软件工程与实践

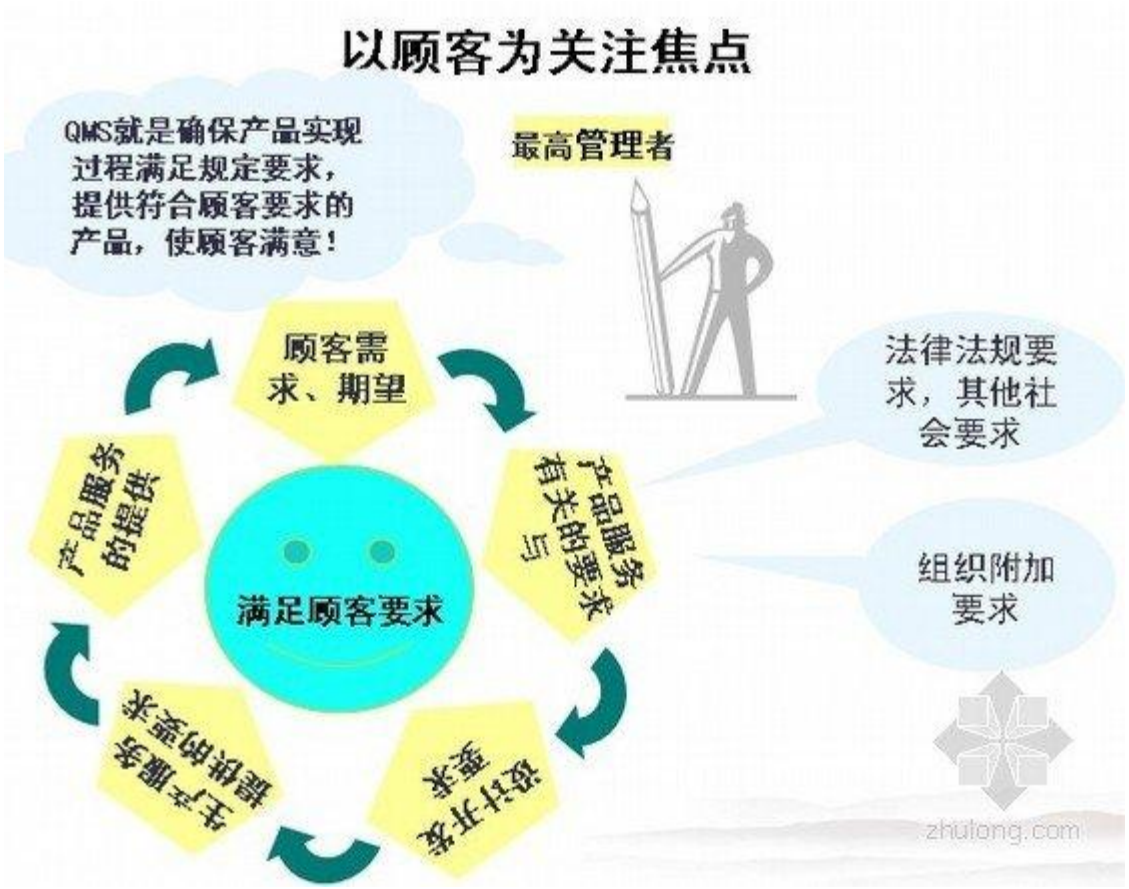
软件工程与实践课程组

电子科技大学信息与软件工程学院

- 6.1 软件质量保证和软件测试
- 6.2软件测试技术
- 6.3 软件测试策略

第六章：软件测试

介绍软件测试的概念，测试策略和测试的技术。



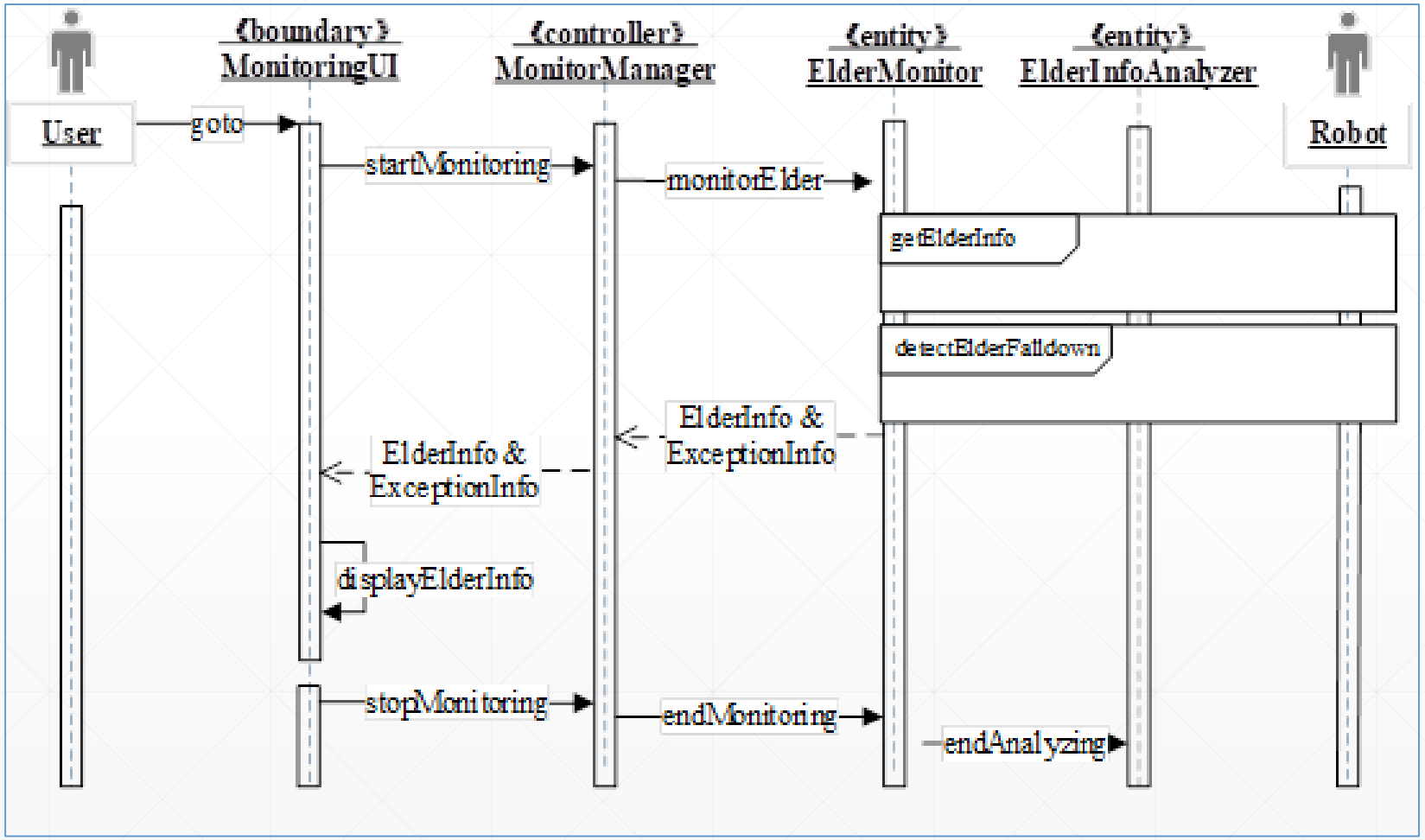
6.1.软件质量保证

- 软件质量相关概念

示例：软件中存在缺陷，导致软件失效

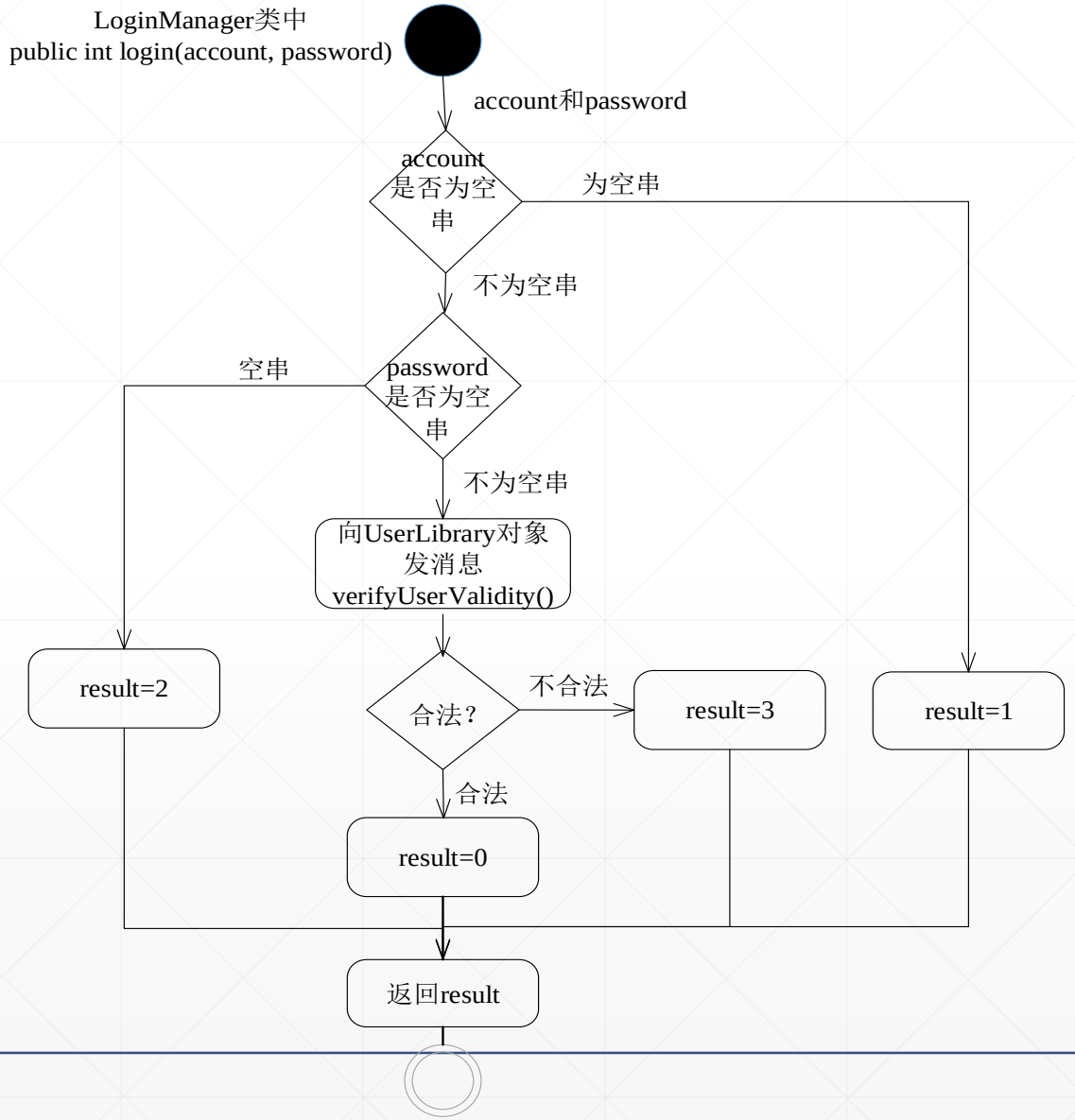
示例：软件中存在缺陷，导致软件失效

示例：软件需求中的潜在问题



- 是否正确描述需求?
- 是否一致描述需求?
- 是否存在描述错误?

示例：软件设计中的潜在问题



- 是否正确实现需求?
- 是否存在设计错误?

示例：程序代码中的潜在问题

Notes × Contact.java ×

```
28 public class Contact {
29     private static HashMap<String, String> sContactCache;
30     private static final String TAG = "Contact";
31
32     private static final String CALLER_ID_SELECTION = "PHONE_NUMBERS_EQUAL(" +
   Phone.NUMBER
33     + ",?) AND " + Data.MIMETYPE + "='" + Phone.CONTENT_ITEM_TYPE + "'"
34     + " AND " + Data.RAW_CONTACT_ID + " IN "
35             + "(SELECT raw_contact_id "
36             + " FROM phone_lookup"
37             + " WHERE min_match = '+')";
38
39     public static String getContact(Context context, String phoneNumber) {
40         if(sContactCache == null) {
41             sContactCache = new HashMap<String, String>();
42         }
43
44         if(sContactCache.containsKey(phoneNumber)) {
45             return sContactCache.get(phoneNumber);
46         }
```

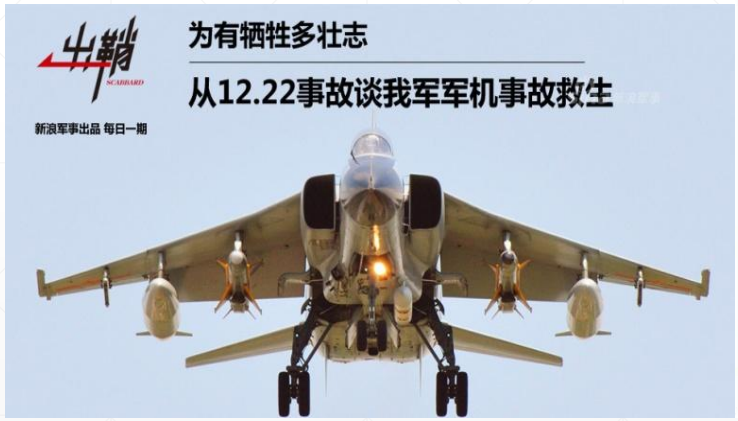
- 是否正确实现功能?
- 是否存在内在缺陷?

需求和设计问题会带到代码中，编码时也会引入问题

软件缺陷的危害

- 无法满足要求
- 不能正常工作
- 引发安全事故
- 影响人员安全
- 产生经济损失
- ……

波音737
MAX事件



原软件用于教练飞机某参数屏显范围±100，重用于新战斗机，
该参数屏显范围应该为±300！

尽可能减少软件缺陷非常重要

软件缺陷不可避免

- 人总是会犯错误的
  - 软件工程师、用户等
  - 软件系统太复杂
- 程序缺陷来自多个源头
  - 需求、设计、编码活动
  - 模型、文档、程序制品
- 缺陷成常态化
  - 对于复杂软件系统而言缺陷不可避免
  - 很难做到无缺陷的软件

你所使用的软件中是否发现有缺陷?

思考和讨论

- 如何应对软件缺陷?
  - 如何避免和减少缺陷?
  - 如何发现和修复缺陷?

如何应对?

- 能否不犯和少犯错误
- 如何发现缺陷
- 如何纠正缺陷
- ……

软件质量保证
(Software Quality Assurance)

软件测试
(Software Testing)

方法之一：找到软件缺陷，以排除缺陷

6.1.1 软件质量相关概念——软件质量

软件质量：明确表示是否符合功能和性能要求，明确地记载开发标准和所有专业开发软件的期望的隐性特点

- **关** 符合明确规定的功能和性能要求
- **键** 符合明确的开发标准
- **点** 符合所有软件开发专业的共性、隐性标准，如易用性、可维护性等

6.1.1 软件质量相关概念——软件质量保证

软件质量保证(SQA)：遵照一定的软件生产标准、过程和步骤对软件质量进行评估的活动。

审查

- 评审既定标准是否得到遵守。如IEEE、ISO、GB/T等

监督

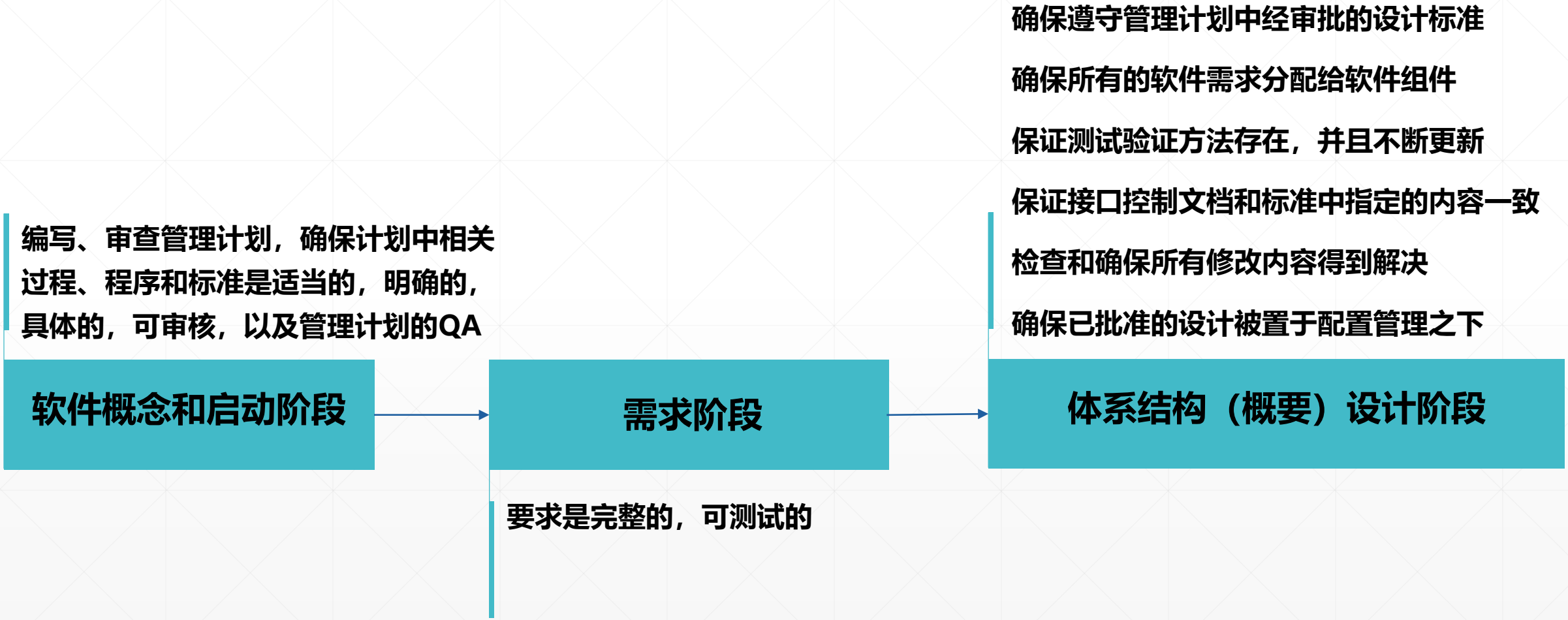
- 对比文档中描述的执行和实际操作步骤，确保执行过程采取适当步骤和操作方式

审计

- 确保开发过程使用了恰当的质量控制措施，以符合相应的标准或过程。

6.1.1 软件质量相关概念——软件质量保证

软件质量保证(SQA)活动



6.1.1 软件质量相关概念——软件质量保证

软件质量保证(SQA)活动

确保批准的设计标准得到遵守
保证分配的模块在详细设计中
确保所有修改内容得到解决

详细设计阶段

实施阶段

设计与构建相一致
所有交付项目的状态
配置管理活动和软件开发库
不符合项报告和纠正措施系统

确保为所有交付项目进行测试
测试计划和程序有效执行，问题解决与报告
保证测试报告是完整和正确的
验证测试已经完成，软件和文件准备交付
参与测试前再审，并保证所有行动项目完成

