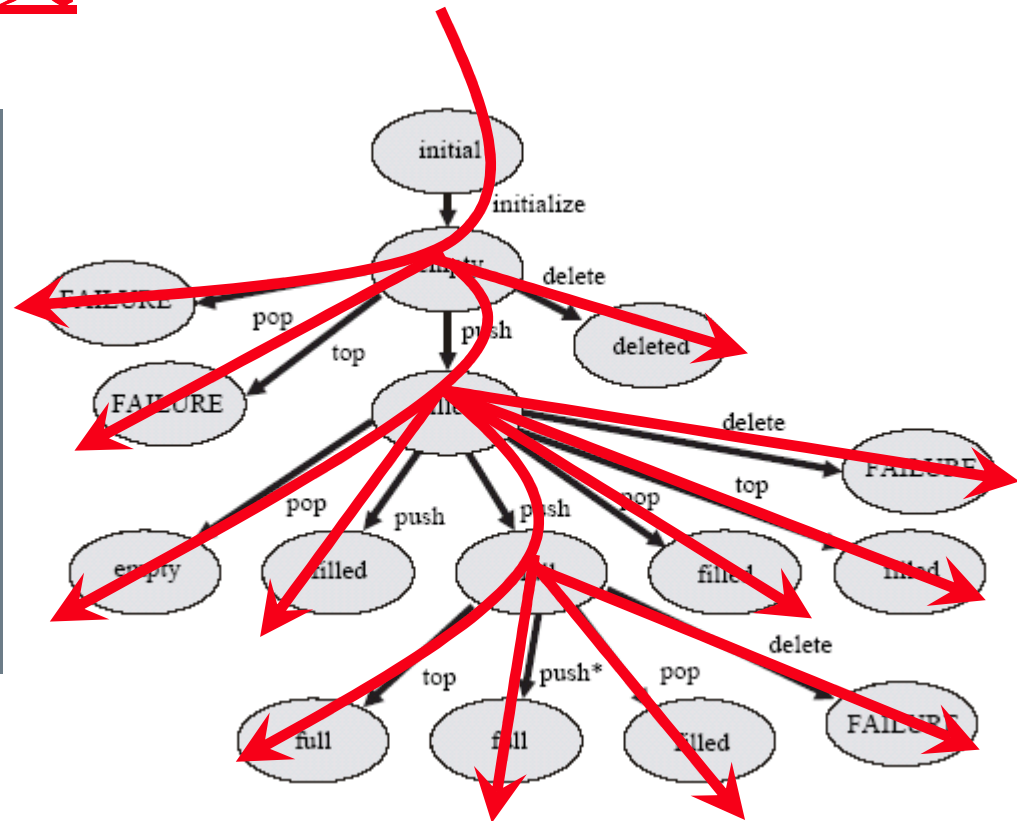
initialize[empty], push[filled]...,
push[full], pop[full]...,
pop[empty], delete;

假设: **Max = 4**



状态转换测试强度定义

状态转换测试对于每个状态都应该将与此状态相关的函数至少执行一遍。通过扩展的状态树来实现；



状态转换测试

状态转换测试覆盖率

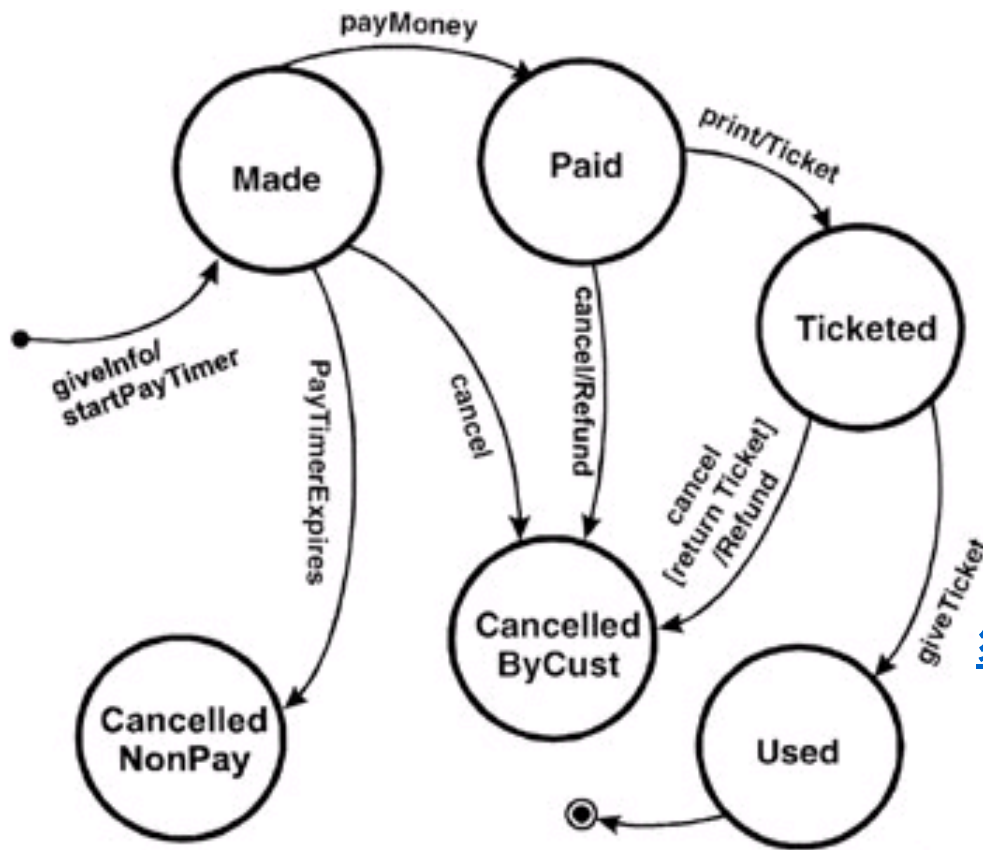
$$= (\text{执行的状态转换数量} / \text{总的数量}) \times 100\%$$

