



西安电子科技大学  
XIDIAN UNIVERSITY

# 软件测试





# 内容

---

## 1. 软件测试概述

- ✓ 软件测试的思想和原理

## 2. 软件测试的过程和策略

- ✓ 软件测试的活动及实施的方法

## 3. 软件测试技术

- ✓ 白盒和黑盒测试技术

## 4. 软件测试计划及输出

- ✓ 测试计划制定及测试结果



# 示例：软件中存在缺陷，导致软件失效

## 高校课程实践社区

当前位置：主页 > 课程实践平台 > 软件工程



ID:342

配置

关闭

### 软件工程

教师 (2) | 学生 (21) | 资源 (2)

主讲教师：毛新军

学时总数：54 学时

课程学期：2015 秋季学期

开设单位：国防科学技术大学

### 动态 (15)

课程作业 (0)

+发布作业

课程通知 (0)

+发布通知

资源库 (2)

+上传文件

讨论区 (7)

+发布新帖

留言 (0)

问卷调查 (1)

+新建问卷

课内搜索

全站搜索

上传：课件 | 软件 | 媒体 | 代码 | 其他

共有 2 个资源

按 时间 / 下载次数 / 引用次数 排序

课件x2

2 软件与软件工程.pdf

选入我的其他课程

公开

预览

文件大小：4.475 MB

1天之前 | 下载2 | 引用0 | 删除

课件 x + 添加标签

1 课程介绍与要求.pdf

选入我的其他课程

公开

预览

文件大小：6.755 MB

6 天之前 | 下载10 | 引用0 | 删除

课件 x + 添加标签

✓ 上传资源后资源数量没有变化  
✓ 查找以前上传的资源找不到

# 示例：软件中存在缺陷，导致软件失效

 学习空间

搜索空间

意见反馈

+

31



 软件工程课程综合实践 (学生)

资源区 286

讨论区 25

贡献区 118

设置



软件工程课程综合实践 (学生)

