软工第7次上机

软件测试

软件测试

为了发现程序中的错误而执行程序的过程

贯穿软件生命周期

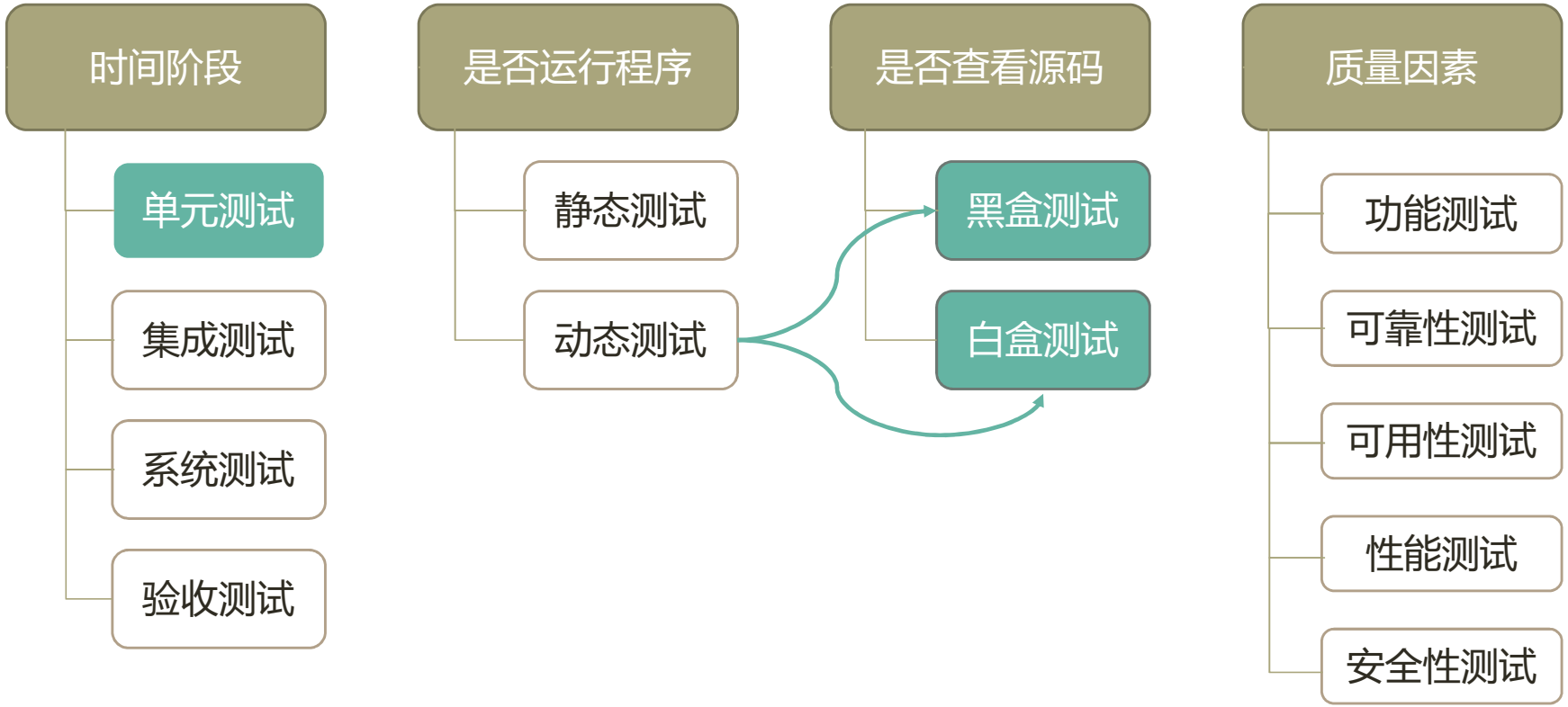
目的为：尽早发现软件缺陷，并确保其得以修复。

再全面的测试也不能完全消除软件缺陷，希望**完全**依托测试确保软件质量是不现实的

软件测试的原则

完全测试是不可能的。测试并不能找出所有错误。
测试中存在风险。
软件测试只能表示缺陷的存在，而不能证明软件产品已经没有缺陷。
软件产品中潜在的错误数与已发现的错误数成正比。
让不同的测试人员参与到测试工作中。
让开发小组和测试小组分立。
尽早并不断进行测试。
在设计测试用例时，应该包括输入数据和预期的输出结果两部分。
集中测试容易出错或错误较多的模块。
长期保留所有的测试用例。