状态转换测试

状态转换测试价值

- ★ 状态转换测试适用于那些状态起着重要作用的测试对象，并且功能也会因为测试对象的状态不同而受到影响；
- ★ 在面向对象的系统中，对象可以有不同的状态，操作对象的方法必须能根据不同的状态做出相应的反应。因此，状态转换测试对于面向对象测试非常重要；

状态转换测试：练习



练习输出

- ★ 列出状态图中包含的状态和事件;
- ★ 设计测试用例覆盖所有的状态;
- ★ 设计测试用例覆盖所有的事件;
- ★ 设计测试用例覆盖所有的路径;
- ★ 设计完整的状态转换表;

决策表测试

决策表测试的定义

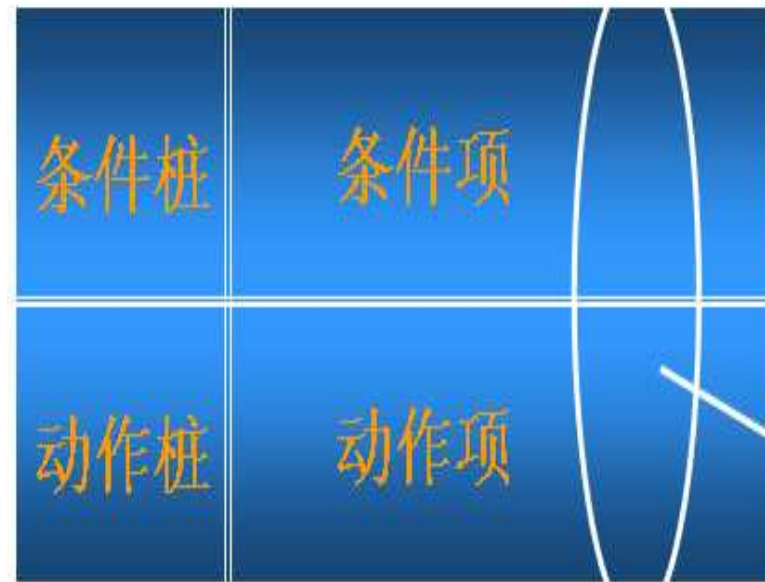
决策表测试是分析和表达多逻辑条件下执行不同操作的情况的工具！

决策表能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏。因此，利用决策表能够设计出完整的测试用例集合；

在一些数据处理问题当中，某些操作的实施依赖于多个逻辑条件的组合，即：针对不同逻辑条件的组合值，分别执行不同的操作。**决策表**很适合于处理这类问题；

决策表测试

决策表测试的组成



规则

决策表测试

决策表组成的定义

- ★ 条件桩（**Condition Stub**）：列出了问题的所有条件。通常认为列出的条件的次序无关紧要；
- ★ 动作桩（**Action Stub**）：列出了问题规定可能采取的操作。这些操作的排列顺序没有约束；
- ★ 条件项（**Condition Entry**）：列出针对它左列条件的取值。在所有可能情况下的真假值；
- ★ 动作项（**Action Entry**）：列出在条件项的各种取值情况下应该采取的动作；

决策表测试

决策表测试规则

任何一个条件组合的特定取值及其相应要执行的动作就称为规则；

在决策表中贯穿条件项和动作项的一列就是一条规则；

决策表中列出多少组条件取值，也就有多少条规则，既条件项和动作项有多少列；

决策表测试

决策表测试步骤

- ★ 确定规则的个数。假如有 n 个条件，每个条件有两个取值（**0,1**），故有 2^n 种规则；
- ★ 列出所有的条件桩和动作桩；
- ★ 填入条件项；
- ★ 填入动作项，得到初始的决策表；
- ★ 简化合并相似的规则，比如相同动作，得到优化的决策表；
- ★ 每列规则设计一个测试用例；

决策表测试

决策表测试例子


| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----|------|---|---|---|---|---|---|---|---|
| 条件桩 | 觉得疲倦 | Y | Y | Y | Y | N | N | N | N |
| | 感兴趣 | Y | Y | N | N | Y | Y | N | N |
| | 糊涂 | Y | N | Y | N | Y | N | Y | N |
| 动作桩 | 重读 | | | | | √ | | | |
| | 继续 | | | | | | √ | | |
| | 跳下一章 | | | | | | | √ | √ |
| | 休息 | √ | √ | √ | √ | | | | |

决策表测试

决策表测试规则合并

有两条或多条规则具有相同的动作，并且其条件项之间存在着极为相似的关系；

| | | 1 | 2 | 3 | 4 |
|-----|------|---|---|---|---|
| 条件桩 | 觉得疲倦 | Y | Y | Y | Y |
| | 感兴趣 | Y | Y | N | N |
| | 糊涂 | Y | N | Y | N |
| 动作桩 | 重读 | | | | |
| | 继续 | | | | |
| | 跳下一章 | | | | |
| | 休息 | √ | √ | √ | √ |




| | | 1 |
|-----|------|---|
| 条件桩 | 觉得疲倦 | Y |
| | 感兴趣 | — |
| | 糊涂 | — |
| 动作桩 | 重读 | |
| | 继续 | |
| | 跳下一章 | |
| | 休息 | √ |

决策表测试

决策表测试规则合并

有两条或多条规则具有相同的动作，并且其条件项之间存在着极为相似的关系；

| | | | |
|-----|------|---|---|
| | | 7 | 8 |
| 条件桩 | 觉得疲倦 | N | N |
| | 感兴趣 | N | N |
| | 糊涂 | Y | N |
| 动作桩 | 重读 | | |
| | 继续 | | |
| | 跳下一章 | √ | √ |
| | 休息 | | |
| | | | |



| | | |
|-----|------|---|
| | | 7 |
| 条件桩 | 觉得疲倦 | N |
| | 感兴趣 | N |
| | 糊涂 | — |
| 动作桩 | 重读 | |
| | 继续 | |
| | 跳下一章 | √ |
| | 休息 | |
| | | |