围绕软件工程课程综合实践，针对综合实践中的需求分析、软件设计、系统建模、编码测试、软件维护等方面的内容，讨论问题，分享成果，交流经验，共享资源

37

28

资源区

新建资源

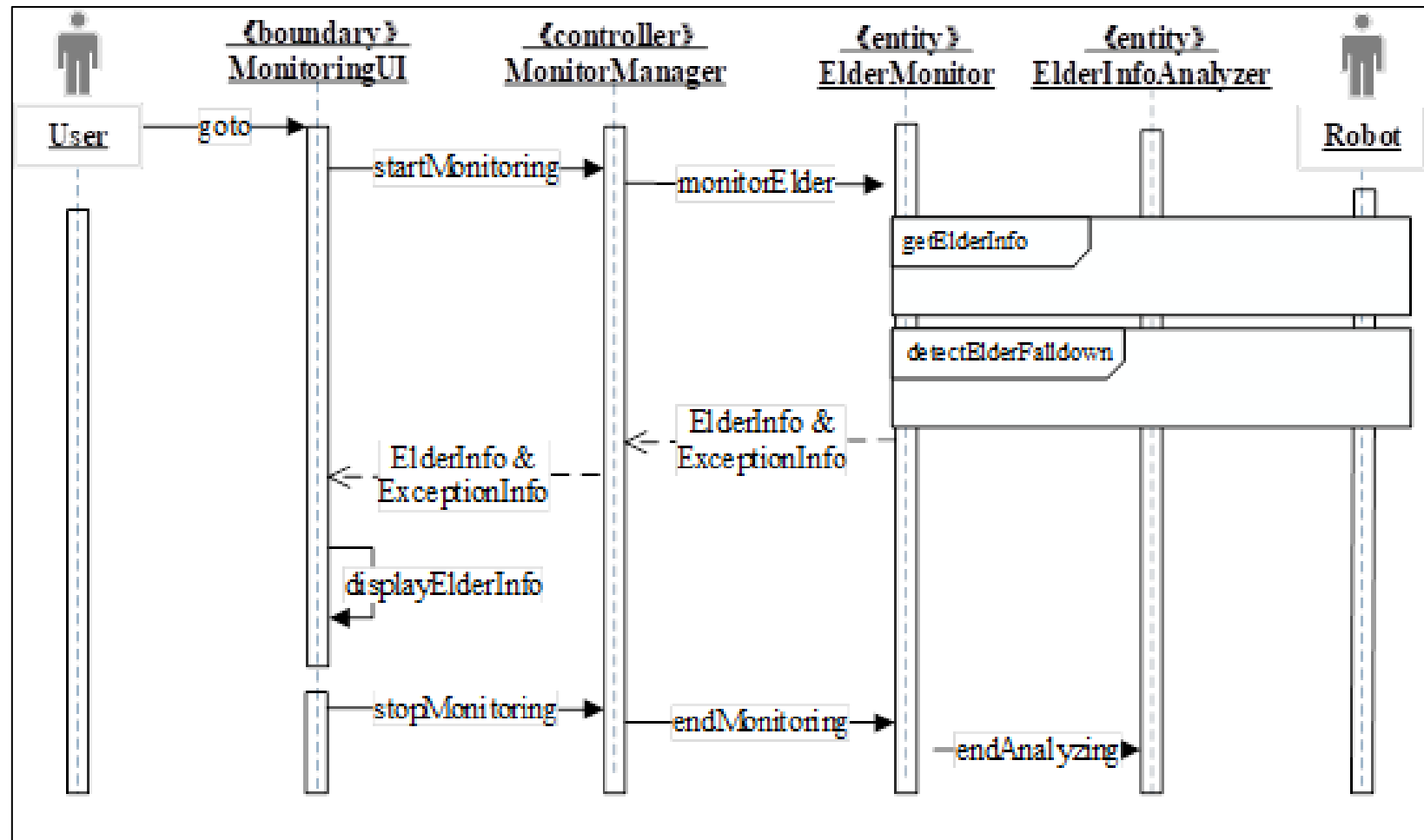
新建目录

搜索资源

资源名	作者	描述	更新时间
资源分享			1 天前
技术博客			7 天前

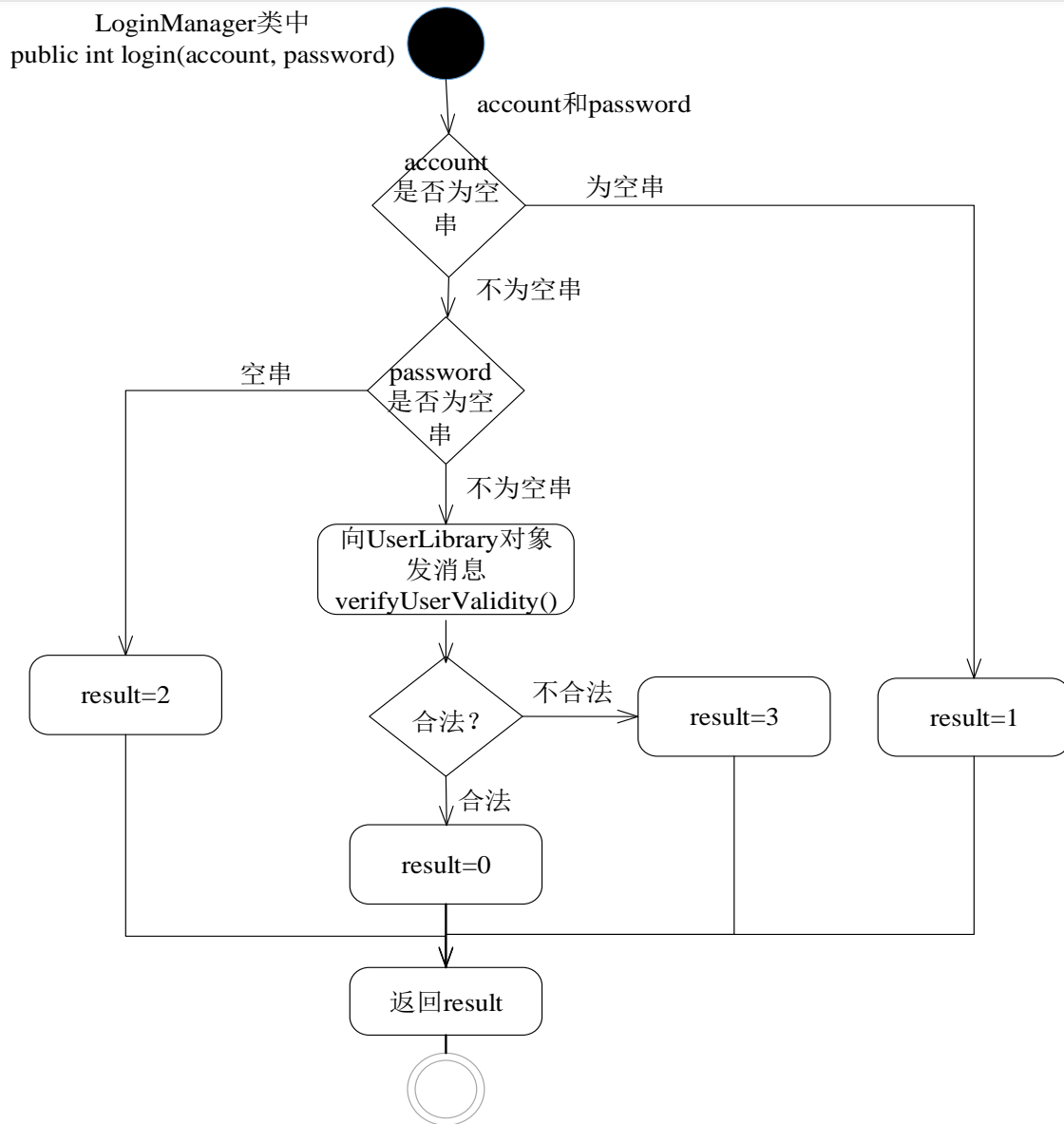
- ✓ 注册成功却无法登陆
- ✓ 增加资源但没有显示
- ✓ .....

