软件工程与实践

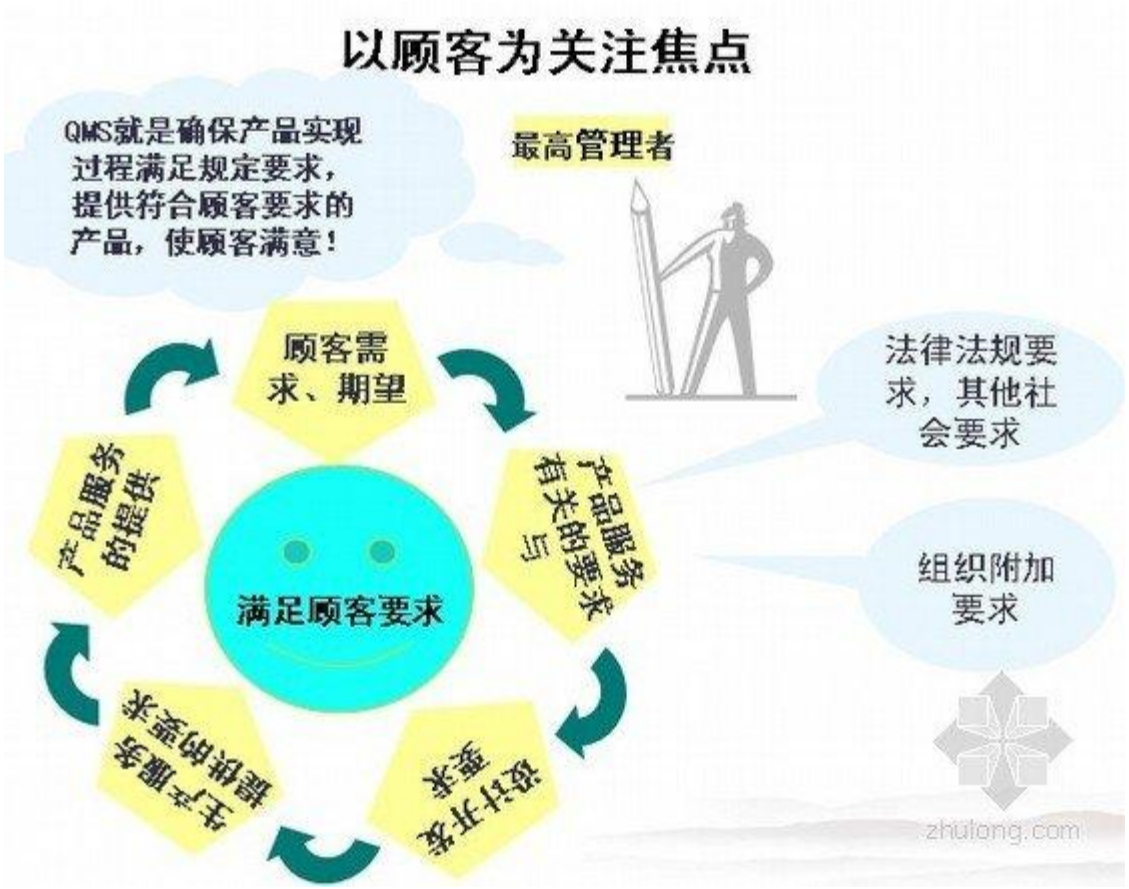
软件工程与实践课程组

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。



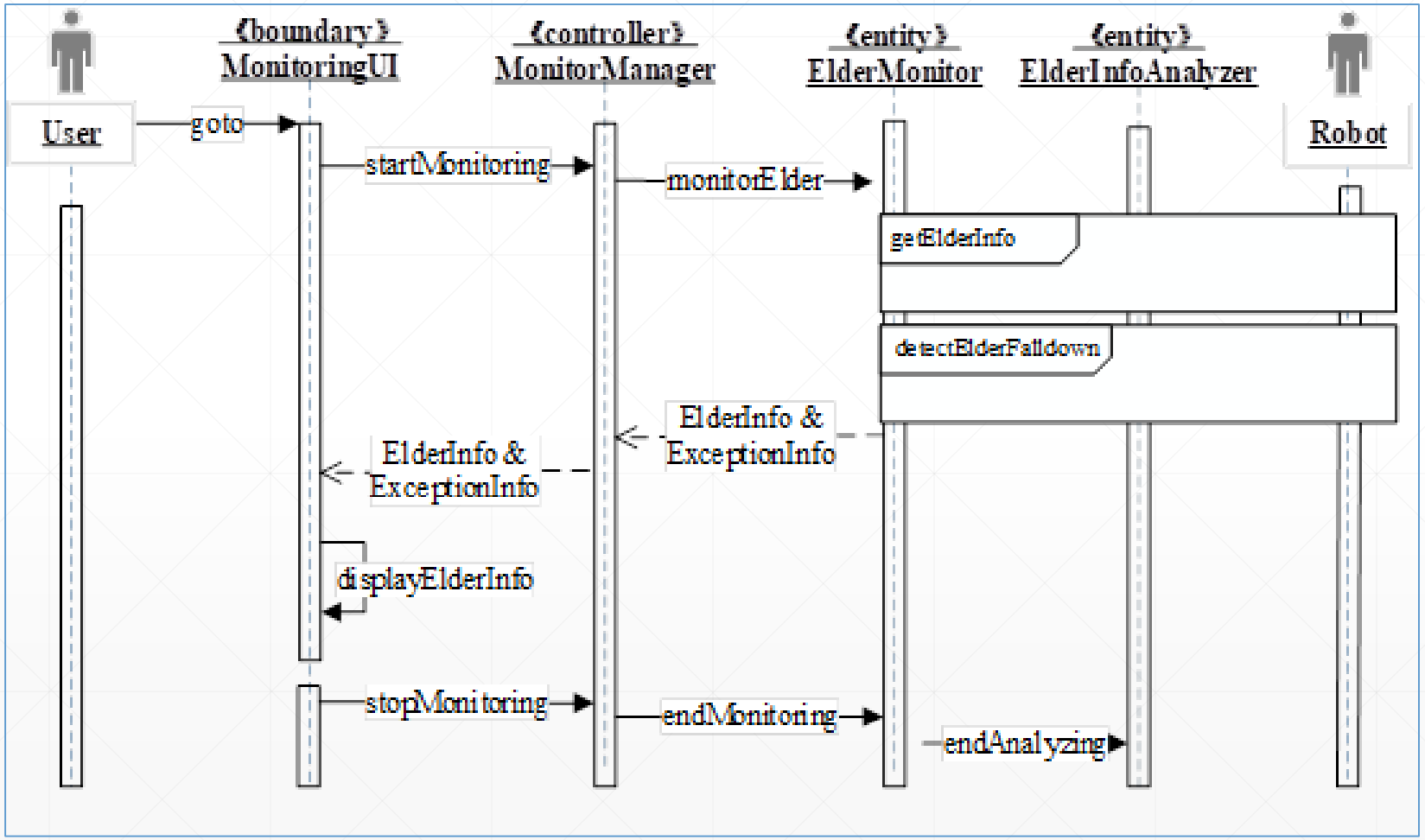
6.1.软件质量保证

- 软件质量相关概念

示例：软件中存在缺陷，导致软件失效

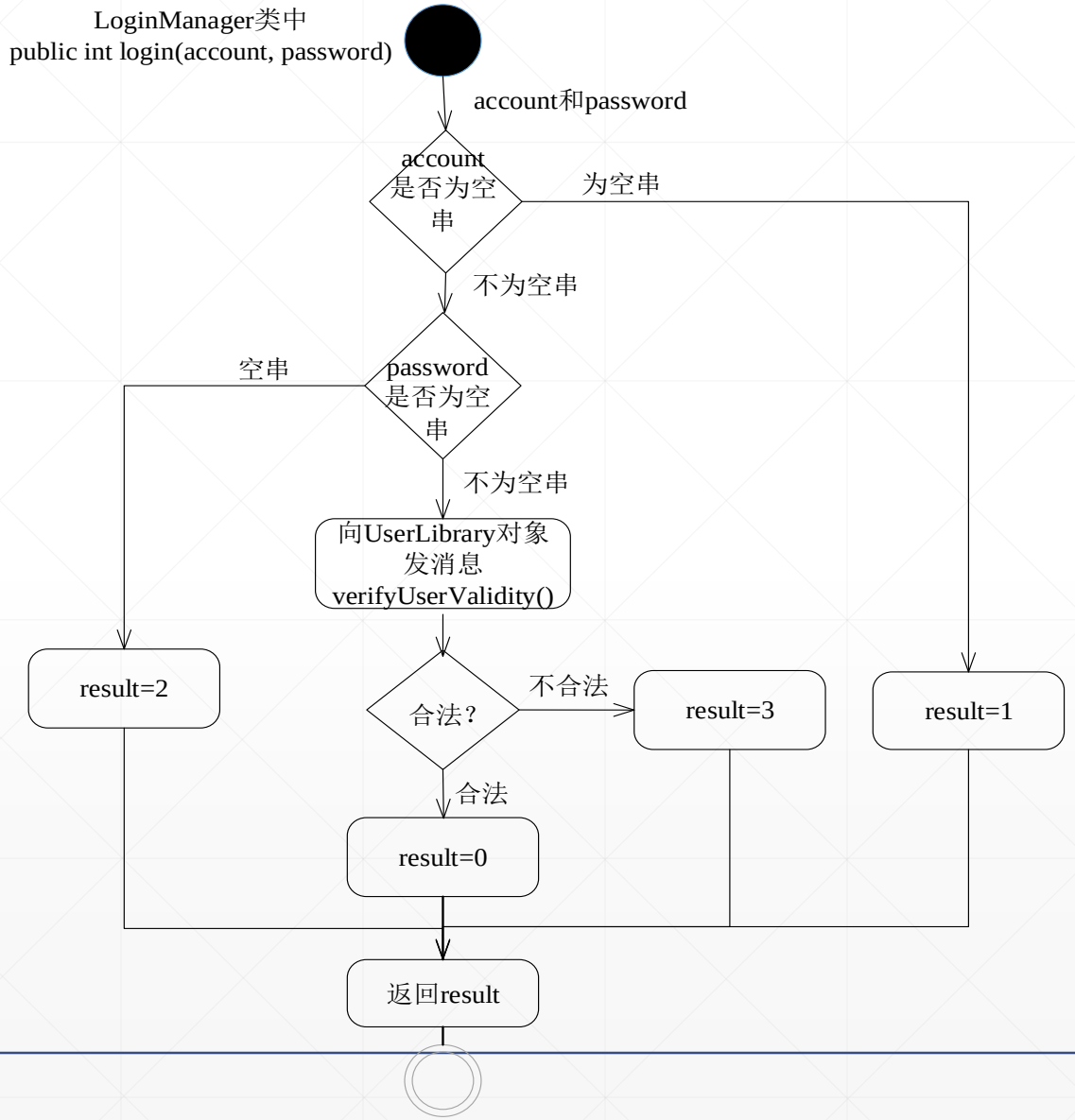
示例：软件中存在缺陷，导致软件失效

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         }
```

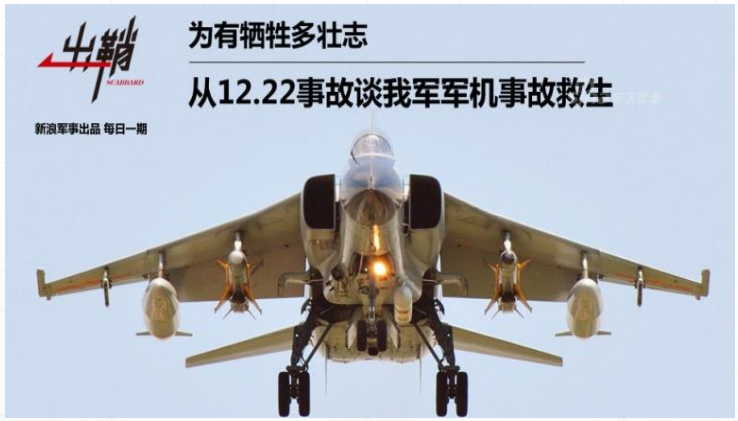
- 是否正确实现功能?
- 是否存在内在缺陷?

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

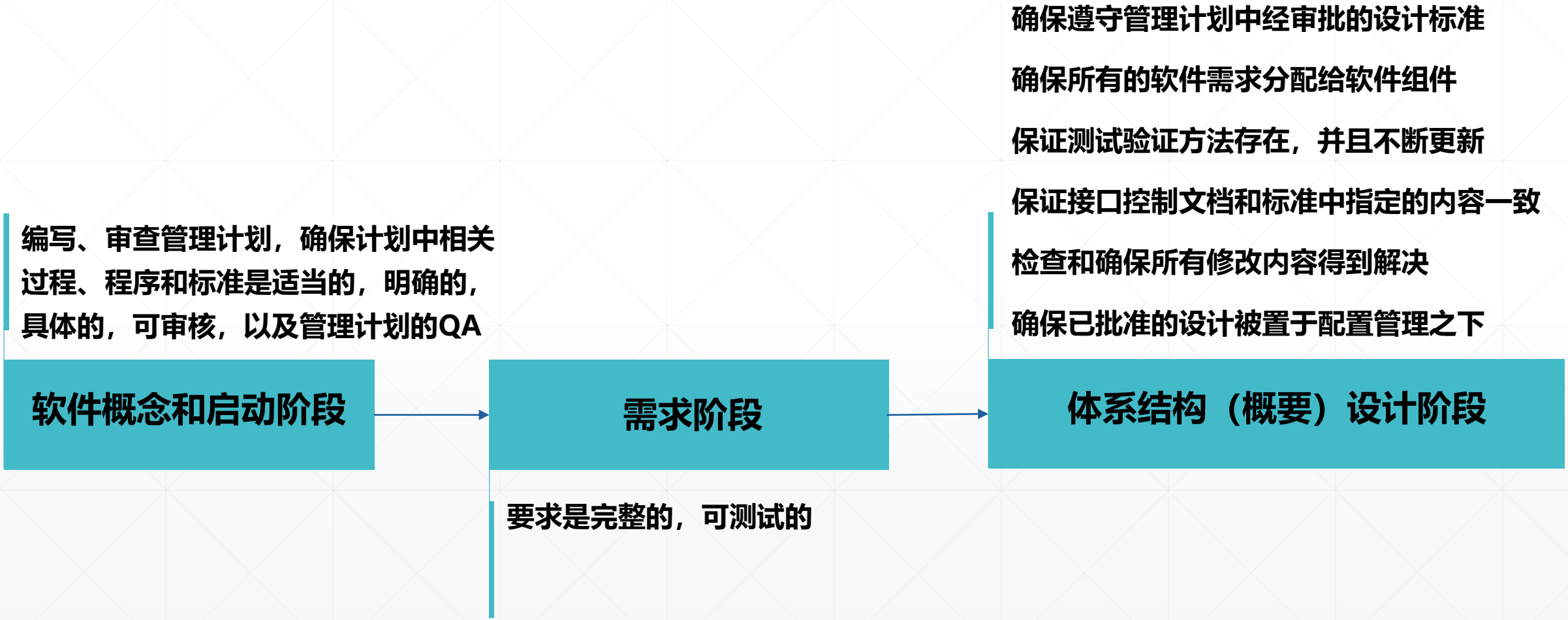
- 对比文档中描述的执行和实际操作步骤，确保执行过程采取适当步骤和操作方式

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

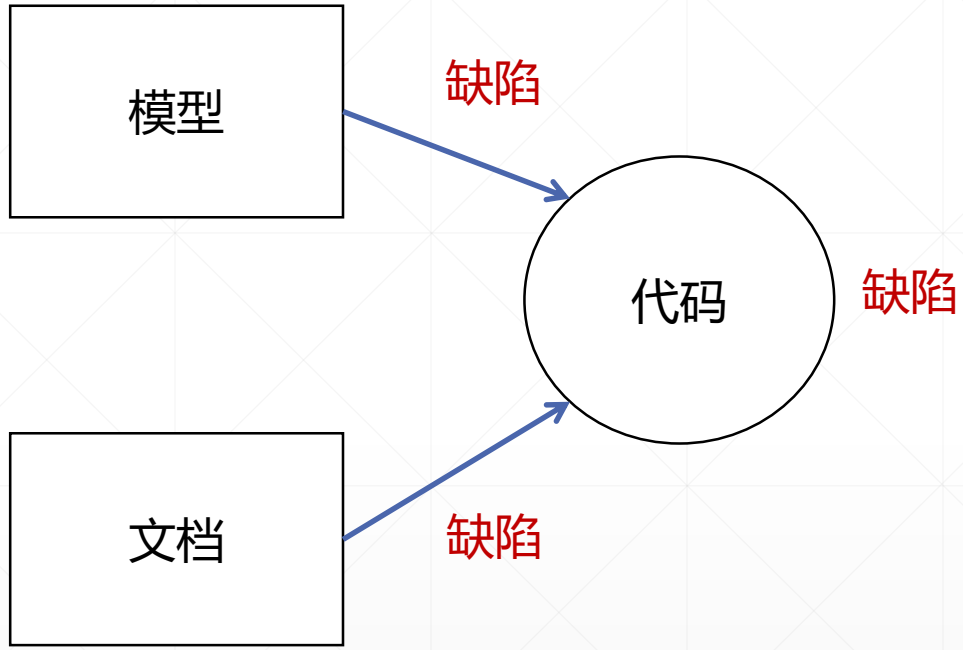
使用较短开发周期来升级和更正软件

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

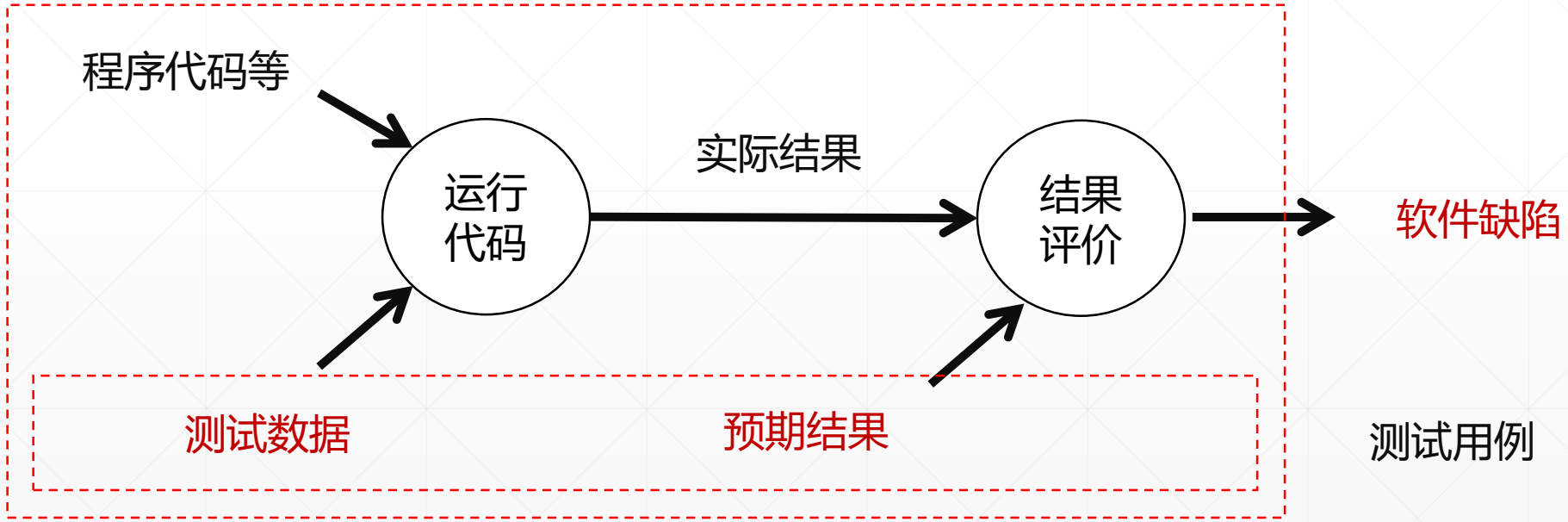


思考和讨论

- 如何从程序代码中找出软件缺陷?