集成和测试阶段

保证最终产品的性能，及所有
交付材料齐备

验收和交付阶段

支持工程和操作阶段

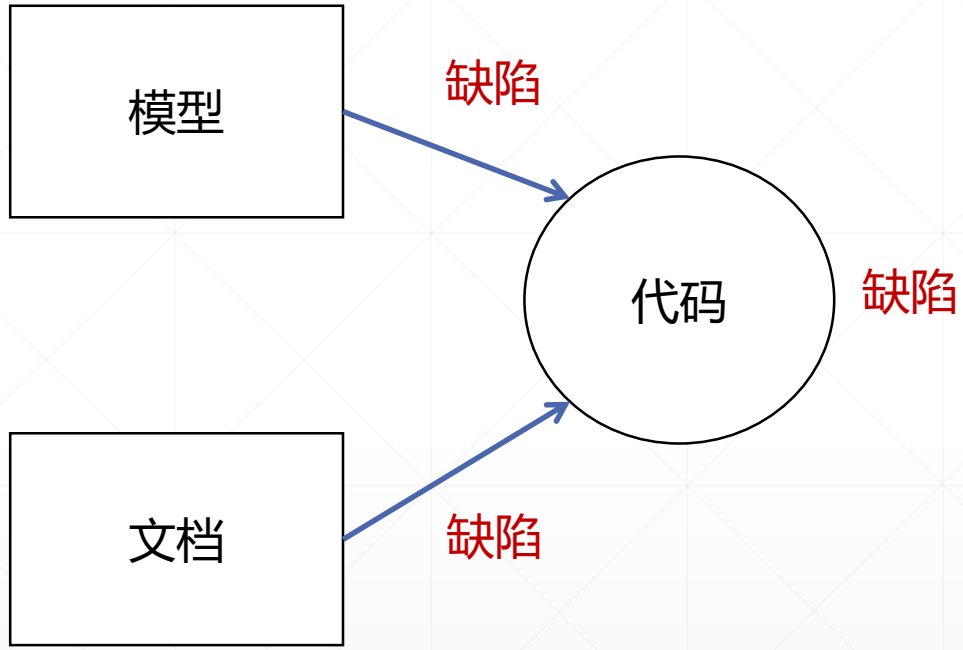
使用较短开发周期来升级和更正软件

6.1.2 何为软件测试?

- **运行软件**或模拟软件的执行，**发现软件缺陷**的过程
- 注意点
  - 软件测试通过运行程序代码的方式来发现程序代码中潜藏的缺陷，这和代码走查、静态分析形成鲜明对比。
  - 软件测试的目的是为了发现软件中的缺陷。它只负责发现缺陷，不负责修复和纠正缺陷

在程序代码中找出软件缺陷

- **程序**是运行软件的载体
  - 通过执行代码运行软件
  - 通过软件运行发现缺陷
- **程序**是软件缺陷的载体
  - 缺陷分布在模型、文档和代码中
  - 最终会反映在程序代码上

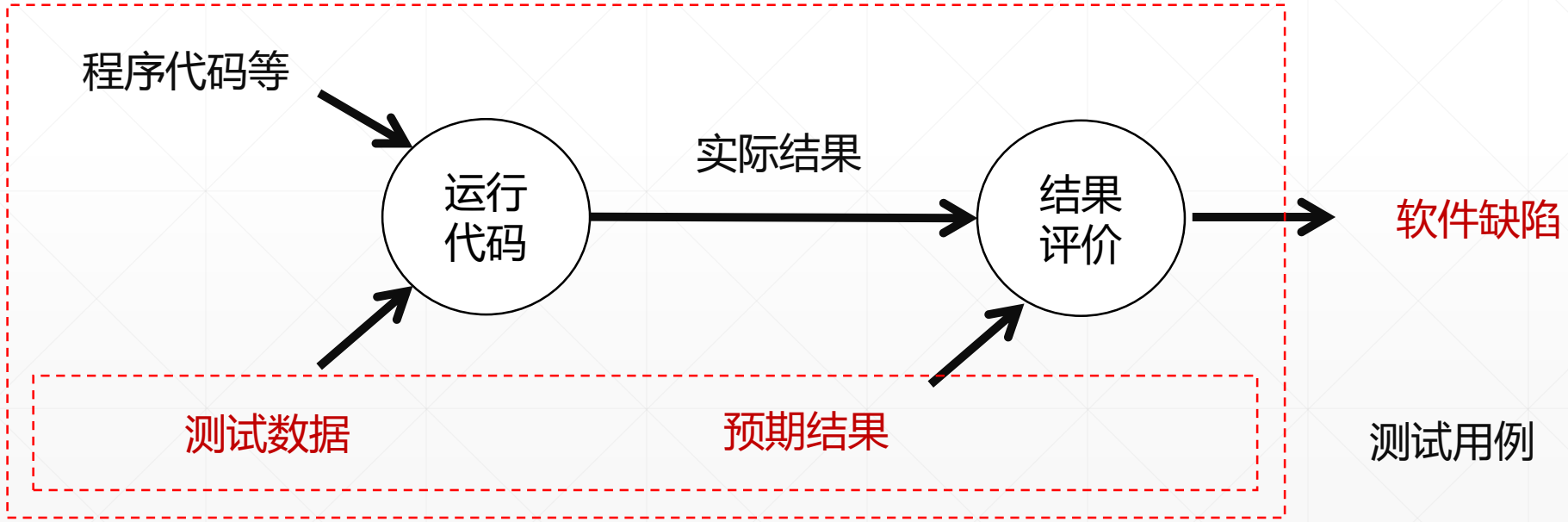


思考和讨论

- 如何从程序代码中找出软件缺陷?

6.1.3 软件测试的原理

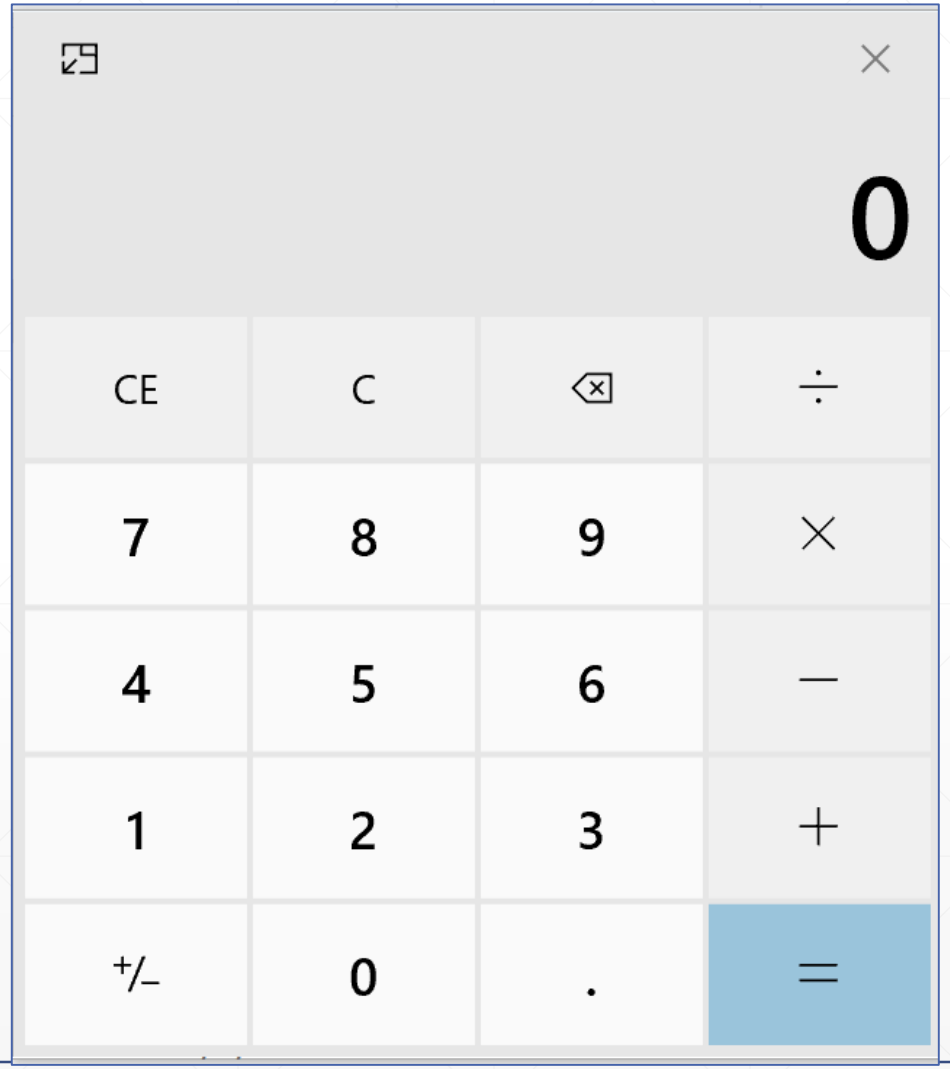
- **程序本质上是对数据的处理**
  - **设计数据(测试用例) ➔ 运行测试用例(程序来处理数据) ➔ 判断运行结果(是否符合预期结果)**



为软件测试而设计的数据称为测试用例(Test Case)

示例：软件测试的原理

- 加法的功能
- A = B + C
- 测试用例
  - 测试数据 1, 2
  - 预期结果 3



测试用例

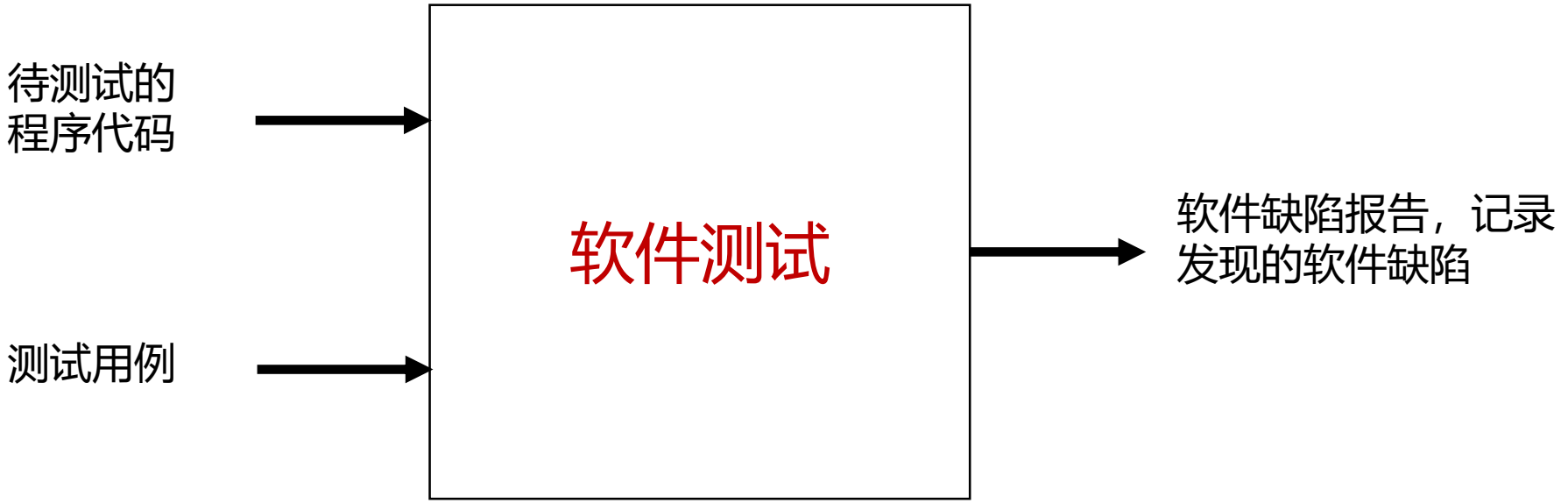
- 测试用例是一个四元偶
  - **输入数据**：交由待测试程序代码进行处理的数据
  - **前置条件**：程序处理输入数据的运行上下文，即要满足前置条件
  - **测试步骤**：程序代码对输入数据的处理可能涉及到一系列的步骤，其中的某些步骤需要用户的进一步输入
  - **预期输出**：程序代码的预期输出结果

示例：测试用例的设计

- “用户登录”模块单元的测试用例设计
  - **输入数据**：用户账号=“admin”，用户密码=“1234”
  - **前置条件**：用户账号“admin”是一个尚未注册的非法账号，也即“T_User”表中没有名为“admin”的用户账号。
  - **测试步骤**：首先清除“T_User”表中名为“admin”的用户账号？；其次用户输入“admin”账号和“1234”密码；第三，用户点击界面的确认按钮；最后，系统提示“用户无法登录系统”的信息
  - **预期输出**：系统将提示“用户无法登录系统”的提示信息

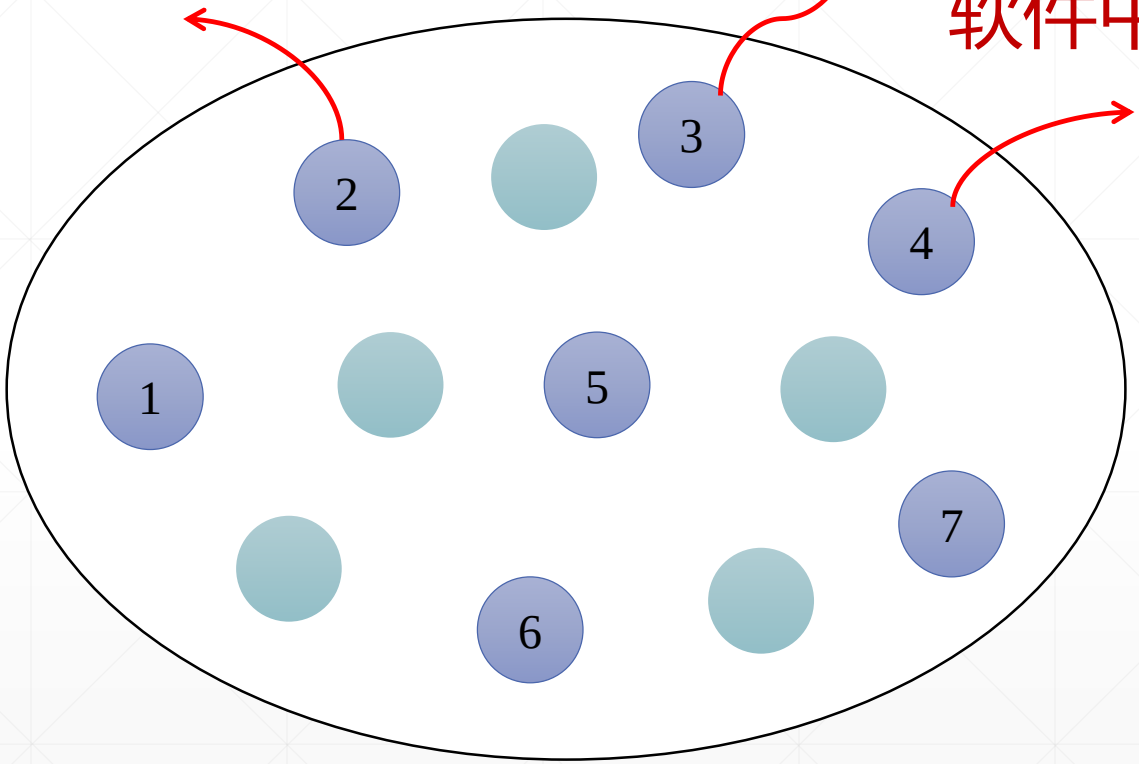
6.1.4 软件测试的任务

➢软件测试的输入和输出



软件测试任务

尽可能多找出
软件中缺陷



软件缺陷可能隐藏的比较深，难以发现

缺陷

软件测试能把软件中的所有缺陷都找出来吗？

软件测试的目的

- 目的
  - 发现软件中的缺陷
  - 最大限度、尽可能多的找到缺陷
- 功效
  - 发现的缺陷越多 → 软件中遗留的缺陷越少
    → 交付的软件质量越高 → 后期维护工作量就越少

思考和讨论

- 软件测试没有发现缺陷是否意味着软件就没有缺陷？
- 为什么?
- 软件测试能够用于证明软件无缺陷吗?