软件测试的分类

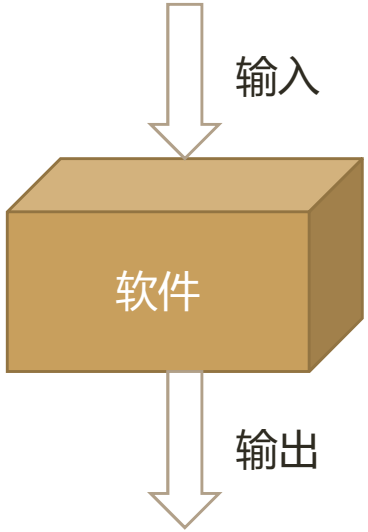

时间阶段
单元测试
集成测试
系统测试
验收测试
是否运行程序
静态测试
动态测试
是否查看源码
黑盒测试
白盒测试
质量因素
功能测试
可靠性测试
可用性测试
性能测试
安全性测试

黑盒测试

将被测试的软件系统看成一个黑盒子，不关心盒子的内部结构和内部特性，只关心软件产品的输入数据和输出结果。

方法：

- 等价类划分法
- 边界值分析法
- 错误推测法
- 因果图法



等价类划分法

将程序的输入域划分为若干子集，然后从每个子集中选取少数具有代表性的数据用作测试用例。一个子集称为一个等价类，同一个等价类中的输入数据对于揭露程序中的错误都是等效的。

等价类分为有效等价类和无效等价类：

- 有效等价类：对程序的规格说明是有意义的、合理的输入数据所构成的集合。
- 无效等价类：对程序的规格说明是无意义的、不合理的输入数据所构成的集合。

等价类划分法

划分等价类时遵循的原则：

1. 如果输入条件规定了取值范围或个数，则可确定一个有效等价类和两个无效等价类。

例：要求输入值x在0到100之间，则有：有效等价类0<=x<=100，无效等价类x<0和x>100。

2. 如果输入条件规定了输入值的集合或是规定了“必须如何”的条件，则可以确定一个有效等价类和一个无效等价类。

例：要求输入值是整数，则有：有效等价类整数数据，无效等价类其他类型数据

等价类划分法

3. 如果输入条件是布尔表达式，则可以分为一个有效等价类和一个无效等价类。

例：要求密码非空，则有：效等价类非空密码，无效等价类空密码。

4. 如果输入条件是一组值，且程序对不同的值有不同的处理方式，则每个允许的输入值对应一个有效等价类，所有不允许的输入值的集合为一个无效等价类。

例：要求输入 初级、中级、高级中的一个，则有：有效等价类初级、中级、高级，无效等价类其他所有输入。

5. 如果规定了输入数据必须遵循的规则，则可划分出一个有效等价类（符合规则）和若干个无效等价类（从不同的角度违反规则）

一个输入可以有多条约束；例：日期

设计测试用例的步骤

1. 对每个输入和外部条件进行等价划分，画出等价类表，并为每个等价类进行编号。

2. 设计若干个测试用例覆盖所有的有效等价类，每个测试用例应覆盖尽可能多的有效等价类。

3. 为每一个无效等价类设计一个测试用例。

示例

测试一个函数dateValidation(int year, int month, int day)，功能是验证输入日期是否合法

输入三个变量（年、月、日），函数返回布尔值，判断该日期是否合法：$1 \leq$ 月份 ≤ 12，$1 \leq$ 日期 ≤ 31，$2000 \leq$ 月份 ≤ 2019。

等价类划分表

输入及外部条件	有效等价类	等价类编号	无效等价类	等价类编号
输入的类型	数字字符	1	非数字字符	8
year	2000<=year<=2019	2	year<2000	9
			year>2019	10
month	1<=month<=12	3	month<1	11
			month>12	12
非闰年的2月	1<=day<=28	4	day<1	13
			day>28	14
闰年的2月	1<=day<=29	5	day<1	15
			day>29	16
month ∈ {1,3,5,7,8,10,12}	1<=day<=31	6	day<1	17
			day>31	18
month ∈ {4,6,9,11}	1<=day<=30	7	day<1	19
			day>30	20