6.1.3 软件测试的原理

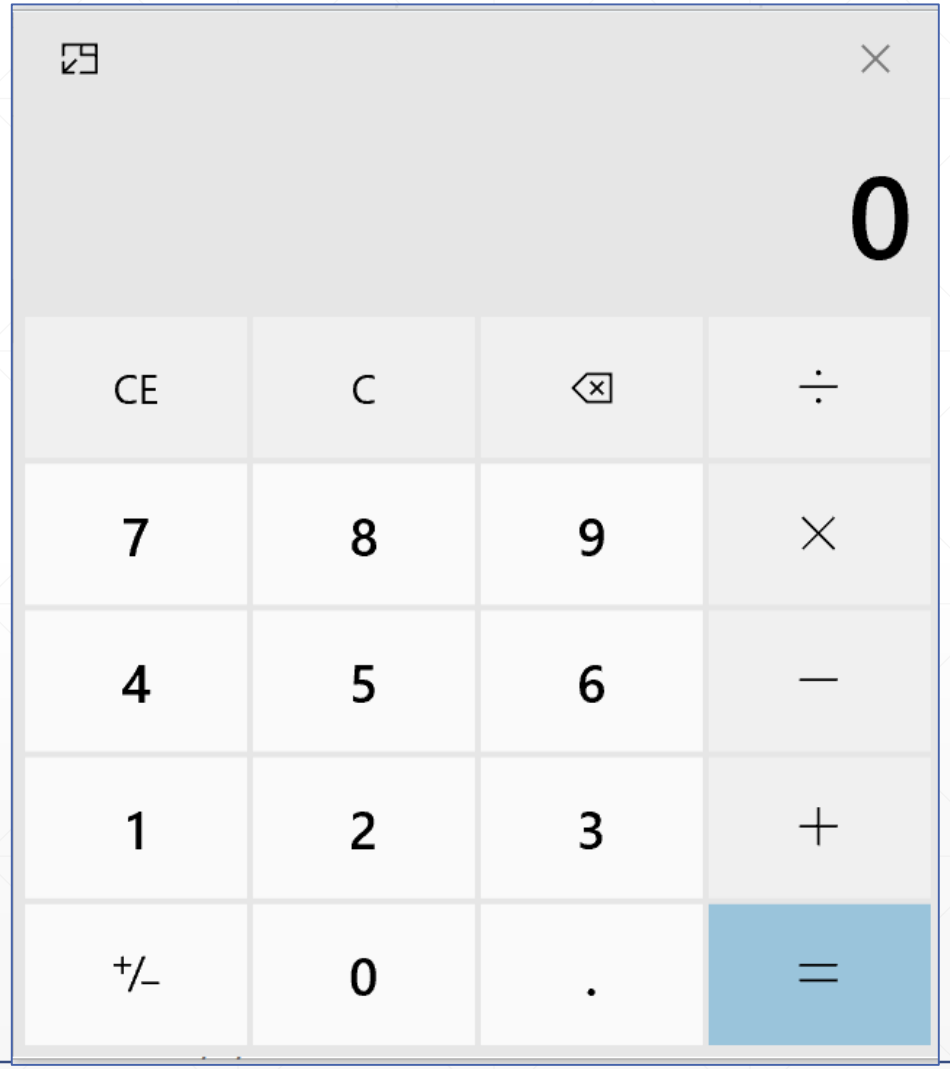
- **程序本质上是对数据的处理**
  - **设计数据(测试用例) ➔ 运行测试用例(程序来处理数据) ➔ 判断运行结果(是否符合预期结果)**



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

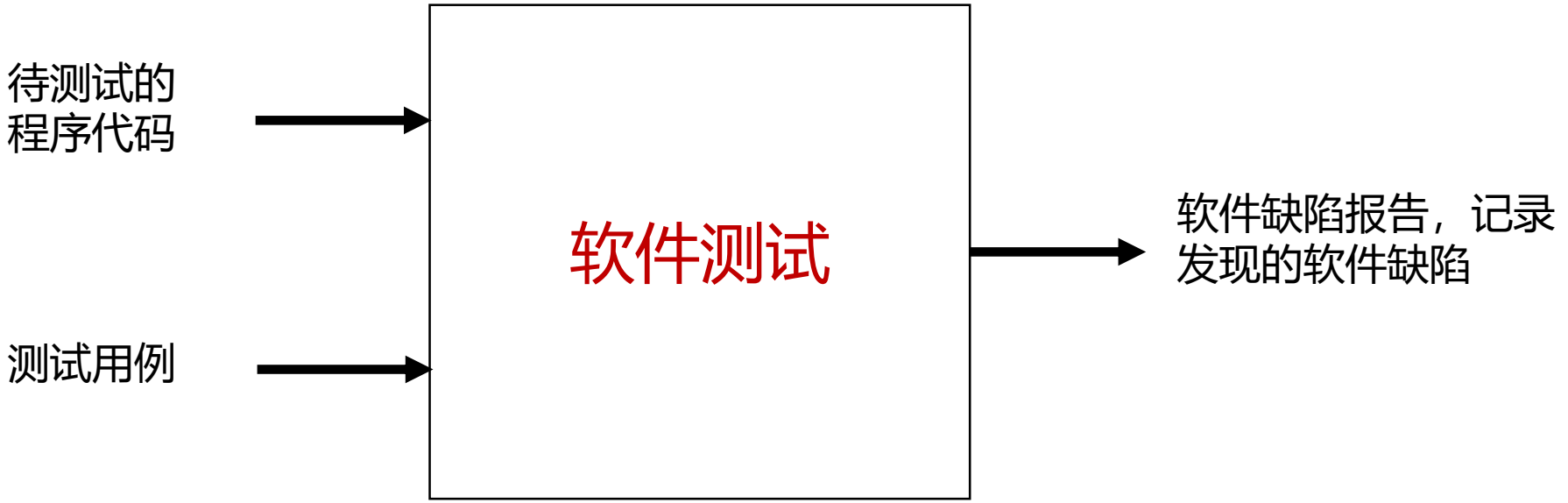
- 测试用例是一个四元偶
  - **输入数据**：交由待测试程序代码进行处理的数据
  - **前置条件**：程序处理输入数据的运行上下文，即要满足前置条件
  - **测试步骤**：程序代码对输入数据的处理可能涉及到一系列的步骤，其中的某些步骤需要用户的进一步输入
  - **预期输出**：程序代码的预期输出结果

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

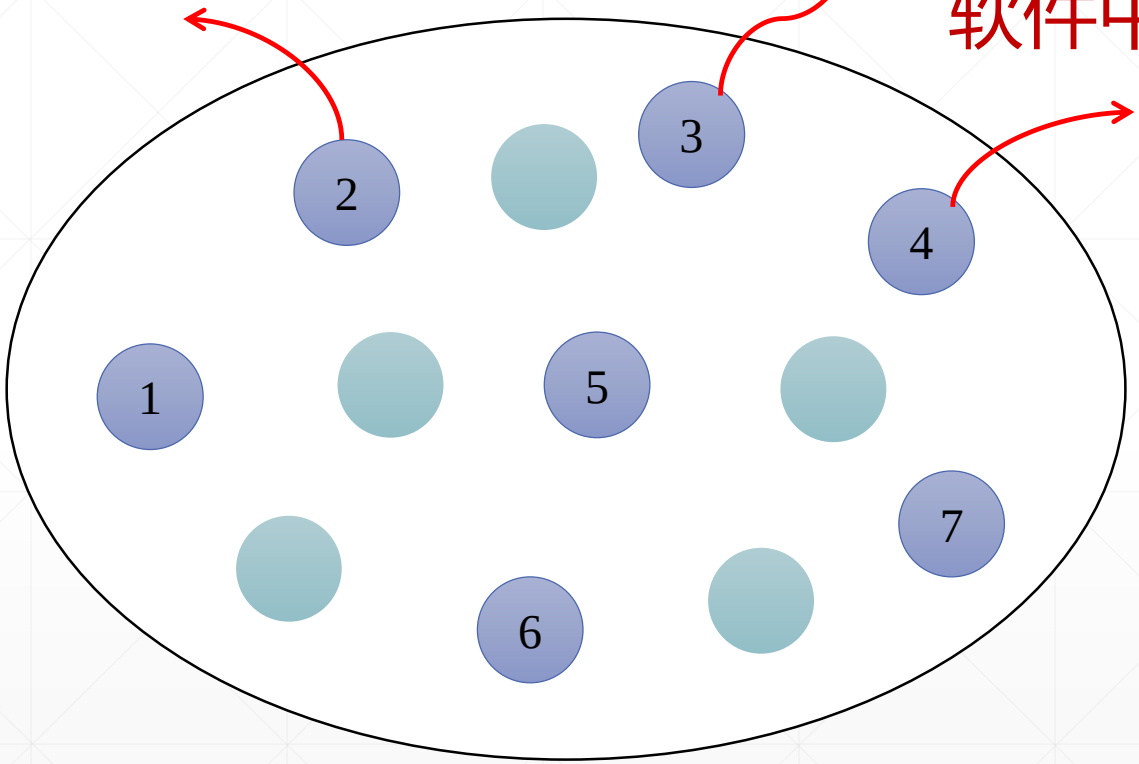
6.1.4 软件测试的任务

➢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?

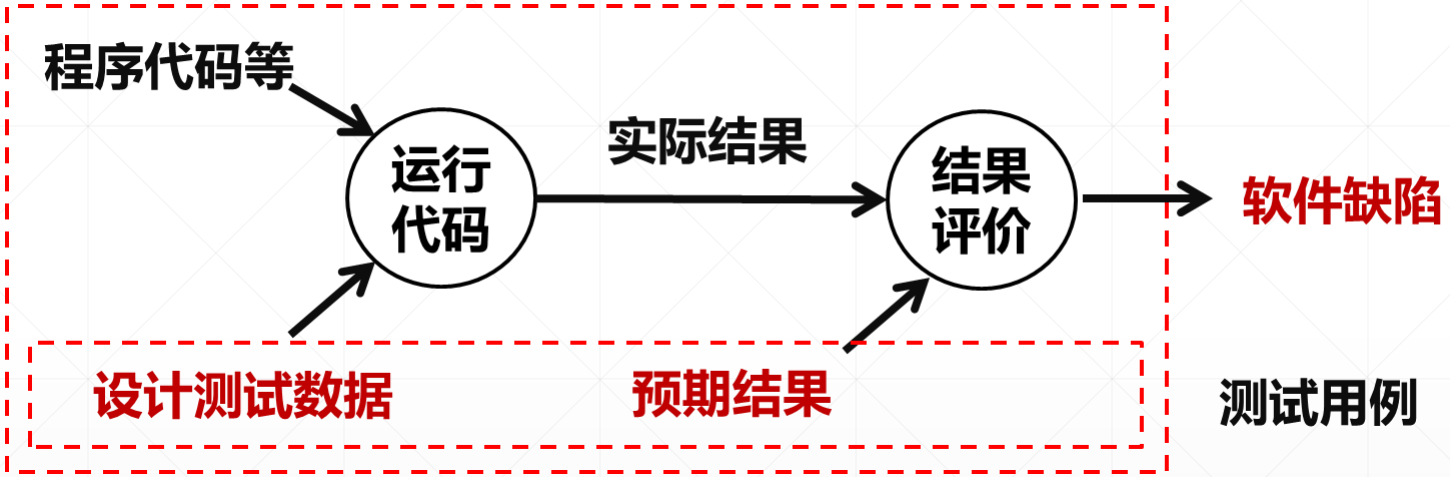
6.1.5 软件测试的步骤

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

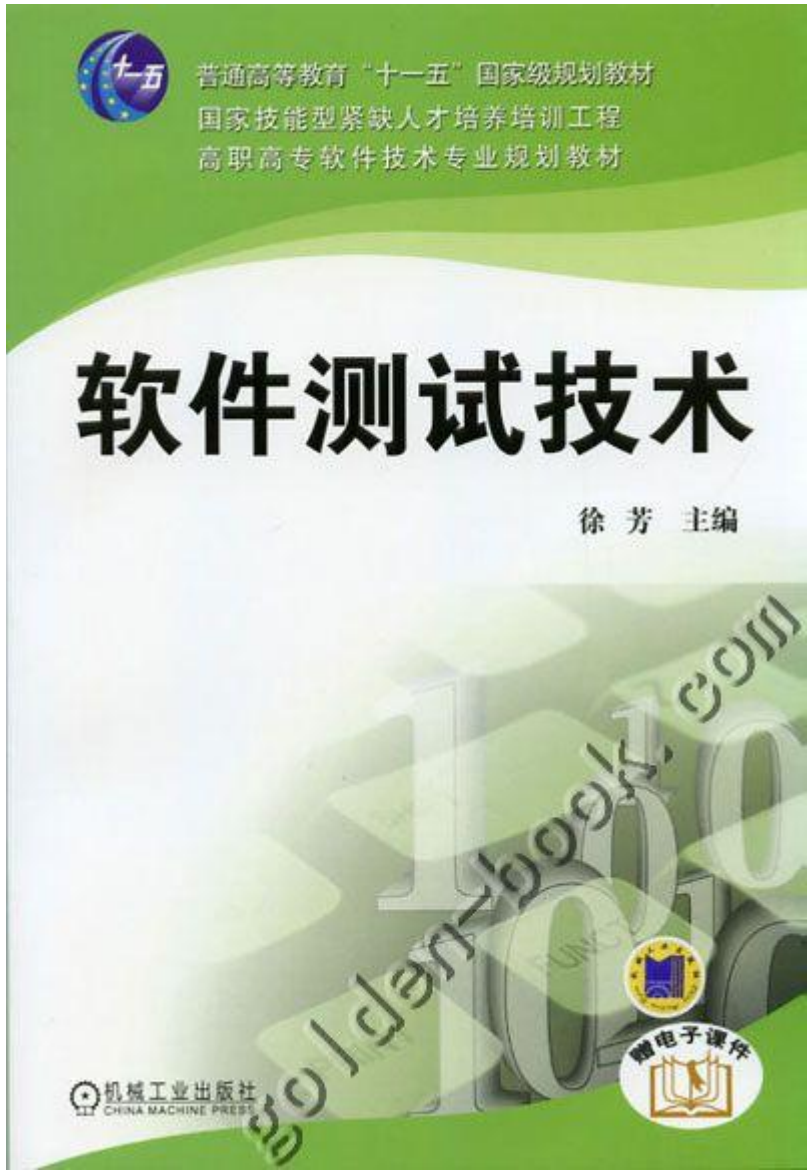
6.1.6 软件测试面临的主要挑战

- 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

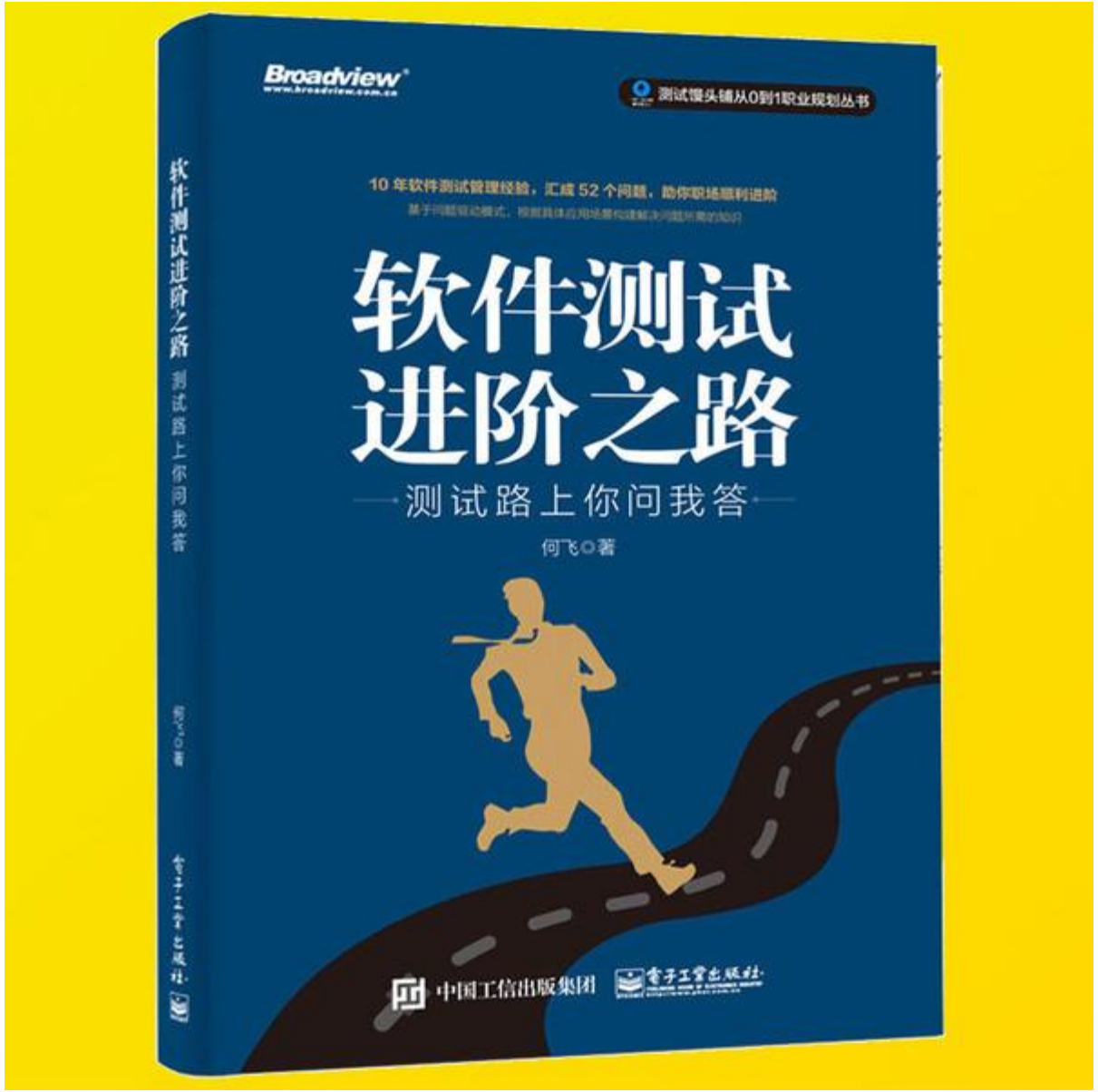
6.1.7 软件测试工程师

- 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

软件测试的定义

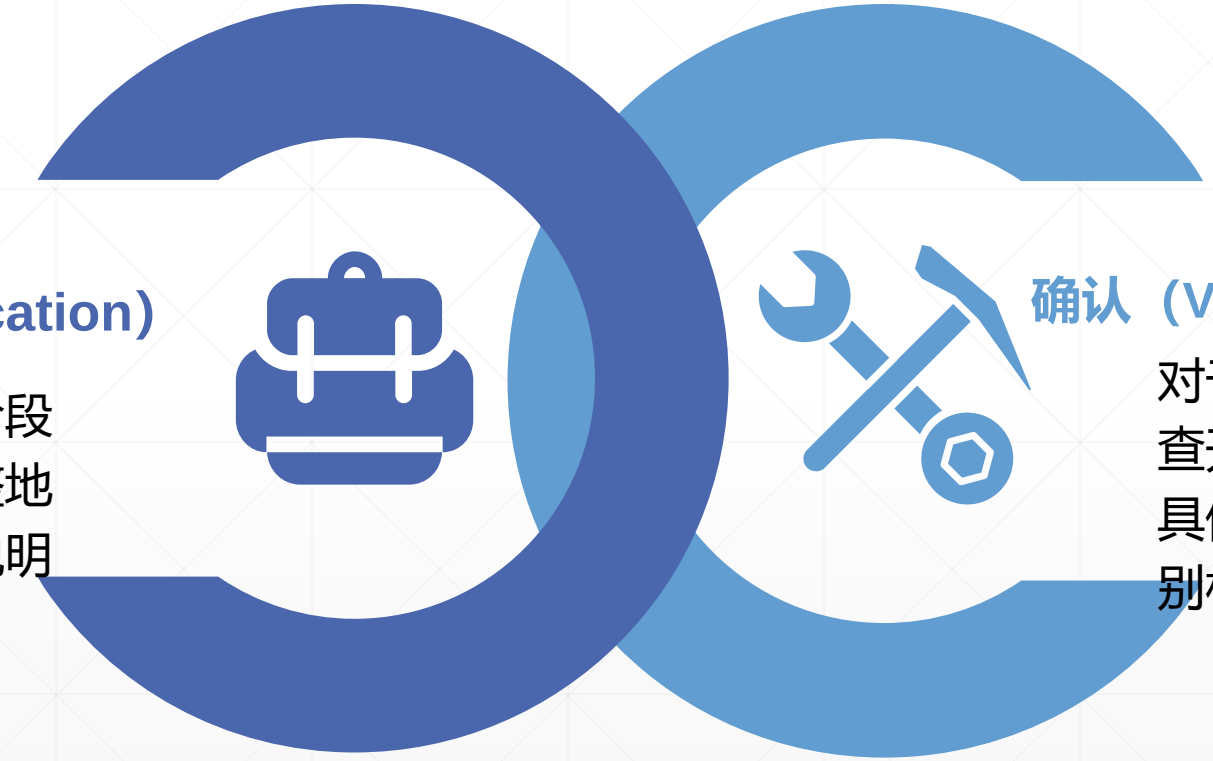
在某种指定的条件下对系统或组件操作，观察或记录结果，对系统或组件的某些方面进行评估的过程。

分析软件各项目以检测现有的结果和应有结果之间的差异，并评估软件各项目的特征的过程。

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）

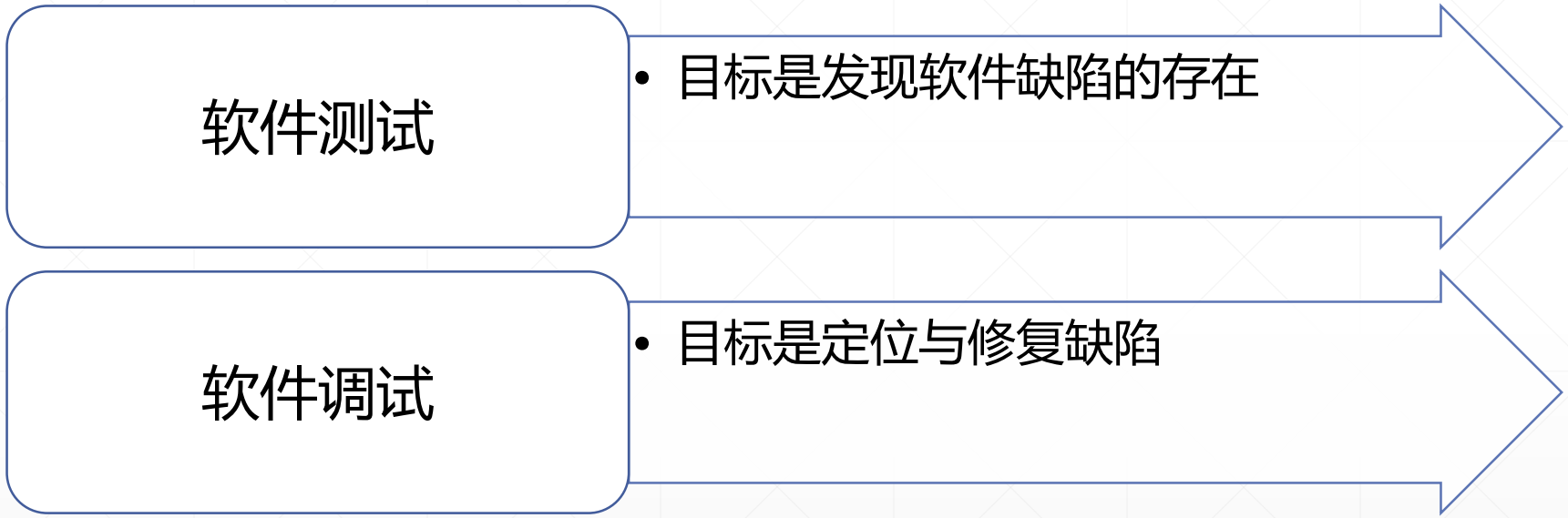
保证软件特定开发阶段的输出已经正确完整地实现了规格说明

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

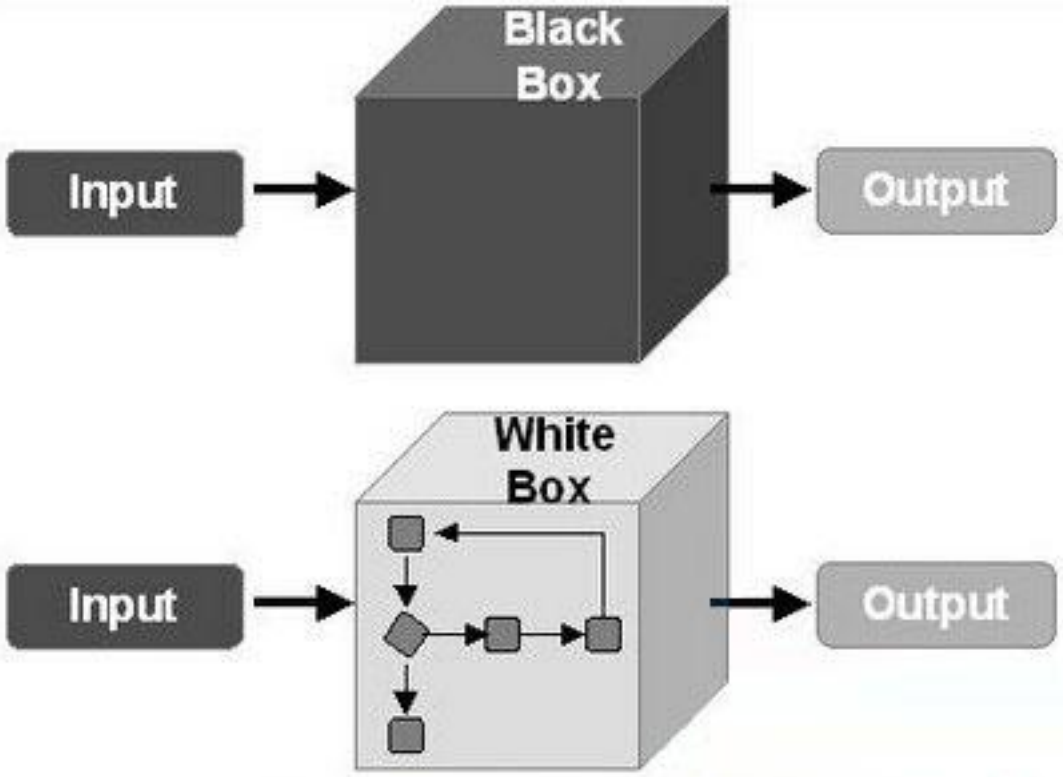
忽略系统或组件的内部机制，仅关注于那些响应所选择的输入及相应执行条件的输出的测试形式

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。