决策表测试

决策表规则合并优化

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----|------|---|---|---|---|---|---|---|---|
| 条件桩 | 觉得疲倦 | Y | Y | Y | Y | N | N | N | N |
| | 感兴趣 | Y | Y | N | N | Y | Y | N | N |
| | 糊涂 | Y | N | Y | N | Y | N | Y | N |
| 动作桩 | 重读 | | | | | √ | | | |
| | 继续 | | | | | | √ | | |
| | 跳下一章 | | | | | | | √ | √ |
| | 休息 | √ | √ | √ | √ | | | | |



| | | 1 | 2 | 3 | 4 |
|-----|------|---|---|---|---|
| 条件桩 | 觉得疲倦 | Y | N | N | N |
| | 感兴趣 | — | Y | Y | N |
| | 糊涂 | — | Y | N | — |
| 动作桩 | 重读 | | √ | | |
| | 继续 | | | √ | |
| | 跳下一章 | | | | √ |
| | 休息 | √ | | | |

决策表测试

决策表转化为概要测试用例

| | | 1 | 2 | 3 | 4 |
|-----|------|---|---|---|---|
| 条件桩 | 觉得疲倦 | Y | N | N | N |
| | 感兴趣 | — | Y | Y | N |
| | 糊涂 | — | Y | N | — |
| 动作桩 | 重读 | | √ | | |
| | 继续 | | | √ | |
| | 跳下一章 | | | | √ |
| | 休息 | √ | | | |

| 编号 | 输入 | 期望结果 |
|----|---------------|------|
| 1 | 觉得疲倦 | 休息 |
| 2 | 不觉得疲倦、感兴趣，但糊涂 | 重读 |
| 3 | 不觉得疲倦、感兴趣且不糊涂 | 继续 |
| 4 | 不觉得疲倦，但不感兴趣 | 跳下一章 |

决策表测试

决策表测试应用

- ★ 规格说明以决策表形式给出，或很容易转换成决策表；
- ★ 条件的排列顺序不会也不影响执行哪些操作；
- ★ 规则的排列顺序不会也不影响执行哪些操作；
- ★ 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则；
- ★ 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要；

决策表测试

决策表测试测试用例

从决策表的每一列可以清楚地看到条件及其输入的依赖关系，以及由这些输入组合得到的相应的输出动作和结果；

决策表定义了逻辑测试用例，为了执行这些测试用例，必须输入具体的数据值并且标识前置条件和后置条件；

决策表测试

决策表测试完成准则

同前面介绍的方法一样，可以简单地定义决策表测试完成的准则。最基本的要求是至少用一个测试用例来执行决策表中的每一列，这样就验证了所有关心的输入条件组合和相应的输出结果；

决策表测试

决策表测试价值

根据可能的输入条件组合来定义决策表是一种系统化而且非常正式的方法，它可以覆盖一些在其他测试用例设计技术中没有包含的输入组合。但是，在优化决策表时可能会引入错误，比如忽略了需要考虑的输入和条件的组合。

正如前面提到，在条件的数量和依赖关系增加时，决策表的规模增加得非常快，从而失去可读性。如果没有工具的支持，使用决策表技术就不太容易；

决策表测试：练习

| | 三角形判定决策表 | | | | | | | | | | | | | | | |
|----------------|----------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| C1:a、b、c构成三角形? | | | | | | | | | | | | | | | | |
| C2:a=b? | | | | | | | | | | | | | | | | |
| C3:a=c? | | | | | | | | | | | | | | | | |
| C4:c=b? | | | | | | | | | | | | | | | | |
| a1:非三角形 | | | | | | | | | | | | | | | | |
| a2:不等边三角形 | | | | | | | | | | | | | | | | |
| a3:等腰三角形 | | | | | | | | | | | | | | | | |
| a4:等边三角形 | | | | | | | | | | | | | | | | |
| 不符合逻辑 | | | | | | | | | | | | | | | | |

练习输出

- ★ 输出初始的三角形判定决策表；
- ★ 根据前面的知识，将初始决策表进行合理的优化，得到简化的决策表；
- ★ 根据简化的决策表，设计概要测试用例；

用例测试

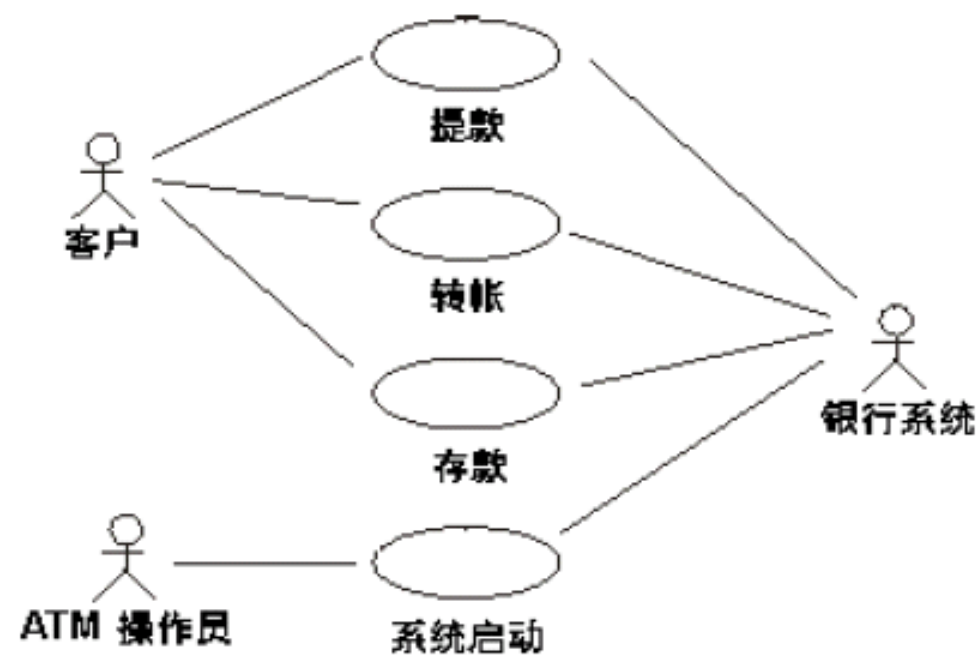
用例测试定义

通过用例（**Use Cases**）或业务场景来设计测试，用例描述了参与者（包括用户与系统）之间的相互作用，并从这些交互产生一个从用户的角度所期望和能观察到的结果；