测试用例

有效等价类

序号	输入数据			预期输出	覆盖范围（等价类编号）
	year	month	day		
1	2008	1	9	True	1, 2, 3, 6
2	2008	2	29	True	1, 2, 3, 5
3	2008	4	30	True	1, 2, 3, 7
4	2018	2	28	True	1, 2, 3, 4

所有的有效等价类被覆盖

测试用例

无效等价类

序号	输入数据			预期输出	覆盖范围（等价类编号）
	year	month	day		
1	x	2	29	False	8
2	1234	3	2	False	9
3	2222	3	2	False	10
4	2018	0	3	False	11
5	2018	13	1	False	12
6	2018	2	0	False	13
7	2018	2	29	False	14
8	2004	2	0	False	15
9	2004	2	30	False	16

测试用例

无效等价类（续）

序号	输入数据			预期输出	覆盖范围（等价类编号）
	year	month	day		
10	2018	3	0	False	17
11	2018	3	32	False	18
12	2018	4	0	False	19
13	2018	4	31	False	20

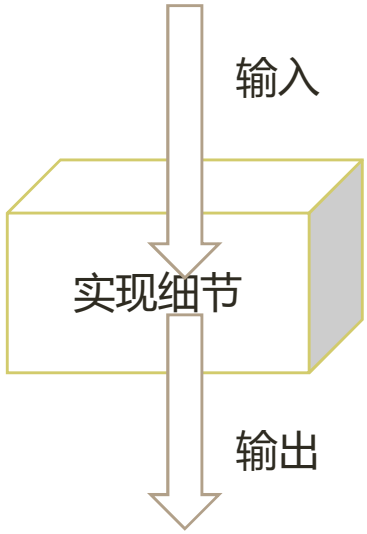
为每一个无效等价类设计
一个测试用例

白盒测试

关注软件产品的内部细节和逻辑结构。

方法：

- 逻辑覆盖测试方法
- 基本路径测试方法



逻辑覆盖法

根据覆盖的目标不同，可分为语句覆盖、分支覆盖、条件覆盖、分支-条件覆盖、条件组合覆盖、路径覆盖。

1.语句覆盖每条语句至少执行一次。

2.判定覆盖每个判定的每个分支至少执行一次。

3.条件覆盖每个判定的每个条件应取到各种可能的值。

4.判定/条件覆盖同时满足判定覆盖条件覆盖。