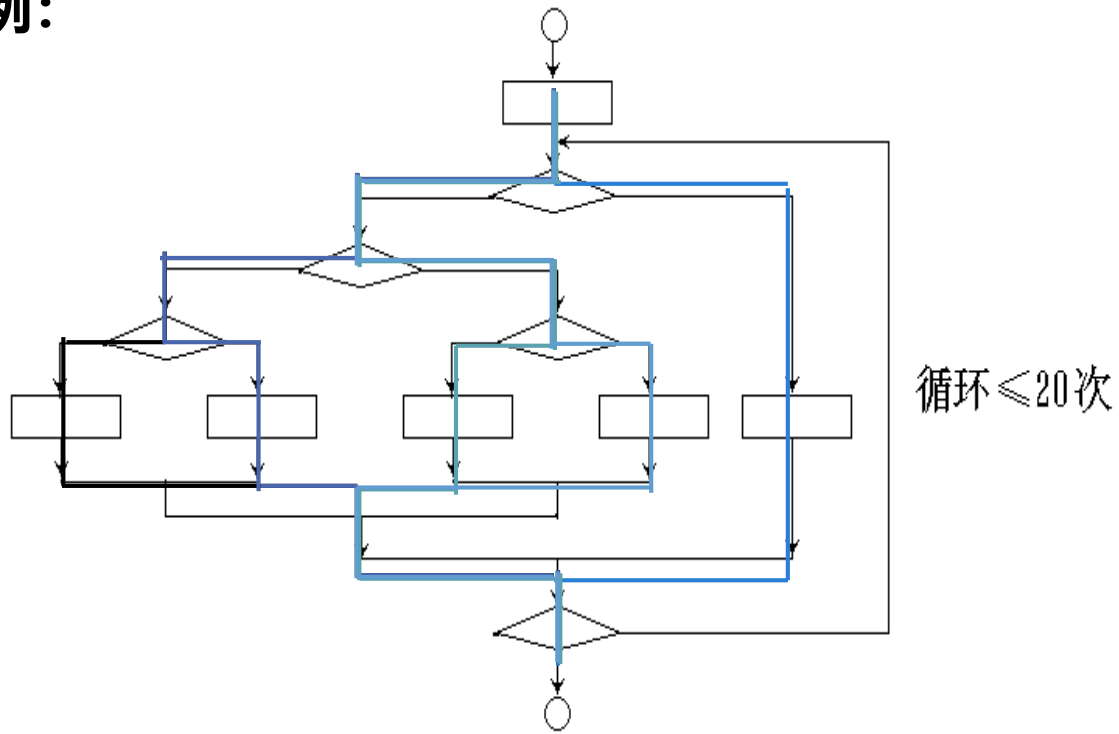
1）白盒测试——检查范围

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

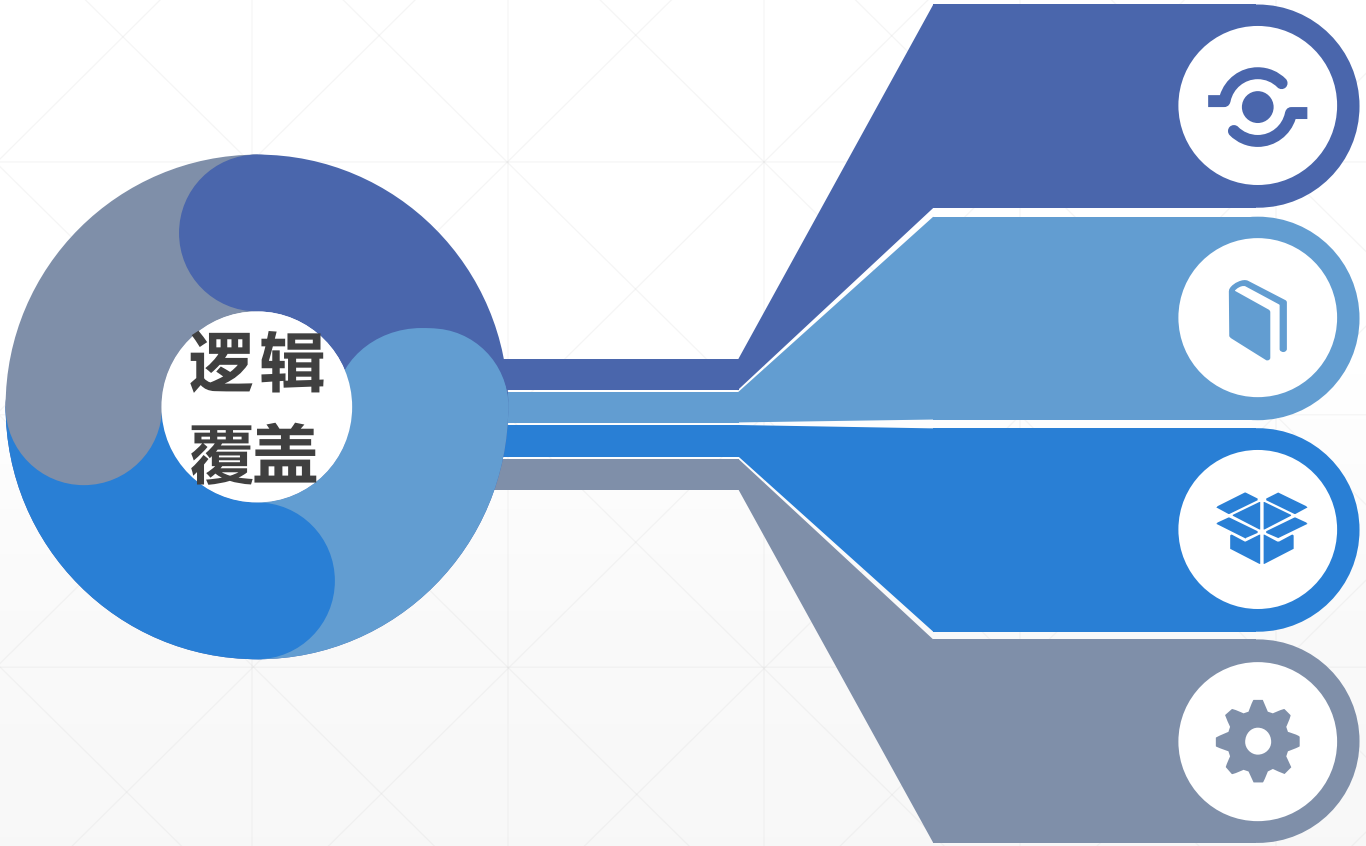
一年工作365 × 24小时

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

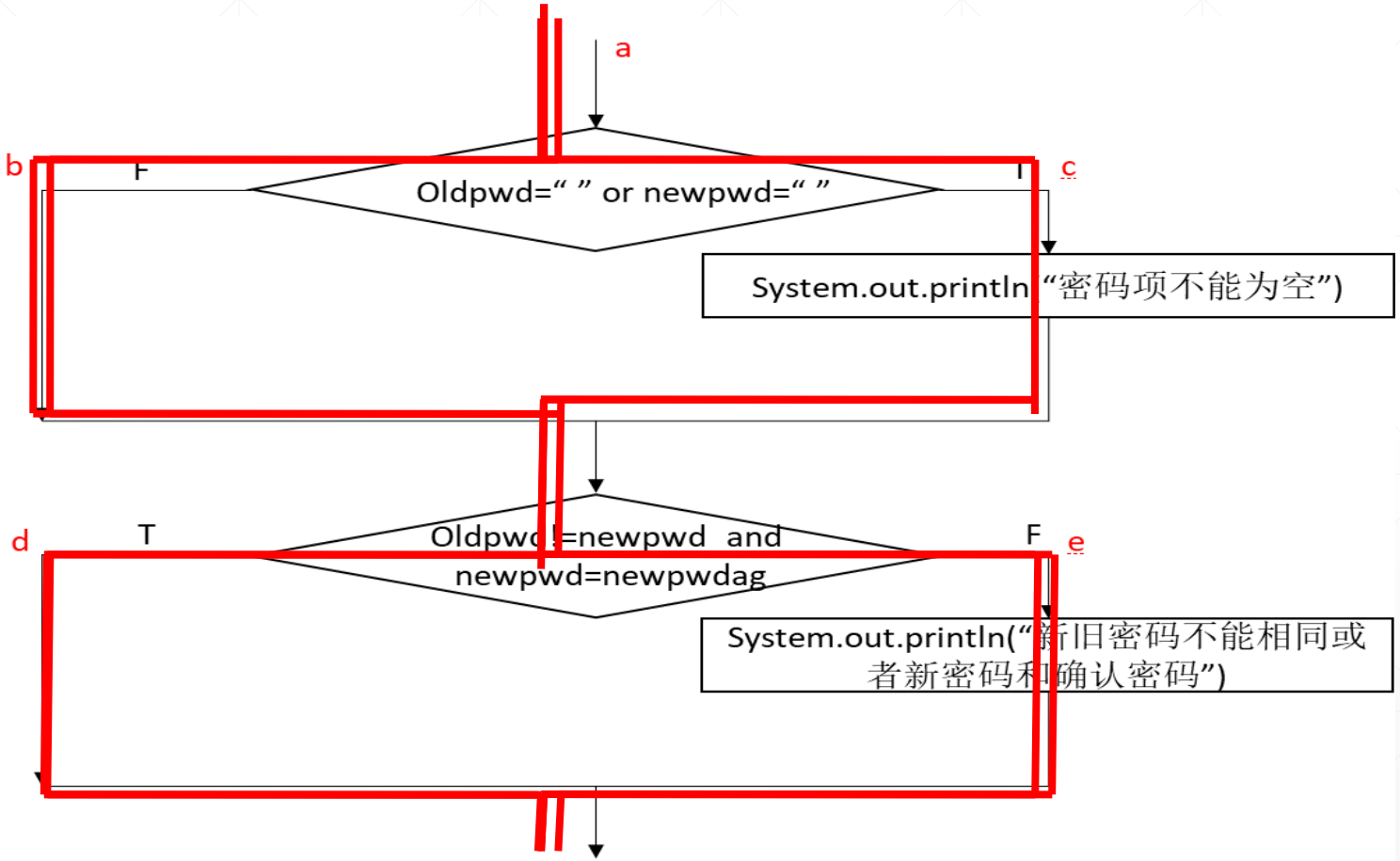
分支覆盖

条件覆盖

条件组合覆盖

语句覆盖

程序流程图示例



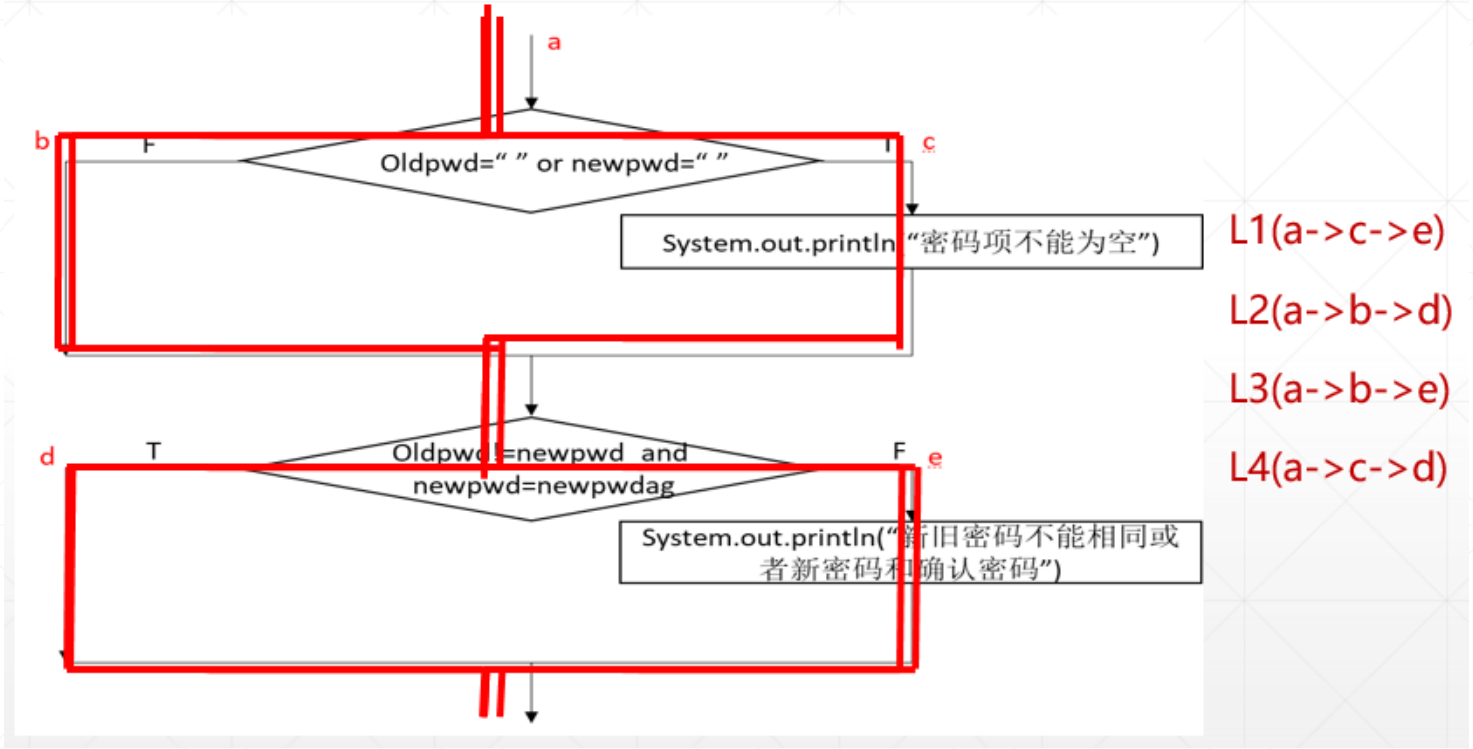
L1(a->c->e)

L2(a->b->d)

L3(a->b->e)

L4(a->c->d)

语句覆盖



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

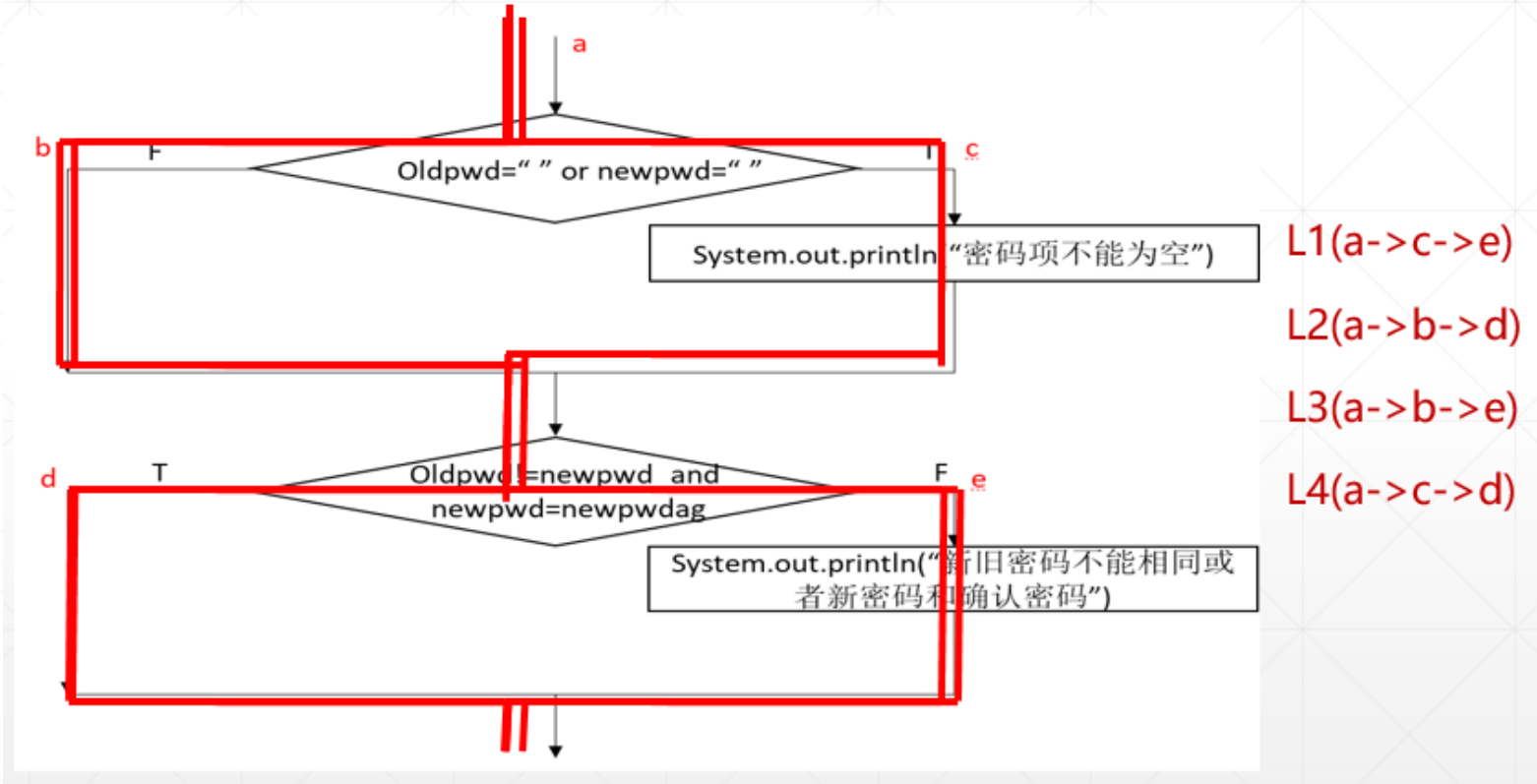
$$= \{oldpwd = "" \text{ or } newpwd = ""\} \text{and } \{oldpwd != newpwd \text{ and } newpwd = newpwdag\}$$

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

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

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

语句覆盖



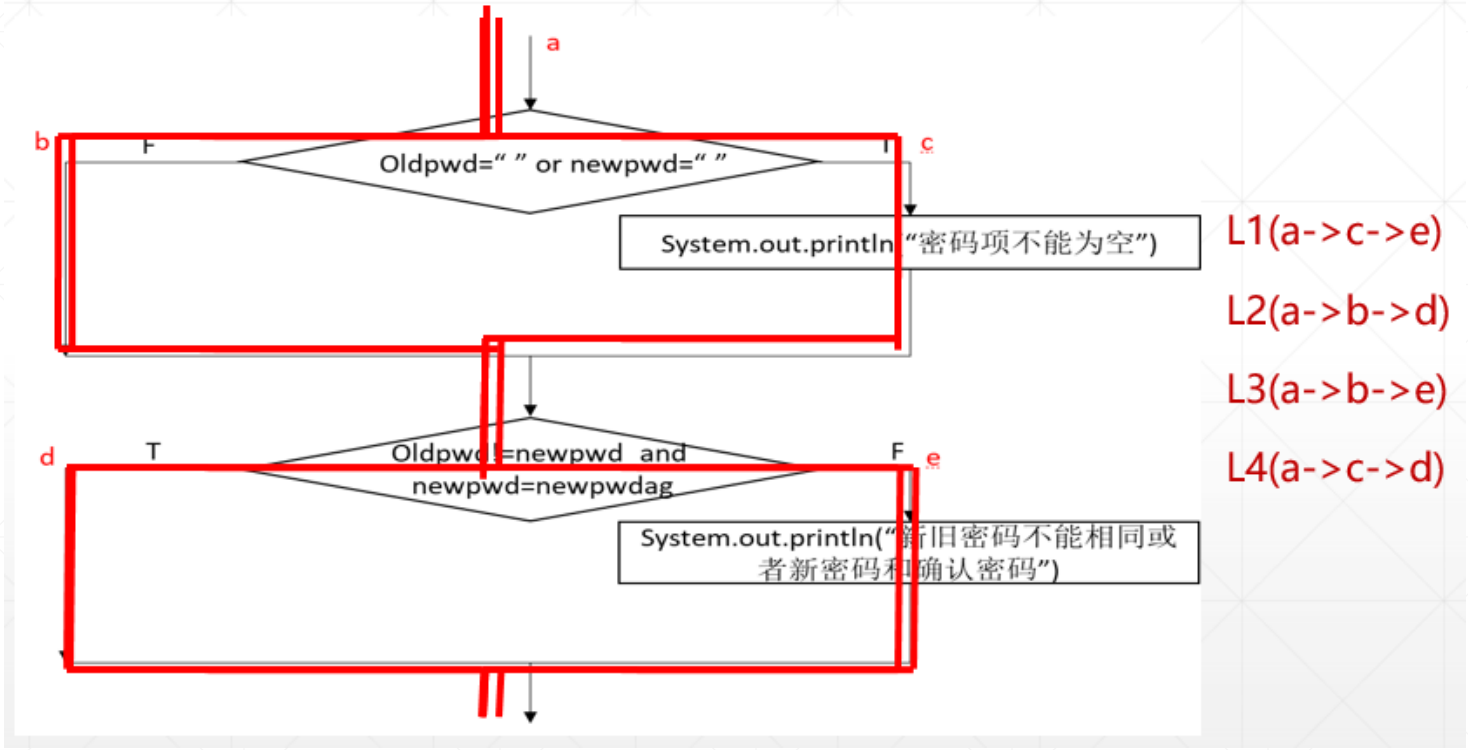
$$L2(a \rightarrow b \rightarrow d)$$

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

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

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

语句覆盖



$$L3(a \rightarrow b \rightarrow e)$$

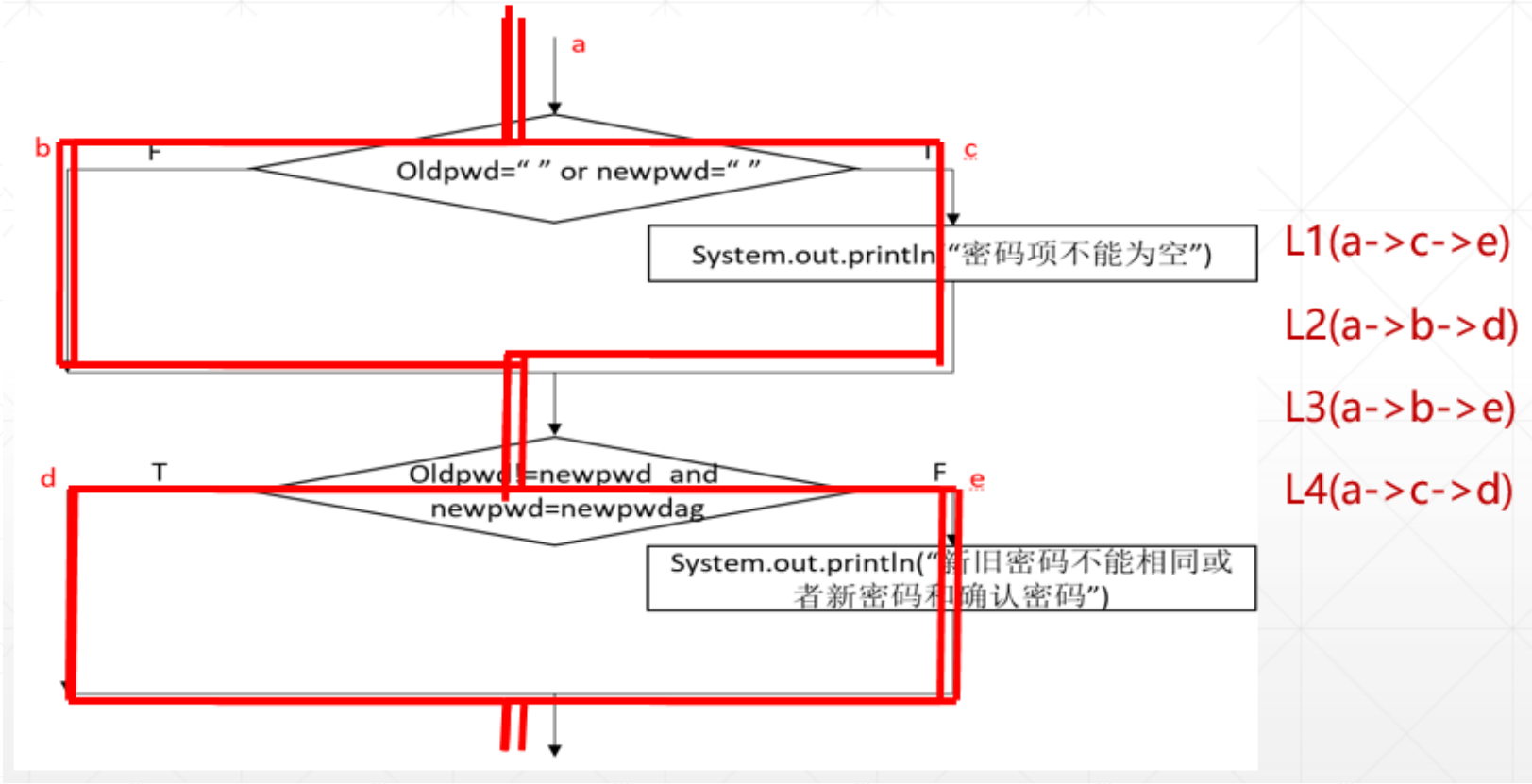
$$= \overline{\{oldpwd = "" \text{ or } newpwd = ""\}} \text{ and } \overline{\{oldpwd != newpwd \text{ and } newpwd = newpwdag}$$

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

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