每个用例都有测试前置条件，这是用例成功执行的必要条件。每个用例结束后都存在后置条件，这是在用例执行完成后能观察到的结果和系统的结束状态；

用例测试

用例测试例子



用例测试

测试用例必要信息

- ★ 开始情况和前置条件;
- ★ 其他可能的条件;
- ★ 期望结果;
- ★ 后置条件;

用例测试

用例测试技术的价值

- ★ 基于用例的测试技术适合于测试典型的用户系统交互。因此，将这种技术应用在验收测试和系统测试中是最好的；
- ★ 另外，测试规格说明工具可以用于支持用例测试技术。其他测试技术（如边界值分析）可以在这方面做些辅助；

其他黑盒测试技术

随机测试

随机选择数据作为测试用例的值；

冒烟测试

冒烟测试通常被理解为“快速且脏”的测试，主要的目的是验证测试对象的最低要求的可靠性；

冒烟测试主要集中在测试对象的主要功能上，不会对测试的输出进行详细的评估；

黑盒测试技术讨论

未检测到规格说明
中的错误

所有黑盒技术的基础是需求和系统的规格说明以及单个组件和组件之间的协作的规格说明。如果需求本身有错或者是按照错误的规格说明来实现就不可能发现这些问题，因为在有错的规格说明与实际的反应间不存在偏差。尽管需求文档或规格说明是错误的，但测试对象是按照这些需求文档或规格说明的要求来运行的。如果测试人员对需求文档或规格说明严格把关，根据测试人员的经验，可以在测试设计时发现需求的错误；否则必须通过评审才能发现规格说明中的不一致性和存在的问题（4.1.2节）。

黑盒测试技术讨论

未检测到规格说明
以外的功能

另外，黑盒测试不能发现规格说明规定之外的额外功能（这些额外的功能经常会引发严重问题），这些额外的功能既没有描述也不是客户要求的，即使真的存在这些额外功能的测试用例，可能也只会偶然去执行。作为测试完成的标准，覆盖率标准是严格基于规格说明或需求文档的，而绝不是没有提到的或者假想的功能。

功能验证

黑盒测试的中心思想是验证测试对象的功能。毫无异议，软件系统可以正确工作是第一位的。因此，实际测试过程中经常会用到黑盒测试技术。

课程内容

1. 测试开发过程
2. 测试设计技术类型
3. 黑盒测试技术
4. 白盒测试技术
5. 基于经验的技术
6. 测试技术的选择

白盒测试技术

ISTQB考试知识点

- ★ 描述代码覆盖的概念及其重要性（**K2**）；
- ★ 解释语句覆盖和判定覆盖等概念，理解这些概念除了可以应用在组件测试外，还可以应用在其他任何测试级别上（**K2**）；
- ★ 根据给定的控制流，使用下面的测试设计技术设计测试用例（**K3**）：
 - ✦ ☐ 语句测试
 - ✦ ☐ 判定测试
- ★ 评估语句覆盖和判定覆盖的完整性（**K3**）；

白盒测试技术

代码覆盖的含义

白盒测试技术的基础是测试对象的代码，因此也称之为基于代码的测试技术或者基于结构的测试技术；

基于代码的测试技术，其代码覆盖可以基于不同的对象进行判断，比如基于语句的语句判断、基于分支的分支判断等；

白盒测试技术的测试期望结果应该是根据需求或规格说明来确定的，而不是代码本身来确定；

白盒测试技术

代码覆盖的步骤

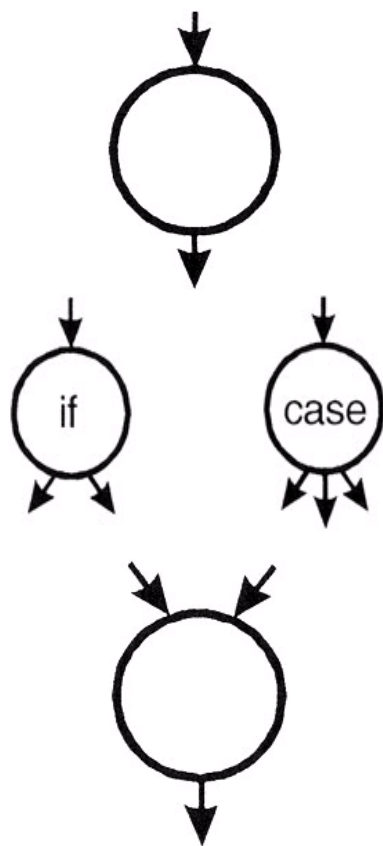
第一步：源代码转换为控制流图。控制流图可以比较直观而详细地描述需要覆盖的语句；

第二步：分析控制流图，根据测试对象的要求选择需要覆盖的代码；

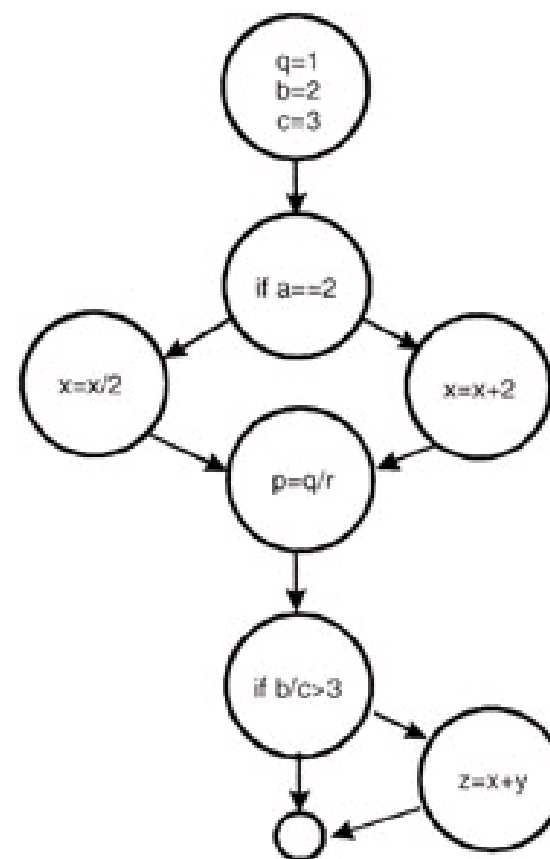
第三步：根据第二步的结果确定测试数据，生成测试用例；

白盒测试技术

控制流图例子



```
q=1;  
b=2;  
c=3;  
if (a==2) {x=x+2;}  
else {x=x/2;}  
p=q/r;  
if (b/c>3) {z=x+y;}
```