# 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 x Contact.java x
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         }
47     }
48 }
```

- 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



# 1.1 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  
 $\pm 100$ ，重用于新战  
斗机，该参数屏显范  
围应该为 $\pm 300$ ！

尽可能减少软件缺陷非常重要



# 1.2 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？**



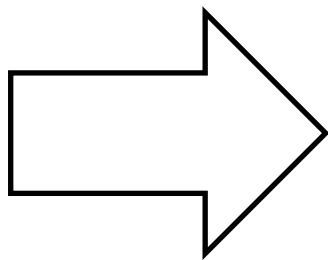
## □如何应对软件缺陷？

- ✓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
- .....



**软件测试**  
(Software Testing)



**方法之一：找到软件缺陷，以排除缺陷**

## 1.3 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

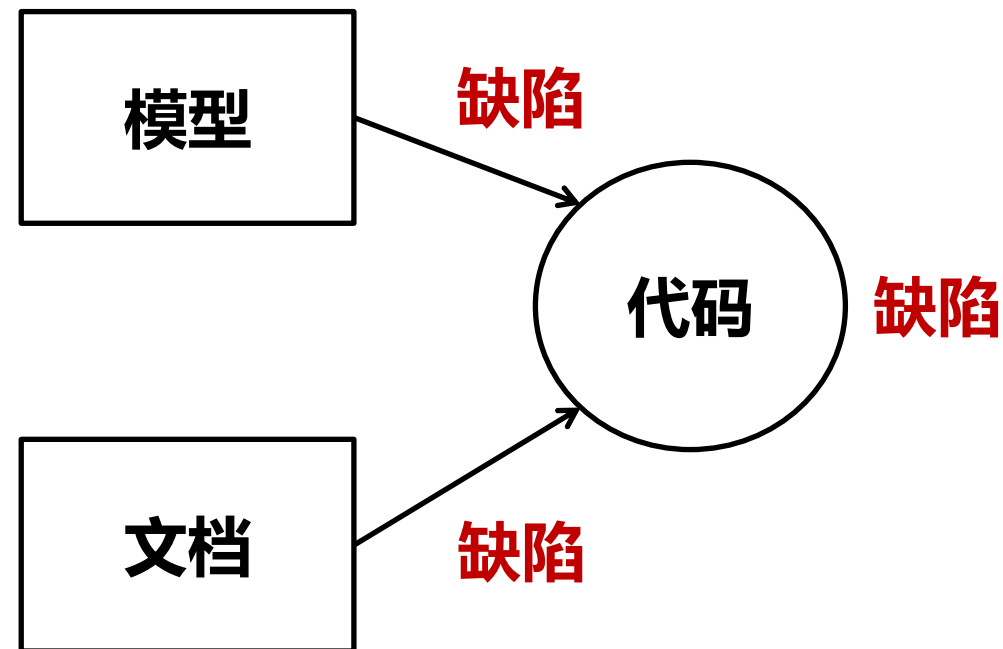
## □程序是运行软件的载体

- ✓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

## □通过测试发现代码中的缺陷，进而回溯到高层的软件模型和文档中



## □如何从程序代码中找出软件缺陷？

- ✓代码走查、静态分析
- ✓软件测试

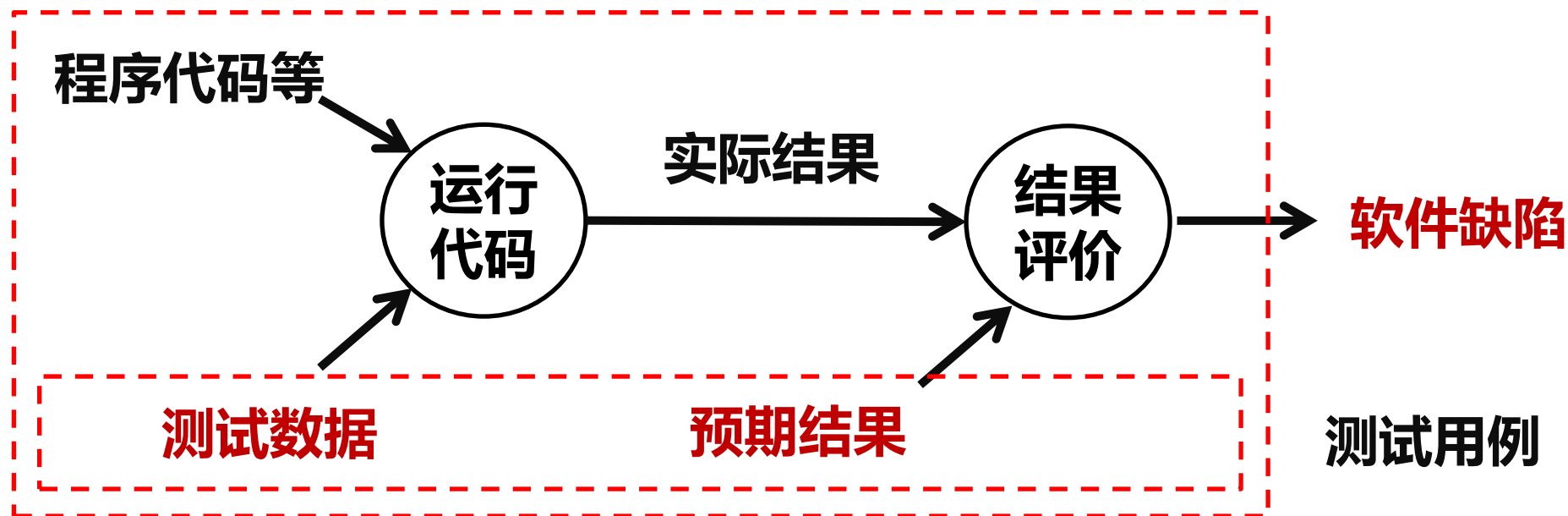




# 1.4 软件测试的原理

## □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

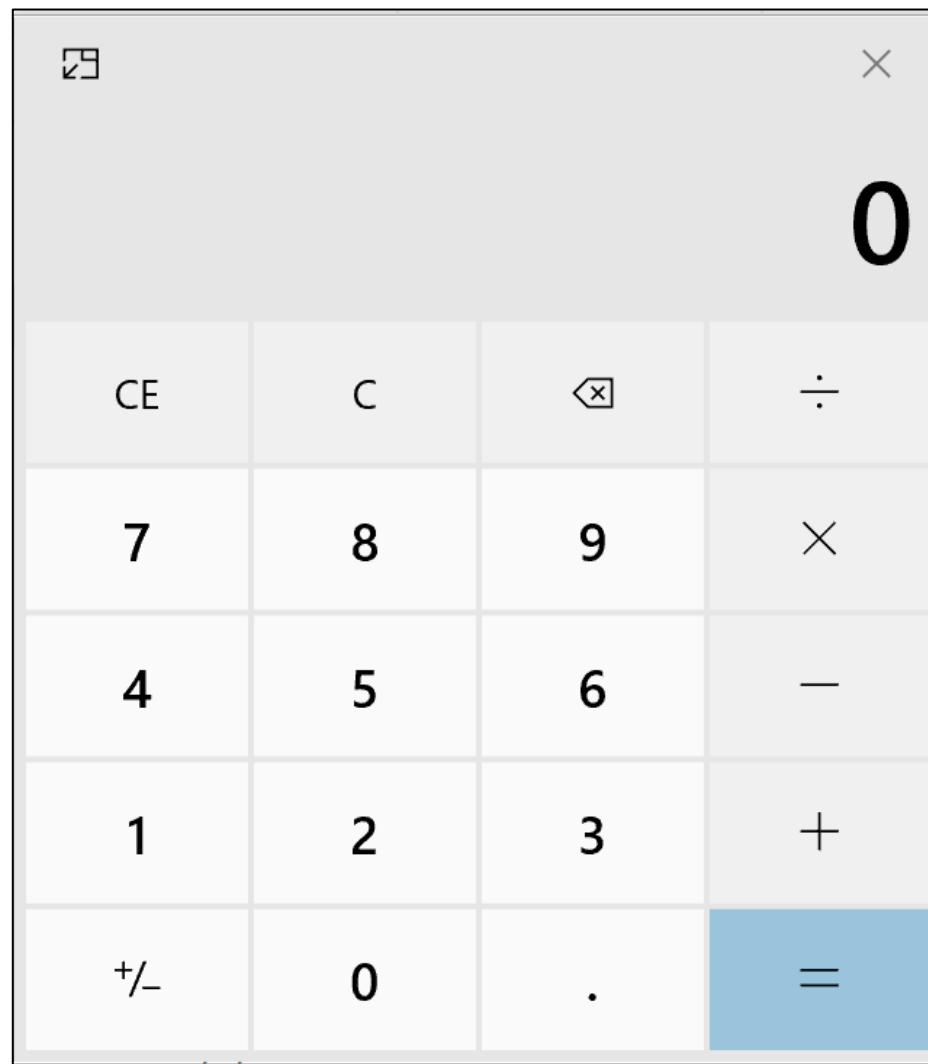
□ 加法的功能

□  $A = B + C$