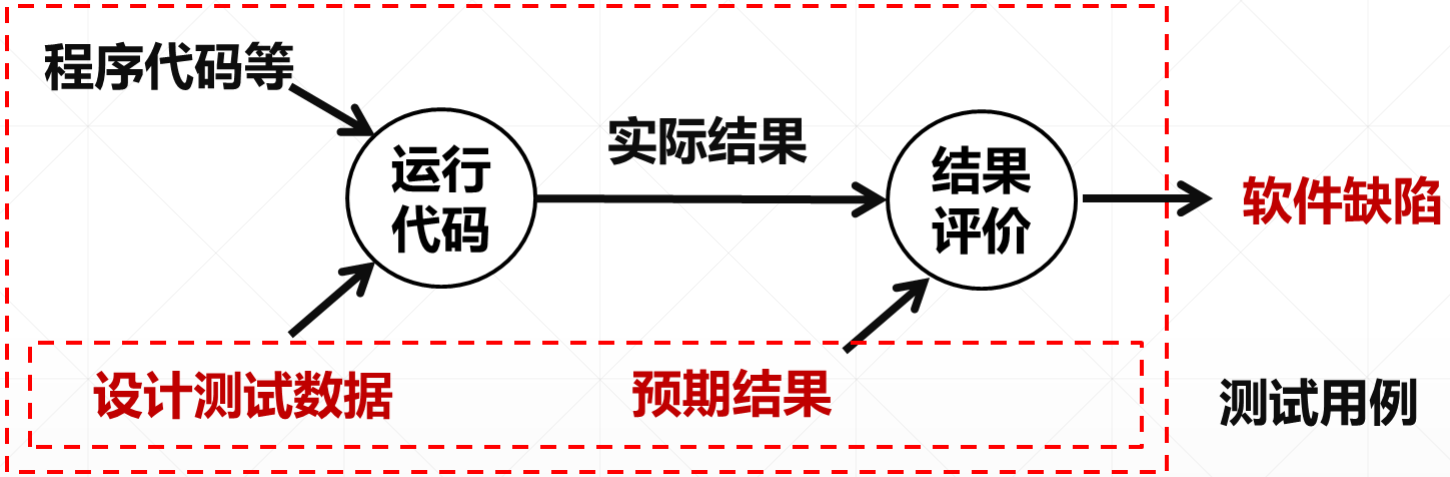
6.1.5 软件测试的步骤

① 明确待测试对象
- 什么粒度的程序代码

② 设计测试用例
- {<Data, Result>}
- 可能有许多

③ 运行代码和测试用例
- 输入和处理测试用例

④ 分析运行结果
- 对比运行结果和预期结果，发现问题和缺陷



示例: 软件测试的步骤

- 明确测试对象
  - 完成用户注册功能的程序模块（类）
- 设计测试用例
  - 输入：UserID = xjmao, Psw=se
  - 预期：在数据库中有该用户密码的数据条目
- 运行测试用例
  - 运行程序，输入数据
- 分析运行结果
  - 用户数据库表中是否有该新用户的密码信息

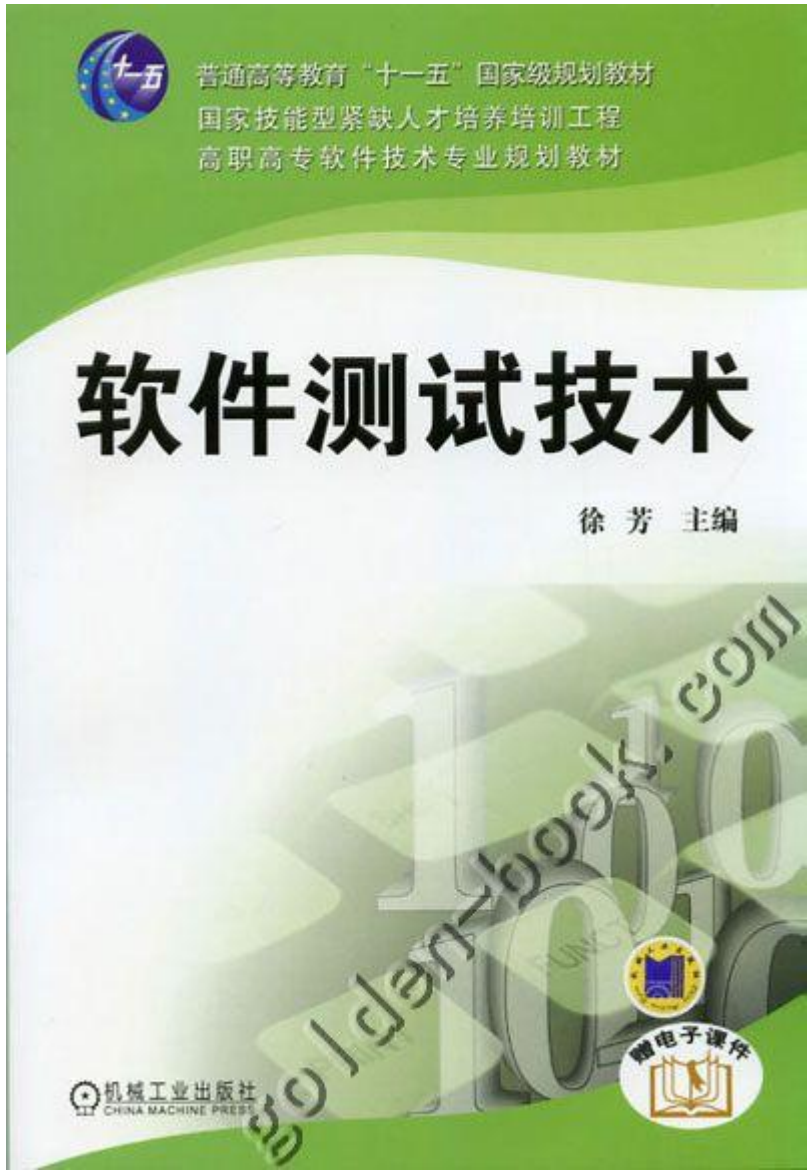
6.1.6 软件测试面临的主要挑战

- 设计测试用例
  - C1: 如何设计有效的测试用例？提高软件测试的质量
  - C2: 如何确保测试用例的合理性：尽可能地发现缺陷?
- 发现程序缺陷
  - C3: 如何运行程序和用例来发现缺陷?
  - C4: 如何采用工具来自动地发现缺陷：提高测试的效率?

软件测试的前提和关键是要设计出有效的测试用例

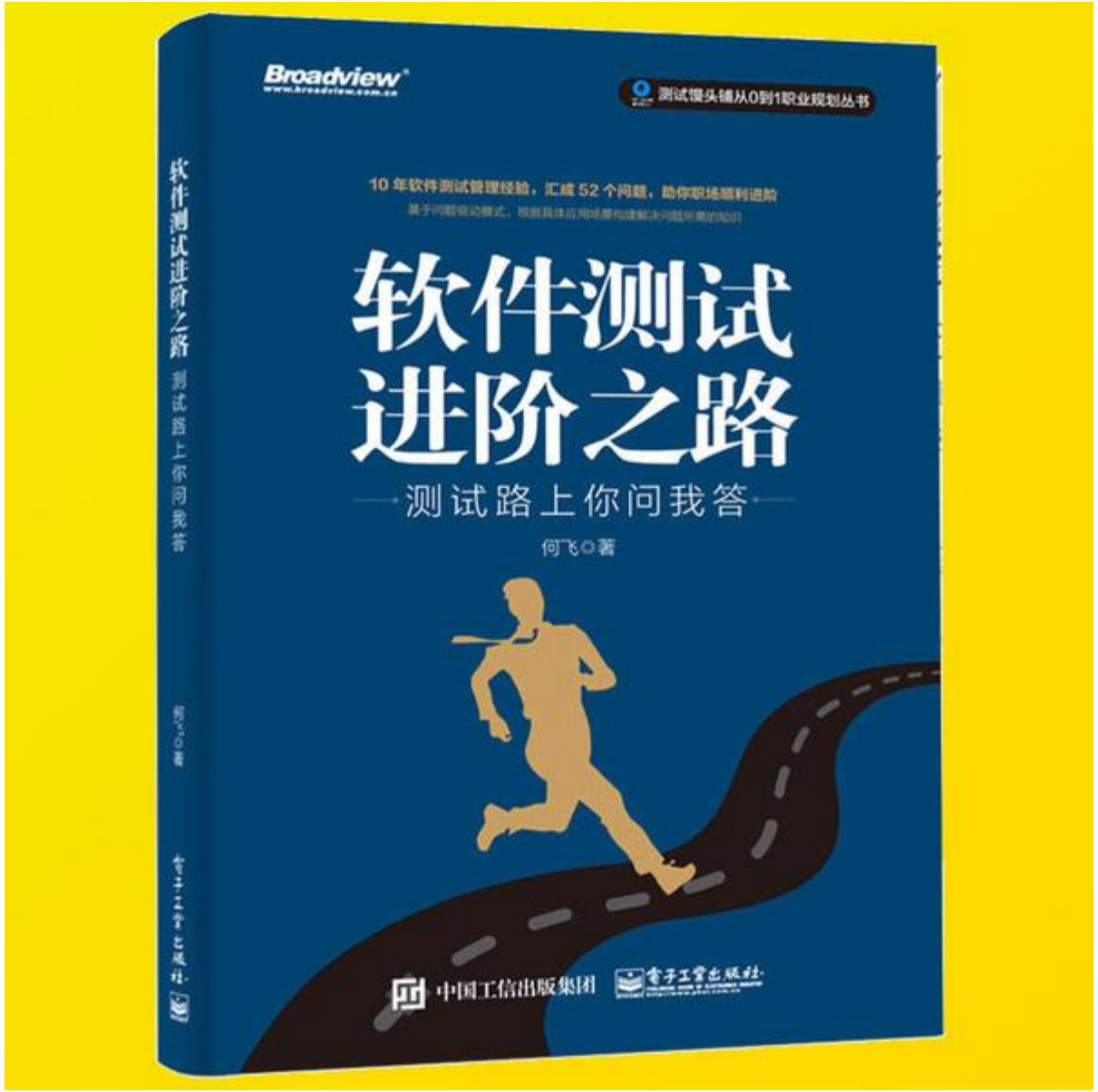
6.1.7 软件测试工程师

- 负责软件系统的测试工作，发现软件系统中的缺陷，协助软件开发工程师定位和修复缺陷
- 服务于软件开发工程师，帮助他们理解和解决软件中的缺陷
- 充当客户的技术代表，帮助客户发现软件中存在的各类问题，通过测试来演练客户验收



6.2 软件测试技术

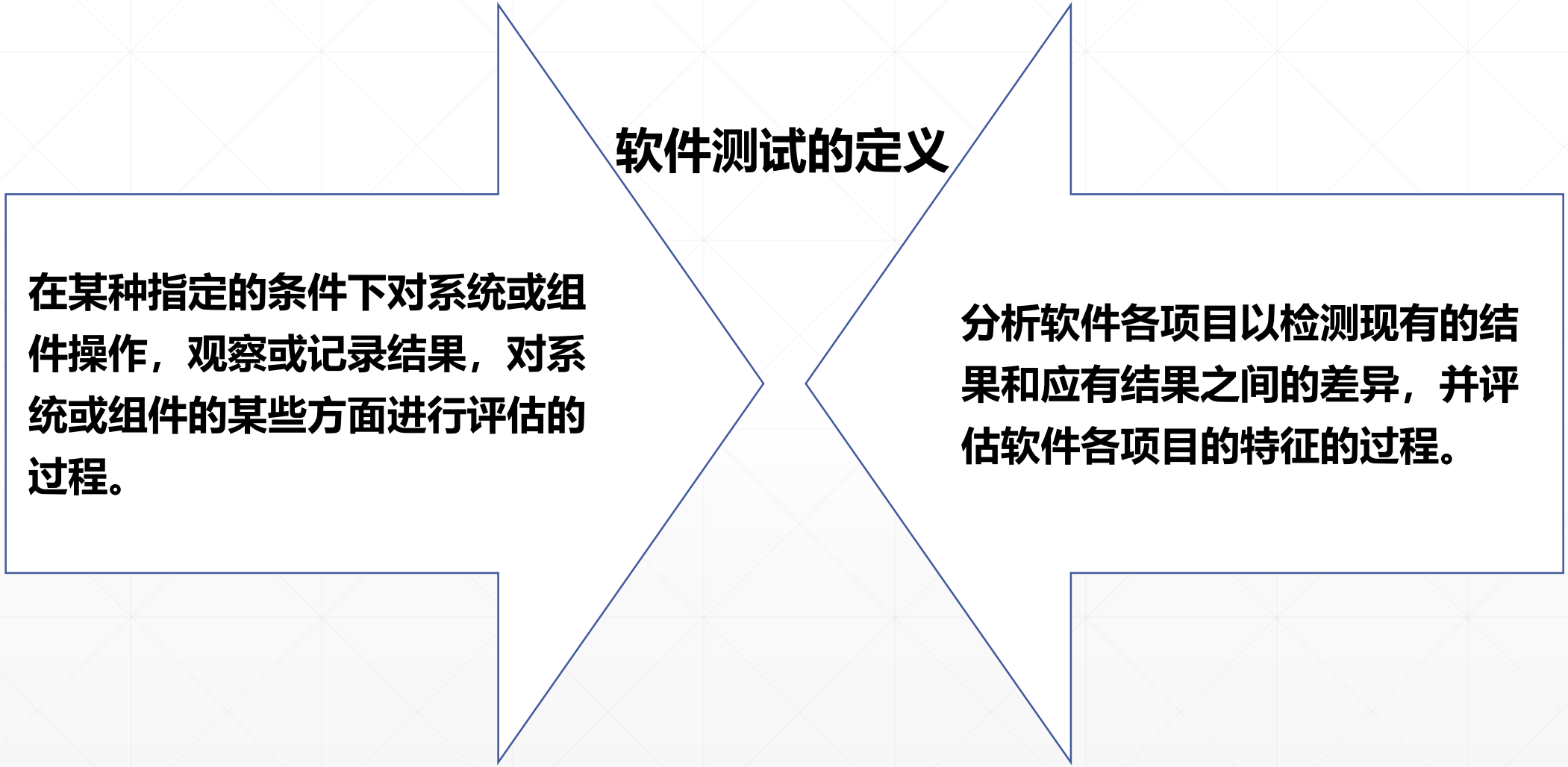
- 软件测试技术相关概念
- 白盒测试
- 黑盒测试



6.2.1软件测试技术——相关概念

- 相关概念
- 目的与原则
- 评估准则
- 主要测试方法

1）相关概念——软件测试定义



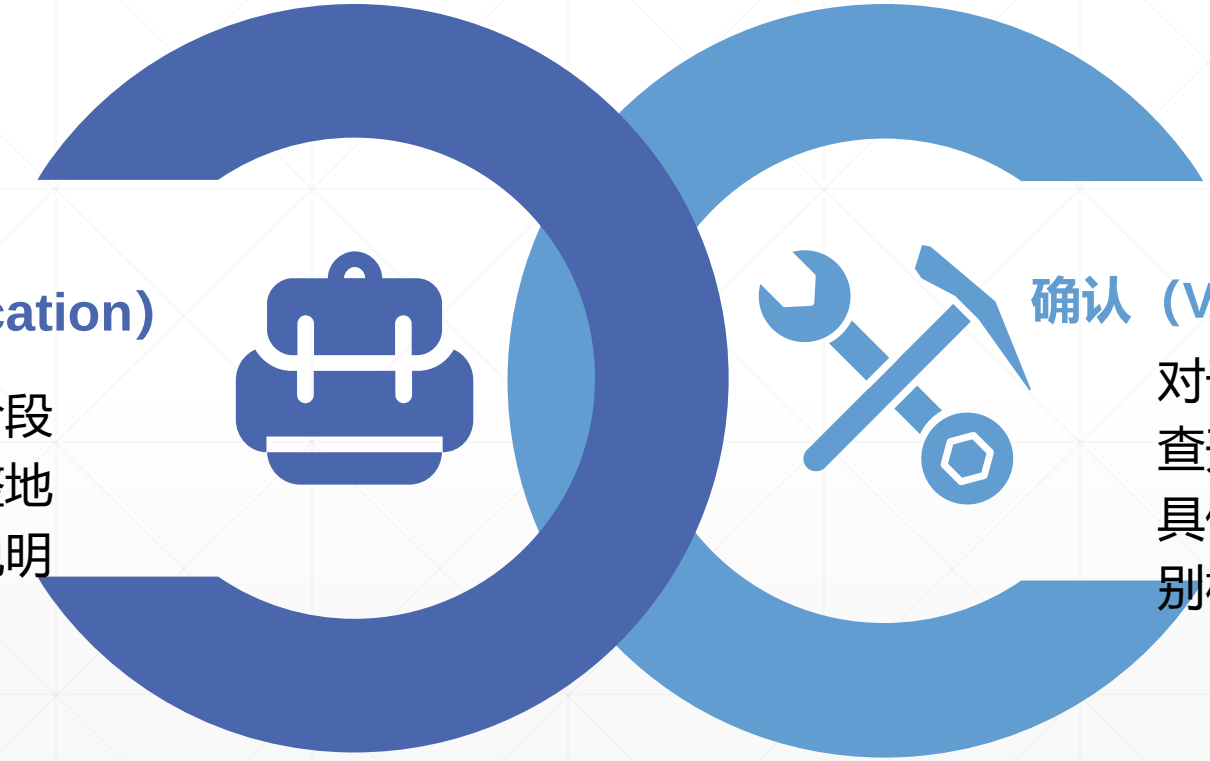
1）相关概念——软件缺陷

软件缺陷	软件未实现产品说明书要求的功能。
	软件出现了产品说明书指明不能出现的错误。
	软件实现了产品说明书未提到的功能。
	软件未实现产品说明书虽未明确提及但应该实现的目标。
	软件难以理解、不易使用、运行缓慢或者——从测试员的角度看——最终用户会认为不好。