语句覆盖（**Statement Coverage**）

语句覆盖的含义

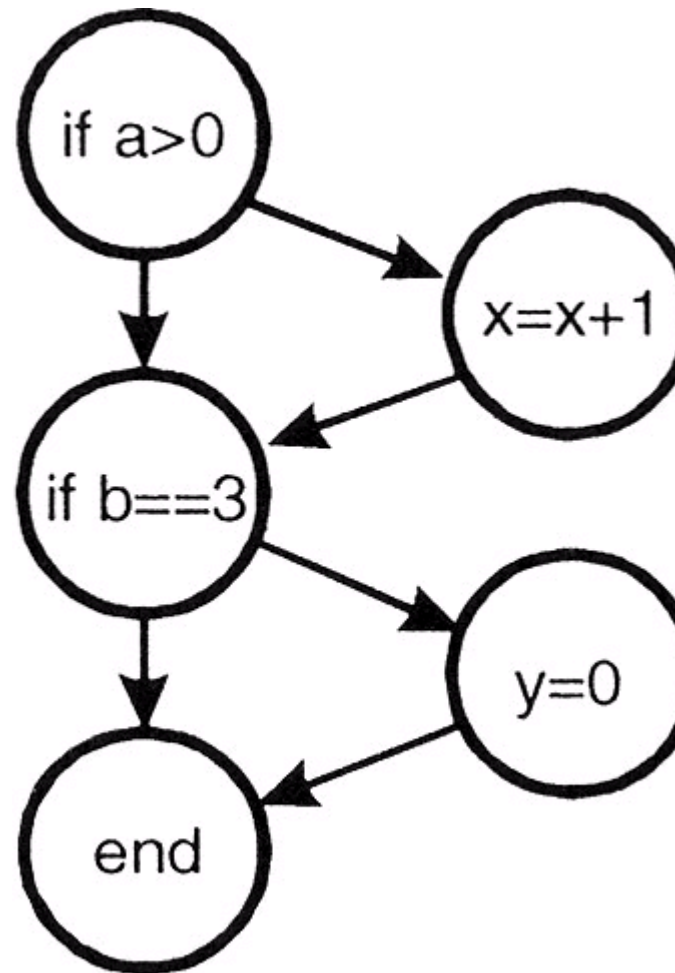
语句覆盖就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次；

语句覆盖的第一步是将源代码转换为控制流图。控制流图可以比较直观而详细地描述需要覆盖的语句；

语句覆盖（Statement Coverage）

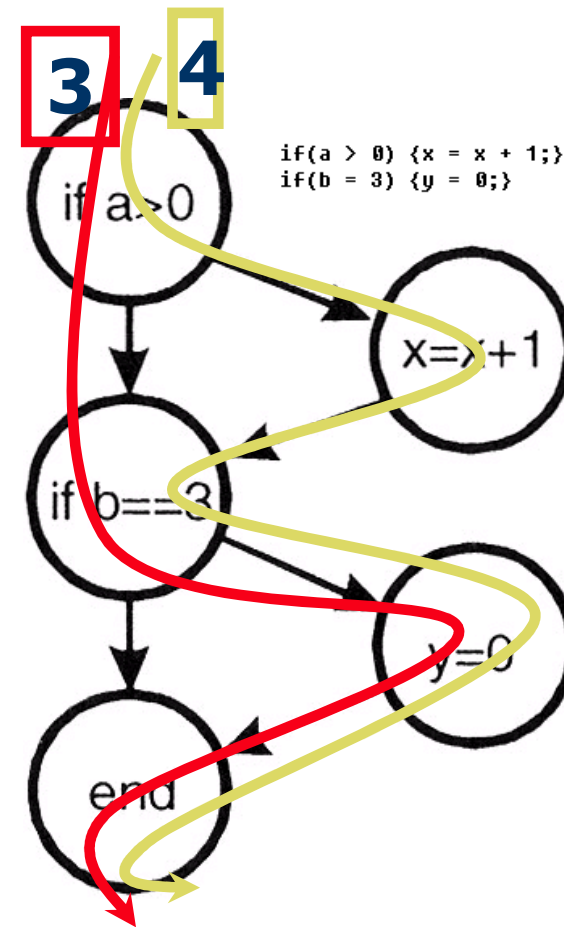
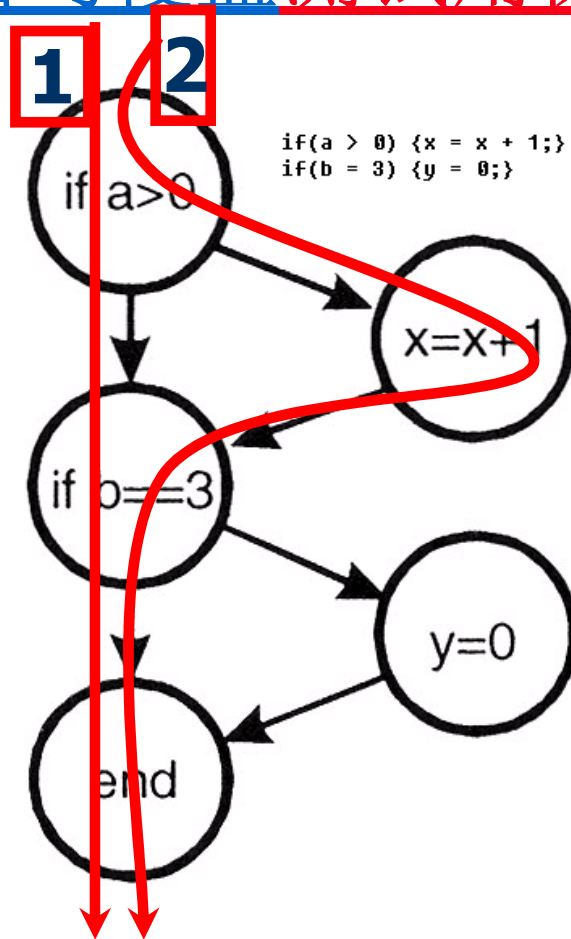
语句覆盖例子

```
if(a > 0) {x = x + 1;}  
if(b = 3) {y = 0;}
```