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

语句覆盖



L4(a → c → d)

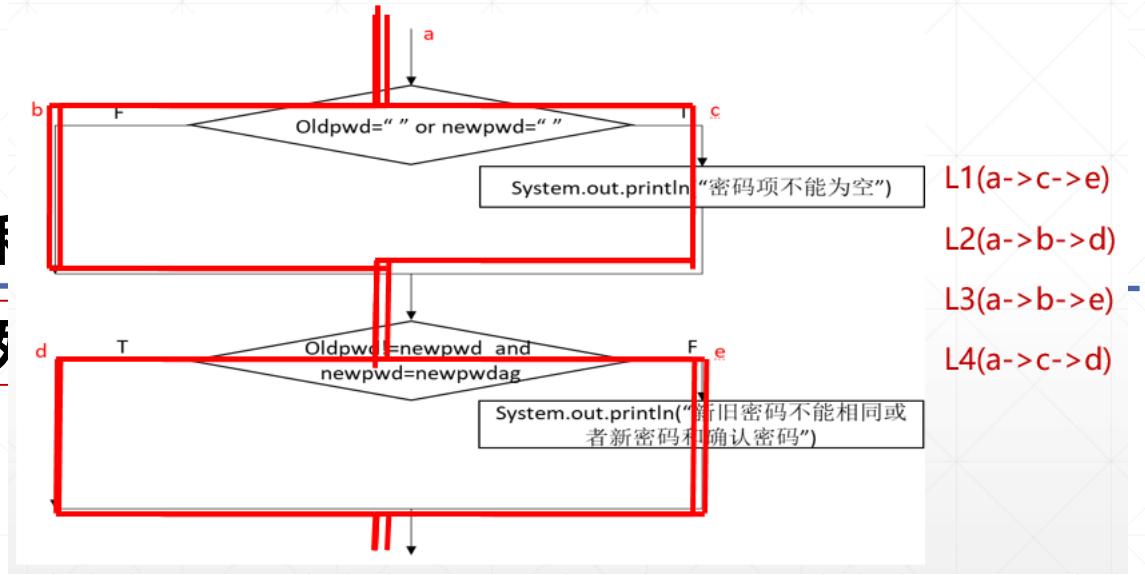
= {oldpwd =" " or newpwd =" "}and {Oldpwd!= newpwd and newpwd = newpwdag}

= 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

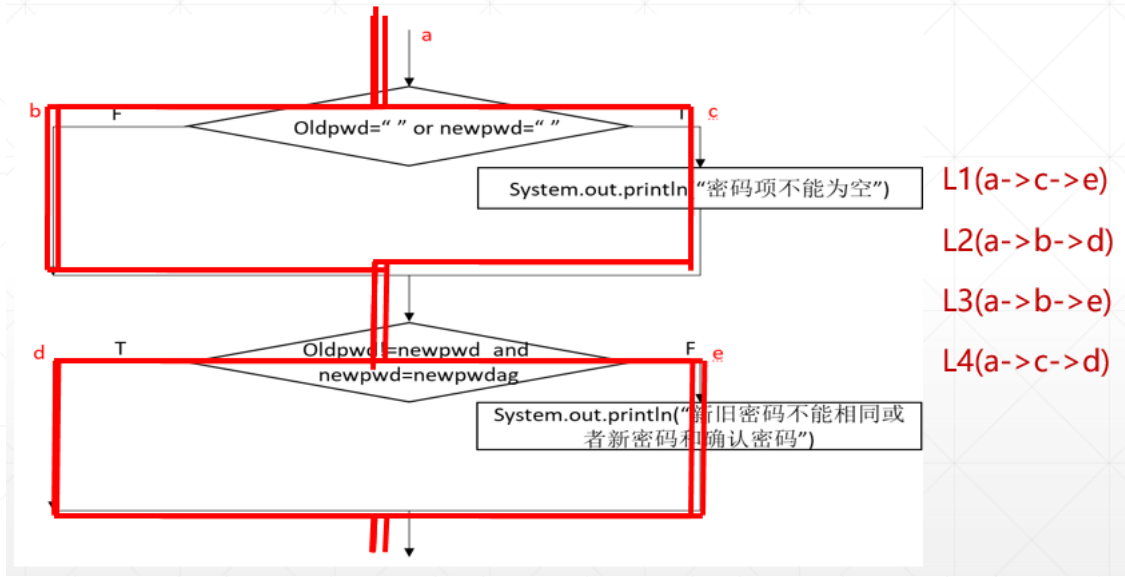
语句覆盖

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:

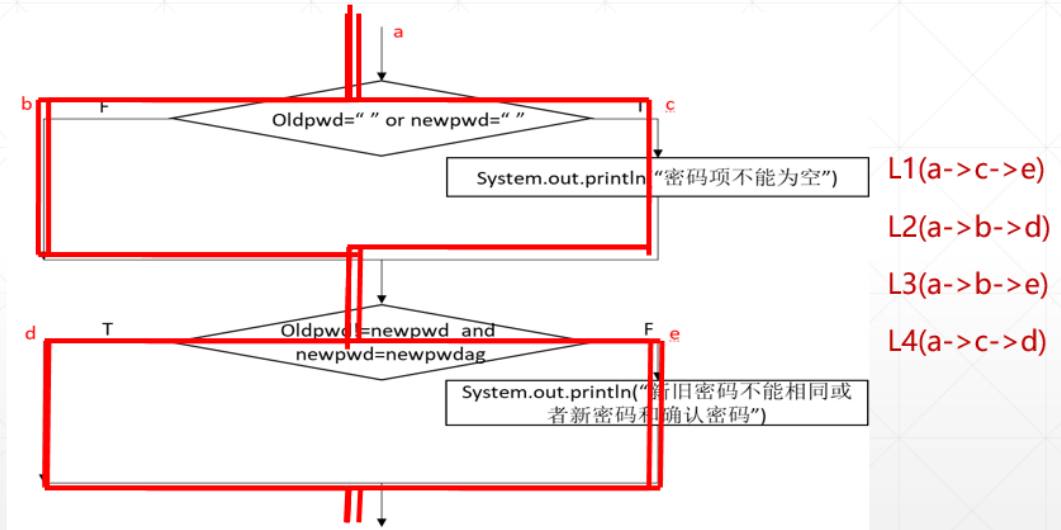
【(“”，“”，“”)，(“密码项不能为空”, “新旧密码不能相同或者新密码和确认密码要相同”】覆盖 ace【L1】

【(“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

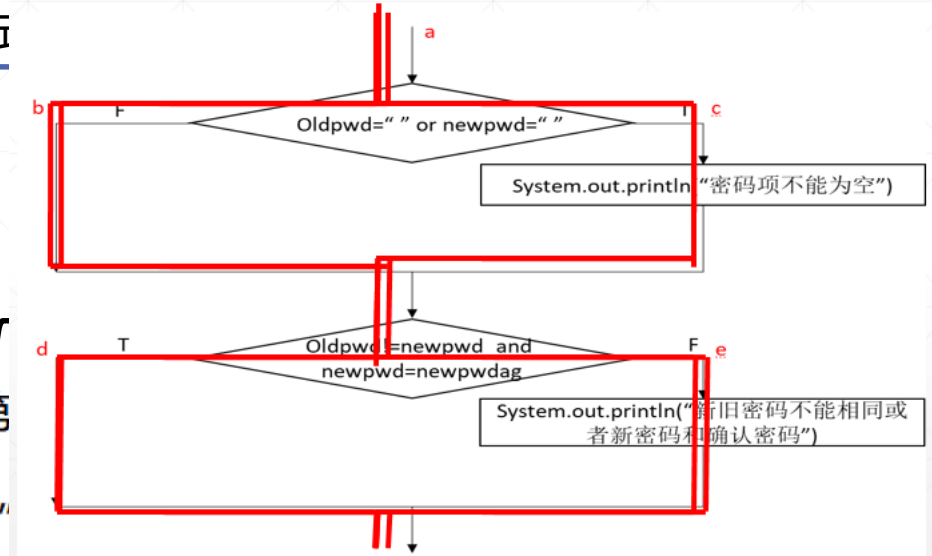
例

- 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，使得每个判定中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[illegible]判断：

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

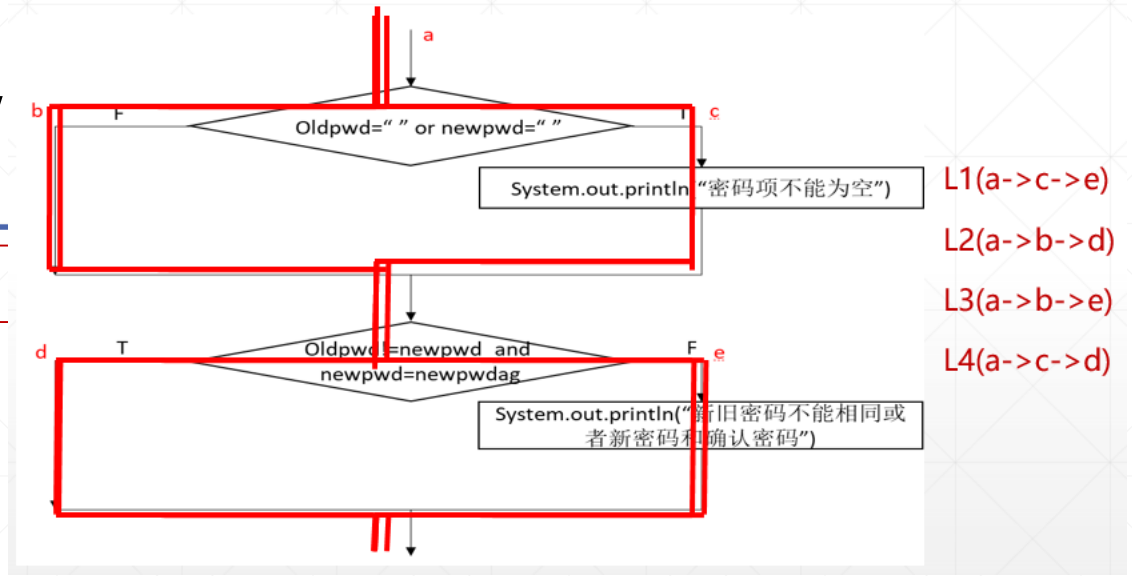
条件 newpwd= “”[illegible]ag取真为T_4，取假为$\overline{T_4}$

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



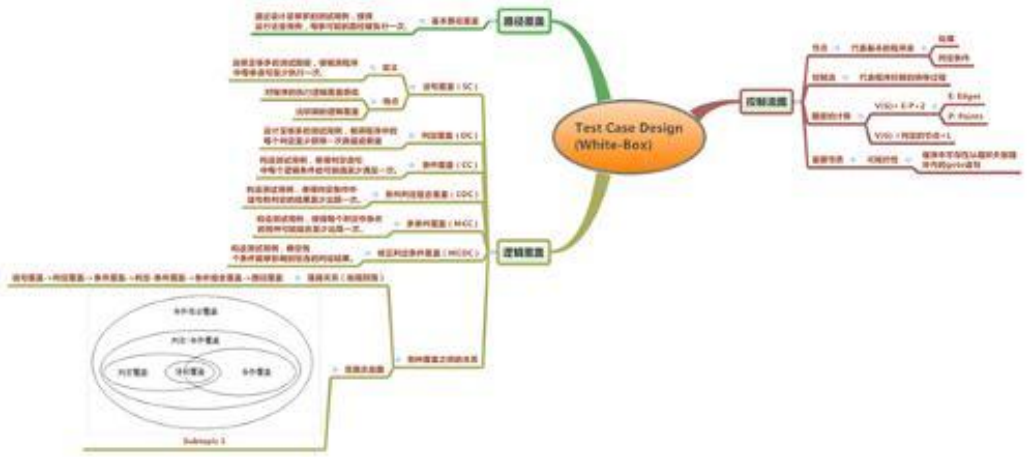
事先对所有条件组合的取值加以标记。

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



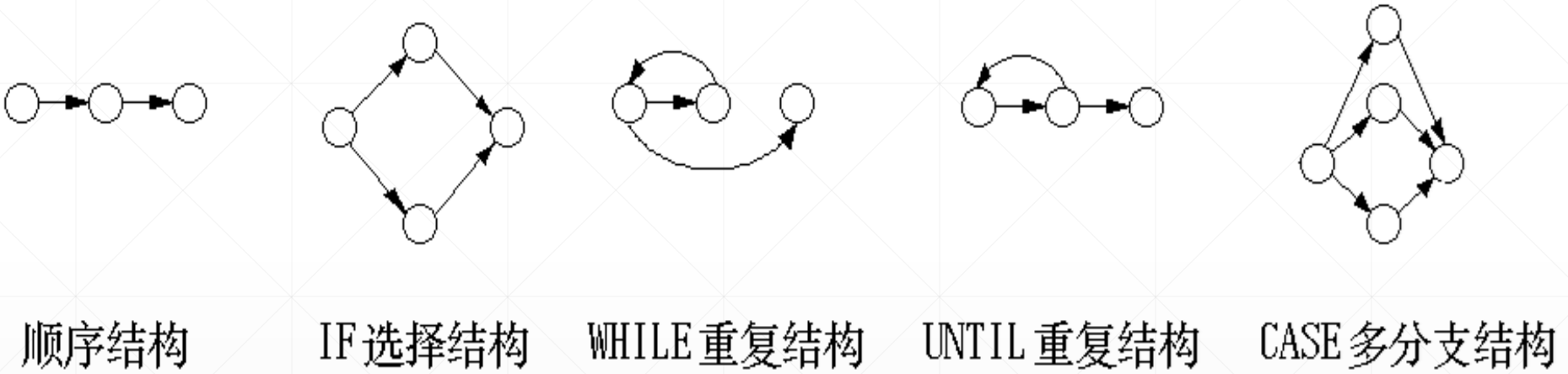
6.2.3.软件测试技术——白盒测试（续）

- 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。

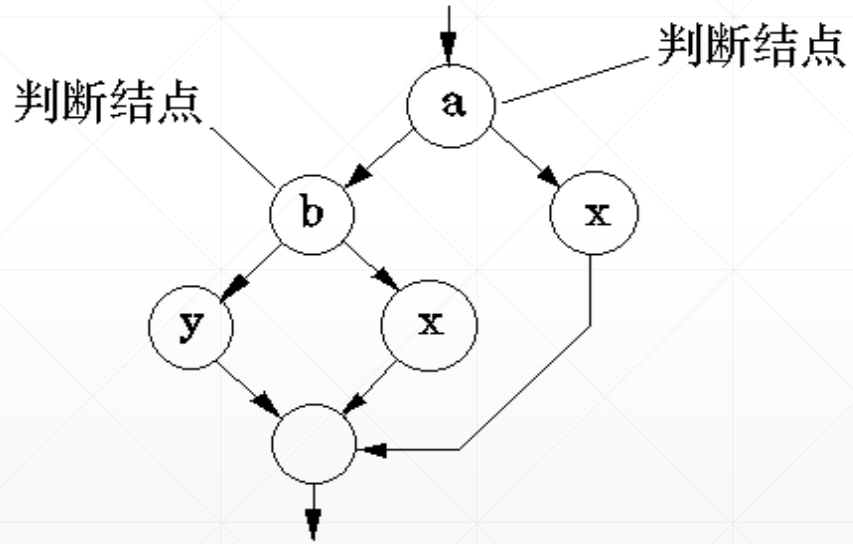
2）基本概念——单条件嵌套

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

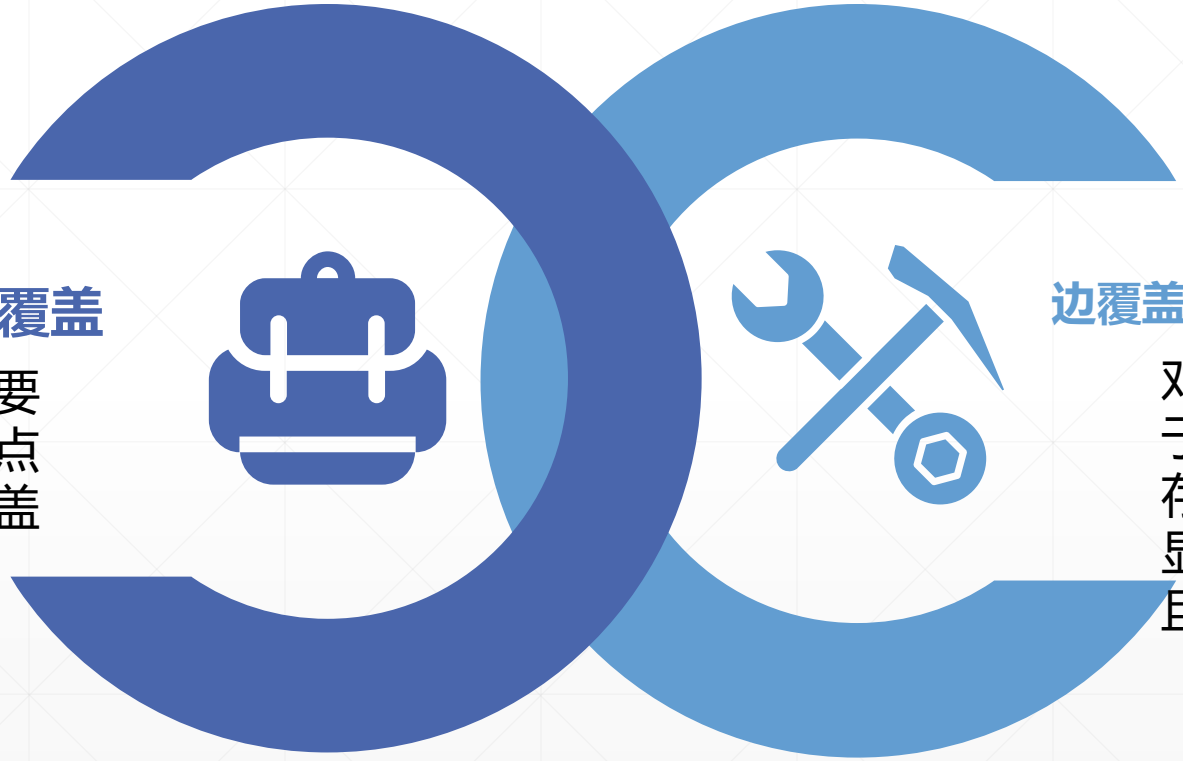
```
.
.
.
if   a OR b
  then  procedure  x
  else  procedure  y;
.
.
.