1）相关概念——验证与确认

验证（Verification）

保证软件特定开发阶段的输出已经正确完整地实现了规格说明

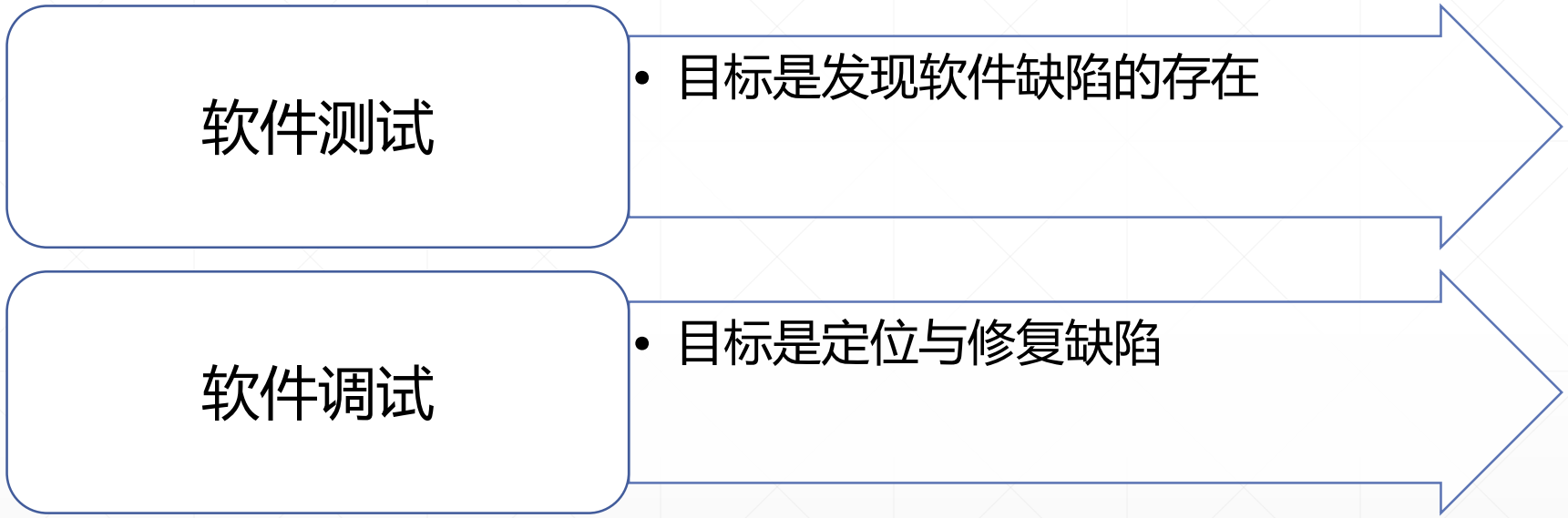


确认（Validation）

对于每个测试级别，都要检查开发活动的输出是否满足具体的需求或与这些特定级别相关的需求

1）相关概念——质量与可靠性

1）相关概念——测试与调试



1）相关概念——测试用例

测试用例（test case）：是测试输入、执行条件、以及预期结果的集合，是为特定的目的开发的，例如执行特定的程序路径或验证与指定的需求相符合。

输入条件	执行条件	预期结果
用户名 =yiyh，密码为空	输入用户名称，按 “登陆” 按钮。	显示警告信息 “请输入用户名和密码！”

输入条件	预期结果	实际结果
典型值		

主题。设计者、类型、测试名称、状态、描述、优先级、comment、步骤名、步骤描述、预期结果、评审人、评审备注、评审时间等……

2）目标与原则——目标

软件测试的目标

- 确认系统满足其预期的使用和用户的需要。
- 确认解决了所需解决的问题
- 为测试的过程建立责任和可解释性。
- 便于及早发现软件和系统的异常。
- 及早提供软件和系统的性能的评估。
- 为管理提供真实信息，以决定当前状态下发布产品在商业上的风险
- 鉴别出程序在功能等方面的异常集聚之处。

3）目标与原则——原则

软件测试的基本原则

穷尽测试是不可能的

测试无法显示潜伏的软件缺陷

测试活动应尽早进行

软件缺陷具有群聚性

注意“杀虫剂”现象

尽量由独立的测试团队进行测试

4）主要测试方法

黑盒测试

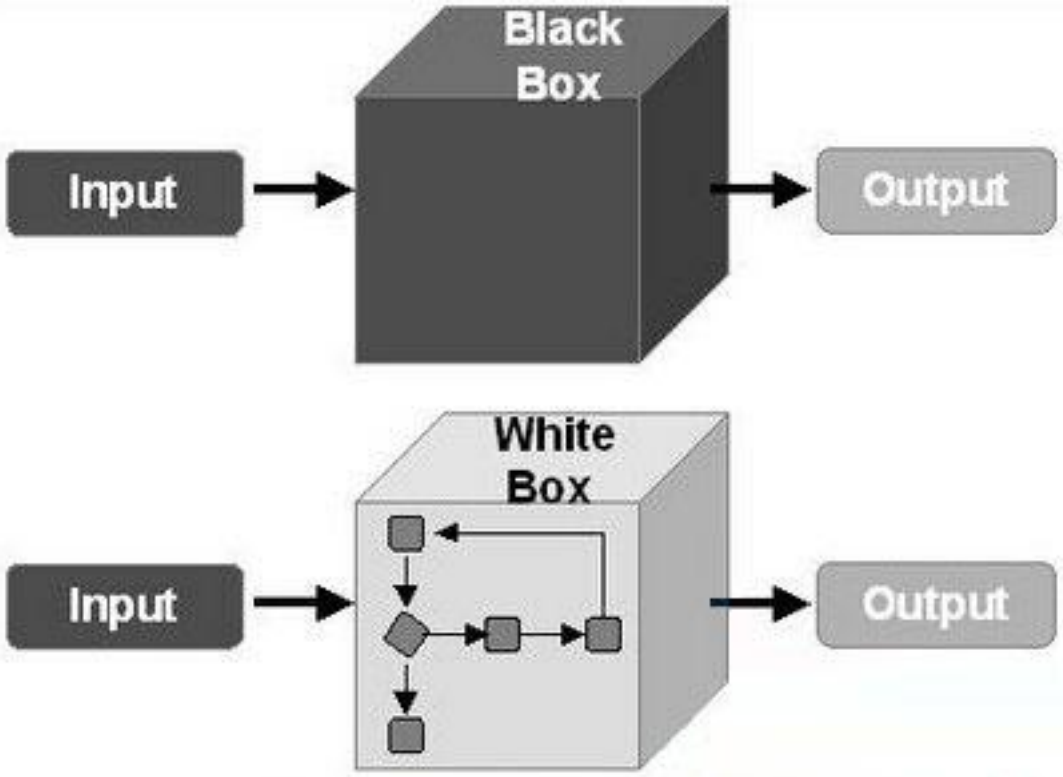
忽略系统或组件的内部机制，仅关注于那些响应所选择的输入及相应执行条件的输出的测试形式

白盒测试

考虑系统或组件的内部机制的测试形式

灰盒测试

介于白盒测试与黑盒测试之间的一种测试，多用于集成测试阶段，不仅关注输出、输入的正确性，同时也关注程序内部的情况。



6.2.2.软件测试技术——白盒测试

- 白盒测试概念
- 语句覆盖
- 分支覆盖
- 条件覆盖
- 条件组合覆盖

1）白盒测试——概念

把测试对象看做一个透明盒子，允许利用程序内部逻辑结构及有关信息，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。

又称为结构测试或逻辑驱动测试。

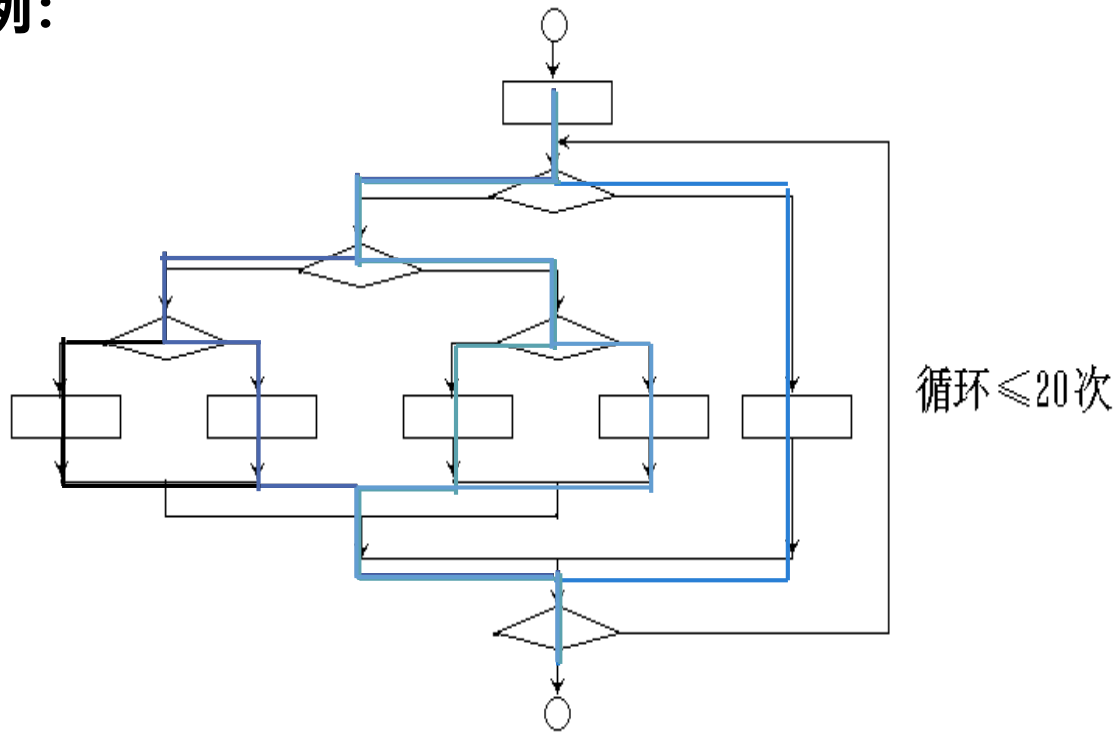
1）白盒测试——检查范围

检查范围	对程序模块的所有独立的执行路径至少测试一次；
	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；
	在循环的边界和运行界限内执行循环体；
	测试内部数据结构的有效性等。