语句覆盖（Statement Coverage）

语句覆盖测试用例



语句覆盖

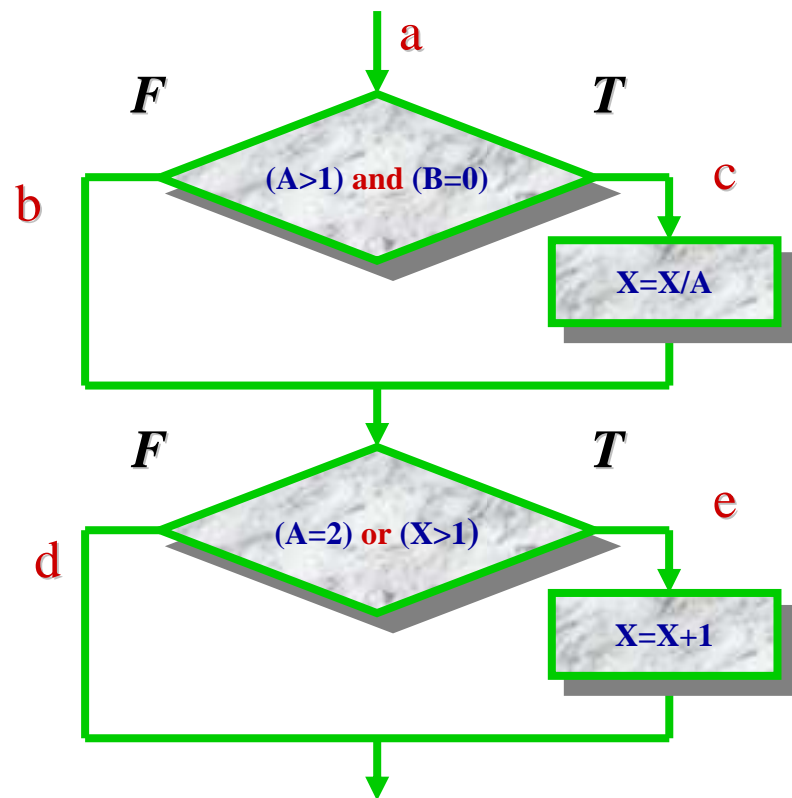
语句覆盖测试完成准则

$$= \left(\frac{\text{被执行的语句数量}}{\text{总的语句数量}} \right) \times 100\%$$

只要执行前面例子中的第四条路径（比如**a=8**, **b=3**作为输入）的测试用例，就可以达到**100%**的语句覆盖率；

语句覆盖，有时候又叫做**C0**覆盖，是最弱的一种覆盖；

语句覆盖：练习



练习输出

- ★ 选择通过哪些测试用例（或者哪条路径），可以达到**100%**的语句覆盖？

分支覆盖（**Branch Coverage**）

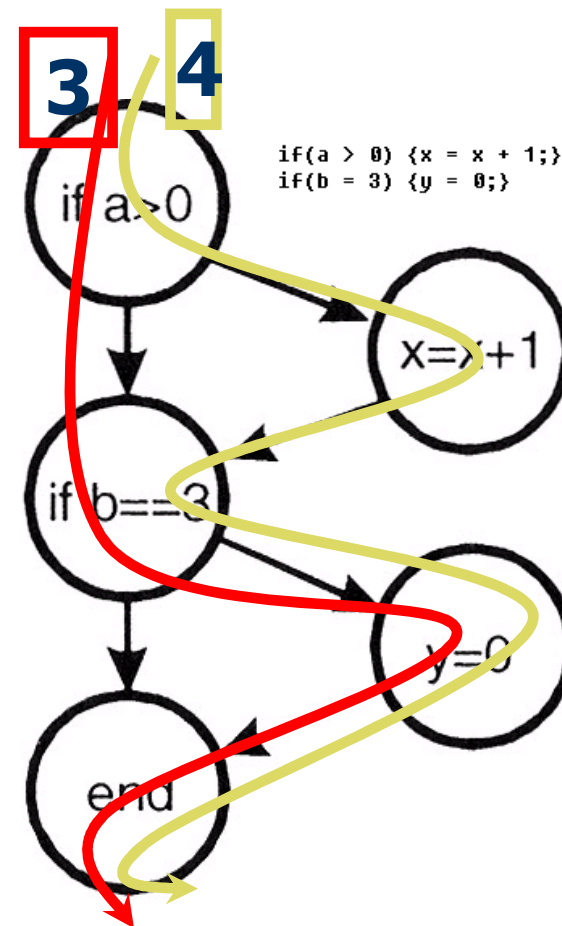
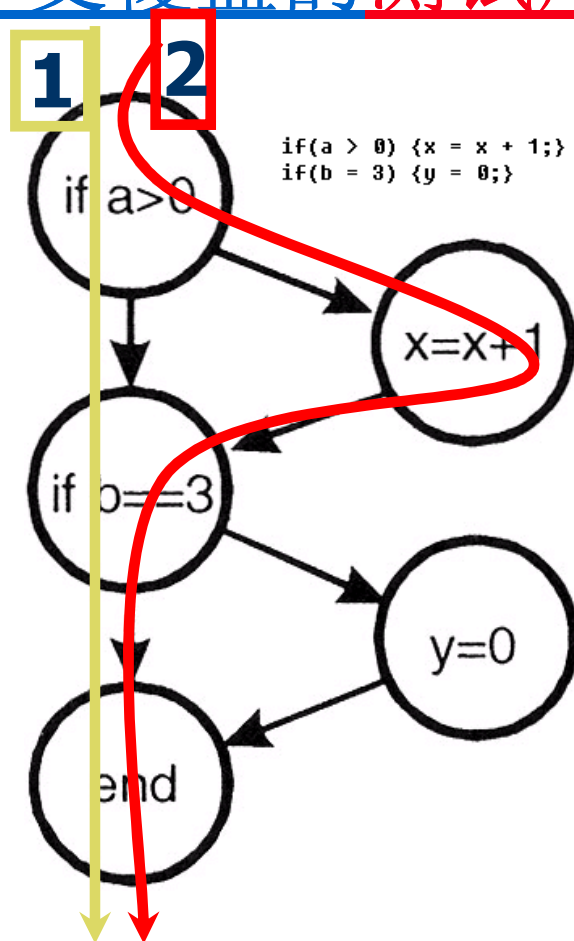
分支覆盖的**含义**