□ 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

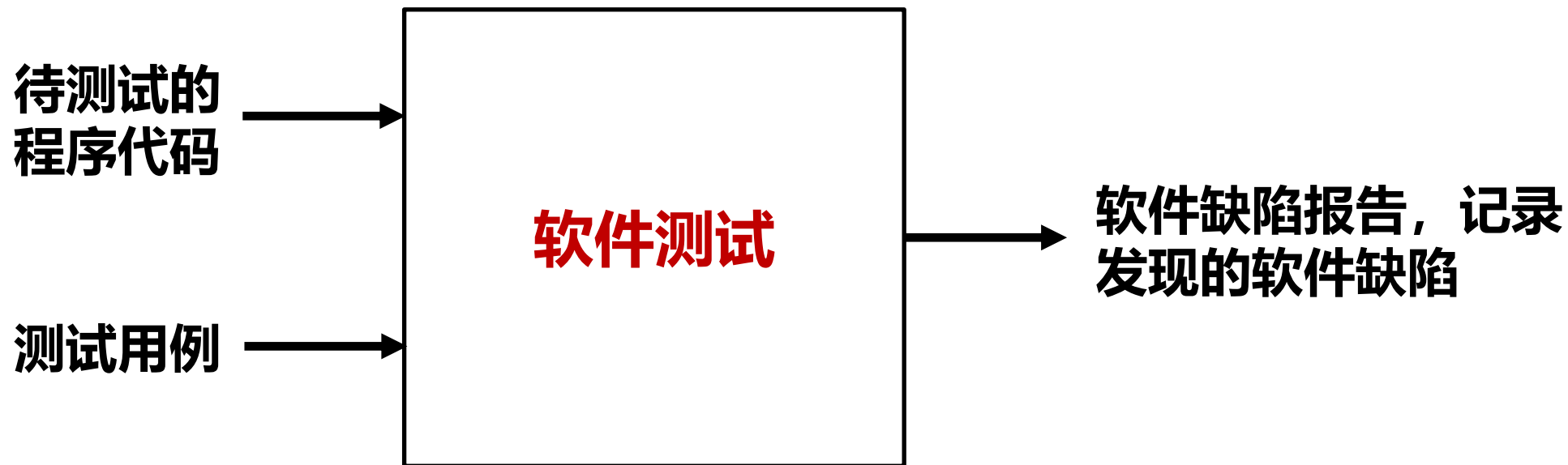
# 示例：测试用例的设计

---

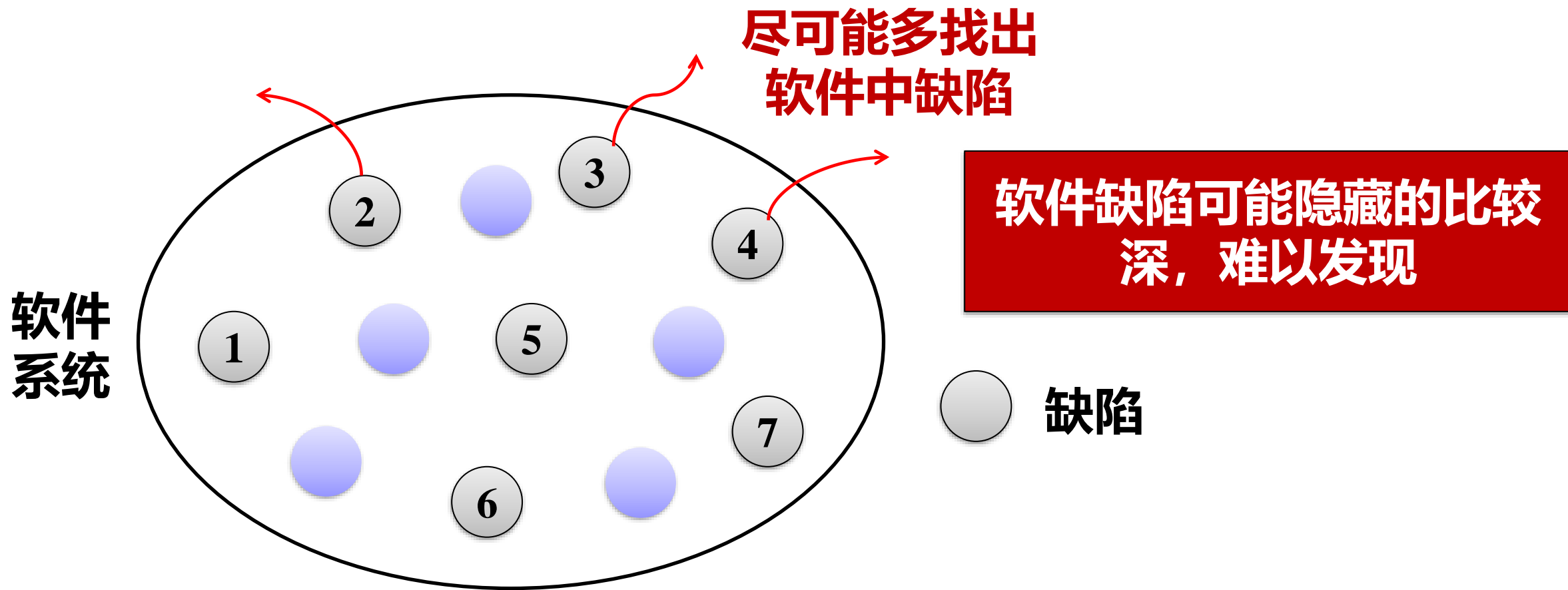
## □ “用户登录” 模块单元的测试用例设计

- ✓ **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

# 1.5 软件测试的任务



# 软件测试任务



采用这种方法能把软件中的所有缺陷都找出来吗？





## □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？



# 1.6 软件测试的步骤

## ① 明确待测试对象

- ✓ 什么粒度的程序代码：过程、函数、类、整个软件系统，需要知道待测试对象意图实现的功能或者内部逻辑

## ② 设计测试用例

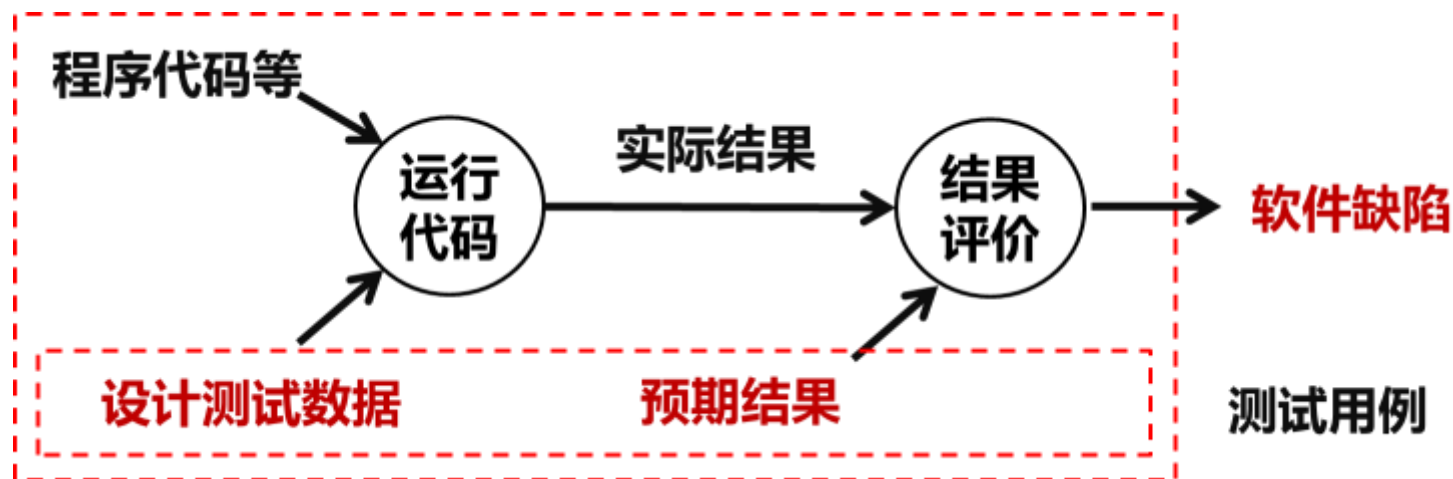
- ✓  $\{ \langle \text{Data}, \text{Result} \rangle \}$
- ✓ 可能有许多

## 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

# 1.7 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**

## 1.8 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





# 内容

## 1. 软件测试概述

✓ 软件测试的思想和原理

## 2. 软件测试的过程和策略

✓ 软件测试的活动及实施的方法

## 3. 软件测试技术

✓ 白盒和黑盒测试技术

## 4. 软件测试计划及输出

✓ 测试计划制定及测试结果





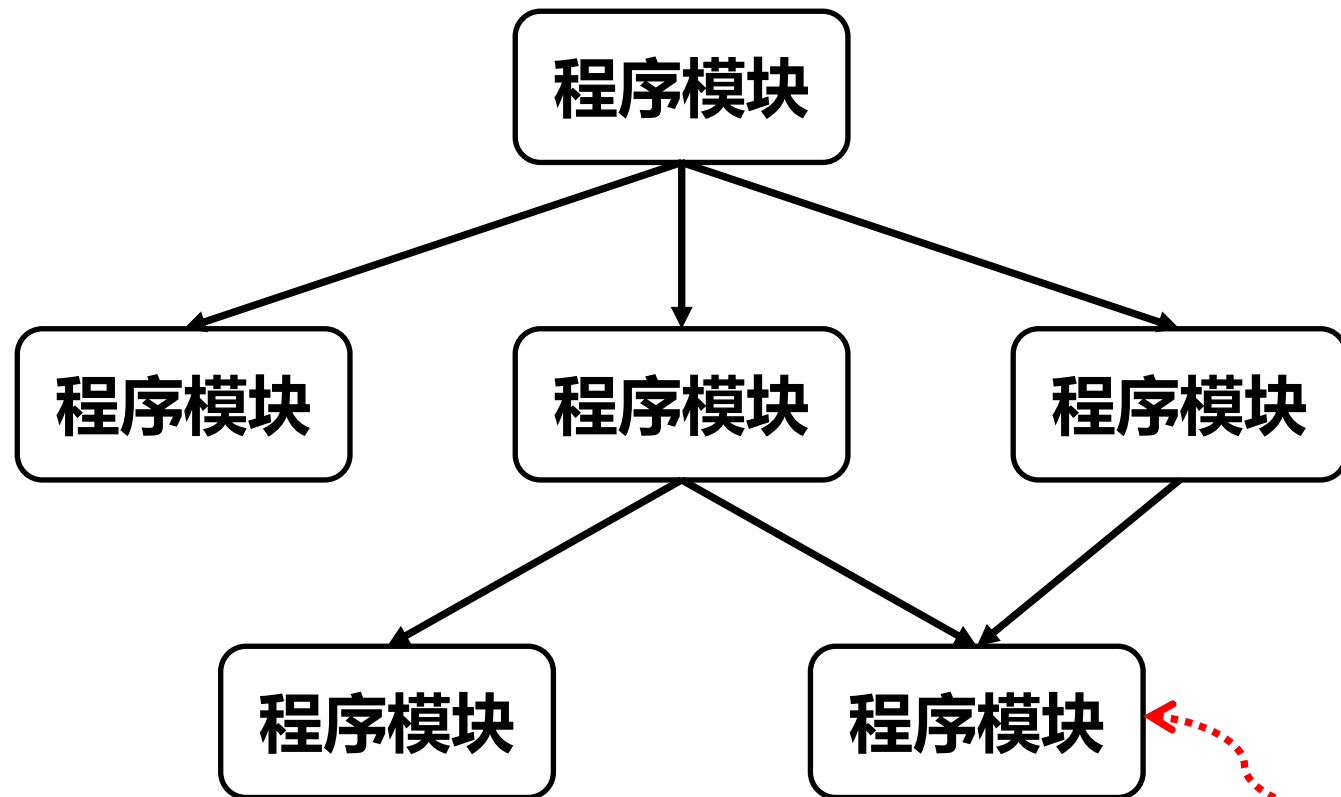
# 思考和讨论

---

□ 软件缺陷会“潜伏”在程序代码的哪些地方？



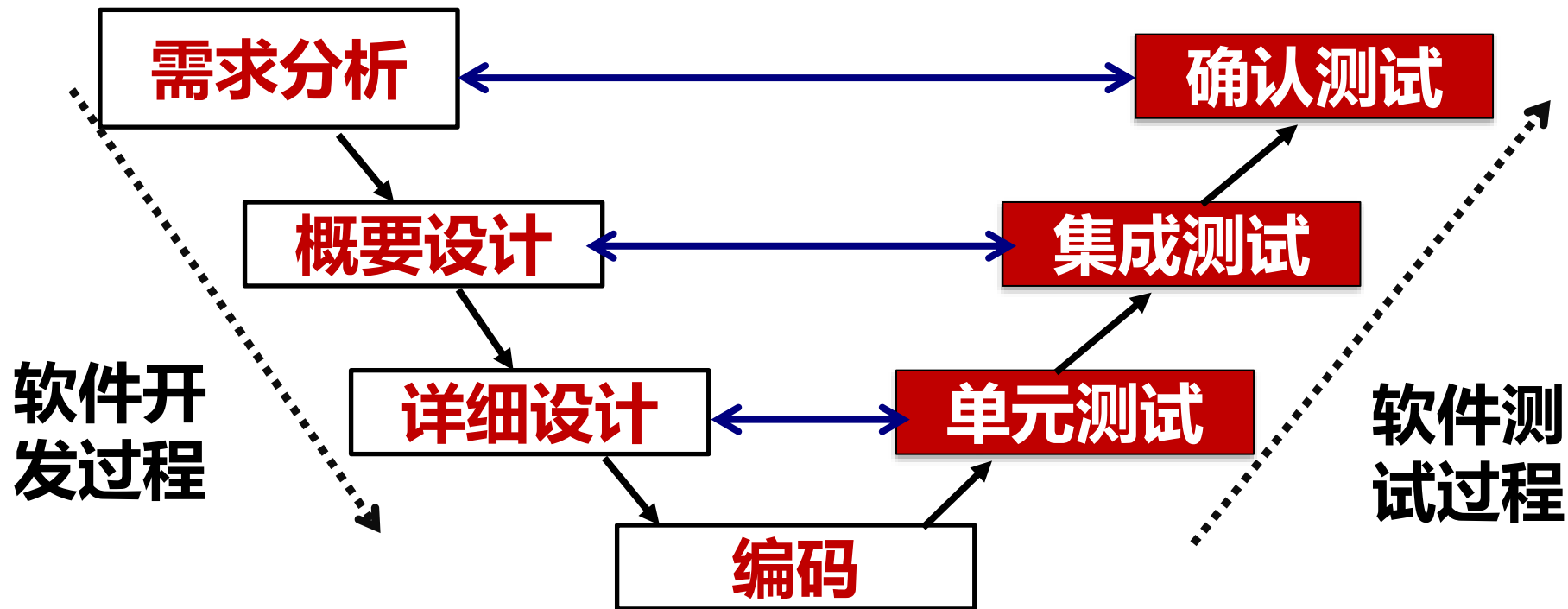
## 2.1 程序构成及其缺陷的潜在位置



- 程序模块内部
- 程序模块接口
- 程序模块间的交互
- 整个软件系统