1）白盒测试——完全测试的困难性

完全测试的困难性：对一个具有多重选择和循环嵌套的程序，不同的路径数目可能是天文数字。

例：



执行路径数：5^{20}条

设：

每一条路径测试需要1毫秒

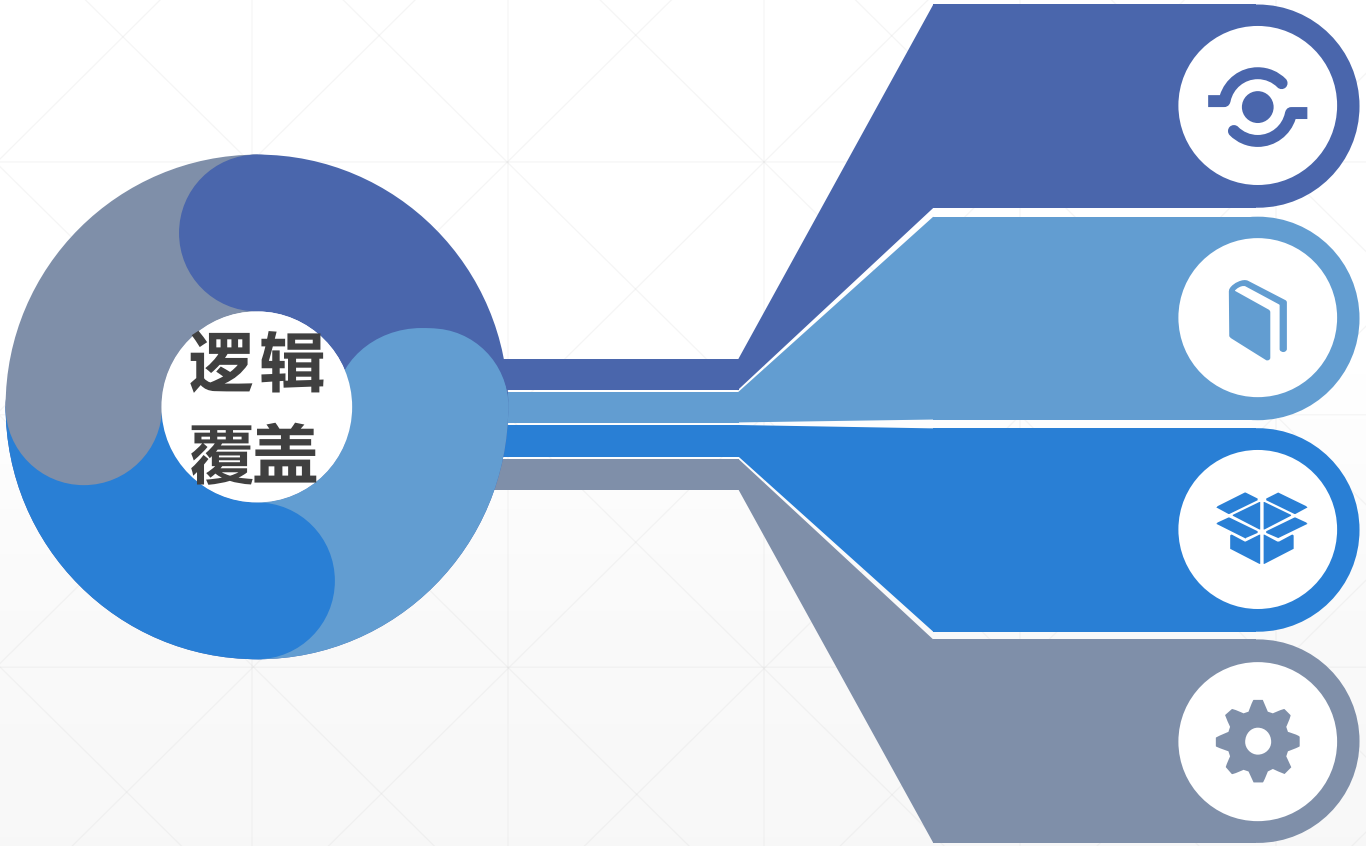
一年工作365 × 24小时

需：

3170年。

2）语句覆盖——概念

逻辑覆盖：以程序内部的逻辑结构为基础设计测试用例的技术。



语句覆盖

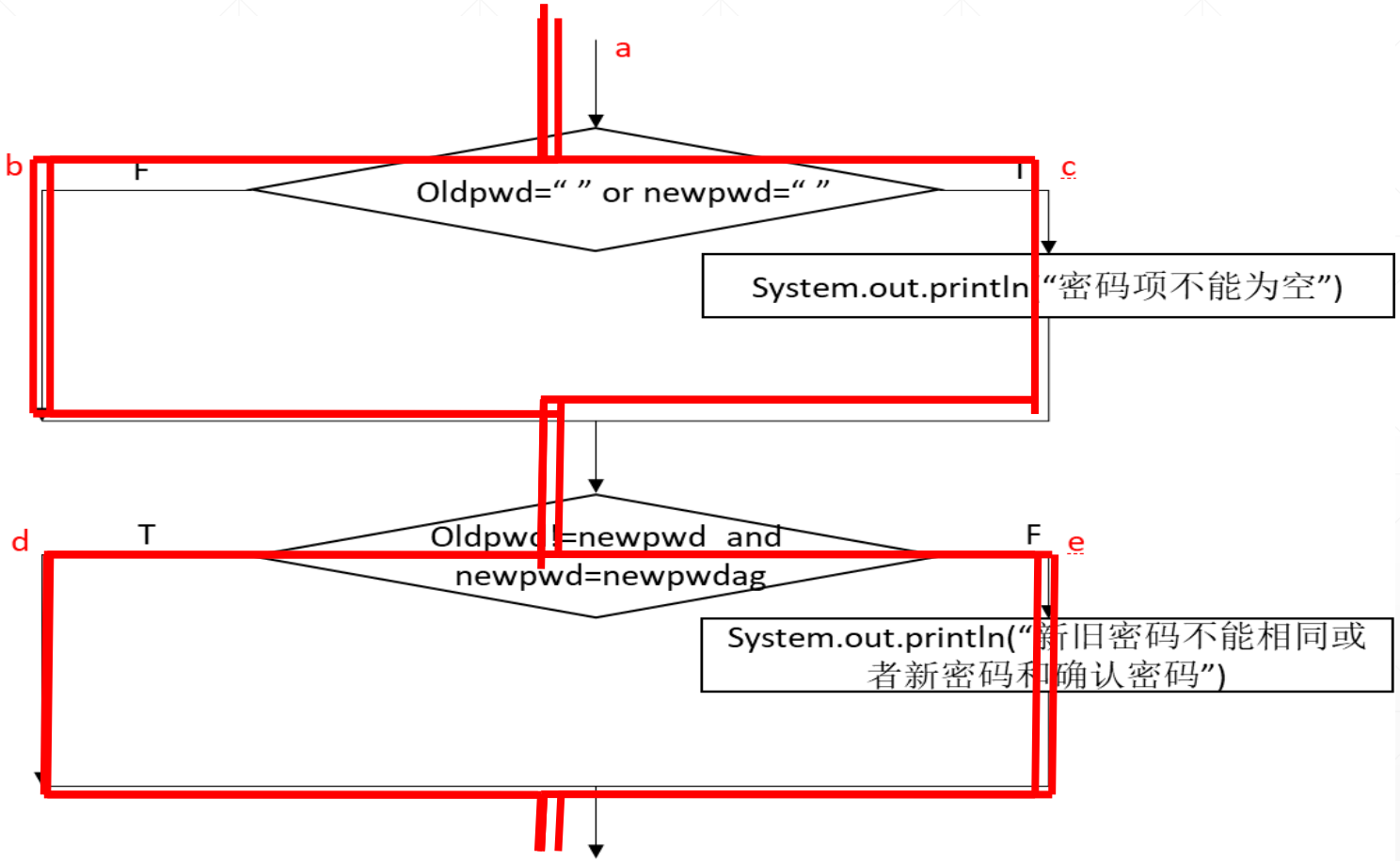
分支覆盖

条件覆盖

条件组合覆盖

语句覆盖

程序流程图示例



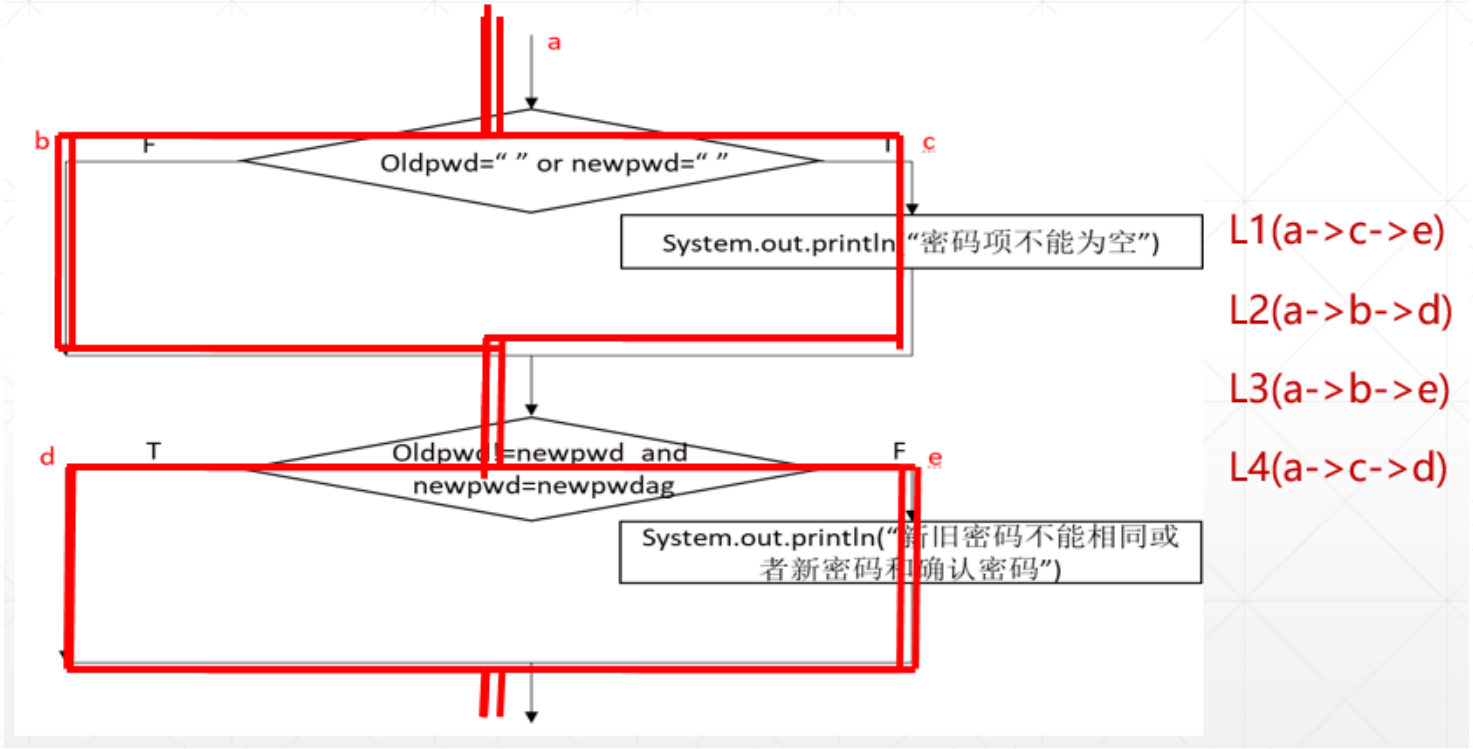
L1(a->c->e)

L2(a->b->d)

L3(a->b->e)

L4(a->c->d)

语句覆盖



$L1(a \rightarrow c \rightarrow e)$

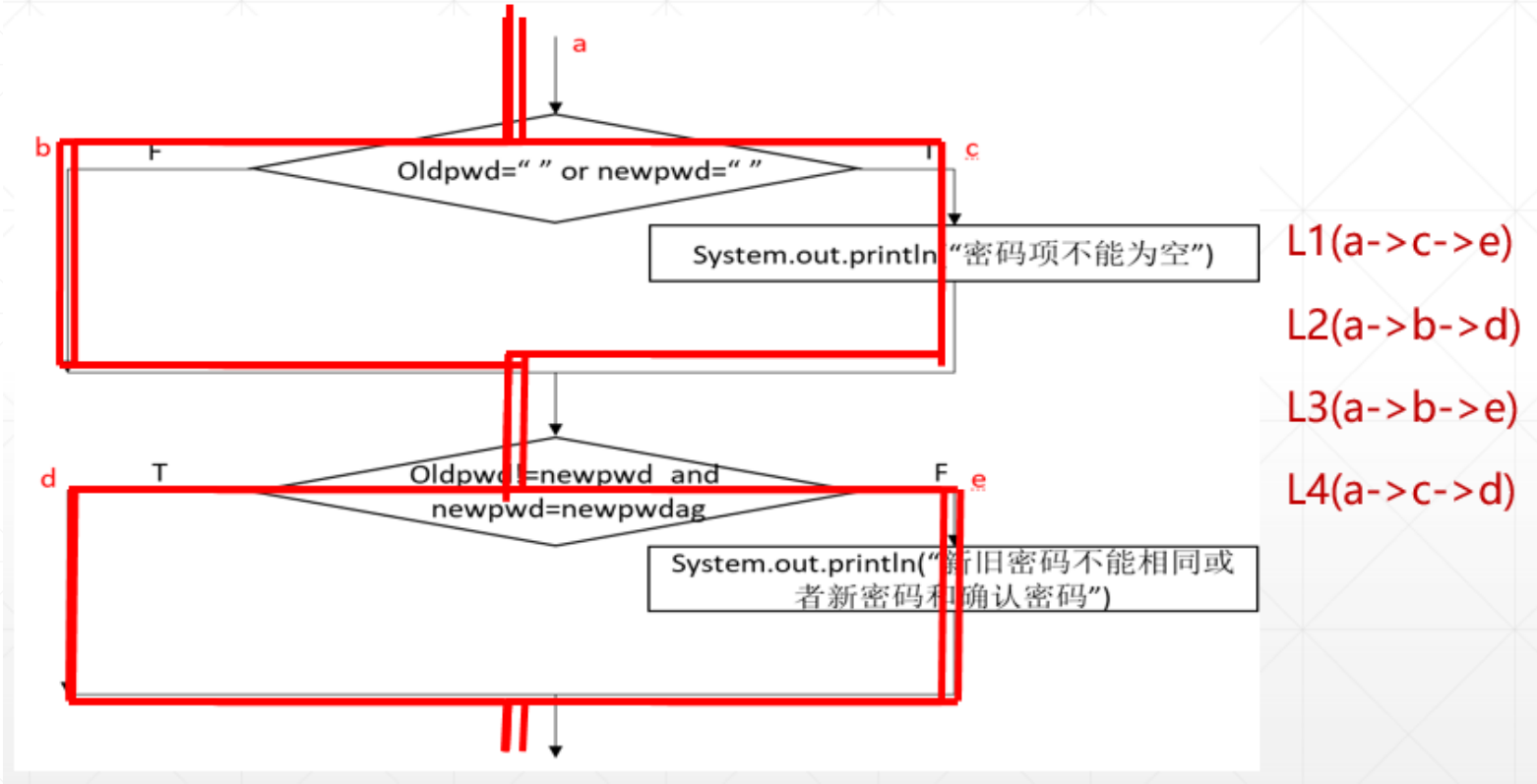
$= \{oldpwd = "" \text{ or } newpwd = ""\} \text{and } \{oldpwd != newpwd \text{ and } newpwd = newpwdag\}$

$= \{oldpwd = "" \text{ or } newpwd = ""\} \text{ and } \{oldpwd = newpwd \text{ or } newpwd != newpwdag\}$

$= oldpwd = "" \text{ and } oldpwd = newpwd \text{ or } oldpwd = "" \text{ and } newpwd != newpwdag$

$\text{or } newpwd = "" \text{ and } oldpwd = newpwd \text{ or } newpwd = "" \text{ and } newpwd != newpwdag$

语句覆盖



L1(a->c->e)

L2(a->b->d)

L3(a->b->e)

L4(a->c->d)

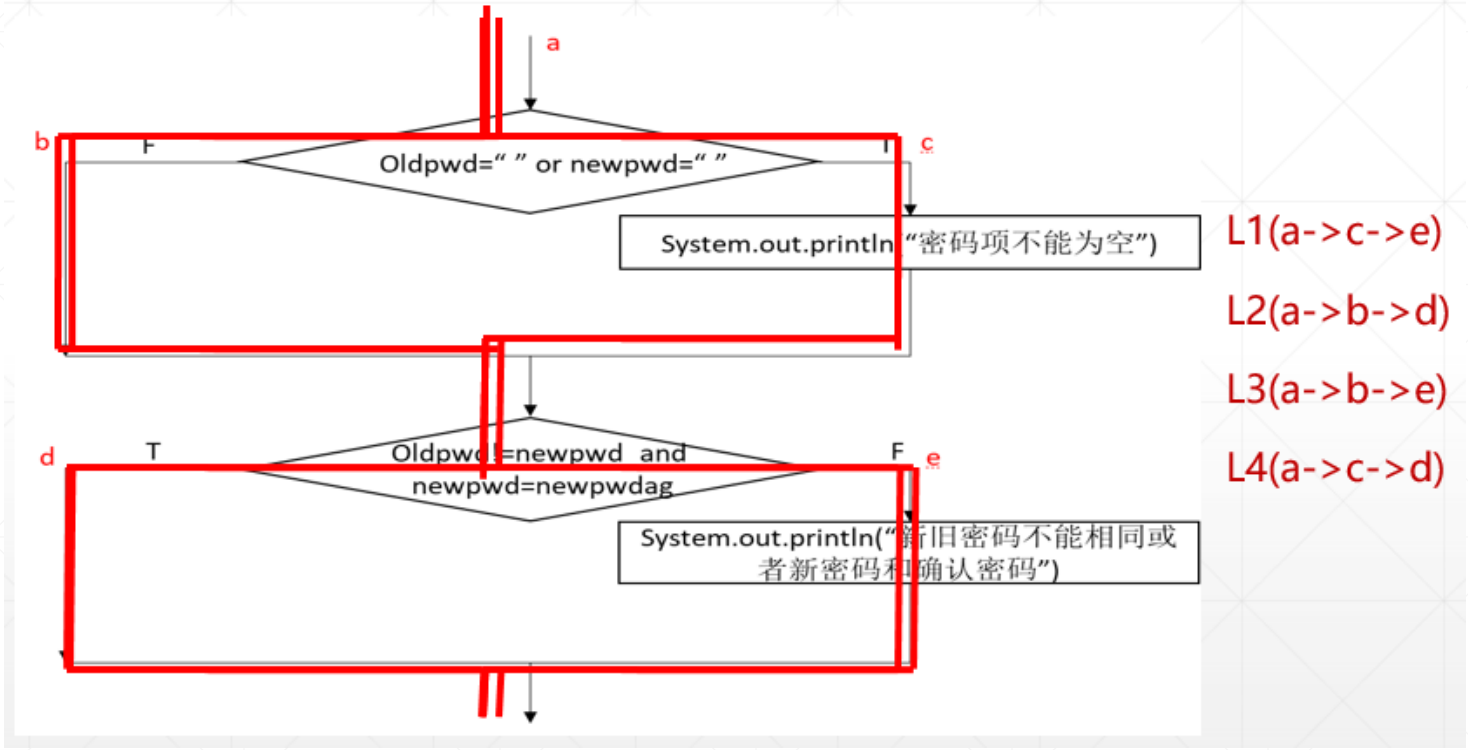
$$L2(a \rightarrow b \rightarrow d)$$

$$= \overline{\{oldpwd = "" \text{ or } newpwd = ""\}} \text{ and } \{Oldpwd != newpwd \text{ and } newpwd = newpwdag\}$$

$$= \{oldpwd != "" \text{ and } newpwd != ""\} \text{ and } \{Oldpwd != newpwd \text{ and } newpwd = newpwdag\}$$

$$= oldpwd != "" \text{ and } newpwd != "" \text{ and } Oldpwd != newpwd \text{ and } newpwd = newpwdag$$

语句覆盖



L3(a → b → e)

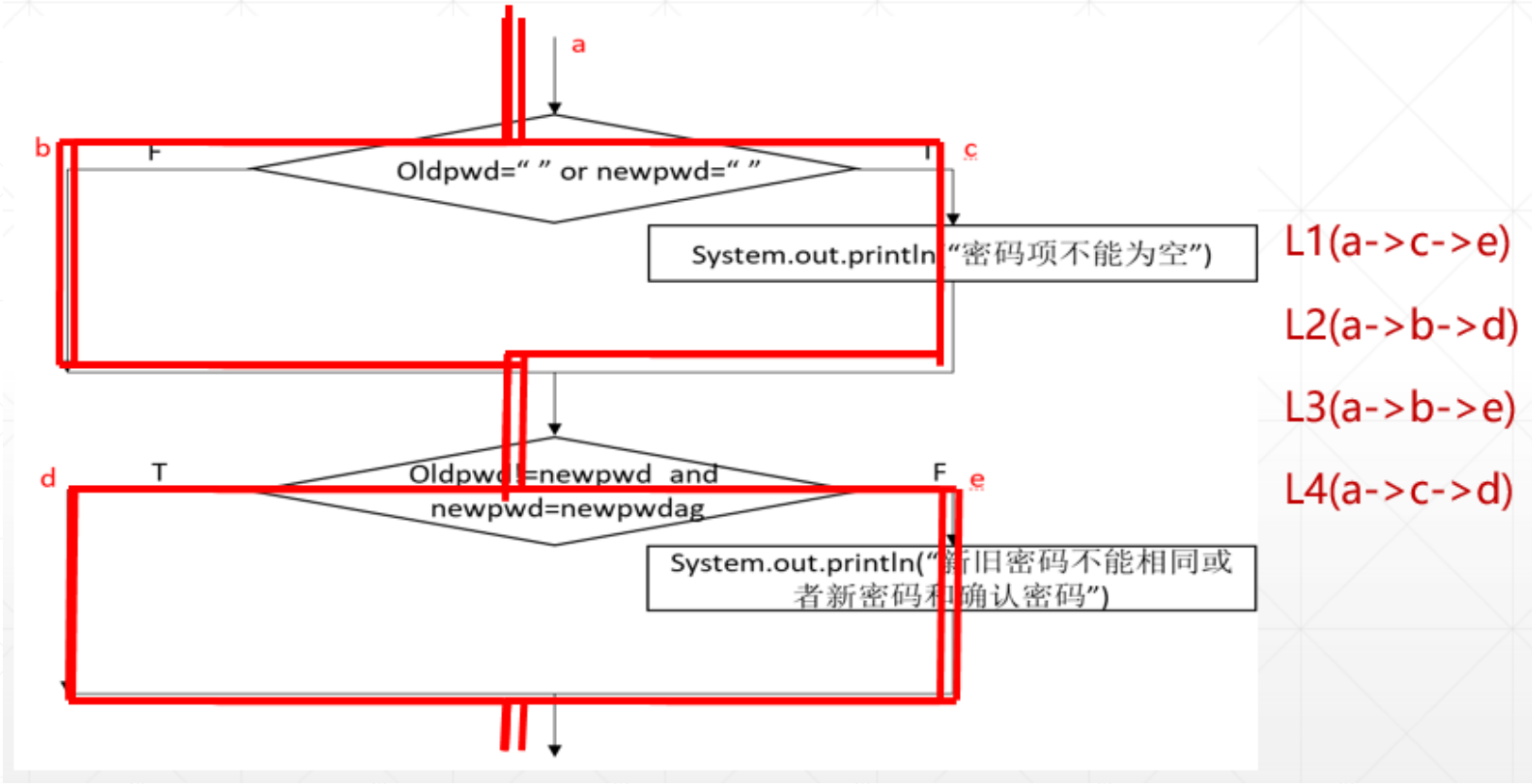
$= \overline{\{oldpwd = "" \text{ or } newpwd = ""\}} \text{ and } \overline{\{oldpwd != newpwd \text{ and } newpwd = newpwdag}$

$= \{oldpwd != "" \text{ and } newpwd != ""\} \text{ and } \{oldpwd = newpwd \text{ or } newpwd != newpwdag\}$

$= oldpwd != "" \text{ and } newpwd != "" \text{ and } oldpwd = newpwd \text{ or}$

$oldpwd != "" \text{ and } newpwd != "" \text{ and } newpwd != newpwdag$

语句覆盖



L4(a → c → d)

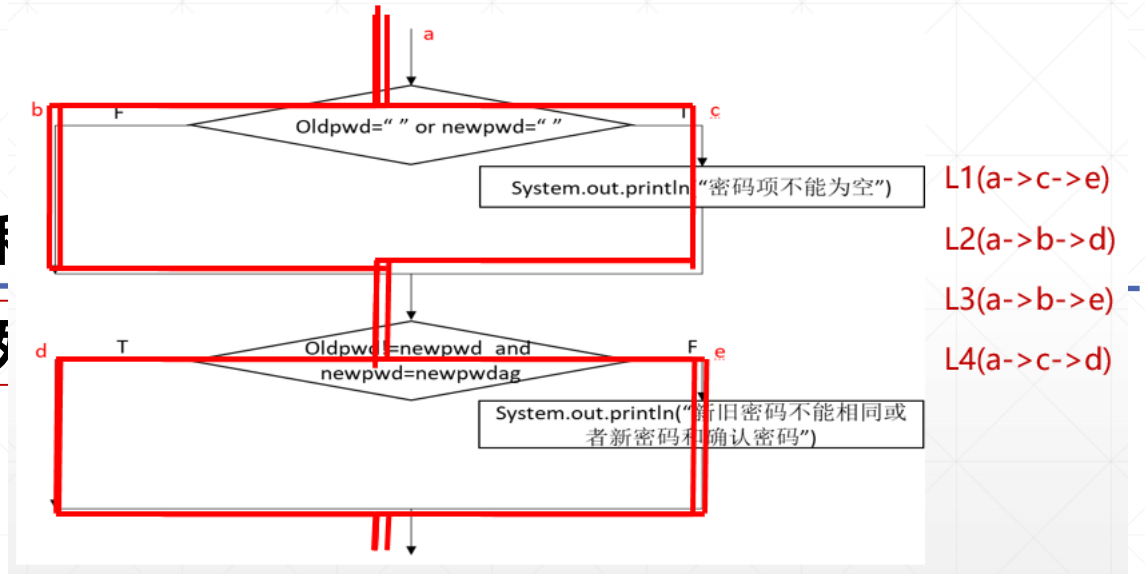
= {oldpwd =″ ″ or newpwd =″ ″ }and {Oldpwd!= newpwd and newpwd = newpwdag}