控制流图中的边是分支覆盖关注的焦点。它不考虑每条语句的执行情况，而是考虑判定的执行情况，由判定的结果来决定执行哪条语句；

测试需要确保每个判定得到了**TRUE**和**FALSE**的结果，即保证每个判定条件取**TRUE**和取假**FALSE**各至少一次。分支覆盖又称为判定覆盖（**decision coverage**）；

分支覆盖（Branch Coverage）

分支覆盖的测试用例



分支覆盖（**Branch Coverage**）

分支覆盖的测试用例

执行**1**和**4**路径达到**100%**的分支覆盖率，比如：

[a = 6, b = 3]和**[a = -1, b = 4]**分别作为两个测试用例的输入：前一个用例实现了分支的**TT**，而后一个是分支**FF**；

执行**2**和**3**路径也可达到**100%**的分支覆盖率，比如：

[a = -1, b = 3]和**[a = 6, b = 4]**分别作为两个测试用例的输入：前一个用例实现了分支的**FT**，而后一个是分支**TF**；

分支覆盖

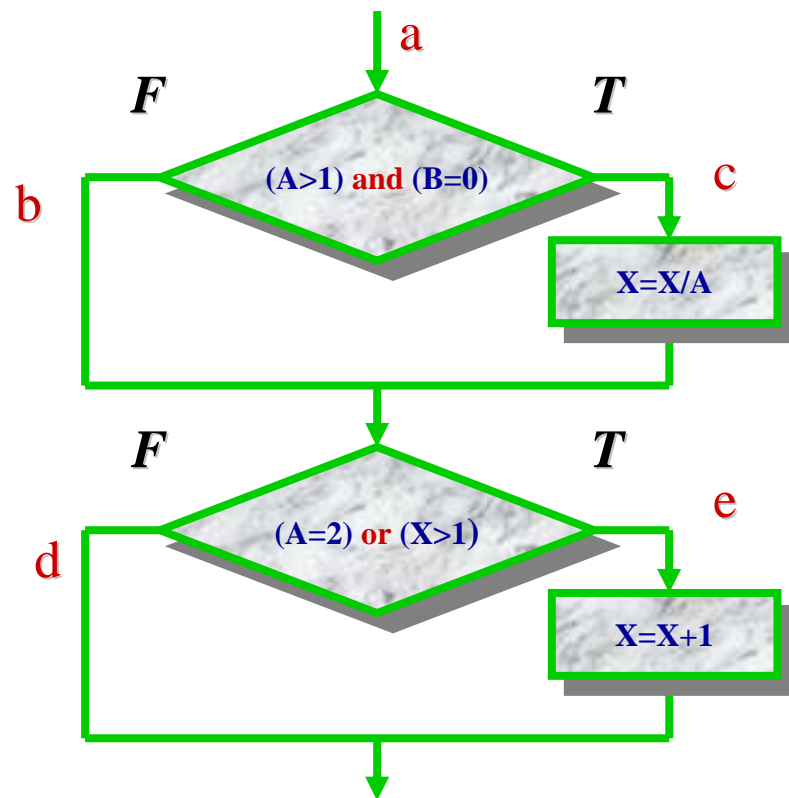
分支覆盖测试完成准则

= （被执行的分支数量 / 总的分支数量） × **100%**

分支覆盖，有时候又叫做**C1**覆盖，**100%**的分支覆盖可以保证**100%**的语句覆盖；

与语句覆盖相反，分支覆盖可以发现在空分支中遗漏的语句；

分支覆盖：练习



练习输出

- ★ 画出分支覆盖的详细表格；
- ★ 选择通过哪些测试用例（或者哪条路径），可以达到**100%**的分支覆盖？

白盒技术的讨论

测试强度的定义

- ★ 所有白盒技术的基础是源代码，可以根据程序结构的复杂程度，选择和应用适当的测试用例设计技术；
- ★ 根据源代码和所选择的技术，比如语句覆盖、分支覆盖、条件覆盖、判定-条件覆盖等，可以确定测试的强度；

白盒测试技术一般应用在低级别的测试中，在高级别的测试中，一般不关注程序的代码！

白盒技术的讨论

白盒技术的不足

白盒技术并不能发现需求没有实现或者需求遗漏的问题。白盒技术只能验证存在的代码，即程序中已实现的需求，而无法检查出应该在系统中实现而未实现的部分。因此，查找遗漏需求需要其他测试设计技术，比如高级别的测试（系统测试等）或者通过静态测试方式！

白盒测试技术一般应该有工具的支持，完全通过人工的方式几乎不可能实现覆盖的目标！

课程内容

1. 测试开发过程
2. 测试设计技术类型
3. 黑盒测试技术
4. 白盒测试技术
5. 基于经验的技术
6. 测试技术的选择

基于经验的技术

ISTQB考试知识点

- ★ 复述在哪些情况下，可以使用基于直觉、基于经验和知识、基于对常见缺陷的认识来编写测试用例（**K1**）；
- ★ 比较基于经验的方法和基于规格说明的方法之间的区别（**K2**）；

基于经验的技术

基于直觉和经验的技术

- ★ 测试基础文档不全或者不完善；
- ★ 基于测试人员的技术、知识和经验；
- ★ 可以作为系统化测试用例设计的补充，它可以发现一些运用系统化方法进行测试时无法发现的问题；
- ★ 基于直觉和经验的测试用例设计方法不能简单地归类为黑盒测试技术或白盒测试技术。这一技术应用在比较高的测试级别上；