发现和消除基本  
模块单元中缺陷

## 2.2 软件测试活动



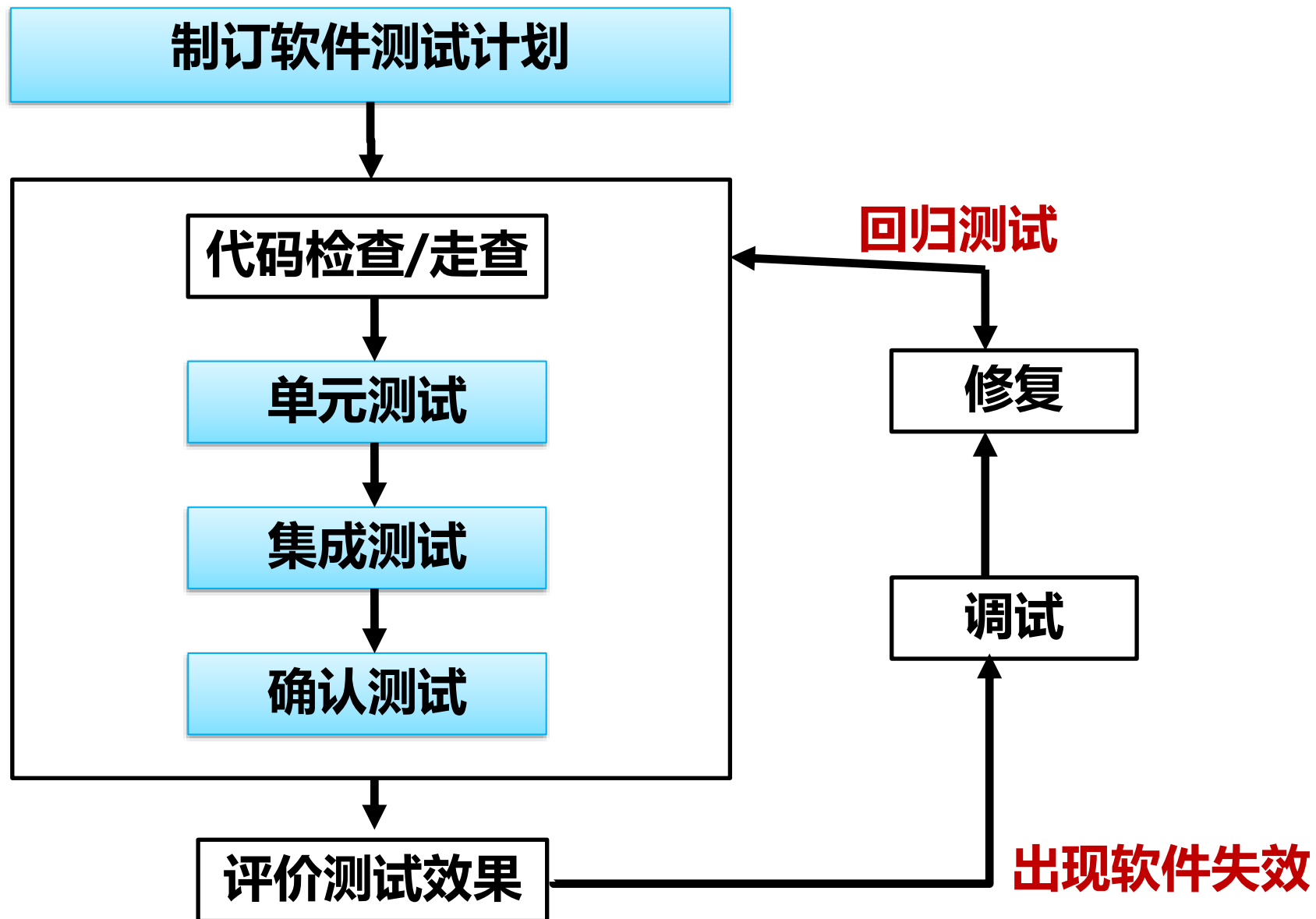
要对软件进行系统的测试

## 2.3 软件测试活动之间的关系

循序渐进、逐层递进地开展软件测试



## 2.4 软件测试过程





## 2.4.1 单元测试

### □测试对象

- ✓对软件基本模块单元进行测试
- ✓过程、函数、方法、类

### □测试方法

- ✓大多采用白盒测试技术

### □测试的内容

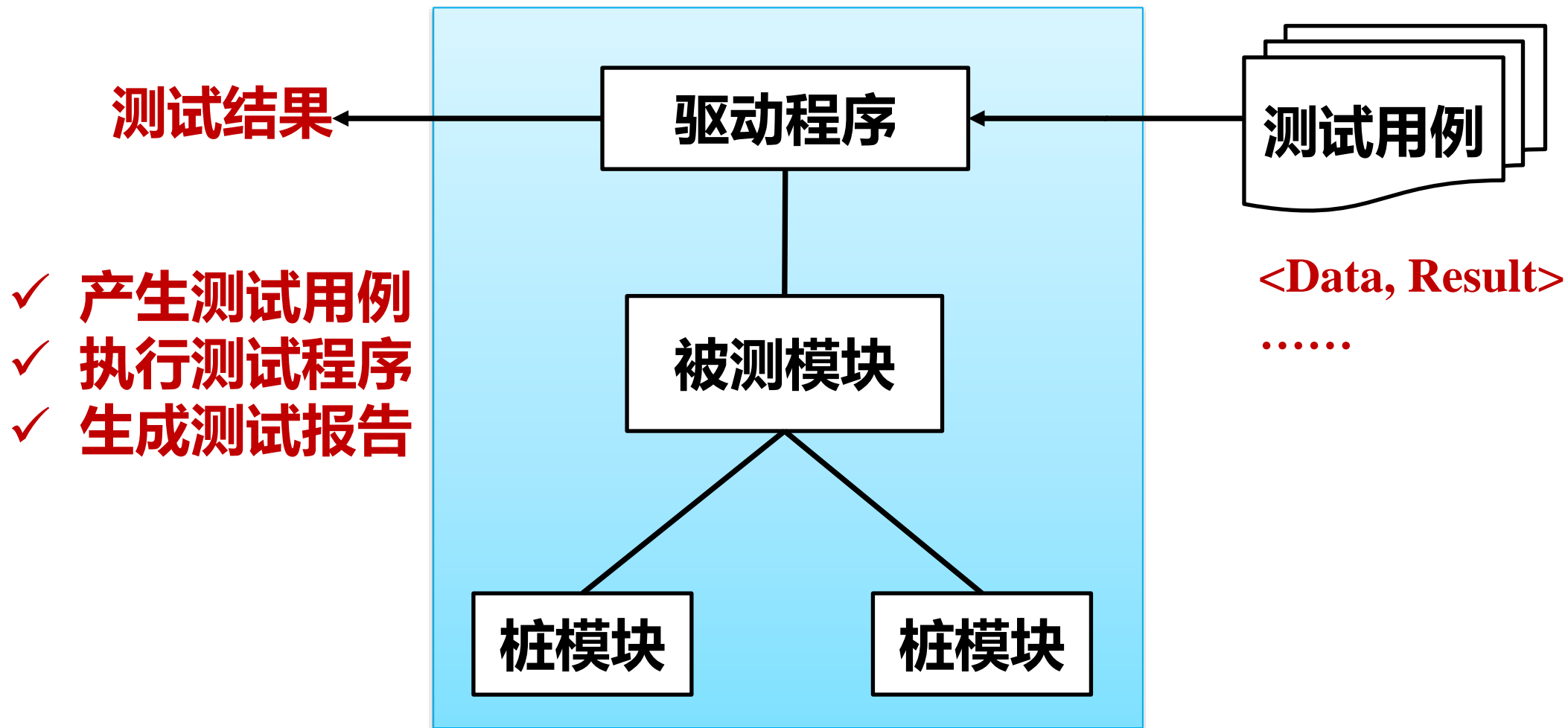
- ✓模块接口测试
- ✓模块局部数据结构测试
- ✓模块独立执行路径测试
- ✓模块错误处理通道测试
- ✓模块边界条件测试



在详细设计阶段就可以设计单元测试用例及计划

# 单元测试的运行环境

## 代码运行支撑



## 2.4.2 集成测试

### □测试对象

- ✓对软件模块之间的接口进行测试
- ✓过程调用、函数调用、消息传递、远程过程调用

### □测试技术

- ✓采用黑盒测试技术

### □集成测试的内容

- ✓过程调用
- ✓函数调用
- ✓消息传递
- ✓远程过程调用
- ✓网络消息

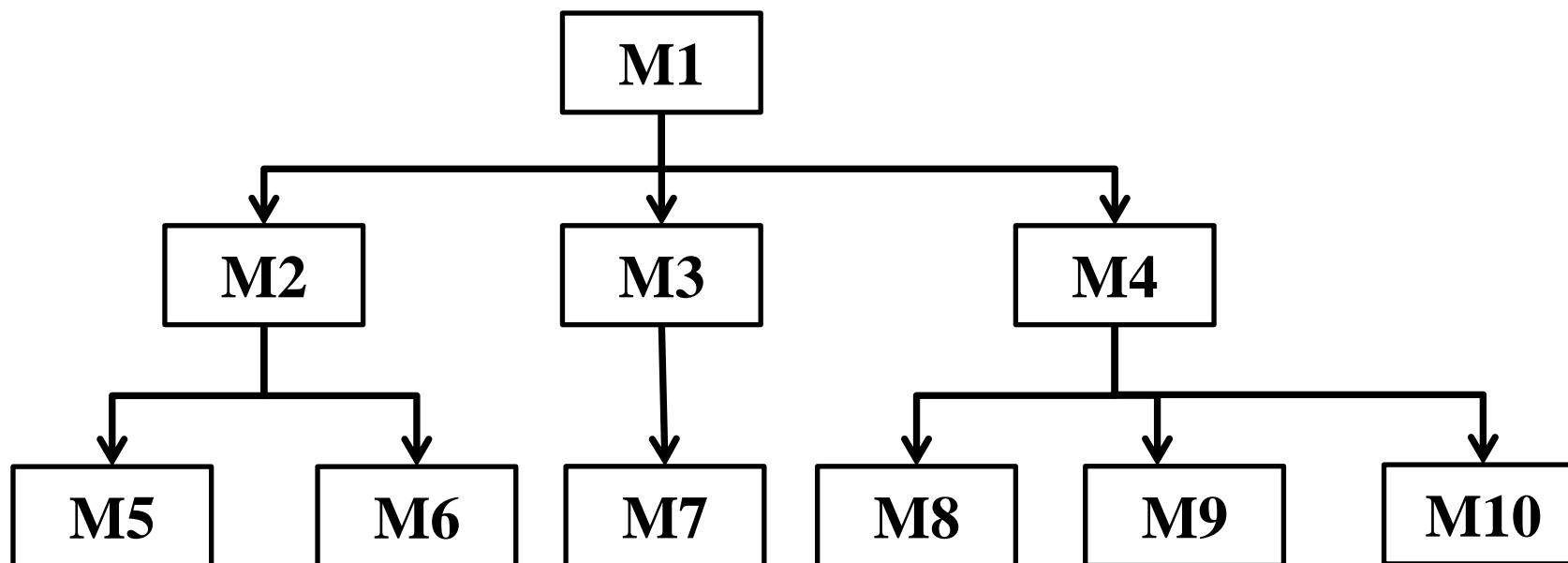


## □ 自顶向下集成

- ✓ 深度优先或者广度优先的策略把各个模块进行组装和测试

## □ 自底向上集成

- ✓ 自底向上进行组装和测试



在概要设计阶段就可以设计集成测试用例及计划

# 自顶向下集成测试的过程

- 以**软件主控模块**作为测试驱动模块，把对主控模块进行单元测试时引入的所有**桩模块**用实际模块替代
- 依据集成策略(深度优先或广度优先)，每次只替代一个桩模块
  - ✓ 每集成一个模块立即测试一遍
  - ✓ 只有每组测试完成后，才着手替换下一个桩模块
- 为避免修改引入新错误，须不断进行回归测试（即全部或部分地重复已做过的测试）
- 从步骤（2）开始循环执行上述步骤，直至整个程序结构集成完毕

# 自底向上集成测试的过程

- 把低层模块集成起来形成实现某个子功能的模块群 (cluster)
- 开发一个测试用驱动模块，控制测试数据的输入和测试结果的输出
  - ✓对每个模块群进行测试
  - ✓删除测试使用的驱动模块，用实际开发的高层模块代替并形成能完成更大功能的新模块群