5.条件组合覆盖每个判定中各条件的每一种组合至少出现一次。

6.路径覆盖使程序中每一条可能的路径至少执行一次。

分支覆盖

基本思想：设计若干个测试用例，运行被测程序，使程序中的每个分支至少被执行一次。

以考察程序if-else结构为基础。

对循环结构，考察循环条件能够满足和不可能满足两种情况。

示例

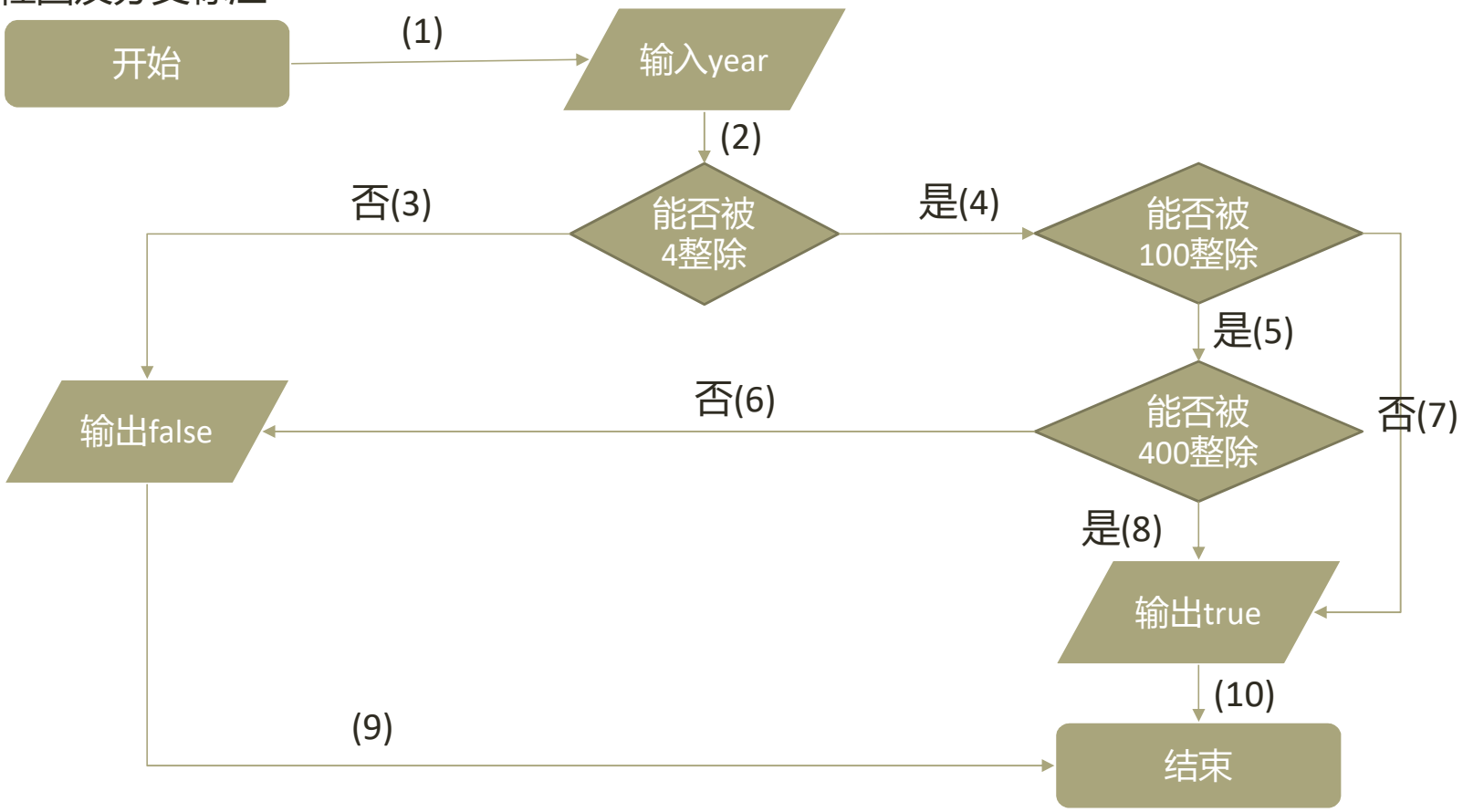
判断是否为闰年的函数isLeapYear(int year)。

闰年的条件是：

- 能被4整除，但不能被100整除；
- 能被100整除，又能被400整除。

示例

流程图及分支标注


开始
(1)
输入year
(2)
能否被
4整除
否(3)
是(4)
能否被
100整除
是(5)
能否被
400整除
否(6)
否(7)
输出false
是(8)
输出true
(10)
(9)
结束

示例

分支覆盖表及用例列表

输入	执行路径
2000	1, 2, 4, 5, 8, 10
2004	1, 2, 4, 7, 10
2006	1, 2, 3, 9
1900	1, 2, 4, 5, 6, 9, 10

单元测试

单元测试（Unit Testing）又称为模块测试, 是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作。

程序单元是应用的最小可测试部件。在过程化编程中，一个单元就是单个程序、函数、过程等；对于面向对象编程，最小单元就是方法，包括基类（超类）、抽象类、或者派生类（子类）中的方法

对软件基本组成单元进行的测试——基本单元不局限于一个具体的函数或类方法，也可以是一个类

单元测试

单元测试的目标是隔离程序部件并证明这些单个部件是正确的。一个单元测试提供了代码片断需要满足的严密的书面规约。

单元测试通常需要自动化，并且执行速度通常需要很快以确保能够常态化运行。（如果一个测试要很久，就会有人懒得执行它）