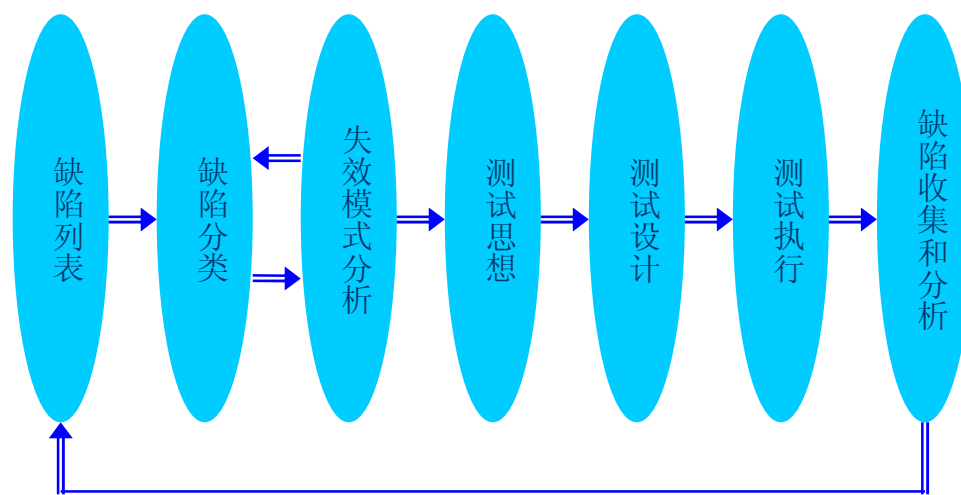
基于经验的技术

错误推测法（**Error Guessing**）

- ★ 测试人员通过选择测试用例来发现预期的问题和它们的征兆；
- ★ 测试用例完全基于什么地方会发生故障的经验，或者测试人员对什么地方可能会发生故障的假设；
- ★ 需要借鉴类似的应用程序开发的经验和类似技术方面的知识，以及测试方面的经验；

基于经验的技术

错误推测法的过程



基于经验的技术

测性测试定义

测性测试是基于经验和技术的，用来为项目相关人员提 产 质量或者产 务信息；

测性测试软件测试的一种，它强调测试人员的自由和主管能动性，在软件生命周期过程中，通过同步的测试学习、测试设计和测试执行来优化测试人员的价值！

基于经验的技术

测试测试目的

Burst of testing触发测试的原因可以是：学习产品、选择覆盖内容、确定测试准则、配置测试系统、操作测试系统、观察被测系统、评估被测系统、组织要求等；根据不同的条件，来确定测试什么、什么时候开始测试，以及如何测试！

基于经验的技术

测试测试关注点

- ★ 为什么要执行测试？测试执行的目标是什么？
- ★ 测试什么内容？
- ★ 如何测试？使用哪种测试方法？
- ★ 需要发现什么问题？
- ★ 下一步的目标；

基于经验的技术

测试测试基本

- ★ 一个测试用例执行的结果会影响后续测试用例的设计和執行；
- ★ 测试期间，为测试的程序构建一个 拟的模型。模型中包含程序是如何工作的，以及它的行为如何或者它应该产生 样的行为；
- ★ 测试应该针对这个模型进行运行。关注点是发现模型中没有的或者与以前发现的不一样的程序的信息和行为；

基于经验的技术

基于经验技术的提示

- ★ 创建和维护缺陷列表： 列所有可能存在的错误、故障和可 环境等是非常有用的。在列表中应标注经常发生的错误、故障和失效，以此与其他测试人员 这些信息。通过标识可能的问题和重的情况，就可以有针对性地设计一些 加的测试用例；
- ★ 与开发人员 缺陷列表：这些列表 至对开发人员也是很有 助的，因为它预示了会发生哪些 在的问题和 。在程序实现时适当考虑这些因素可以有效地预 这些错误；

课程内容

- 1. 测试开发过程**
- 2. 测试设计技术类型**
- 3. 黑盒测试技术**
- 4. 白盒测试技术**
- 5. 基于经验的技术**
- 6. 测试技术的选择**

测试技术的选择

ISTQB考试知识点

- ★ 针对不同类型的问题选择不同的测试用例设计技术，列举出会影响设计技术选择的因素，比如系统的类型、
、用户需求、用例建模的模型、需求模型或测试员的知识水平等（**K2**）；

测试技术的选择

测试设计的目的

用尽可能少的工作量，生成足够多的不同的测试用例，并能最大程度地发现可能存在的缺陷和故障！

合理的选择我们的测试技术

测试技术的选择

影响测试设计的因素

- ★ **测试对象的类型**。不同的测试对象，其复杂度也是 然不同；
- ★ **正规的文档和工具的可用性**。如果有正规的规格说明或者模型信息，这些就可以直接作为测试设计工具的输入，从而得到具体的测试用例；
- ★ **标准的符合性**。行业标准和法规标准会要求使用特定的测试技术和覆盖准则，特别是对 全关的软件或者对完整性要求很高的软件；
- ★ **测试人员的经验**。不同经验的测试人员会选择不同的测试技术。比如，测试人员会使用以前找到过 重故障的技术；

测试技术的选择

影响测试设计的因素（续）

- ★ **户的期望。** 户可能要求采用特殊的测试技术，并且达到一定的测试覆盖率（当使用白盒测试设计技术的时候；
- ★ **的分析。** 分析或多或少可以指导测试工作，即具体测试技术的选择和测试执行强度的确定。对于高的区域应该进行更彻底的测试；
- ★ **其他因素。** 最后，还有其他一些因素，比如规格说明和其他文档的可用性、测试人员的知识和技能、时间和预算的制、测试级别和以的经验；

测试技术的选择

测试技术选择的提示

- ★ 因为没有一种测试技术能够覆盖测试中需要考虑的所有方方面面，所以测试过程中经常会采用不同测试技术的组合；
- ★ 失效的 重程度和预期的 可以指导测试技术的选择和测试执行的强度的确定；
- ★ 选择白盒测试技术的基础是测试对象的结构。例如，如果测试对象中不包含复杂的条件，应用条件确定测试就没有什么意义了；