```

(a)



(b)

3）节点、边覆盖



节点覆盖

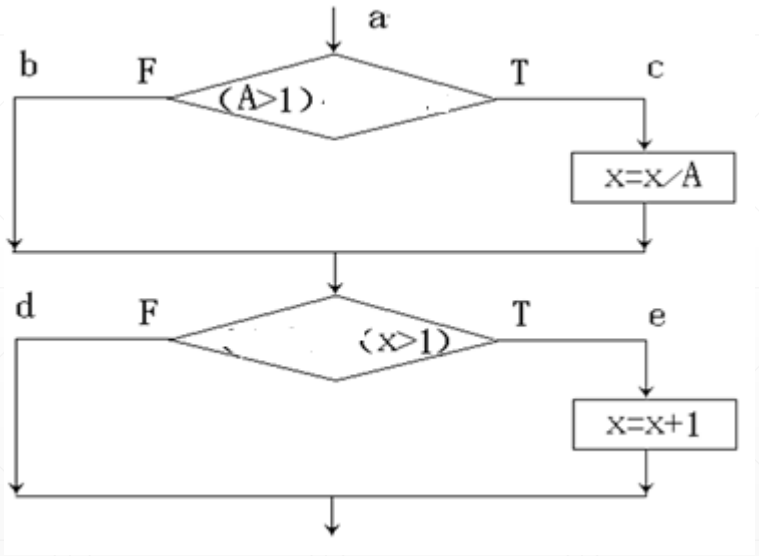
对图中的每个节点，至少要有一条测试路径访问该节点
显然，节点覆盖=语句覆盖

边覆盖

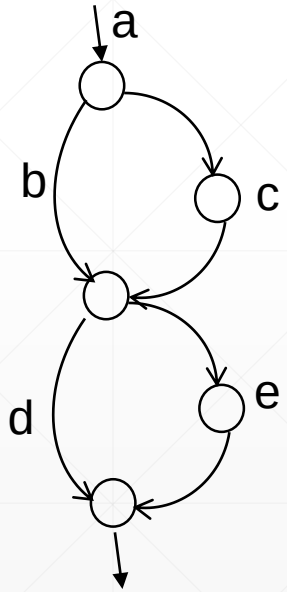
对图中每一个可到达的长度小于(无边图)等于1 的路径，至少存在一条测试路径覆盖。
显然，边覆盖包含节点覆盖，且边覆盖也可以实现分支覆盖。

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

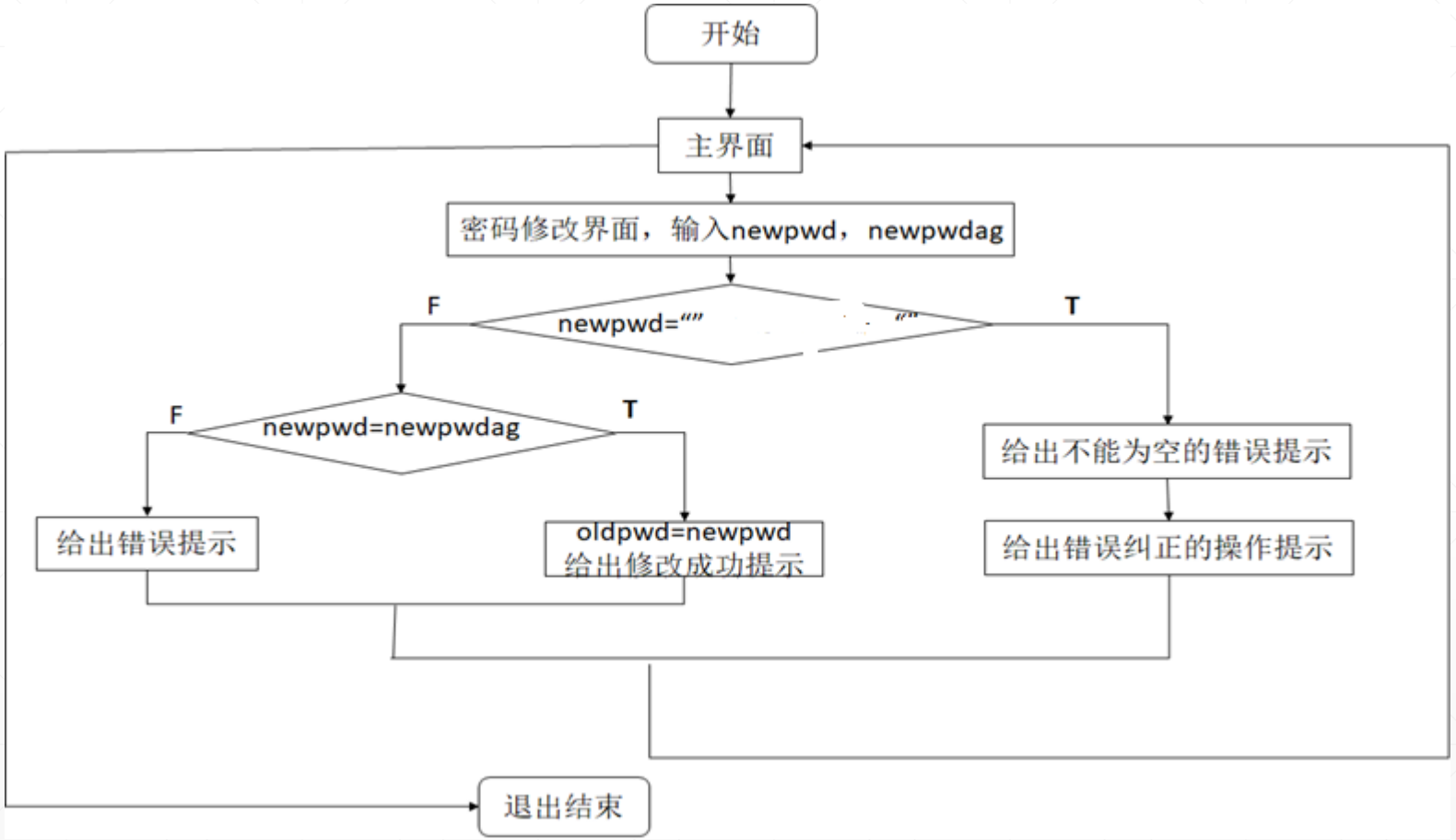
从源节点（控制流图的入口点）开始，一直走到汇节点（控制流图的出口点）。该路径作为基线路径。

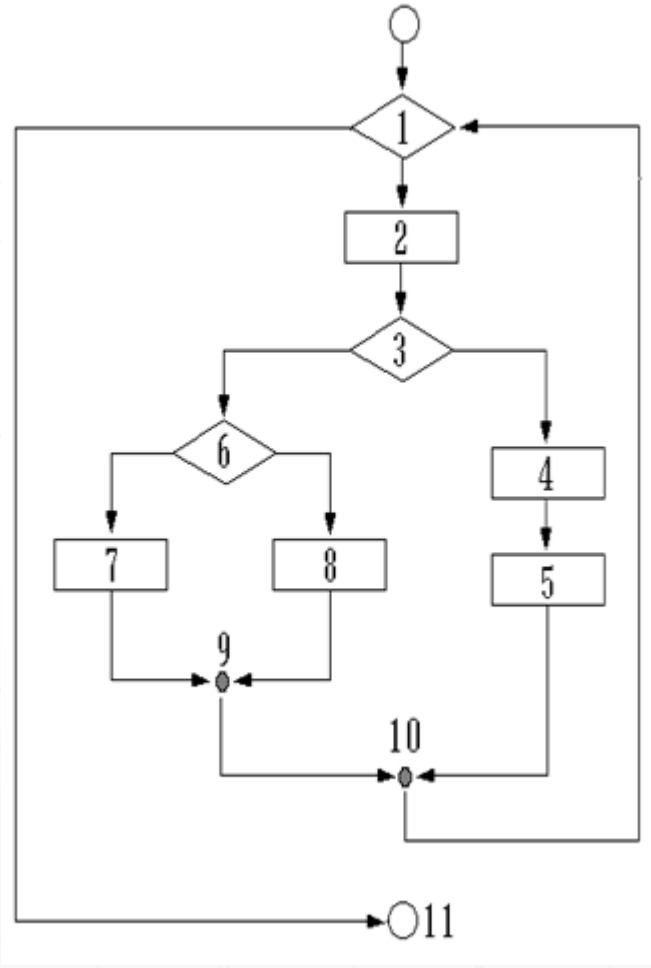
接下来，重新回溯基线路径，依次“翻转”在判断节点上原来选择的路径。即当遇到节点的出度大于等于2 时，必须选择不同的边。

重复以上过程，直到得到的路径数目等于V(G)

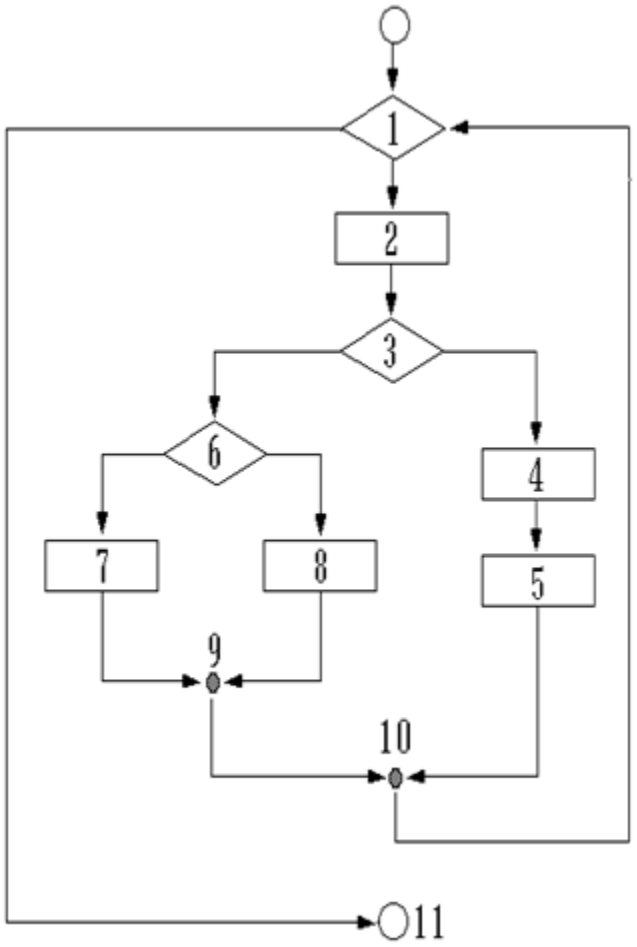
5)基本路径覆盖

例

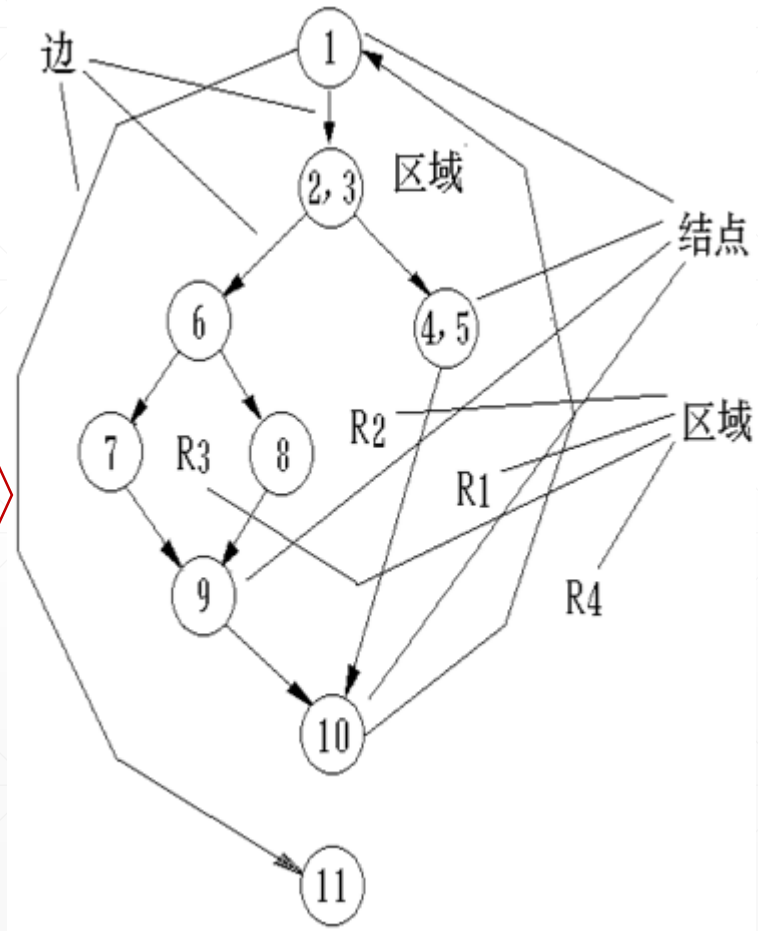

开始
主界面
密码修改界面，输入newpwd，newpwdag
F
newpwd=""
T
F
newpwd=newpwdag
T
给出不能为空的错误提示
给出错误提示
oldpwd=newpwd
给出修改成功提示
给出错误纠正的操作提示
退出结束


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

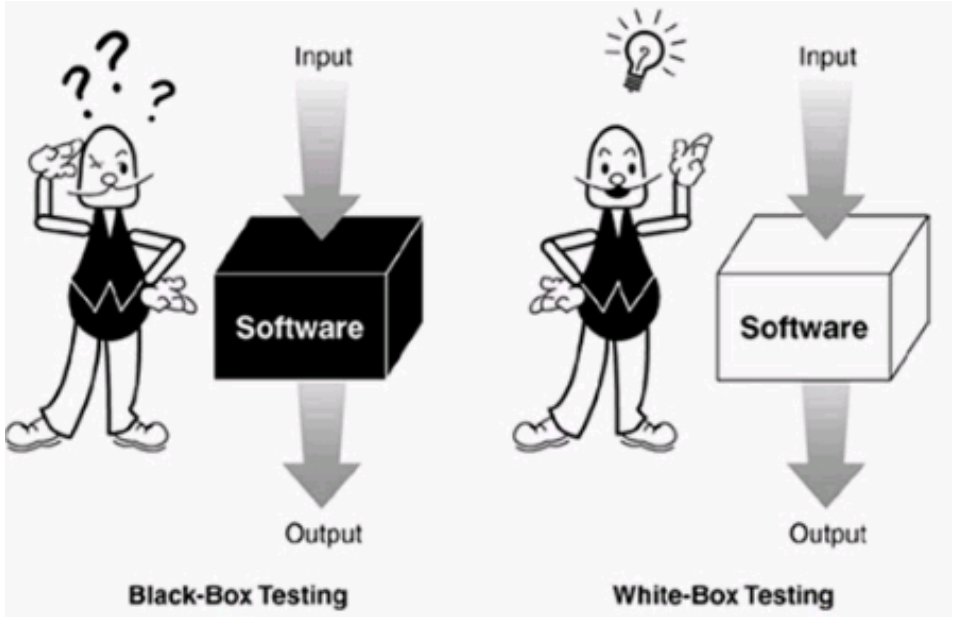
根据判断结点给出的条件，选择合适用例以保证某一条路径可以被测试到

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

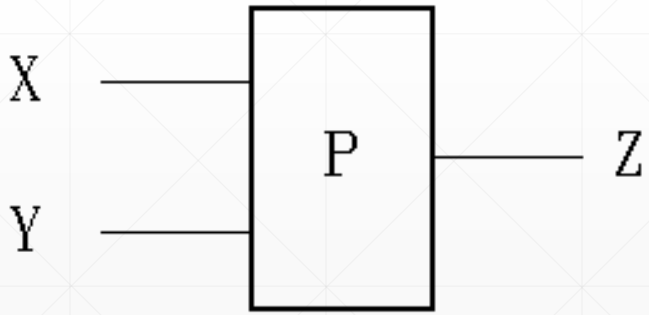
检查范围	是否有不正确或遗漏了的功能?
	在接口上，输入能否正确地接受? 能否输出正确的结果?
	是否有数据结构错误或外部信息访问错误?
	性能上是否能够满足要求?
	是否有初始化或终止性错误?

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

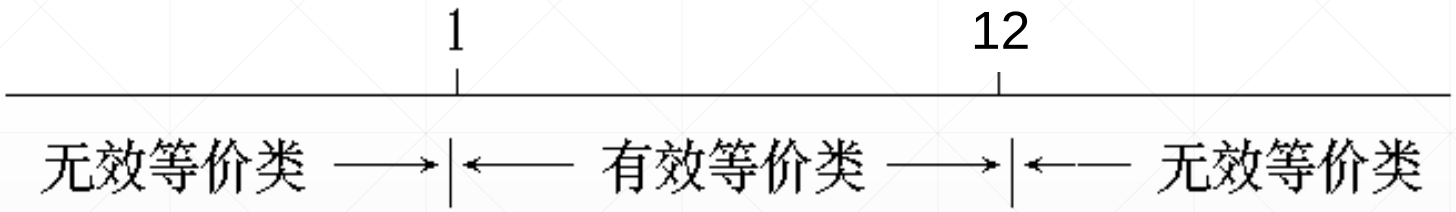
等价类：某个输入域的子集合。在该子集合中，各个输入数据对于测试程序中的缺陷都是等效的。	
有效等价类：对于程序的规格说明来说，是合理的，有意义的输入数据构成的集合。	无效等价类：对于程序的规格说明来说，是不合理的，无意义的输入数据构成的集合。

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)

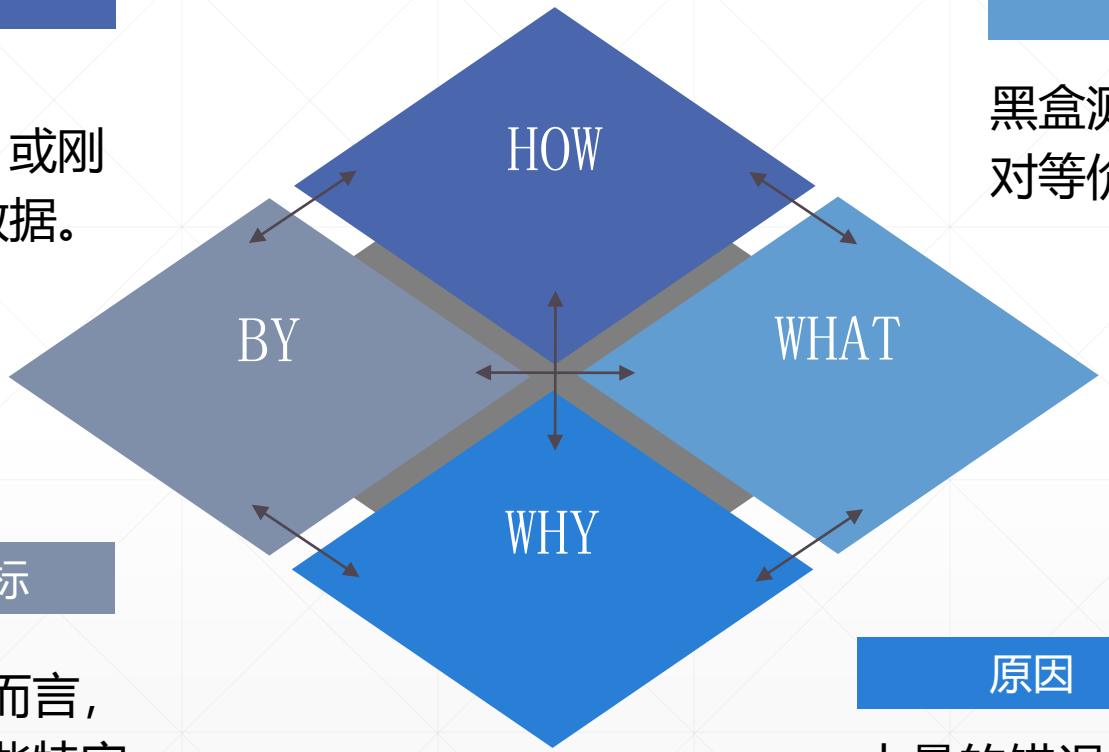
⑧ (z0300,例,是) 覆盖 (11)

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

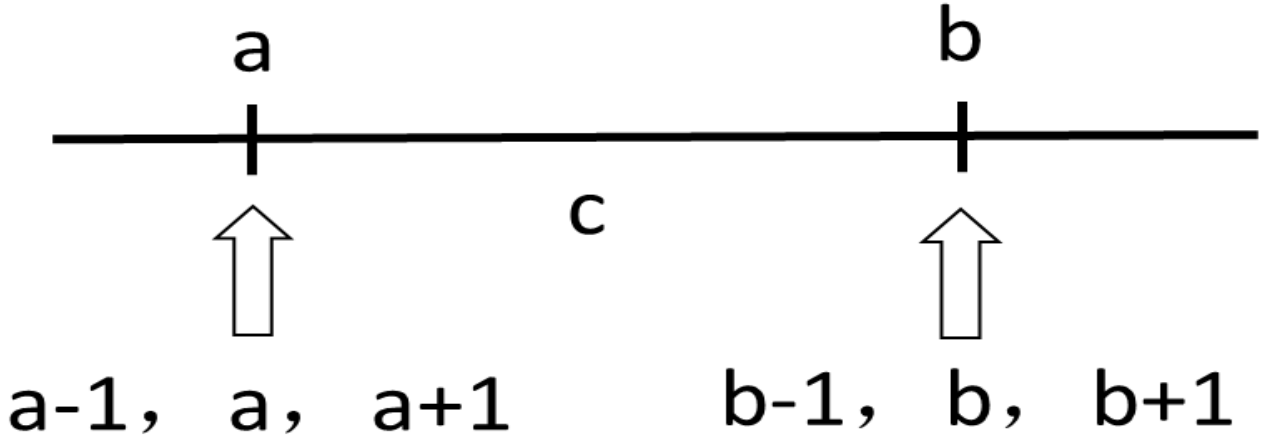
边界值分析-区间范围

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。

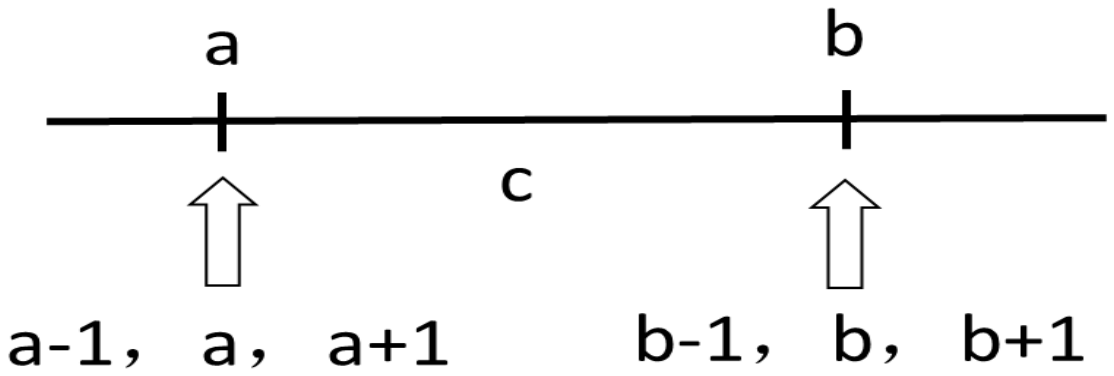
上点，即边界上的点，不管是开区间还是闭区间。

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。

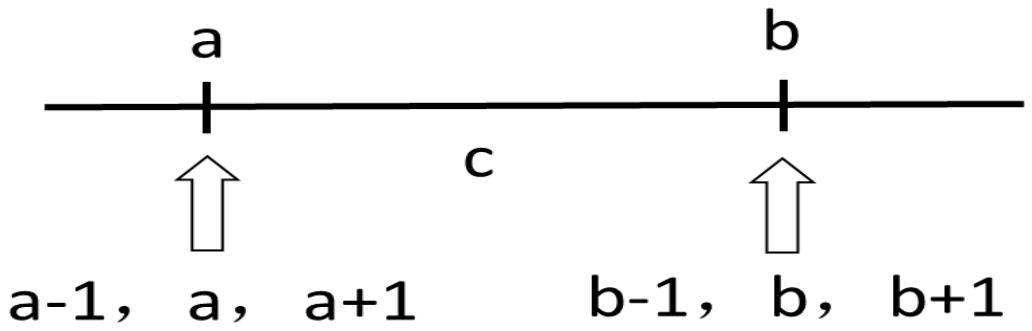
上点：为85和100

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。

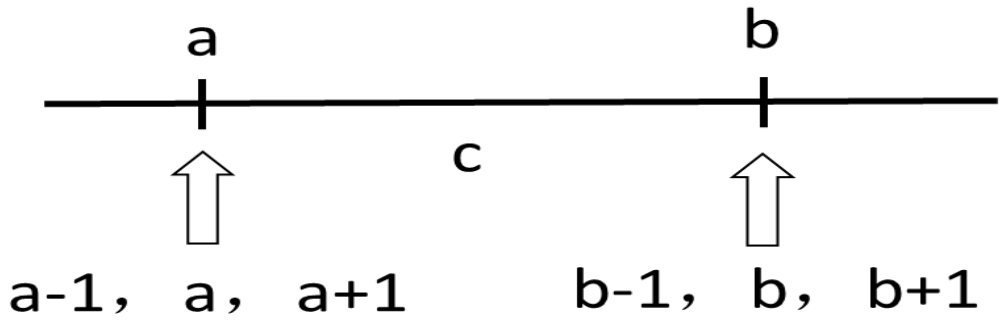
上点：为0和60

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

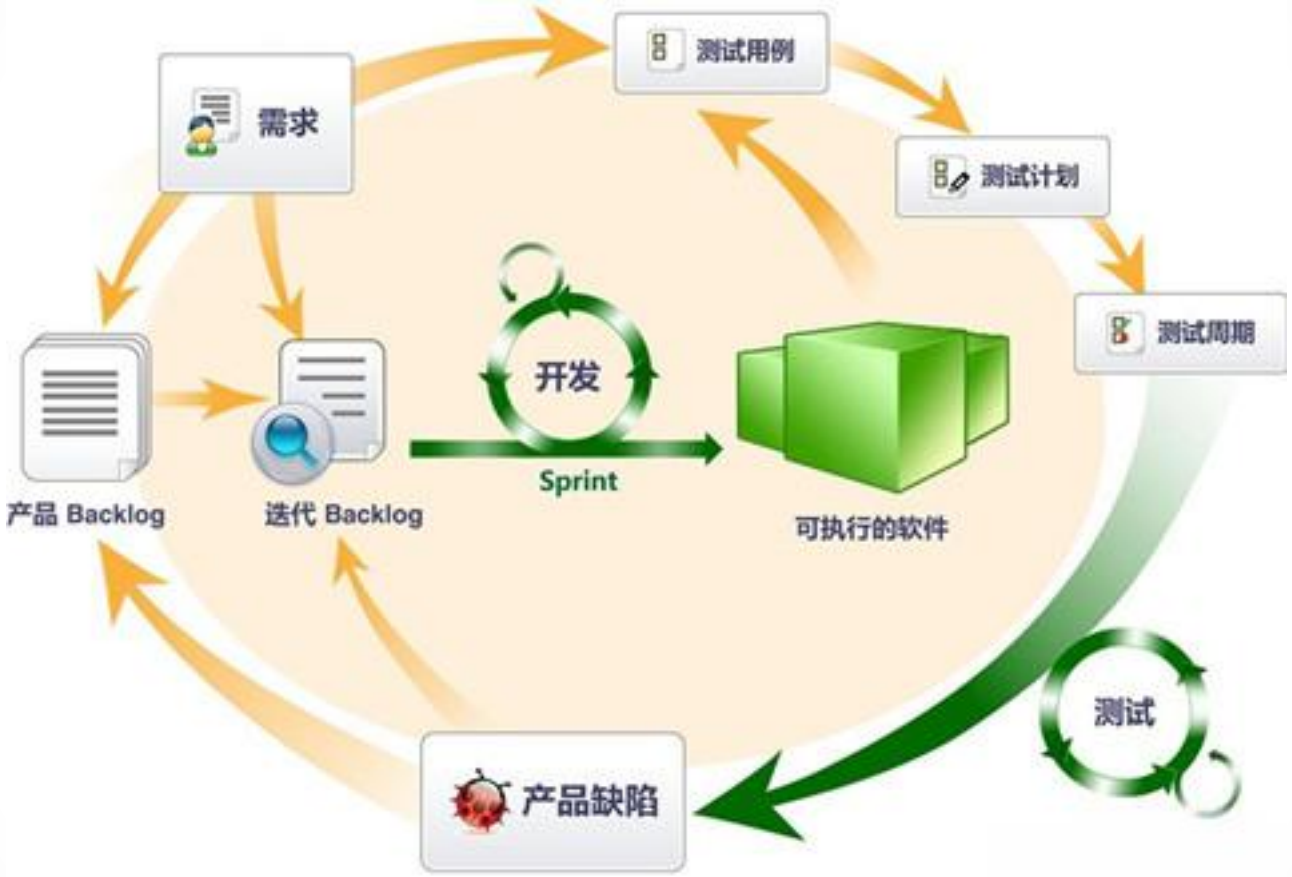
内点：即60到75之间的任何一点，如：67。

离点：为59和74。

测试用例为{59，60，67，74，75}

6.3 软件测试策略

- 软件测试策略概述
- 单元测试
- 集成测试
- 系统测试
- 验收测试



6.3.1 软件测试策略概述

- 软件测试策略含义
- 软件测试策略V模型
- 回归测试
- 软件测试策略注意事项
- 软件测试策略基本步骤

1）软件测试策略含义

软件测试策略

软件测试策略为软件开发人员、质量保证组织、和客户提供了一个路线图

规定了测试的主要步骤

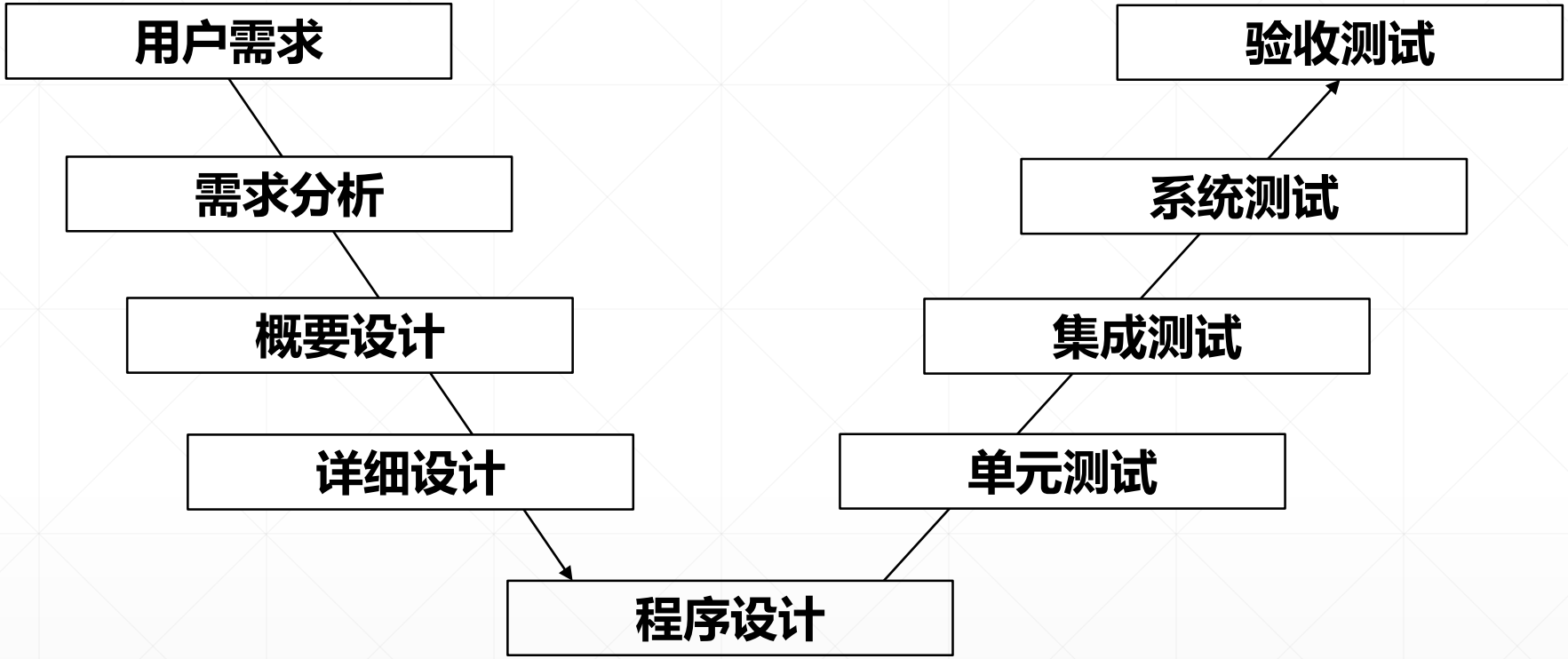
为保证可行性，须考虑人力成本、时间和资源
故应结合：测试计划、测试用例设计、测试执行、测试结果数据的收集与分析