= oldpwd =″ ″ and Oldpwd!= newpwd and newpwd = newpwdag or

newpwd =″ ″ and Oldpwd!= newpwd and newpwd = newpwdag

语句覆盖

语句覆盖：就是设计若干个测试用例，运行被测程




```
if Oldpwd=“ ” or newpwd=“ ”
    then System.out.println(“密码项不能为空”)
end if
if Oldpwd=newpwd  or newpwd!=newpwdag
    then System.out.println(“新旧密码不能相
            同或者新密码和确认密码要相同”)
end if
```

在图例中，**正好所有的可执行语句都在路径L1上**，所以选择路径 L1设计测试用例，就可以覆盖所有的可执行语句

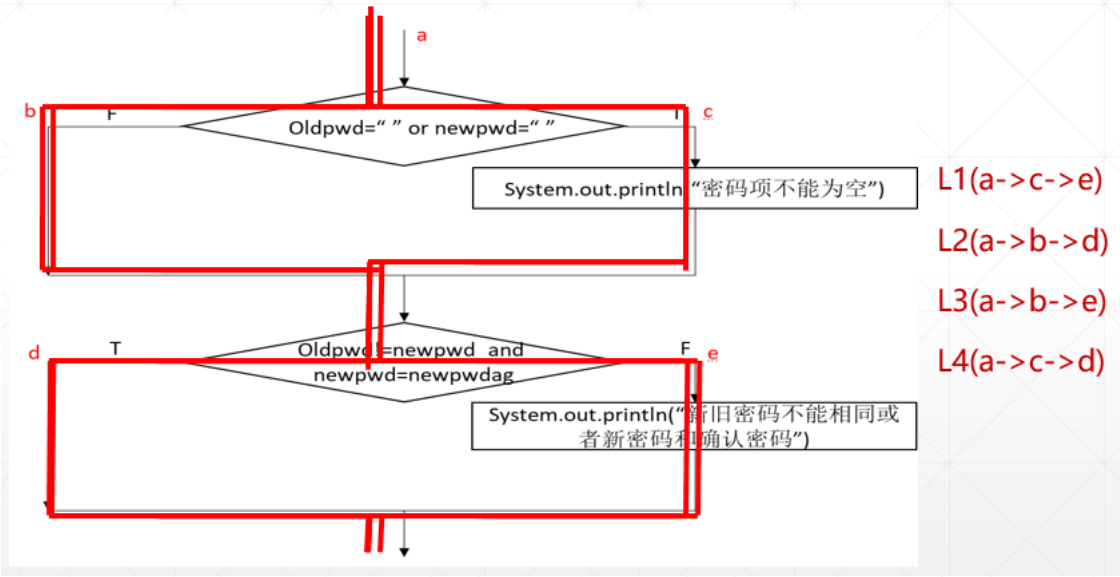
语句覆盖

测试用例的设计格式如下

【输入的(oldpwd, newpwd, newpwdag)，输出的(提示1,提示2)】

为图例设计满足语句覆盖的测试用例是:

【(“”，“”，“”)，(“密码项不能为空”, “新旧密码不能相同或者新密码和确认密码不同”】 覆盖ace【L1】



oldpwd =”” and oldpwd = newpwd or oldpwd =”” and newpwd!= newpwdag

or newpwd =”” and oldpwd = newpwd or newpwd =”” and newpwd!= newpwdag

分支覆盖

分支覆盖：就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。分支覆盖又称为判定覆盖。

选择路径L1和L2:

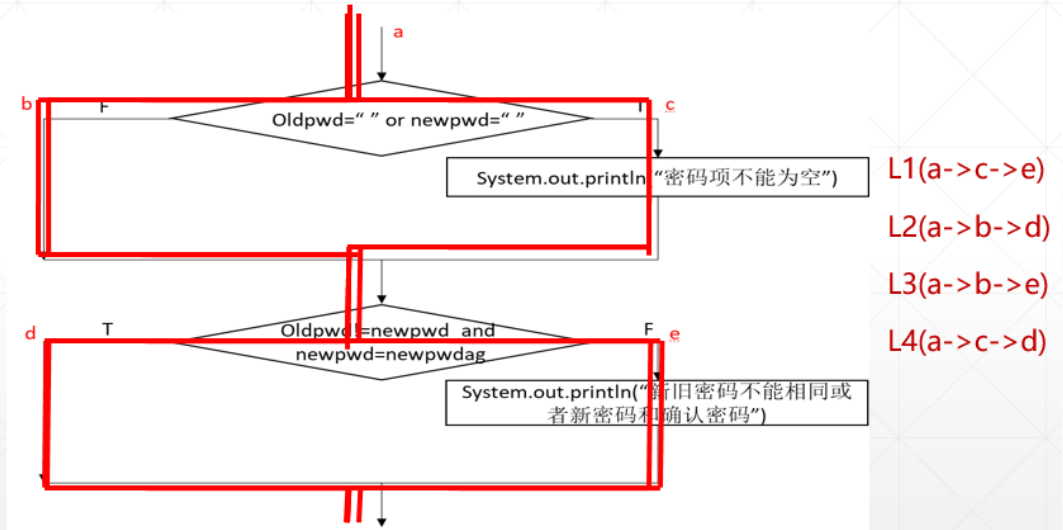
【（“”，“”，“”），(“密码项不能为空”, “新旧密码不能相同或者新密码和确认密码要相同”】覆盖 ace【L1】

【（“123”，“234”，“234”），(“ ”, “ ”】覆盖 abd【L2】

所有取真和取假分支均可经历一次，满足分支覆盖要求

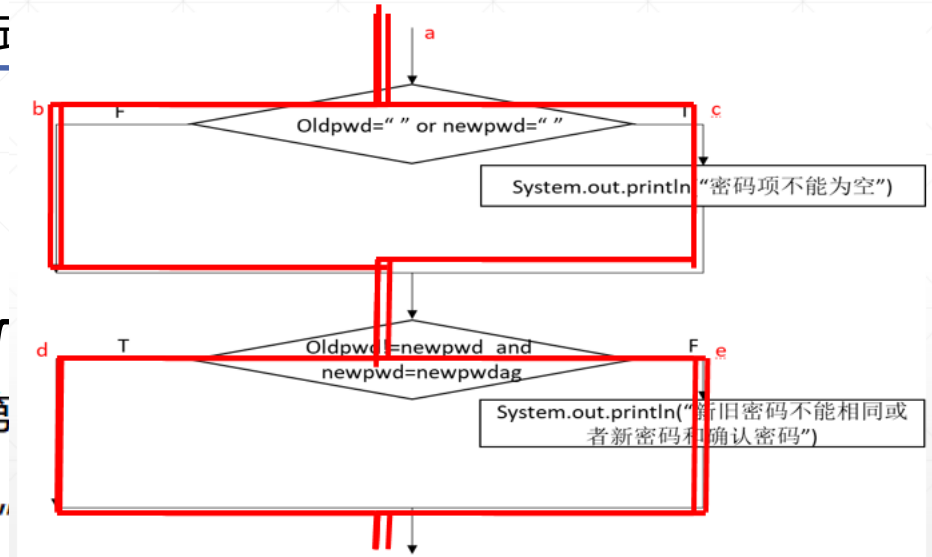
例

- 思考，如果选择路径L3和L4呢，是否也可以满足要求?



条件覆盖

条件覆盖：设计若干个测试用例，运行被测程序，使得每个判断的每个条件的可能取值至少执行一次。



L1(a->c->e)

L2(a->b->d)

L3(a->b->e)

L4(a->c->d)

对于第 判断：

条件 oldpwd= "" 取真为T_3，取假为 $\overline{T_3}$

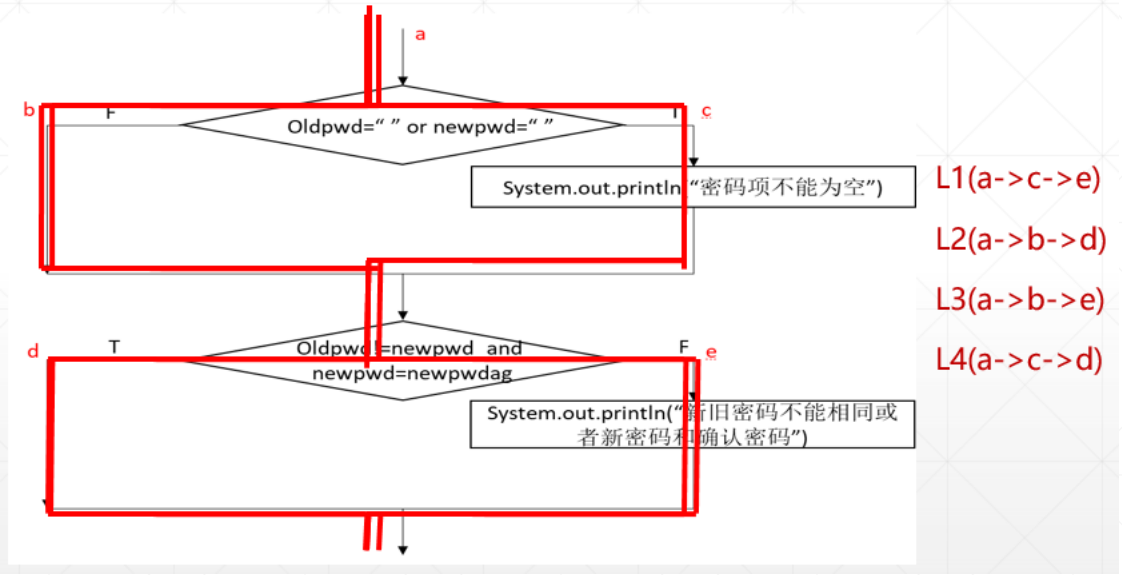
条件 newpwd= "" ag取真为T_4，取假为$\overline{T_4}$

测试用例	覆盖分支	条件取值
【（“123”，“123”，“345”），（“”，“新旧密码不能相同或者新密码和确认密码不同”】	L3(a,b, e)	$\overline{T_1}\overline{T_2}\overline{T_3}\overline{T_4}$
【（“”，“123”，“123”），（“密码项不能为空”，“”】 【（“123”，“ ”，“ ”），（“密码项不能为空”，“”】	L4(a, c, d)	$T_1\overline{T_2}T_3T_4$ $\overline{T_1}T_2T_3T_4$

条件组合覆盖

条件组合覆盖：设计足够的测试用例，运行被测程序，至少执行一次。

例



事先对所有条件组合的取值加以标记。

① oldpwd=“” , newpwd= “ ” T_1T_2
② oldpwd=“” , newpwd！ = “ ” $T_1\overline{T_2}$
③ oldpwd！ =“” , newpwd= “ ” $\overline{T_1}T_2$
④ oldpwd！ =“” , newpwd！ = “ ” $\overline{T_1}\overline{T_2}$
⑤ Oldpwd!=newpwd, newpwd=newpwdag T_3T_4
⑥ Oldpwd!=newpwd, newpwd!=newpwdag $T_3\overline{T_4}$
⑦ Oldpwd=newpwd, newpwd=newpwdag $\overline{T_3}T_4$
⑧ Oldpwd=newpwd, newpwd!=newpwdag $\overline{T_3}\overline{T_4}$

设计测试用例，覆盖所有条件组合

测试用例	覆盖条件	覆盖组合	覆盖判断组合
【(“”, “”, “”), (“密码项不能为空”, “新旧密码不能相同或者新密码和确认密码不同”】	L1	①,⑦	$T_1T_2\overline{T_3}T_4$
【(“”, “536”, “531”), (“密码项不能为空”, “新旧密码不能相同或者新密码和确认密码不同”】	L1	②,⑥	$T_1\overline{T_2}T_3\overline{T_4}$
【(“123”, “ ”, “ ”), (“密码项不能为空”, “”】	L4	③,⑤	$\overline{T_1}T_2T_3T_4$
【(“123”, “123”, “345”), (“”, “新旧密码不能相同或者新密码和确认密码不同”】	L3	④,⑧	$\overline{T_1}\overline{T_2}\overline{T_3}\overline{T_4}$

控制流图覆盖测试

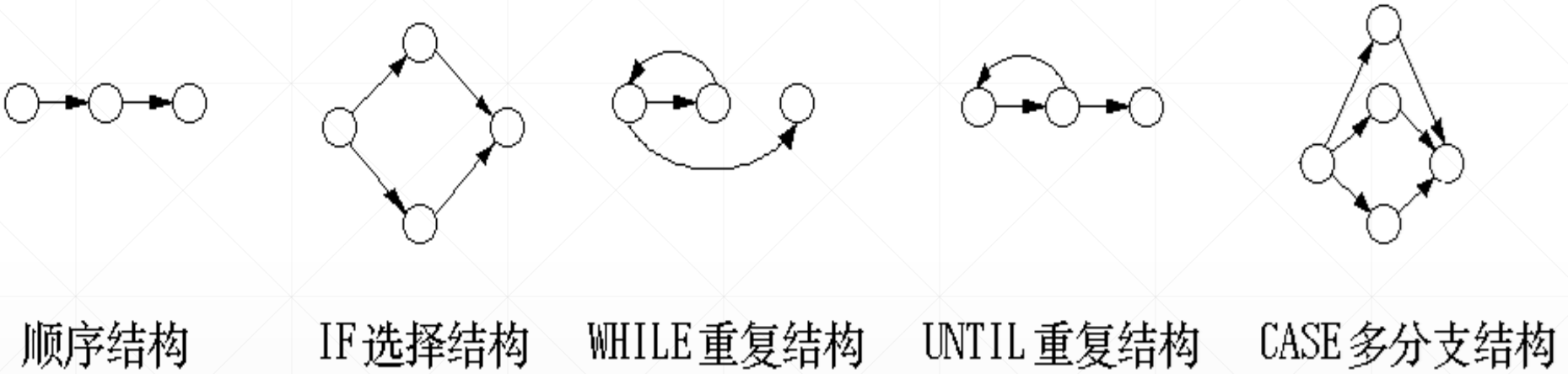
6.2.3.软件测试技术——白盒测试（续）

- 基本概念
- 节点、边覆盖
- 路径覆盖
- 基本路径覆盖

1）基本概念

控制流图覆盖测试：是将代码转变为控制流图（CFG），基于其进行测试的技术。

程序的控制流图



结点：符号○，表示一个或多个无分支的PDL语句或源程序语句。

边：箭头，表示控制流的方向。

汇聚节点：在选择或多分支结构中，分支的汇聚处应有一个汇聚结点。

区域：边和结点圈定的区域。对区域计数时，图形外的区域也应记为一个区域。

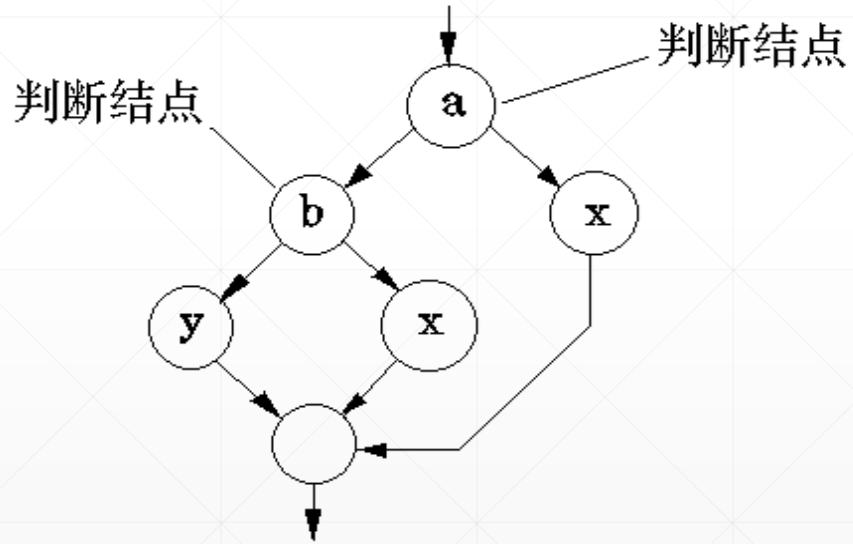
2）基本概念——单条件嵌套

单条件嵌套：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, NAND, NOR）连接的复合条件表达式，则需要改为一系列只有单个条件的嵌套的判断。

例

```
.
.
.
if   a OR b
  then  procedure  x
  else  procedure  y;
.
.
.
```

(a)