- 从步骤（1）开始循环执行上述各步骤，直至整个程序集成完毕
  - ✓首先测试底层，再测试顶层，主要故障的发现会推迟到后期，而顶层故障更多反映设计中的故障，应该再开发中尽快改正而不是等到后期

## 2.4.3 确认测试

### □测试对象

- ✓对软件的功能和性能进行测试
- ✓判断目标软件系统是否满足用户需求

### □依据和标准

- ✓软件需求规格说明书

### □测试技术

- ✓采用黑盒测试技术



在需求分析阶段就可以设计确认测试用例及计划

# $\alpha$ 测试和 $\beta$ 测试

## □ $\alpha$ 测试

- ✓ **软件开发公司内部人员**模拟各类用户行为对即将面市的软件产品（称为 $\alpha$ 版本、内部测试版）进行测试，发现错误并修正
- ✓ 尽可能逼真地**模拟实际运行环境和用户**对软件产品的操作，并尽最大努力涵盖所有可能的用户操作方式
- ✓ 经 $\alpha$ 测试并进行修改后的软件产品称为 **$\beta$ 版本**（也称外部测试版）

## □ $\beta$ 测试

- ✓ 软件开发公司组织各方面的**典型用户**在日常工作中实际使用 $\beta$ 版本，或为对外进行宣传而将 $\beta$ 版本免费赠送给典型用户（很多情况下， $\beta$ 版本可以通过Internet免费下载，也可以向软件公司索取），并要求用户报告异常情况、提出批评意见
- ✓  $\beta$ 测试是在与开发者无法控制的环境下进行的软件现场应用



## 2.4.5 非功能性测试

- 性能测试
- 强度测试
- 配置和兼容性测试
- 安全性测试
- 可靠性测试
- 用户界面测试
- 本地化测试
- Web测试
- 安装测试
- .....



在需求分析阶段就可以设计非功能测试用例及计划

# 性能测试用例示例

<b>用例标识:</b>	<b>KCZC-XN-1</b>
<b>测试项:</b>	课程注册管理系统的性能能否支持同时有100个用户登陆使用。并且平均响应时间少于1.5秒。
<b>测试输入:</b>	由模拟软件模拟100个用户使用系统。
<b>前提条件:</b>	软件系统已经完成并且在服务器上安装成功。
<b>环境要求:</b>	使用1台服务器，配置为××××； 使用5台微机作为客户机，配置为××××； 模拟软件在客户机安装；网络通畅。