要求

灵活性：有足够的可塑性来应付所有的大软件系统

严　格：保证对项目进程进行合理的计划和跟踪管理

2）软件测试策略V模型



V 模型非常明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段对应关系。

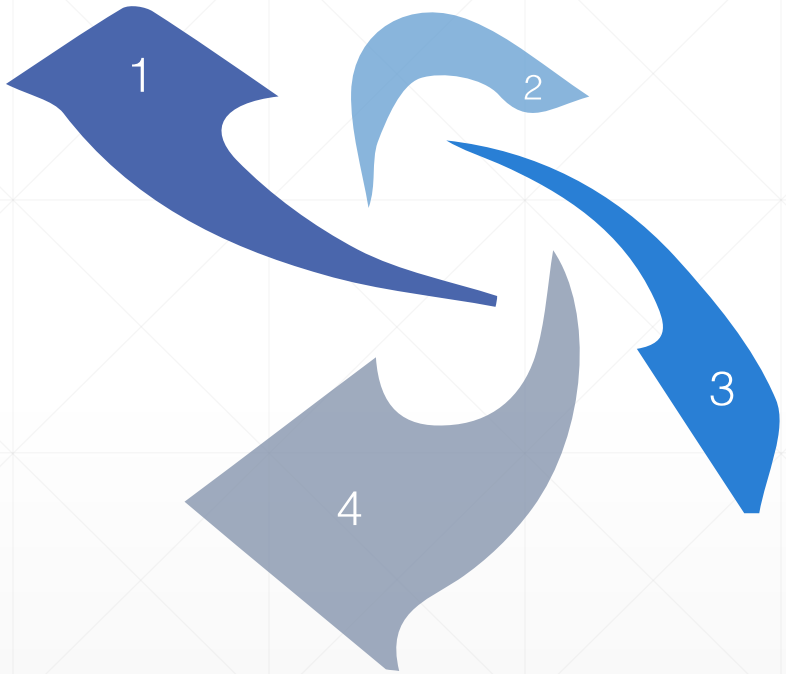
2）软件测试策略V模型

单元测试

主要目的是验证软件模块是否按详细设计的规格说明正确运行。

验收测试

从用户的角度检查系统是否满足合同中定义的需求，以及以确认产品是否能符合业务上的需要。



集成测试

主要目的是检查多个模块间是否按概要设计说明的方式协同工作。

系统测试

主要目的是验证整个系统是否满足需求规格说明。

3）回归测试

WHY

1、测试中，如有缺陷修正、功能增加，变化的部分必须再测试。
2、软件的修改可能会导致新的缺陷及其他问题。为防止，需再测试。

what

回归测试:指有选择地重新测试系统或其组件，以验证对软件的修改没有导致不希望出现的影响，以及系统或组件仍然符合其指定的需求。

BY

回归测试应该尽量采用自动化测试。

3）回归测试

回归测试可以在所有的测试级别执行，并应用于功能和非功能测试中。

范围

缺陷再测试：重新运行所有发现故障的测试，而新的软件版本已经修正了这些故障。

功能改变的测试：测试所有修改或修正过的程序部分。

新功能测试：测试所有新集成的程序。

完全回归测试：测试整个系统。

自动化测试

将以人为驱动的传统测试转为以机器执行的测试方法。

步骤

测试需求的分析，如果确认项目适合自动化测试，则确定自动化测试的范围和相应的测试用例和测试数据，并形成文档；

定义测试需要调用的文件、结果、过程和涉及的结构，其中会被重复使用的对象、方法、环境可以抽取出来或封装、重用

一般先进行脚本录制，通过测试工具提供的录制功能，运行程序自动录制生成脚本，再对脚本进行手工编写加工，添加结构化调用、函数调用等内容。

检查脚本的正确性；进行自动化测试，测试结果分析。

自动化测试

适用范围	回归测试，重复单一的测试操作造成了不必要的时间浪费和人力浪费；
	测试人员对程序的理解和对设计文档的验证通常也要借助于测试自动化工具；
	采用自动化测试工具有利于测试报告文档的生成和版本的连贯性；
	自动化工具能够确定测试用例的覆盖路径，确定测试用例集对程序逻辑流程和控制流程的覆盖。

自动化测试工具：QTP、WinRunner、Rational Robot、AdventNet QEngine、SilkTest 、Test Partner、Holodeck、Telelogic TAU、AutoRunner、Phoenix Framework等。

4）软件测试策略注意事项

- 在着手开始测试之前，要对产品的需求进行量化。
- 明确指出测试目标。
- 为每类用户建立描述交互场景的用例。
- 建立一个强调“快速循环测试”的测试计划。
- 设计一个能够测试自身是否“强壮”的软件。
- 在进行测试之前，对软件进行有效的正式技术审核。
- 使用正式技术审核来评估测试策略和测试用例本身。
- 为测试过程建立一种持续的改进方法。

5）软件测试策略基本步骤

单元测试、集成测试和系统测试

制定计划

编写与评审测试用例

编写测试脚本和准备测试环境

搭建测试环境

构造测试数据

执行测试并记录问题

确认问题

撰写测试报告

6.3.2.软件测试策略——单元测试

- 单元测试概念
- 进入与退出条件
- 测试内容
- 测试用例设计
- 测试环境搭建

1）单元测试概念

针对软件设计的最小单位－程序模块，进行正确性检验的测试工作。

单元测试

测试方法

单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

不同环境含义不同，面向过程：函数、过程等，面向对象：类、类中成员函数等。

单元内涵

主要依据

详细设计

2）进入和退出条件

√ 所用测试用例执行通过
√ 单元测试覆盖率达到预定要求
√ 单元测试未被执行的代码进行正式审查。

√ 被测代码编译链接通过
√ 被测代码静态检查工具检查通过
√ 已完成至少一轮代码检视或走读
√ 单元测试用例的检视通过
√ 单元测试代码写完并通过检测

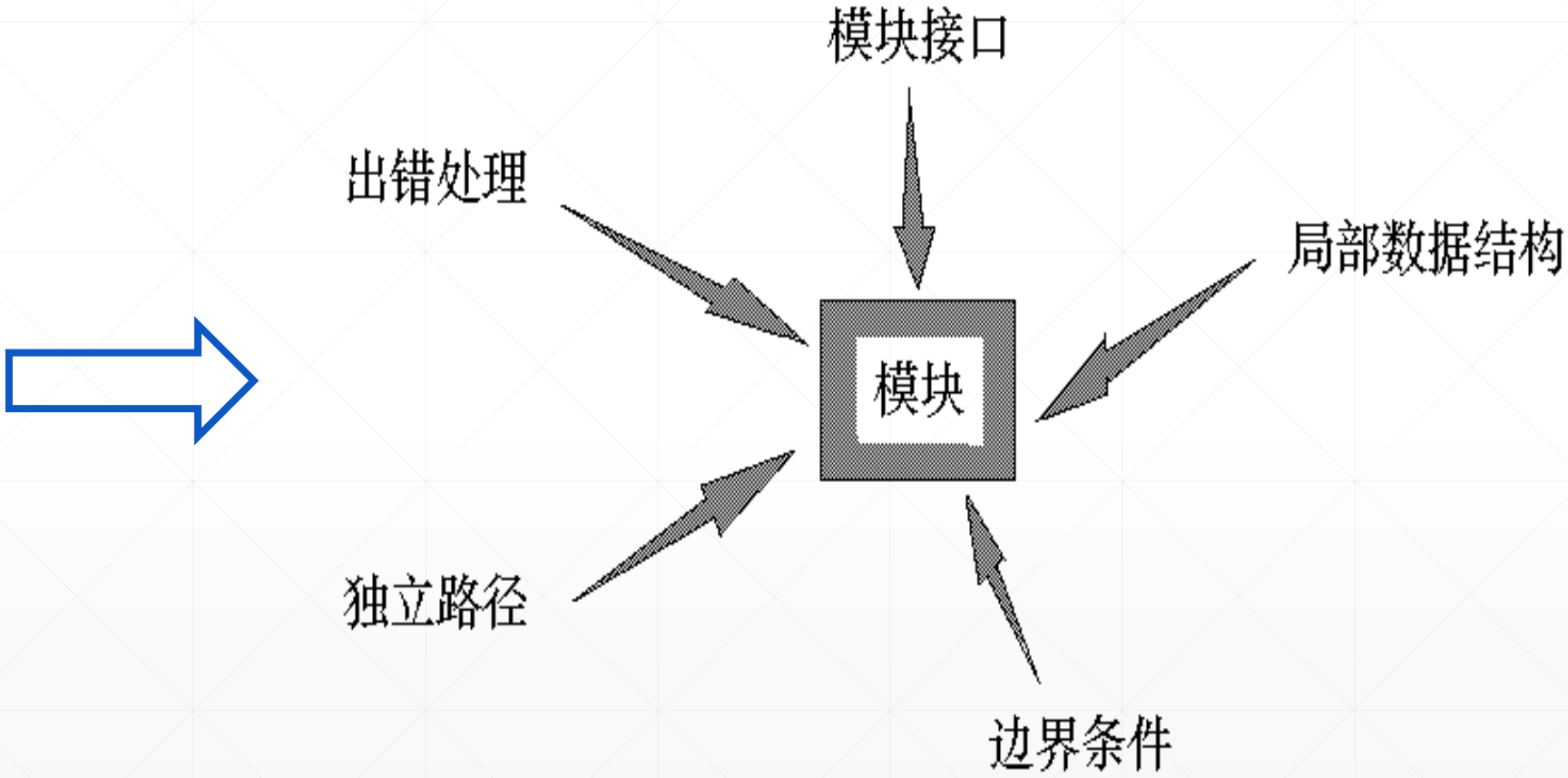
进入条件

退出条件

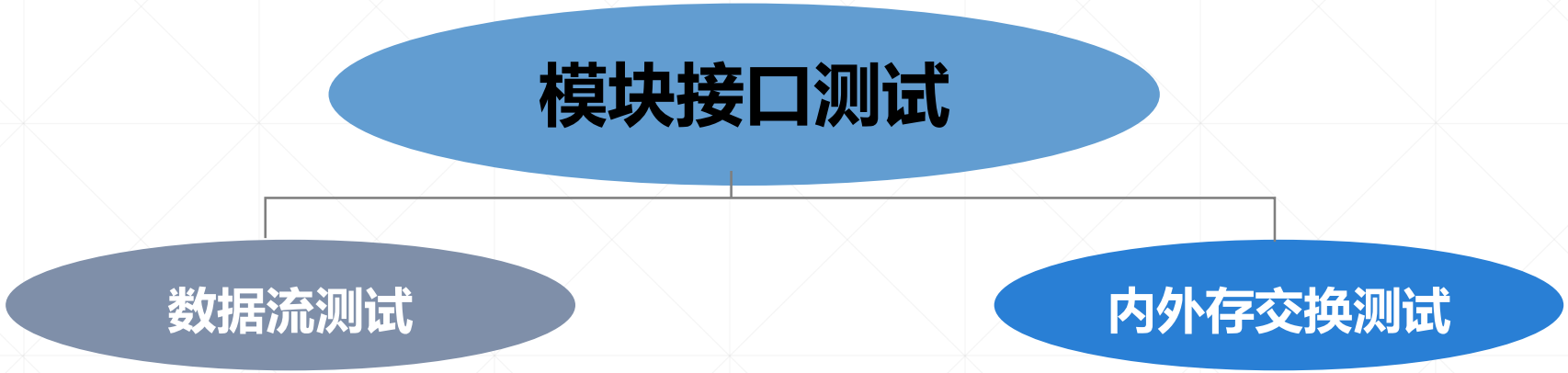
3）测试内容

单元测试

主要内容



3-1）测试内容：模块接口测试



√ 调用本模块的输入参数是否正确；

√ 本模块调用子模块时输入给子模块的参数是否正确；

√ 全局量的定义在各模块中是否一致；

√ 文件属性是否正确；

√ OPEN与CLOSE语句是否正确；

√ 缓冲区容量与记录长度是否匹配；

√ 在进行读写操作之前是否打开了文件；

√ 在结束文件处理时是否关闭了文件；

√ 正文书写 / 输入错误，

√ I / O错误是否检查并做了处理。

3-2）测试内容——局部数据结构测试

局部数据结构测试	不正确或不一致的数据类型说明
	使用尚未赋值或尚未初始化的变量
	错误的初始值或错误的缺省值
	变量名拼写错或书写错误