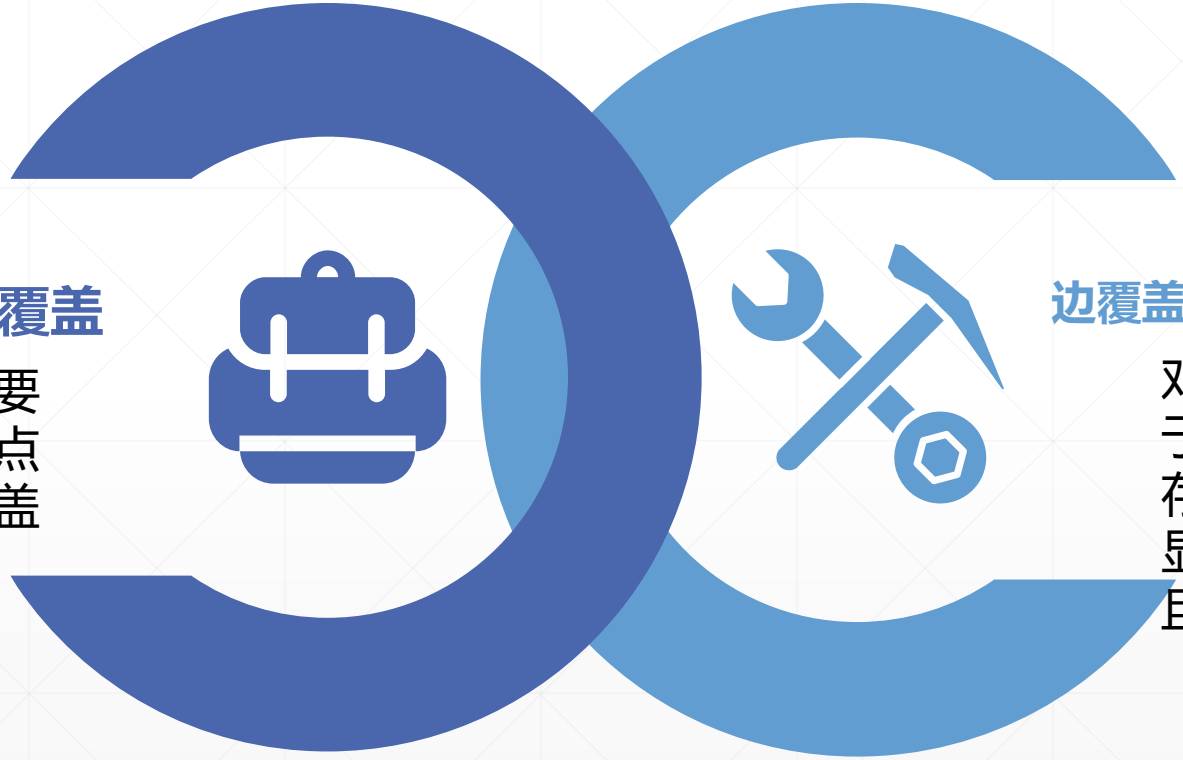


(b)

3）节点、边覆盖

节点覆盖

对图中的每个节点，至少要有一条测试路径访问该节点
显然，节点覆盖=语句覆盖

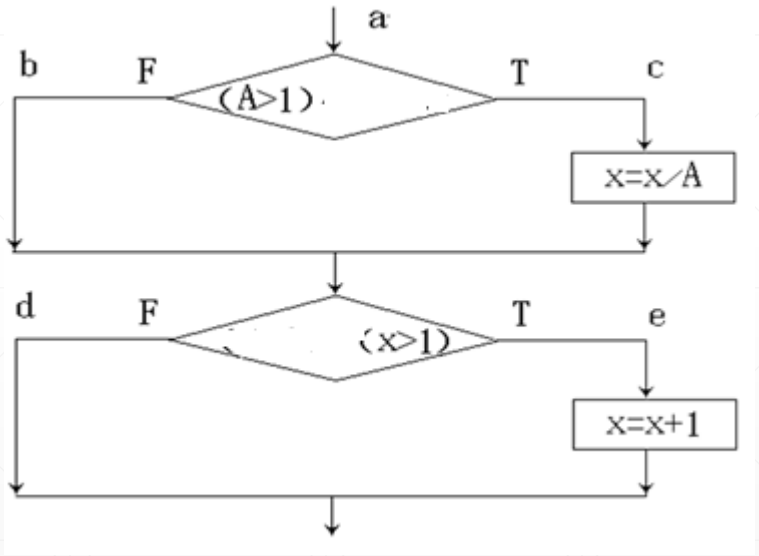


边覆盖

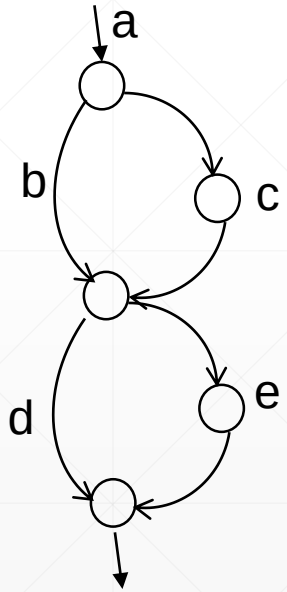
对图中每一个可到达的长度小于(无边图)等于1 的路径，至少存在一条测试路径覆盖。
显然，边覆盖包含节点覆盖，且边覆盖也可以实现分支覆盖。

4）路径覆盖

路径覆盖测试：就是设计足够的测试用例，覆盖程序中所有可能的路径



例



测试用例	覆盖路径
【(2, 4】	ace (L1)
【(1, 1)】	abd (L2)
【(1, 2)】	abe (L3)
【(3, 1)】	acd (L4)

5）基本路径覆盖

基本路径测试：将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。

1. 绘制程序控制流图
2. 分析控制构造的环路复杂性
3. 导出基本可执行路径集合
4. 设计测试用例，保证在测试中，程序的每一个可执行语句至少要执行一次。

5）基本路径覆盖

程序的环路复杂性：程序基本路径集中的独立路径条数，这是确保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。

独立路径：从控制流图来看，一条独立路径是至少包含有一条在其它独立路径中从未有过的边的路径。

计算方法：V(G) = e−n+2。

其中，e 为图中边的数目；n 为节点数目。

5）基本路径覆盖

确定线性独立路径的基本集合

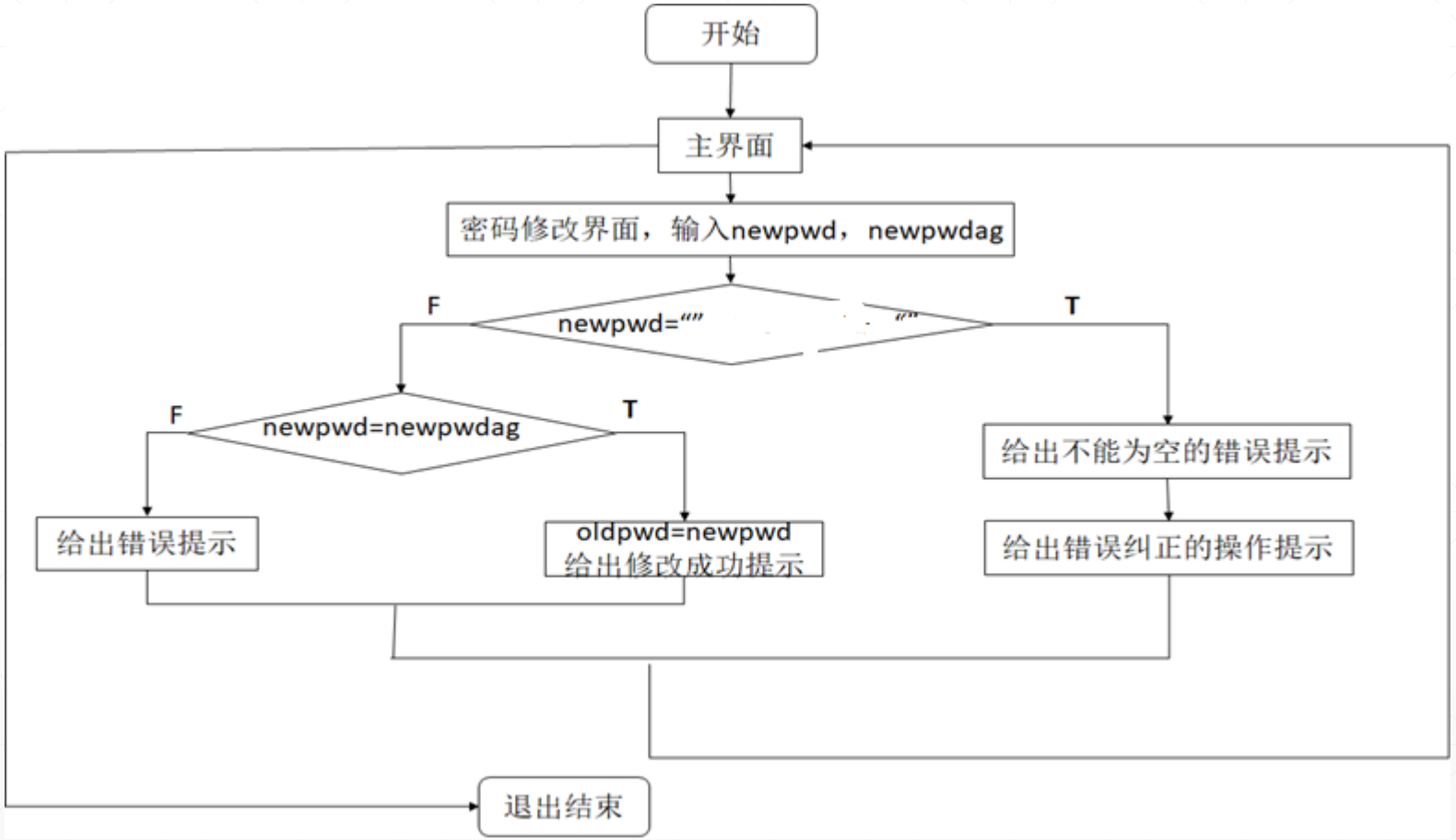
从源节点（控制流图的入口点）开始，一直走到汇节点（控制流图的出口点）。该路径作为基线路径。

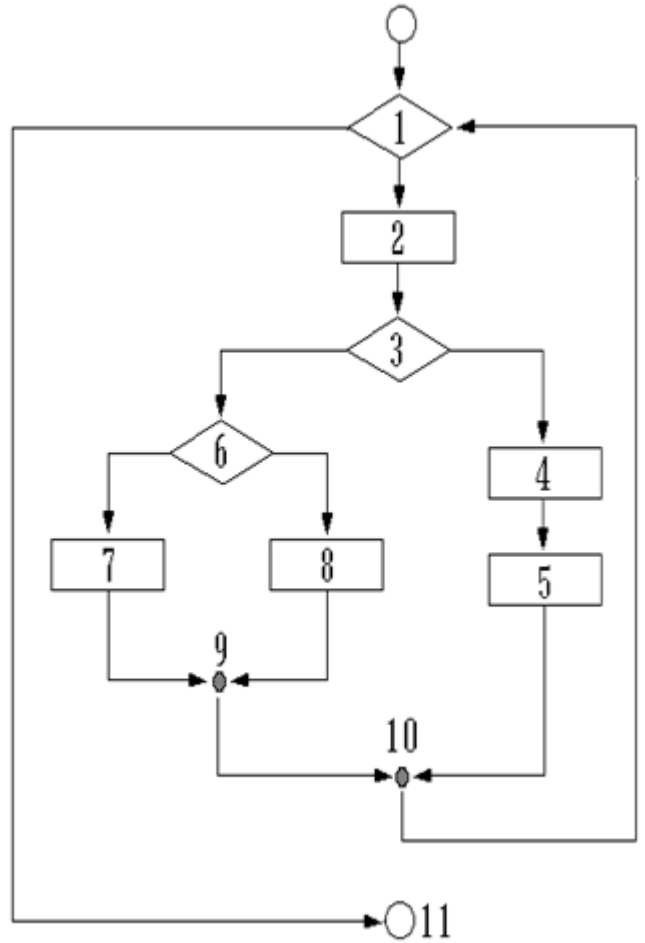
接下来，重新回溯基线路径，依次“翻转”在判断节点上原来选择的路径。即当遇到节点的出度大于等于2 时，必须选择不同的边。

重复以上过程，直到得到的路径数目等于V(G)

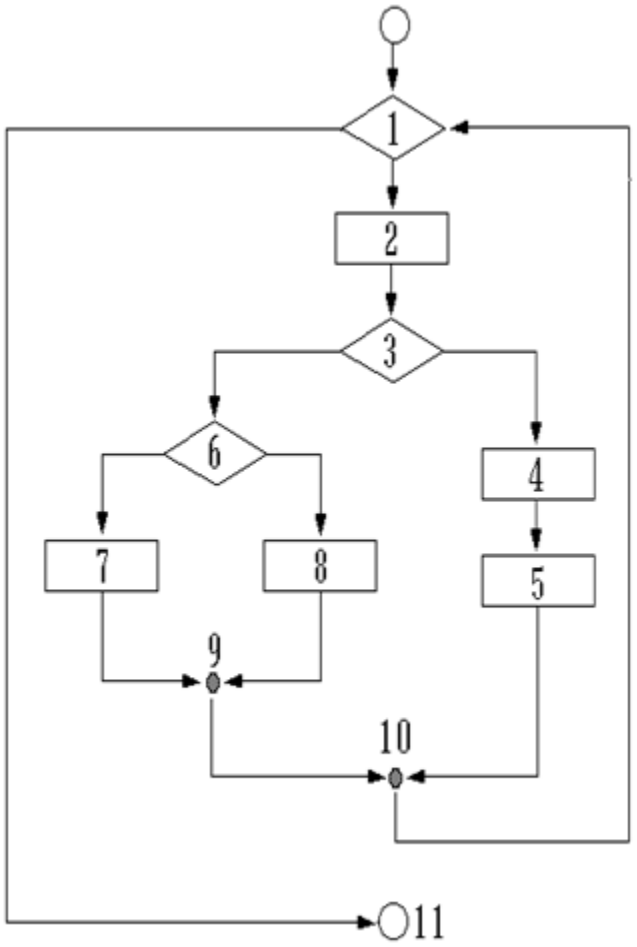
5)基本路径覆盖

例

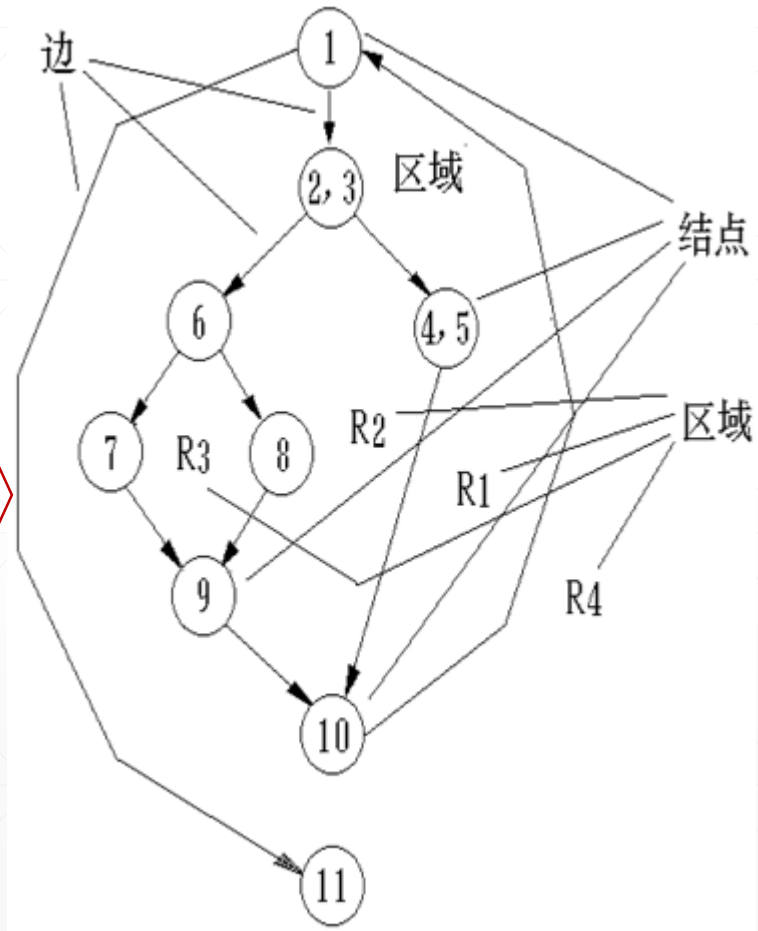

开始
主界面
密码修改界面，输入newpwd，newpwdag
F
newpwd=""
T
F
newpwd=newpwdag
T
给出不能为空的错误提示
给出错误提示
oldpwd=newpwd
给出修改成功提示
给出错误纠正的操作提示
退出结束


1
2
3
6
4
7
8
5
9
10
11

5)基本路径覆盖



例



计算程序环路复杂性：

V(G)=e(11)-n(9)+2=4

确定基本路径级：

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

基本路径集：path1，path2，path3，path4

5）基本路径覆盖

导出测试用例

确保基本路径集的每一条路径的执行。

选择测试用例

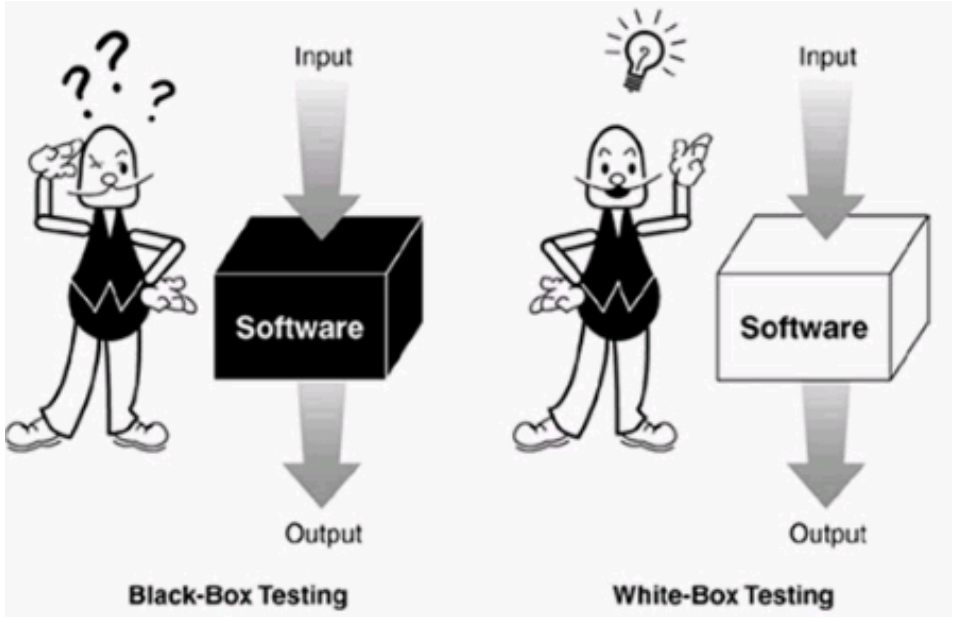
根据判断结点给出的条件，选择合适用例以保证某一条路径可以被测试到

测试结果比较

测试执行后，与预期结果进行比较。

注意事项

非孤立的独立路径可以是另一条路径测试的一部分。



6.2.4.软件测试技术——黑盒测试

- 黑盒测试
- 等价类划分
- 边界值分析

1）黑盒测试：概念

测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性

只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明

又叫做功能测试或数据驱动测试。

1）黑盒测试：检查范围

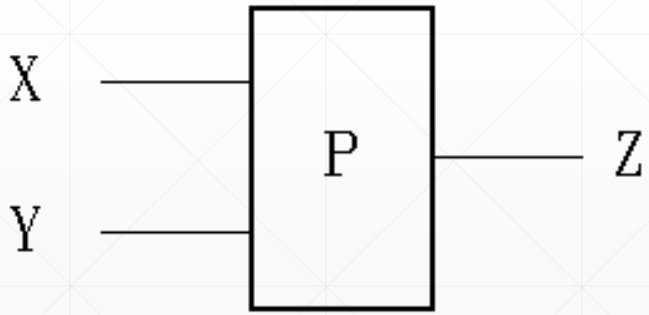
检查范围	是否有不正确或遗漏了的功能?
	在接口上，输入能否正确地接受? 能否输出正确的结果?
	是否有数据结构错误或外部信息访问错误?
	性能上是否能够满足要求?
	是否有初始化或终止性错误?