<b>测试步骤:</b>	(1)启动服务器系统； (2)由模拟软件在每个客户机分别依次发送20个用户登陆请求，并模拟选课过程； (3)监控程序把每个用户的请求和服务器响应过程、时间记录在日志中； (4)对记录结果进行处理，判断是否每个用户均获得成功的服务，并计算平均响应时间。
<b>预期输出:</b>	每个模拟用户都得到正确服务，且对请求的平均响应时间小于1.5秒。

## □强度测试可被视为性能测试一部分

- ✓ 性能测试是为测试软件系统在正常使用时是否能够达到一定性能指标
- ✓ 强度测试则是对系统不断施加更大的压力和使用强度，确定系统瓶颈或者不能接受性能点，来获得系统能提供最大服务能力

## □强度测试示例

- ✓ 机票预订系统同时500个用户使用情况下正常运行属于性能测试
- ✓ 把测试场景定为上千用户、甚至上万用户同时使用，那么就是一种强度测试了，以判断系统能够承受最大的强度是多少

## □强度测试适用于在可变负载系统中运行的软件

- ✓ 强度测试可与性能测试一起进行，但需对测试用例中定义的测试环境、条件、输入、步骤等内容进行调整，以适应强度测试

## □设计测试用例来暴露软件安全漏洞的过程

- ✓如设法破坏数据库管理系统的数据安全机制、突破重要领域软件系统的用户访问控制等

## □安全性测试的过程

- ✓明确潜在的安全隐患，包括容易遭受攻击的内容、可能作为潜在入侵者的人员角色等
- ✓明确潜在的入侵行为及时机，例如事务初始化、系统输入、执行存储和检索操作等时候
- ✓列出每种潜在安全隐患可能遭到的入侵行为及其可能性
- ✓确定风险较高的入侵点，这可通过对包括发生可能性、后果严重程度等因素进行分析得到
- ✓根据风险的顺序，设计测试用例来实施安全性测试

**□也称易用性测试，测试用户界面是否易于使用且具有较强实用性**

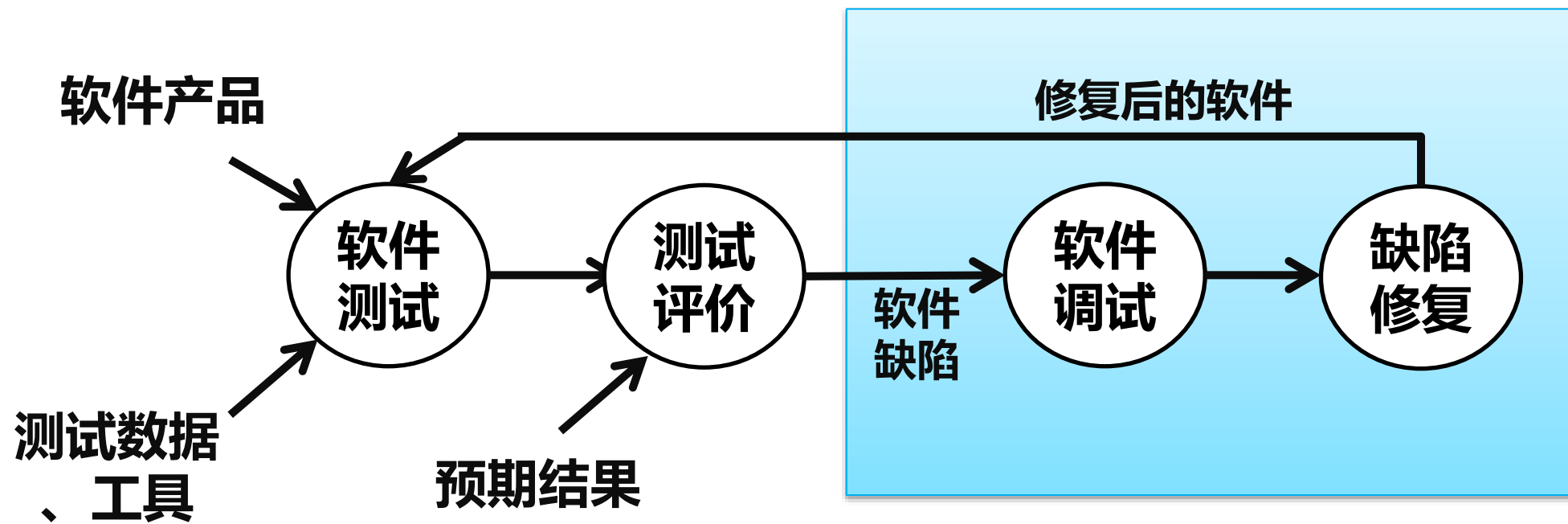
- ✓ 是否把常用的按钮或菜单项放在醒目的位置
- ✓ 软件的输出是否清晰、有序
- ✓ 错误信息是否直观和易于理解
- ✓ 软件错误顺序是否符合用户的业务过程
- ✓ 界面的风格是否一致

□基于Web软件利用浏览器通过超链接把各个网页联系在一起，满足用户对信息的获取

□需要测试大量的对象

- ✓文本测试，对其中的术语、题目、内容素材、电子邮件地址、电话号码等信息的准确度、时效性方面进行检查，文字的拼写是否有误也要经常进行检查
- ✓由于窗口缩放会引起文字段落改变，应测试是否会导致格式混乱