	不一致的数据类型
	全局数据对模块的影响

3-3）测试内容——路径测试

内容

- 重要路径：对模块中重要的执行路径进行测试。
- 计算、控制流：查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。
- 基本、循环：对基本执行路径和循环进行测试可以发现大量的路径错误。

3-4）测试内容——错误处理测试

出错的描述是否难以理解

在对错误进行处理之前，错误条件是否已经引起系统的干预等

出错的描述是否能够对错误定位

对错误条件的处理正确与否

显示的错误与实际的错误是否相符

3-5）测试内容——边界测试

边界测试

流边界

- 注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，加以测试。

关键路径

- 如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。

4）测试用例设计

依据

详细设计说明书和源程序清单

了解

该模块的I/O条件和模块的逻辑结构

手段

主：白盒测试

辅：黑盒测试

结果

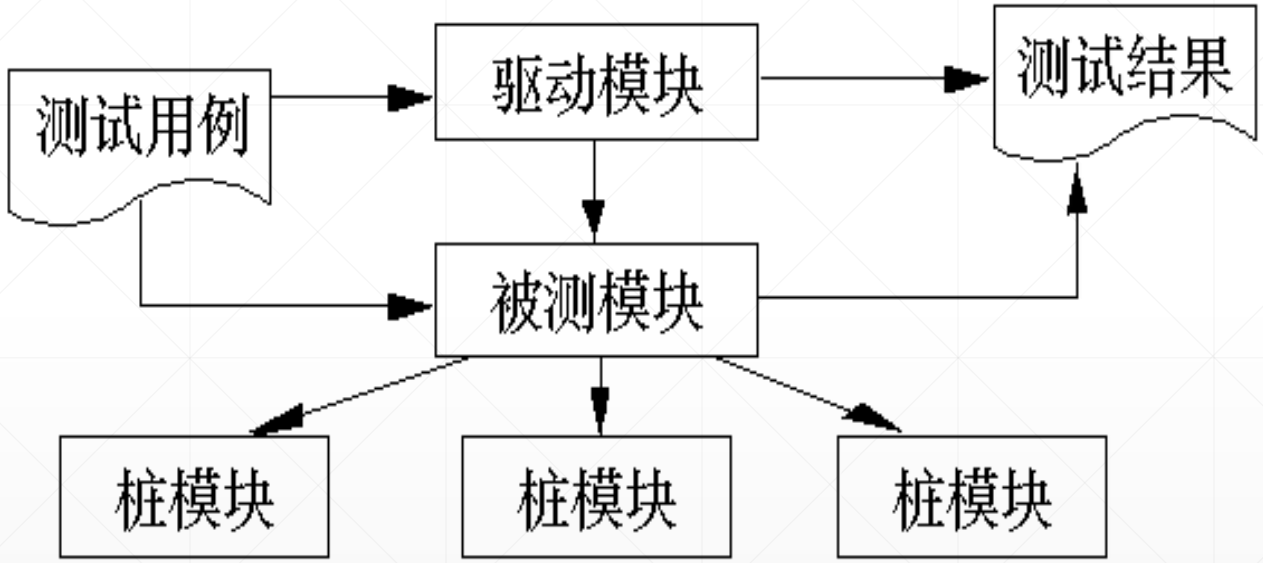
合理和不合理输入的鉴别

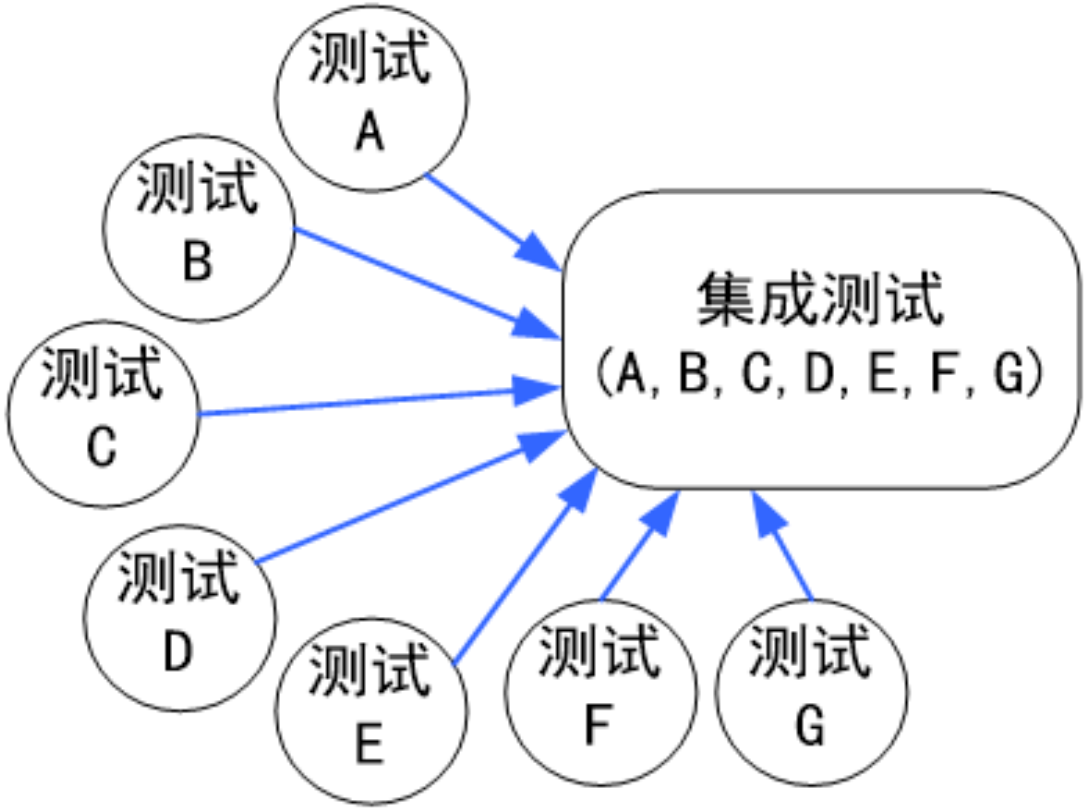
比较预计和实际结果

5）测试环境搭建

模块并非独立程序，进行测试时，要考虑它和外界的联系，需用一些辅助模块去做相应模拟

驱动模块	用来模拟被测试模块的上一级模块，相当于被测模块的主程序。
桩模块	模拟被测试的模块所调用的模块，而不是软件产品的组成的部分。





6.3.3.软件测试策略——集成测试

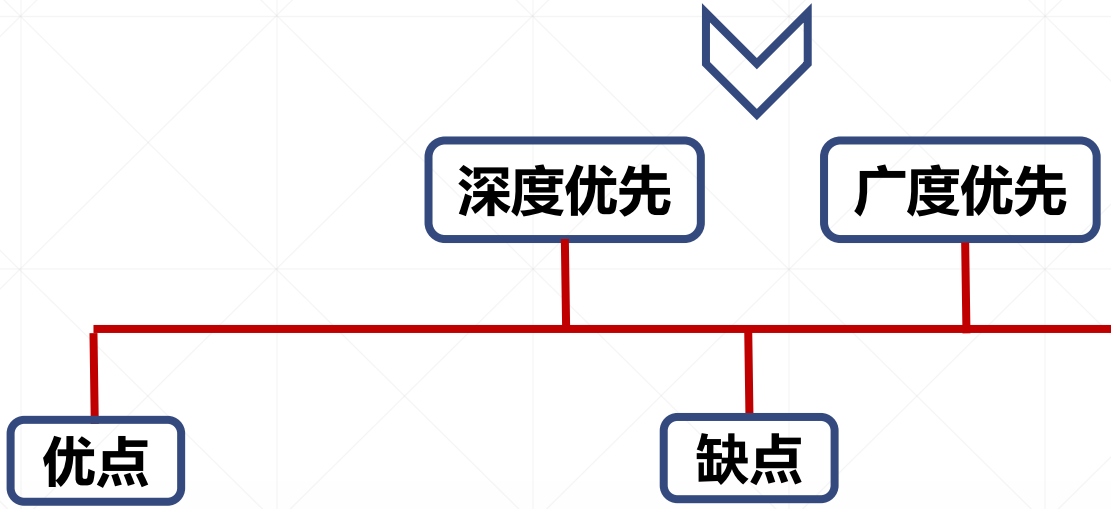
- 集成测试概念
- 自顶向下的集成方法
- 自底向上的集成方法
- 应用注意事项
- SMOKE方法
- 测试用例设计

1）集成测试概念

含义	将软件集成起来后进行测试。 别名：子系统测试、组装测试、部件测试等。
目的	检查诸如两个模块单独运行正常，但集成起来运行可能出现问题的情况。
主要方法	自顶向下的集成方法 自底向上的集成方法 SMOKE方法

2）自顶向下的集成方法

基本思想：该集成方式将模块按系统程序结构，沿控制层次自顶向下进行集成。



优点

可以较早地验证主要的控制和判断点。
按深度方向，可首先实现和验证一个完整的软件功能。

缺点

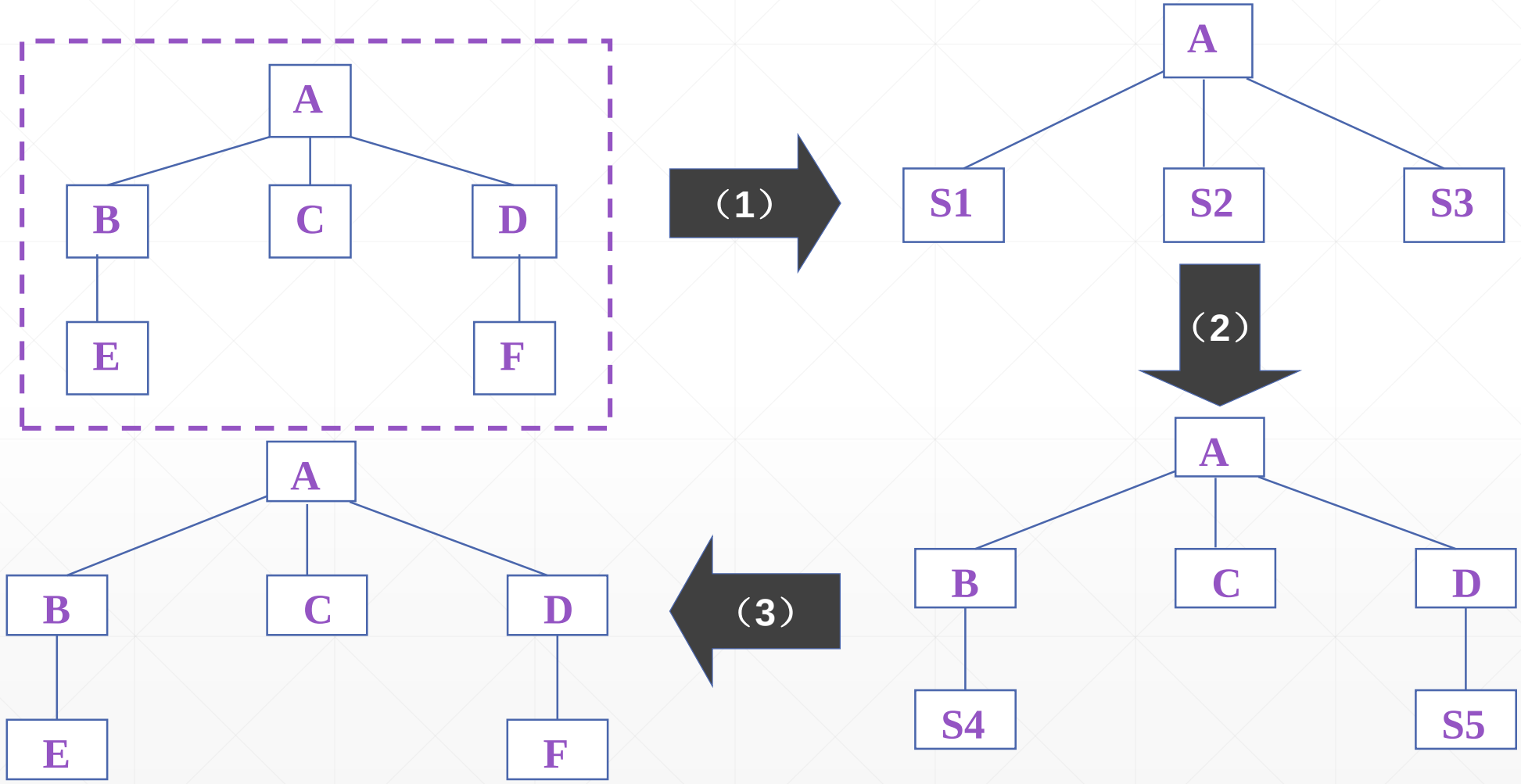
桩模块的开发量较大

适用

控制结构清晰稳定；
高层接口变化较小；
底层接口未定义或经常可能被修改；
接口控制组件具有较大的技术风险，需要尽早被验证；
希望尽早能看到产品的系统功能行为。

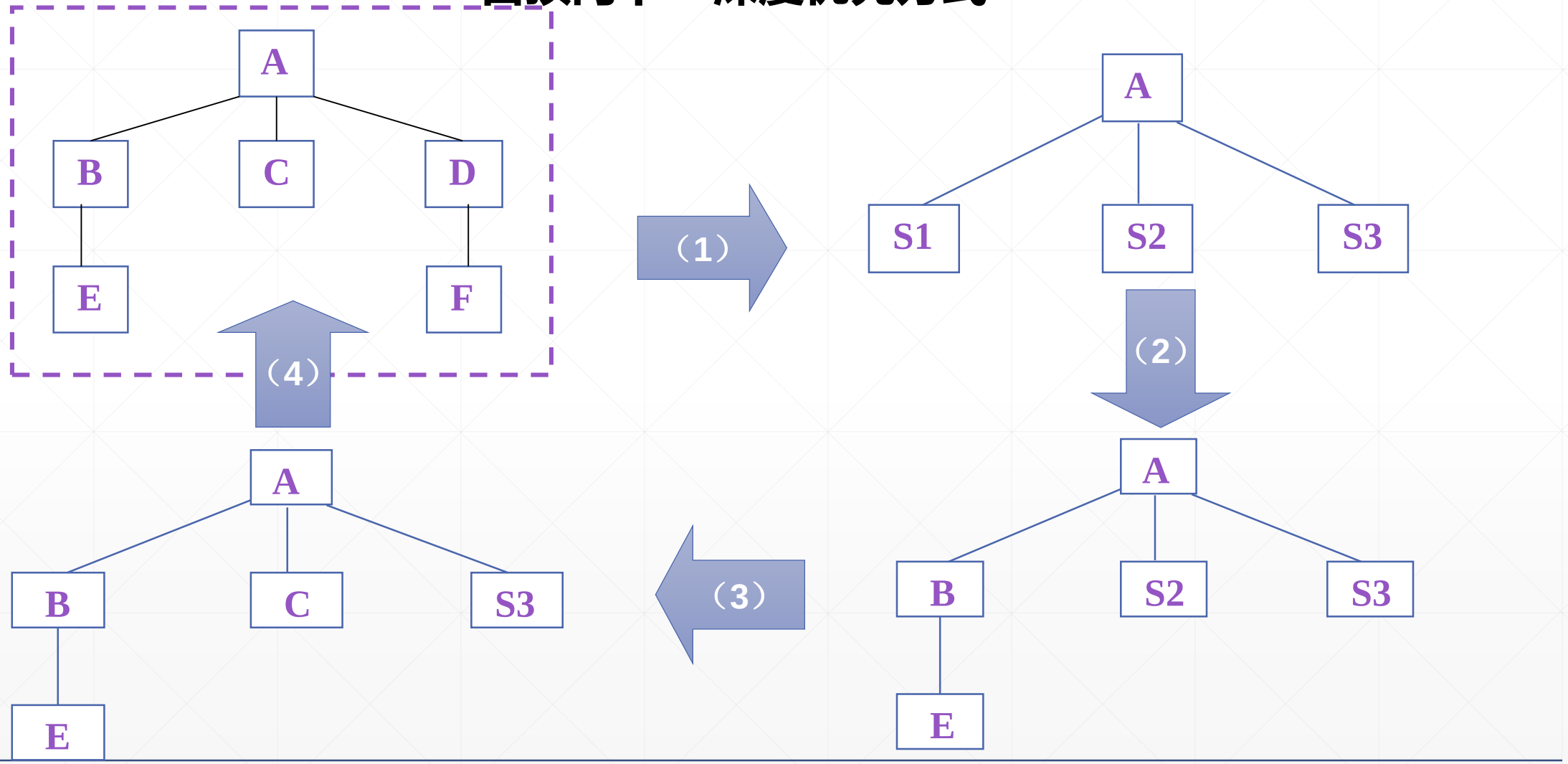
2）自顶向下的集成方法

自顶向下—广度优先方式



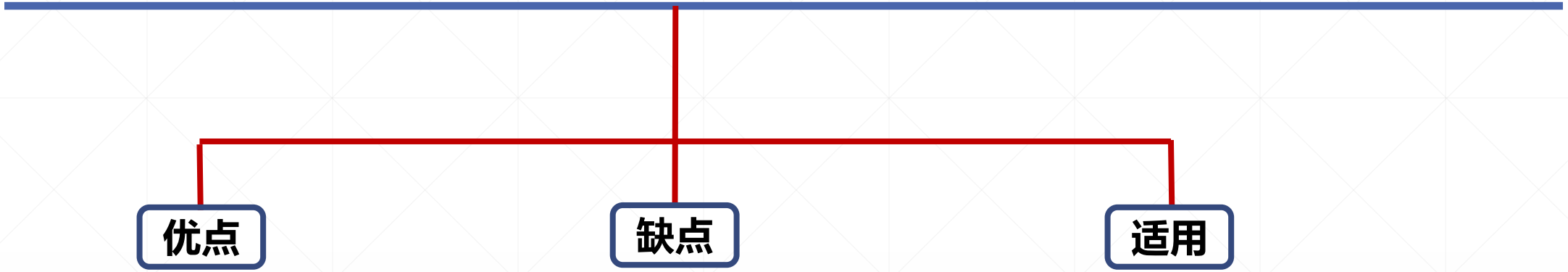
3）自顶向下的集成方法

自顶向下—深度优先方式



4）自底向上的集成方法

基本思想：从软件结构最底层的模块开始，按照接口依赖关系逐层向上集成以进行测试。

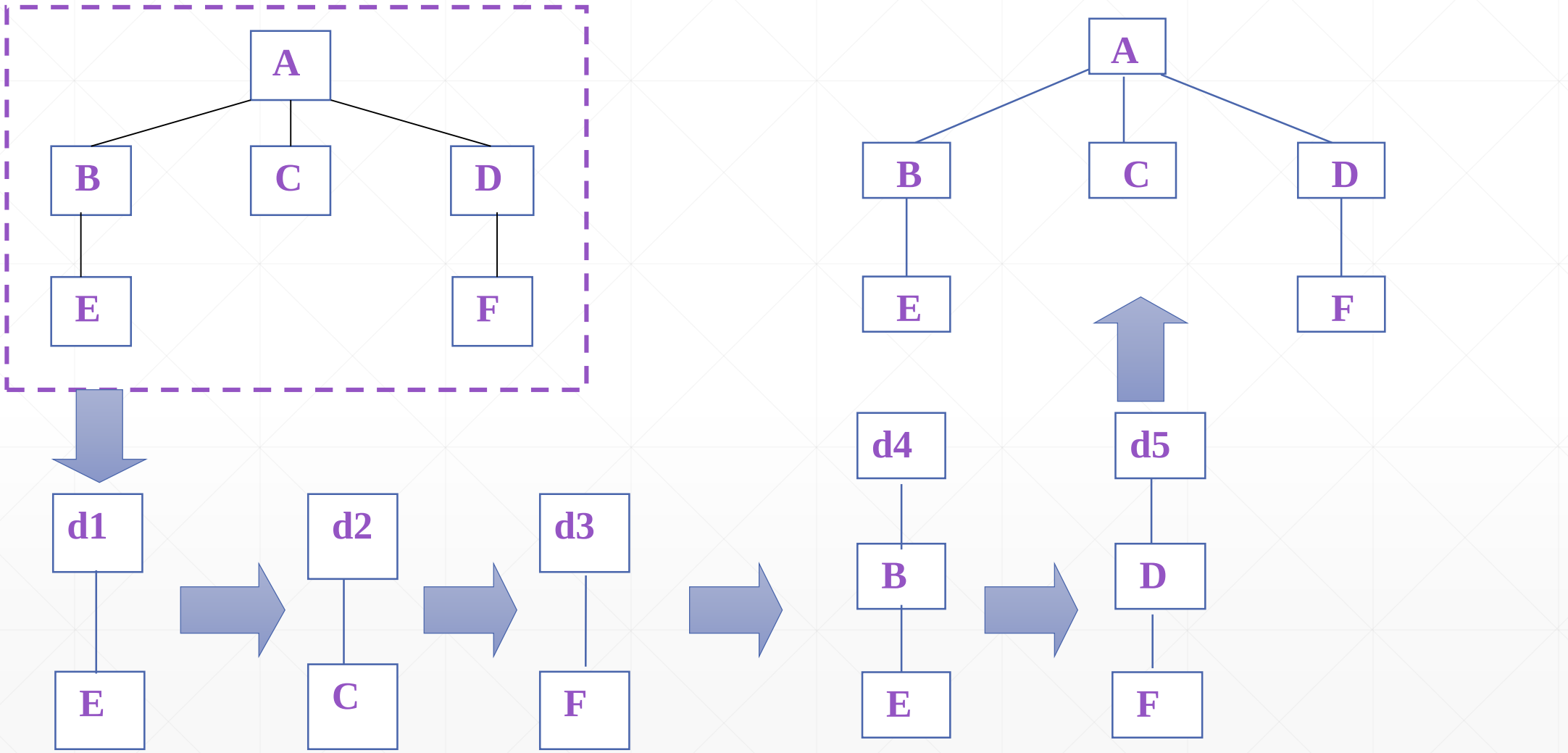


每个模块调用其他底层模块都已经测试，不需要桩模块；

必须编写驱动模块；顶层主控模块在最后才能得到测试。

底层接口比较稳定；
高层接口变化比较频繁；
底层组件较早被完成。

4）自底向上的集成方法



5）应用注意事项

实际工作中，常综合使用：自底向上、自顶向下

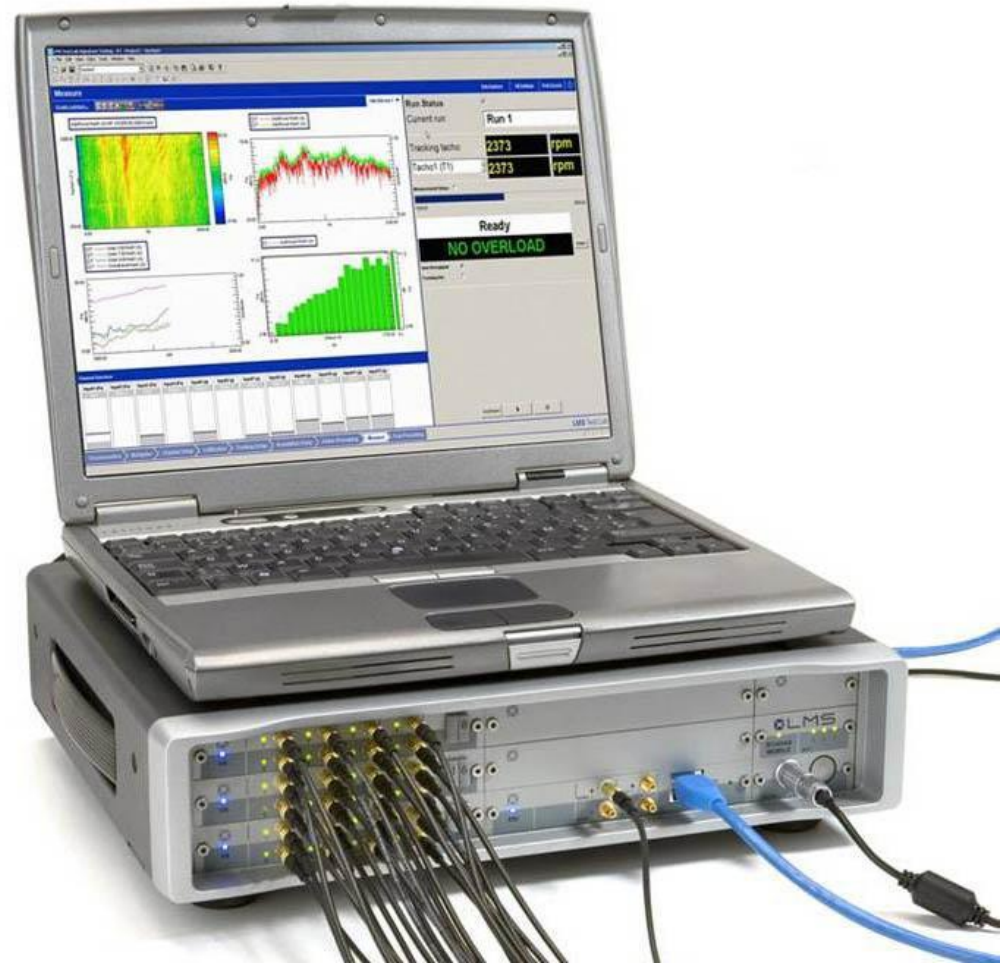
如：按进度选择优先测试已经完成的模块

If：被测模块所调用的模块未完成，用自顶向下，打桩测试。

If：被测模块的上层模块未完成，用自底向上，模拟根模块。

6）集成中的SMOKE方法

基本思想	将已经转换为代码的软件构件集成为构造（build）。一个构造包括所有的数据文件、库、可复用的模块以及实现一个或多个产品功能所需的工程化构件。
目　　标	设计影响构造正确地完成其功能的错误的测试。以发现极有可能造成项目延迟的业务阻塞错误。
方　　法	每天将该构造与其他构造，及整个软件产品集成，冒烟测试。 两种方法：自顶向下，自底向上，均可。



6.3.4.软件测试策略——系统测试

- 系统测试概念
- 系统测试必要性
- 系统测试内容

1）系统测试概念


含义
从用户使用的角度进行测试，将完成了集成测试的系统放在真实的运行环境下进行。
目的
功能确认和验证
测试方法
黑盒测试

2）系统测试必要性

系统测试：软件开发过程必不可少的一环，软件质量保证的最重要环节。

测试内容方面

面向：外部输入层测试，如不做，则外部输入层向接口层转换的代码就没有得到测试。
面向：系统所有组件相互协调。

测试角度方面

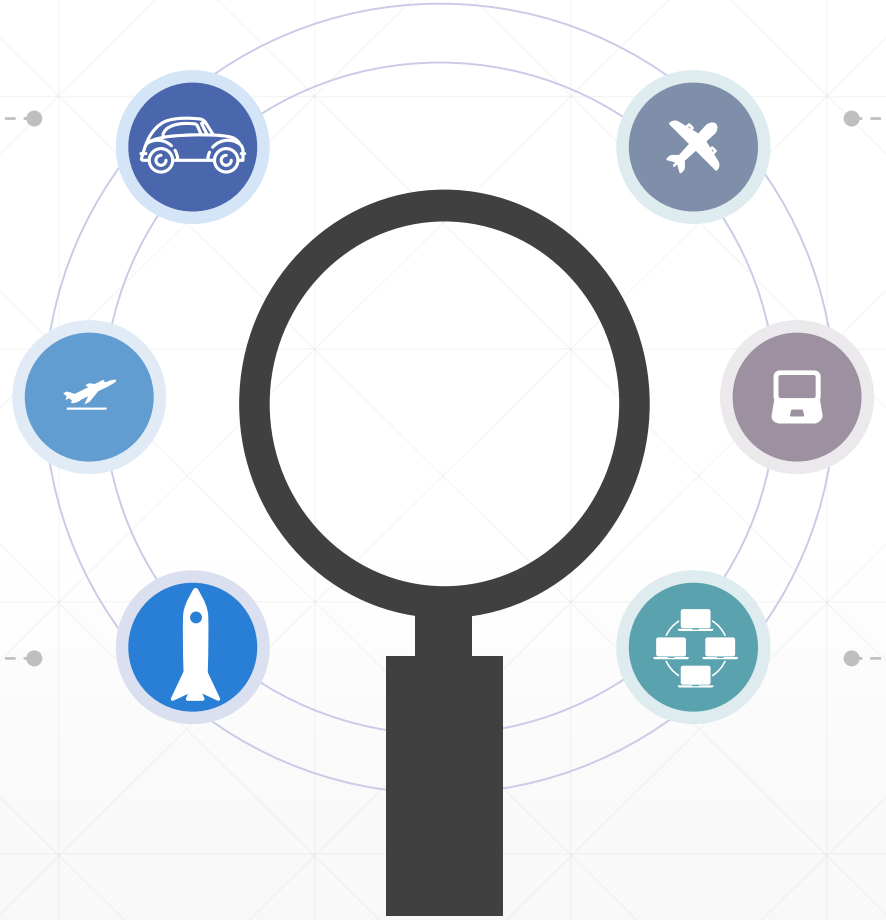
系统测试依据：需求规格说明
单元、集成测试依据：技术规格说明

3）系统测试内容

压力测试

性能测试

功能测试



恢复测试

安全测试