1）黑盒测试：完全测试的困难性

完全测试的困难性：如果考虑所有可能的输入条件和输出条件，黑盒测试用例数可能是天文数字。

例

程序P有输入量X和Y及输出量Z。在字长为32位的计算机上运行。若X、Y取整数



可能采用的测试数据组：$2^{32} \times 2^{32} = 2^{64}$

设：

每一条路径测试需要1毫秒

一年工作365 × 24小时

需：

5亿年。

2）等价类划分：思想

基本思想

把所有可能的输入数据，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有代表性的数据做为测试用例。

测试步骤

划分等价类

选取测试用例

2）等价类划分：等价类

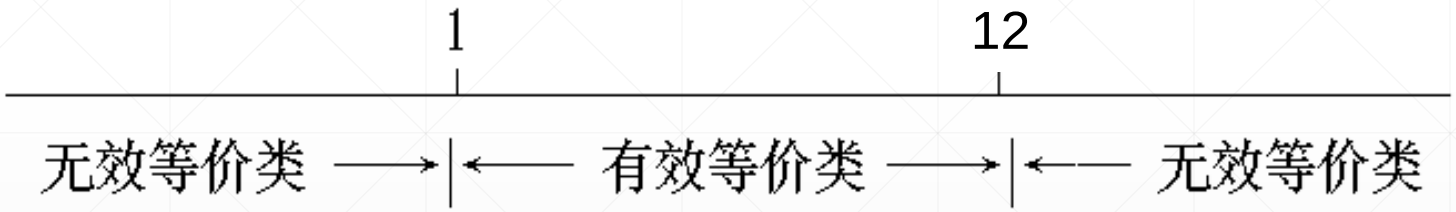
等价类：某个输入域的子集合。在该子集合中，各个输入数据对于测试程序中的缺陷都是等效的。	
有效等价类：对于程序的规格说明来说，是合理的，有意义的输入数据构成的集合。	无效等价类：对于程序的规格说明来说，是不合理的，无意义的输入数据构成的集合。

等价类划分

原则1： **如果输入条件规定了取值范围，或值的个数，则可以确立一个有效等价类和两个无效等价类。**

例

学生信息的出生年月的字段部分约束：“有效月份为数字1-12”



输入条件	有效等价类	无效等价类
学生出生月份	1 ≤出生月份≤12 (1)	出生月份 < 1 (2)；出生月份 > 12 (3)

等价类划分

原则2：如果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，这时可确立一个有效等价类和一个无效等价类。

例

学生信息的学号字段字段部分约束：“学号必须为13位数字串”

输入条件	有效等价类	无效等价类
学生的学号	13位数字串(1)	非13位数字串(2)

等价类划分

原则3：如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。

例

学生信息的在读状态字段部分约束：“为布尔量”

输入条件	有效等价类	无效等价类
学生的在读状态字段	是(1)	否(2)

等价类划分

原则4：如果规定了输入数据的一组值，而且要对每个输入值分别进行处理。可为每一个输入值确立一个有效等价类，所有不允许的输入值集合为一个无效类。

例

学生信息的性别字段部分约束：“性别为{“男”，“女”}，且在系统中需要进行分别计数”。

教师信息中

输入条件	有效等价类	无效等价类
学生的性别字段	“男”(1)，”女”(2)	不在{“男”，“女”}的其他值集合(3)

等价类划分

原则5：**如果规定了输入数据必须遵守的规则，则可以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）。**

例

学生信息的学院字段部分约束：“必须为字符型”

输入条件	有效等价类	无效等价类
学生的学院字段数据类型	字符型(1)	整数型(2),浮点型(3)布尔型(4)

等价类划分

等价类划分步骤

(一)
- 确定等价类

(二)
- 建立等价类表，列出所有划分出的等价类

(三)
- 为每一个等价类规定一个唯一编号；

(四)
- 设计一个新的测试用例，尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；

(五)
- 设计一个新的测试用例，仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止

等价类划分

例

- 在学生管理系统添加教师信息（账号、性别、教师在岗状态）的部分约束：“账号字段由字母开头，后跟字母或数字的任意组合构成，账号不能为空，账号字符数不超过8个；性别字段为{“男”，“女”}，且在系统中需要进行分别计数；教师在岗状态字段为布尔量。”

等价类划分

例

1、确定等价类

2、建立等价类表

3、为等价类唯一编号

输入条件	有效等价类	无效等价类
第一个字符	字母（1)	非字母（2)
标识符组成	字母、数字的任意组合 (3)；	非字母数字字符（4)；保留字（5)
账号长度	0<账号长度≤8 (6)	0个（7)；>8个（8)
性别	“男”(9)，”女”(10)	不在{“男”，“女”}的其他值集合(11)
教师在岗状态	是(12)	否(13)

唯一编号

等价类划分

例

4、设计用例，覆盖尽量多的有效等价类

重复，直至覆盖所有有效等价类

① (z030520,男,是)　覆盖　(1), (3), (6), (9), (12)

② (z030520,女,是)　覆盖　(1), (3), (6), (10), (12)

等价类划分

无效等价类
非字母 (2)
非字母数字字符 (4)；保留字 (5)
0个 (7)；>8个 (8)
不在{“男”，“女”}的其他值集合(11)
否(13)

5、设计用例，仅覆盖一个无效等价类

重复，直至覆盖所有无效等价类

例

③ (6030520,男,是)　　覆盖　　(2)

④ (z0300##,男,是)　　覆盖　　(4)

⑤ (boolean,男,是)　　覆盖　　(5)

⑥ (,男,是)　　覆盖　　(7)

⑦ (z0300xd88,男,是)　　覆盖　　(8)

⑧ (z0300,例,是)　　覆盖　　(11)

⑨ (z030020,男,否)　　覆盖　　(13)

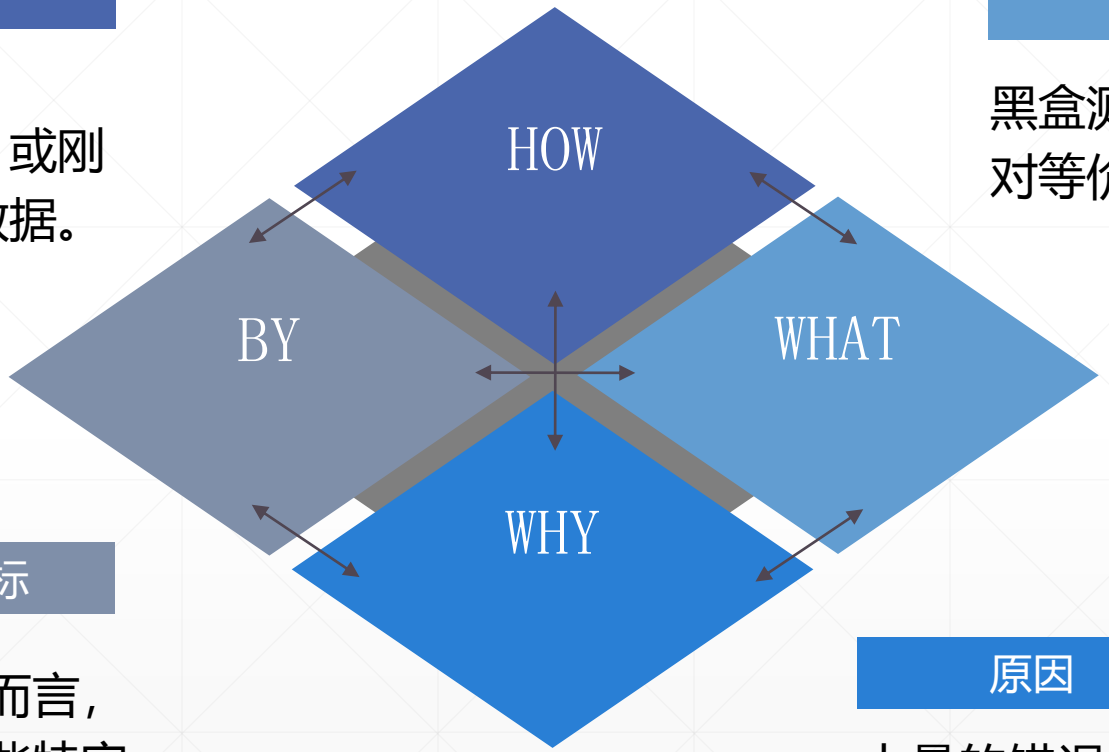
边界值分析

方法

确定边界情况
选取正好等于，刚刚大于，或刚刚小于边界的值做为测试数据。

含义

黑盒测试方法
对等价类划分方法的补充



边界指标

相当于输入、输出等价类而言，稍高、低于其边界值的一些特定情况

原因

大量的错误是发生在输入或输出范围边界上
边界测试可以查出更多的错误

边界值分析-单侧边界

第一种是单侧边界为闭区间的情况，选择边界点、边界点加一个步长的点和边界点减一个步长的点设计测试用例

例：可以在学生成绩管理系统中查询优秀成绩信息，查询标准为成绩字段至少85分，采用针对条件采用边界值分析法设计测试用例。

测试条件	输入值	预期结果
查询优秀成绩信息	85	符合要求
	86	符合要求
	84	错误提示

边界值分析-单侧边界

第二种是单侧边界为开区间的情况，选择边界点、边界点范围内方向移动一个步长的点设计测试用例。

例：可以在学生成绩管理系统中查询不及格成绩信息，查询标准为成绩字段低于60分，采用针对条件采用边界值分析法设计测试用例。

测试条件	输入值	预期结果
查询及格成绩信息	59	符合要求
	60	错误提示

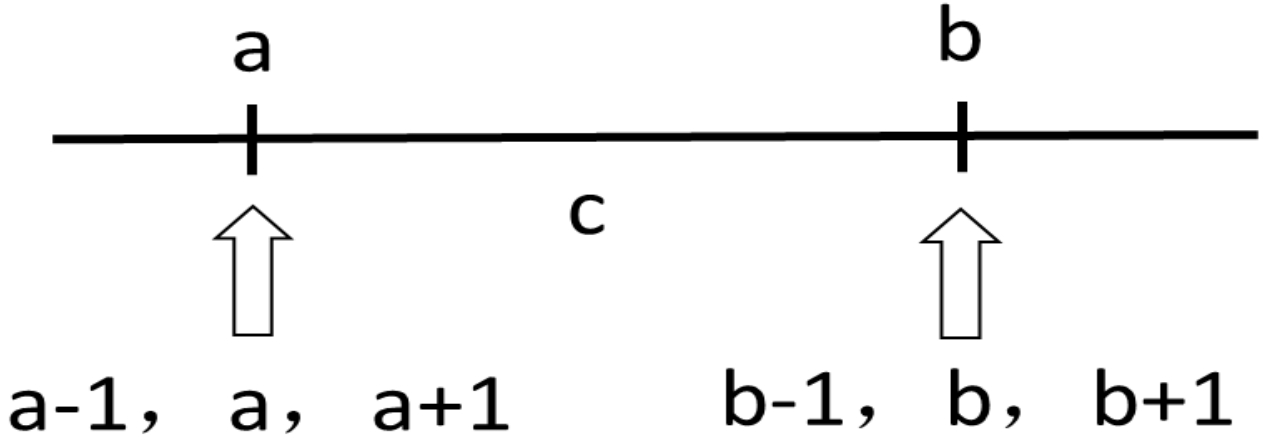
边界值分析-区间范围

结合等价类划分的具体情况，针对边界值的选择就包括开区间、闭区间以及半开半闭区间，涉及3个测试点，命名为上点、离点和内点。

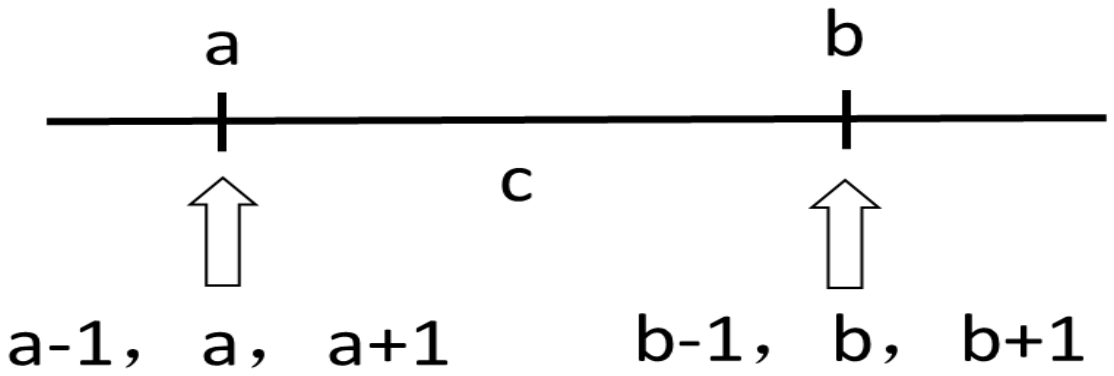
上点，即边界上的点，不管是开区间还是闭区间。

内点，上点范围内的任意一点。

离点，离上点最近的点称为离点。开区间为上点范围内加一个步长，闭区间为上点范围外加一个步长。



边界值分析-区间范围



闭区间：闭区间中的情况，上点为可以取值的点，在上点之间任取一点就是内点。而紧邻上点范围之外第一对点为离点。以上图为例，上点为a、b，内点为c，离点为a-1、b+1。

这里的1是一个步长，可能不是数字的1

例：学生成绩管理系统的成绩输入中，输入分数范围为[85,100]，则成绩等级为优秀，采用边界值分析法设计测试用例。

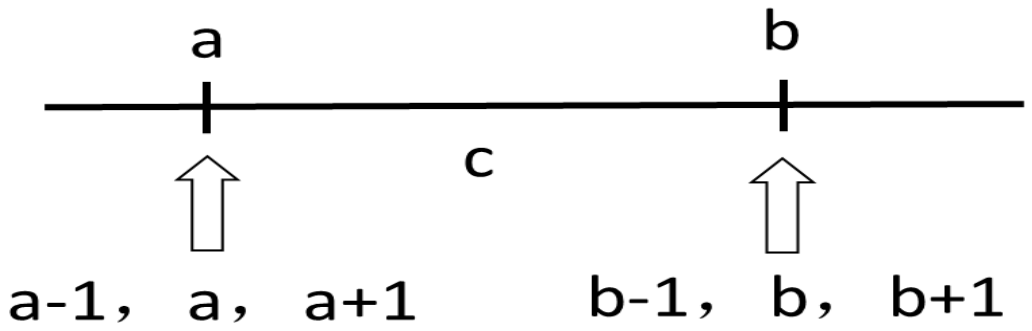
上点：为85和100

内点：即85到100之间的任何一点，如：93。

离点：为84和101。

测试用例为{84，85，93，100，101}

边界值分析-区间范围



开区间：开区间中，上点与内点的定义仍然不变。而离点就是上点内部范围内紧邻上点的一对点。以上图为例，上点为a、b，内点为c，离点为a+1、b-1。

例：学生成绩管理系统的成绩输入中，输入分数范围为（0,60），则成绩等级为不及格，采用边界值分析法设计测试用例。

上点：为0和60

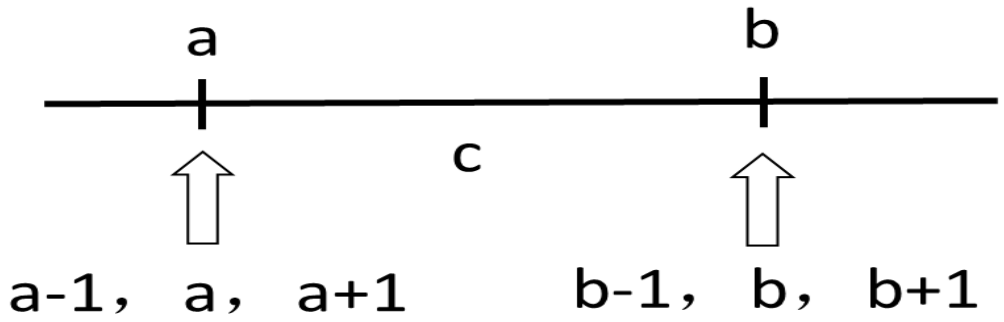
内点：即0到60之间的任何一点，如：33。

离点：为1和59。

测试用例为{0，1，33，59，60}

边界值分析-区间范围

半开半闭区间：半开半闭区间中，上点与内点的定义不变。离点是开区间一侧上点内部范围内紧邻的点，而在闭区间一侧是上点范围外紧邻的点。



例：学生成绩管理系统的成绩输入中，如输入分数范围为[60,75），则成绩等级为及格，采用边界值分析法设计测试用例。

上点：为60和75

内点：即60到75之间的任何一点，如：67。

离点：为59和74。

测试用例为{59，60，67，74，75}