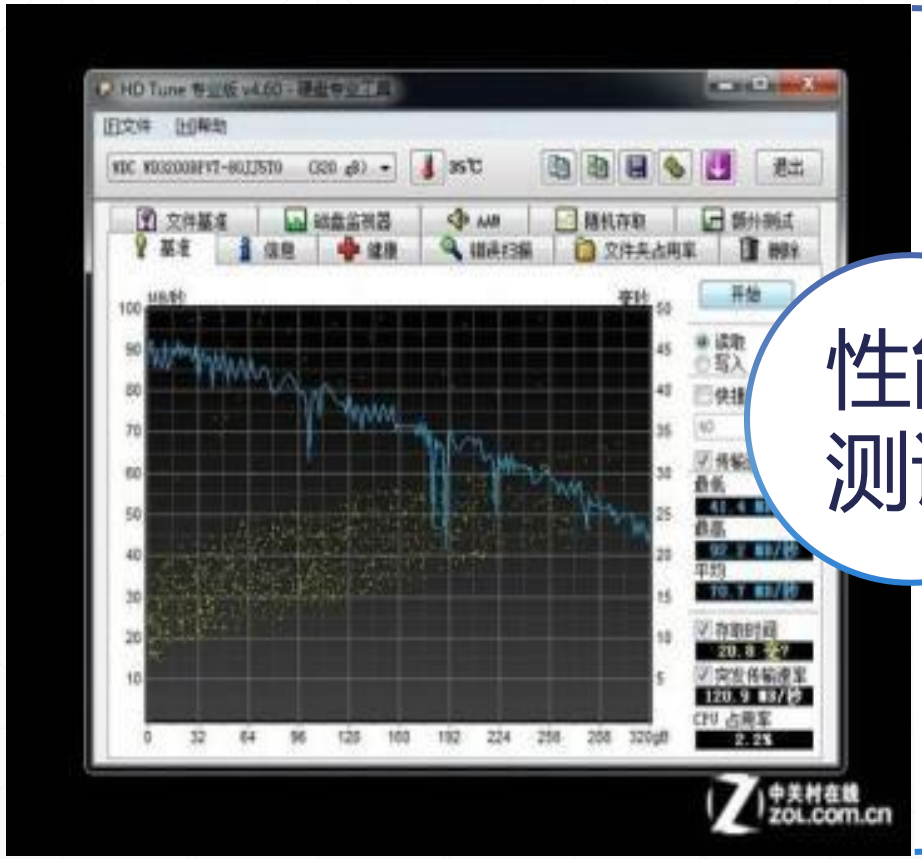
配置测试
兼容性测试
本地化测试
文档测试
易用性测试等

3-1) 系统测试内容——功能测试

功能测试

含　义	测试方法	错误类型
• 在规定的一段时间内运行软件系统的所有功能，以验证这个软件系统有无严重错误。	• 黑盒测试	• 功能错误或遗漏 • 界面错误 • 数据结构或外部数据库访问错误 • 性能错误 • 初始化

3-2) 系统测试内容——性能测试



性能测试

- 含义
  - 检查系统是否满足在需求说明书中规定的性能。
- 结合
  - 常常要与压力测试结合进行，硬件和软件检测同时进行。
- 内容
  - 响应时间
  - 吞吐量
  - 辅助存储区，如缓冲区、工作区的大小、处理精度等。

3-3) 系统测试内容——压力测试

压力测试：在系统运行环境不正常乃至发生故障的情况下，系统可以运行到何种程度的测试

测试方法	把输入数据速率提高一个数量级，确定输入功能将如何响应
	设计需要占用最大存储量或其它资源的测试用例进行测试。
	设计出在虚拟存储管理机制中引起“颠簸”的测试用例进行测试。
	设计出会对磁盘常驻内存的数据过度访问的测试用例进行测试。

敏感性测试：压力测试的一个变种。在程序有效数据界限内一个小范围内的一组数据可能引起极端的或不平稳的错误处理出现，或者导致极度的性能下降。

3-4) 系统测试内容——恢复测试

恢复测试：是要证实在克服硬件故障后，系统能否正常地继续进行工作，并不对系统造成任何损害。

手　　段：人工干预等

测试方法	错误探测功能——系统能否发现硬件失效与故障；
	设备故障时，能否切换或启动备用的硬件；
	故障发生时，能否保护正在运行的作业和系统状态；
	系统恢复后，能否从最后记录下来的无错误状态开始继续执行作业等。
	掉电测试：电源中断时，能否保护当时的状态且不毁坏数据，然后在电源恢复时从保留的断点处重新进行操作。

3-5) 系统测试内容——安全性测试

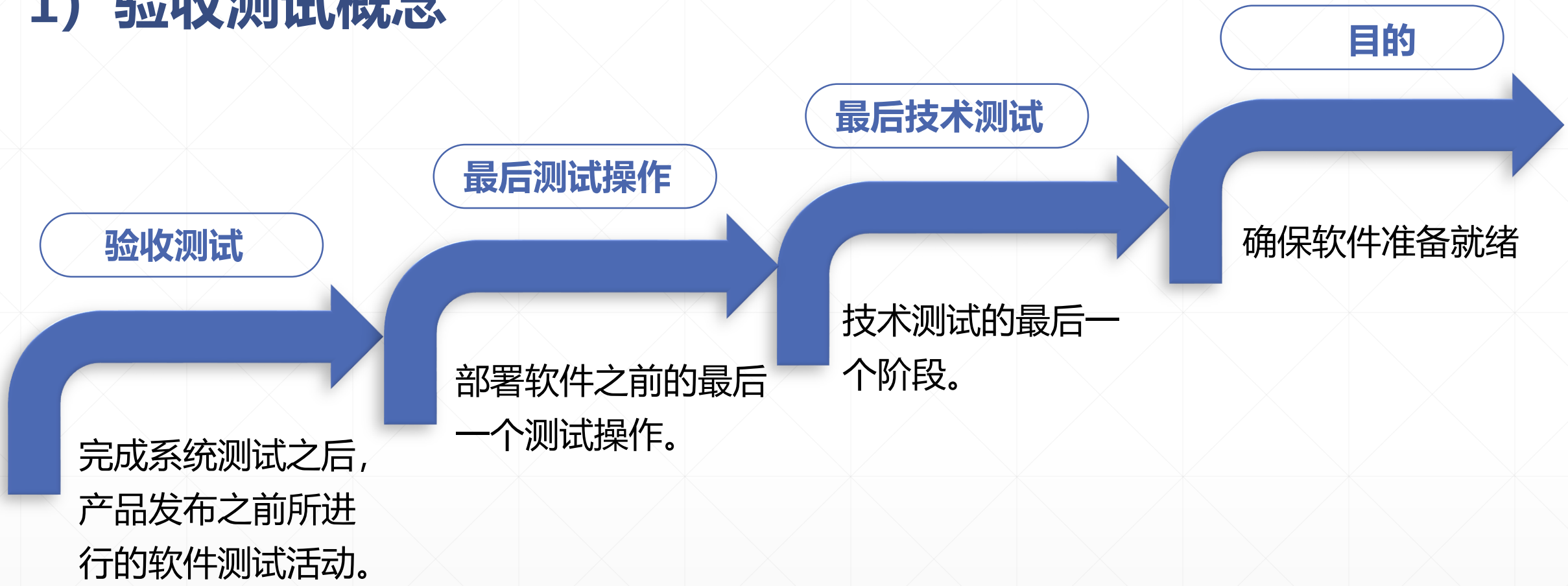
安全性测试：是要检验在系统中已经存在的系统安全性、保密性措施是否发挥作用，有无漏洞。

主要攻击方法	正面、侧面或背面攻击系统中易受损坏的那些部分；
	以系统输入为突破口，利用输入的容错性进行正面攻击；
	申请和占用过多的资源压垮系统，以破坏安全措施，从而进入系统；
	故意使系统出错，利用系统恢复的过程，窃取用户口令及其它有用的信息；
	通过浏览残留在计算机各种资源中的垃圾（无用信息），以获取如口令，安全码，译码关键字等信息；
	浏览全局数据，期望从中找到进入系统的关键字；
	浏览逻辑上不存在，但物理上还存在的各种记录和资料等

6.3.5.软件测试策略——验收测试

- 验收测试概念
- 验收测试过程
- 验收测试形式
- 测试停止标准
- 软件测试的组织

1）验收测试概念


验收测试
完成系统测试之后，产品发布之前所进行的软件测试活动。
最后测试操作
部署软件之前的最后一个测试操作。
最后技术测试
技术测试的最后一个阶段。
目的
确保软件准备就绪

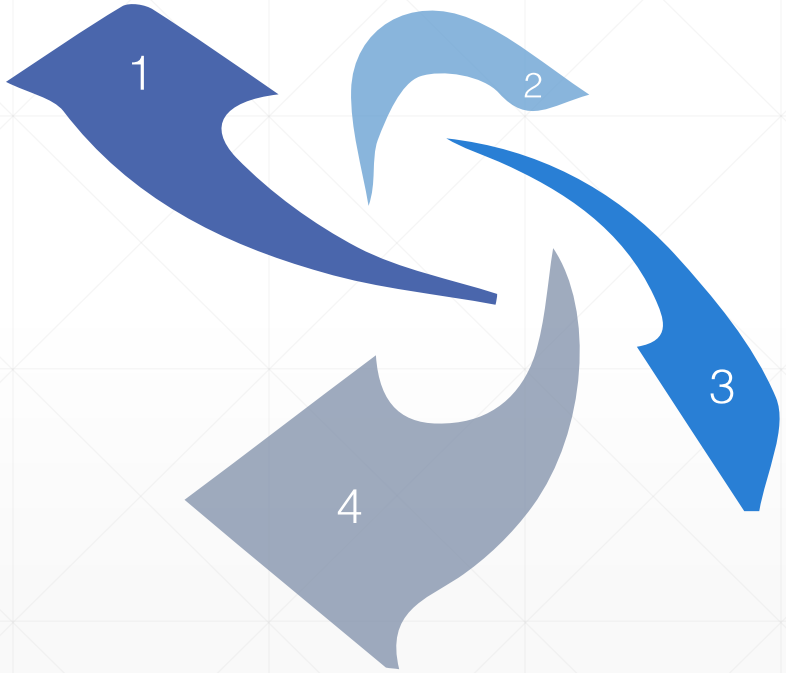
2）验收测试概念

时间节点

系统的有效性测试及软件配置审查通过之后。

测试内容

功能和性能外
可移植性、兼容性、可维护性、错误的恢复功能等



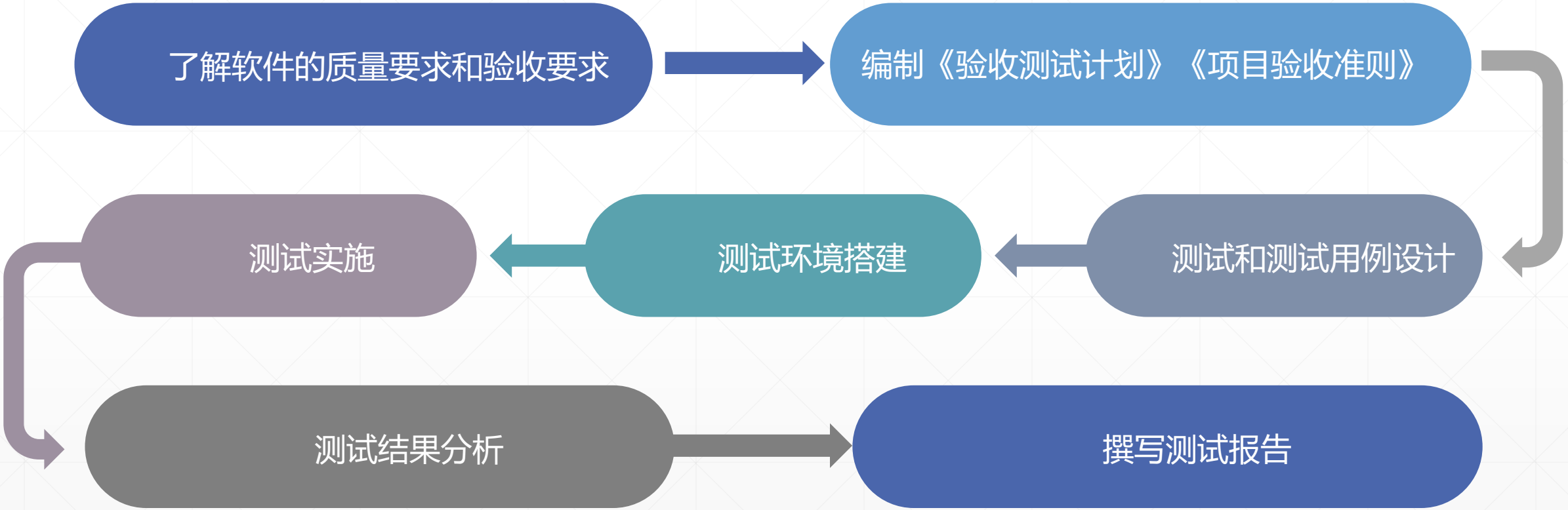
人员组织

以用户为主
开发人员
质量保证人员

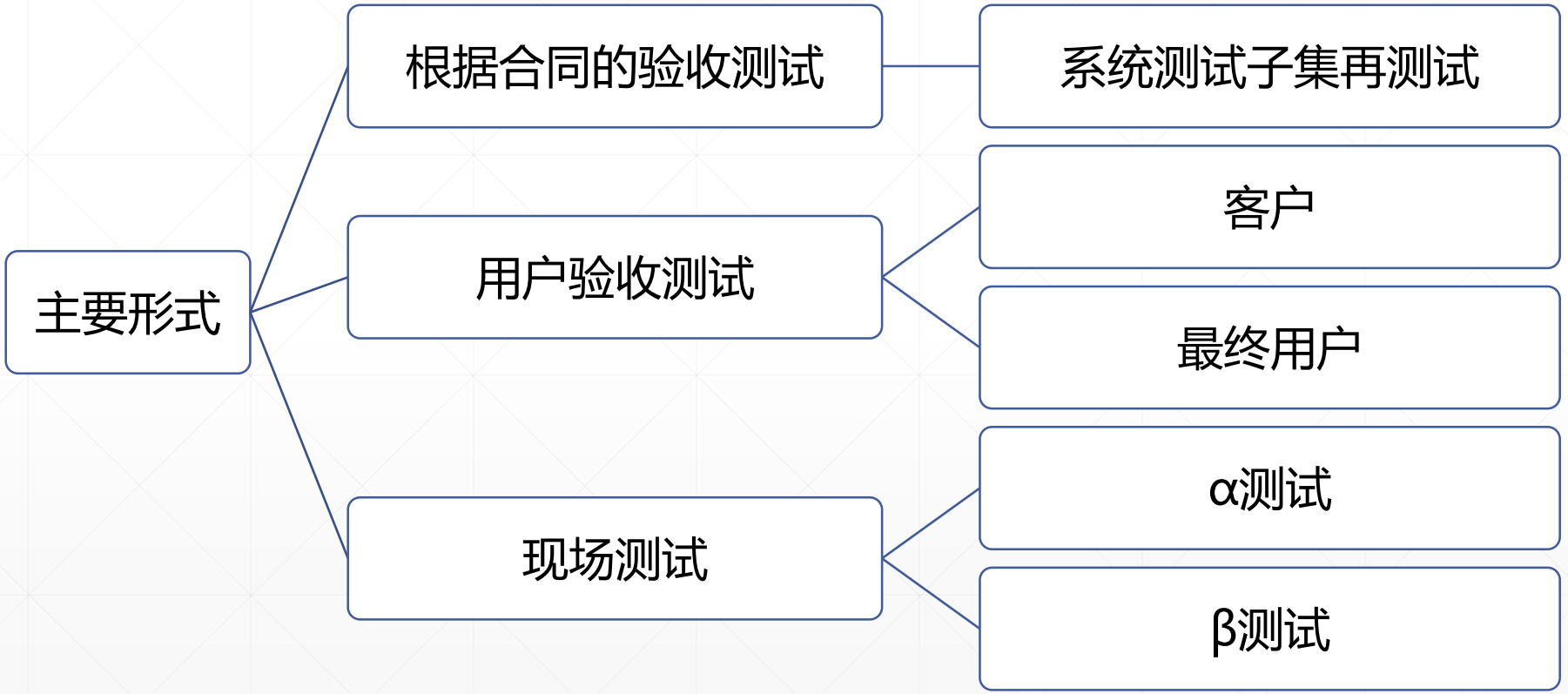
测试数据

实际生产数据

3）验收测试过程


了解软件的质量要求和验收要求
编制《验收测试计划》《项目验收准则》
测试和测试用例设计
测试环境搭建
测试实施
测试结果分析
撰写测试报告

4）验收测试形式



5）验收测试形式

含　义

用户在开发环境、模拟用户在模拟实际操作环境下的测试

原　因

交付后，用户的实际使用程序是无法预测的

目　的

评价FLURPS特性（功能、本地化、可使用性、可靠性、性能和支持）。尤其界面和特色

开始时间

编码结束后

模块（子系统）测试完成后

系统测试过程中产品达到一定的稳定和可靠程度后

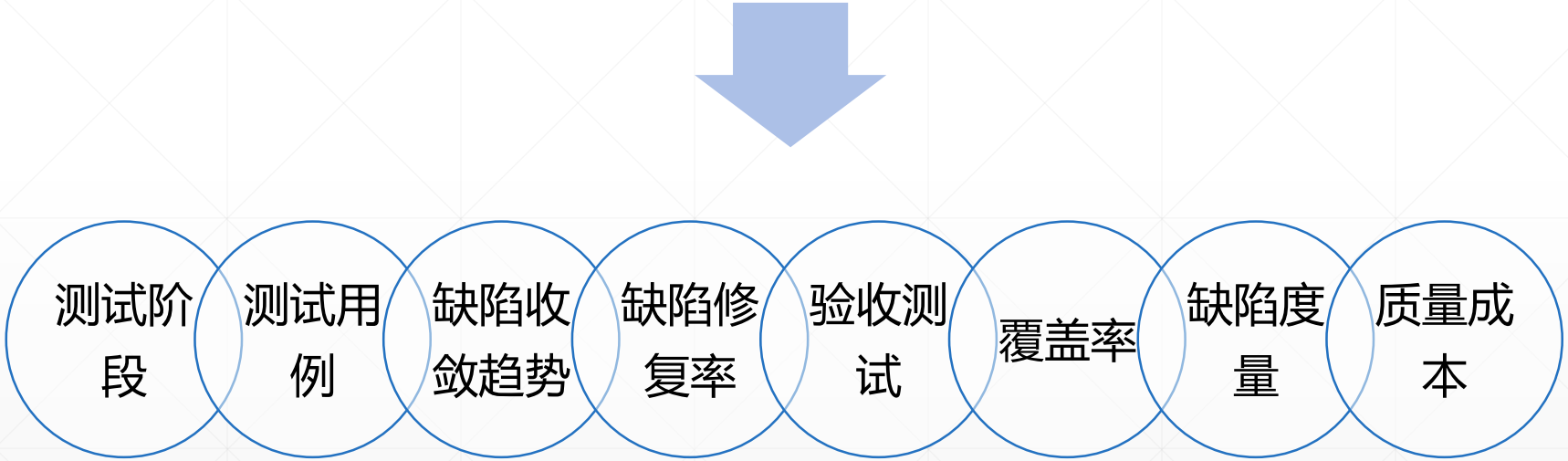
6）验收测试形式

β测试

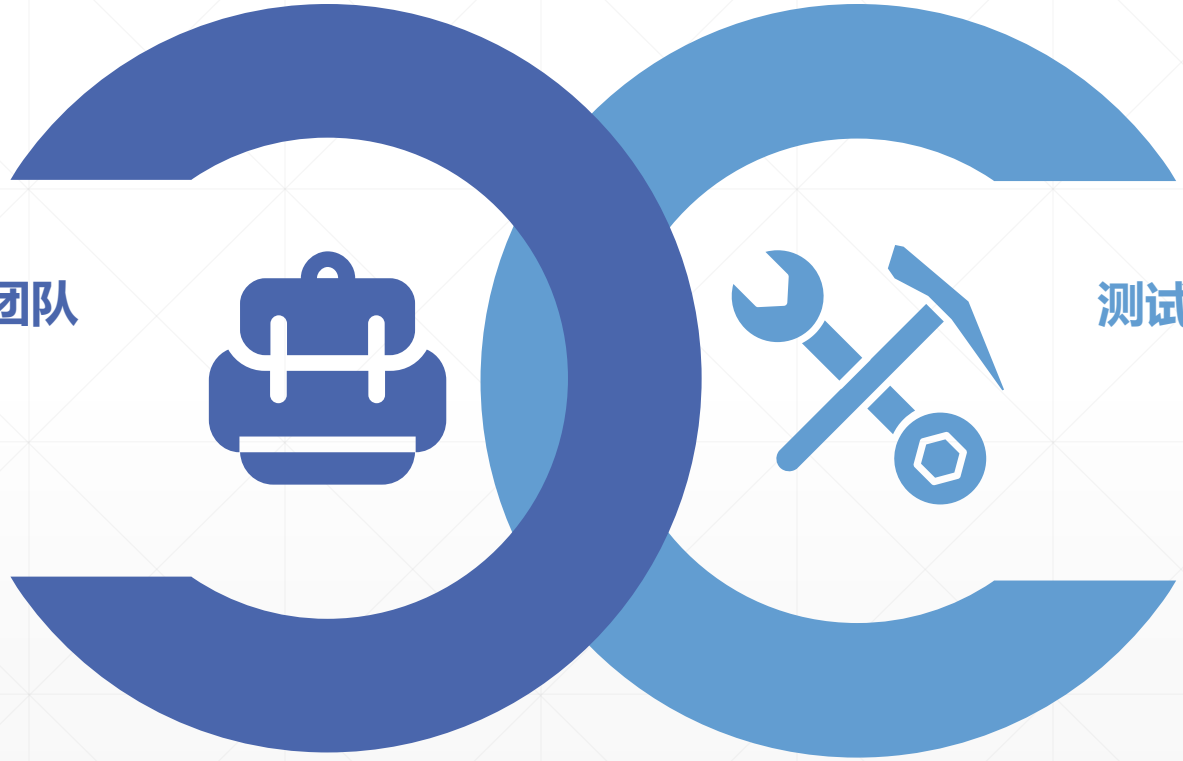
测试人员	• 多个用户在实际使用环境下进行测试。这些用户返回有关错误信息给开发者。
测试形式	• 开发者通常不在测试现场，开发者无法控制的环境下进行的软件现场应用。
测试步骤	• 用户记录所有问题（真实的、主观的），定期向开发者报告。
测试目标	• 产品的FLURPS。着重产品的支持性（文档、客户培训和支持产品生产能力）
开始条件	• α测试达到一定的可靠程度时开始，测试的最后阶段，所有手册文本此阶段完全定稿。

7）测试停止标准

软件是无法完全测试的 → 何时停止测试



8）测试组织



测试团队

开发团队

开发团队中配备测试人员

项目团队中若干测试团队

独立测试专家

单独测试部门

测试人员

测试经理

测试设计人员

测试自动化人员

测试管理员

测试人员