- ✓测试超级链接
- ✓测试图片和视频
- ✓测试表单

## 2.5 软件测试的后续工作



测试是为了发现错误，从而跟错误信息来纠正错误，提高软件质量

## □修改程序可能会引入新的错误

- ✓原先“正常”的程序现在变得“不正常”

## □回归测试目的

- ✓验证软件新版本是否从正常状态回归/退化到不正常状态

## □方法

- ✓重新进行测试，再次运行所有的测试用例来发现缺陷

## □回归测试最好能够自动化

- ✓单元测试是回归测试的基础



## □目的

- ✓测试发现缺陷
- ✓调试定位缺陷
- ✓排错纠正错误

## □独立性不同

- ✓测试由独立的测试小组进行
- ✓调试和排错由开发人员完成

# 测试的重要性和特殊性

---

## □软件测试是软件质量保证活动中关键步骤

- ✓对SRS、设计规格说明书以及编码的最后复审
- ✓其工作量往往占软件开发总工作量的40%以上
- ✓确保软件质量的一种有效 和可操作的重要手段

## □软件测试有其特殊性和规律

- ✓软件是逻辑产品
- ✓软件有其复杂性

## 2.6 软件测试原则(1/2)

### □测试应该有计划

- ✓ 在开发初期就应制定测试计划，测试计划应该在测试之前就应该制定

### □所有的测试都应该追溯到用户需求

### □将Pareto原则应用于软件测试

- ✓ 80%的错误都可以在大概20%的模块中找到根源

### □测试应该从“微观”开始，逐步转向“宏观”

### □穷举测试是不可能的

### □每个测试用例都必须定义预期的输出或结果

### □尽量避免程序的开发人员来测试他自己编写的代码

# 软件测试原则(2/2)

- 应详细检查每次测试的结果。
- 测试用例中不仅要说明合法有效的输入条件，还应该描述那些不期望的、非法的输入条件。
- 检查一个程序没有完成希望它做的事情只是测试的一半任务，还应检查程序是否执行了不希望它做的事情。
- 避免随意舍弃任何测试用例，即使非常简单的测试用例。
- 不应在事先假设不会发现错误的情况下制定测试计划。
- 一个程序模块存在更多错误的可能性与该模块已经发现的错误数量成正比。
- 测试是一个具有相当创新性和智力挑战的活动。



# 内容

## 1. 软件测试概述

- ✓ 软件测试的思想和原理

## 2. 软件测试的过程和策略

- ✓ 软件测试的活动及实施的方法
- ✓ 测试的原则

## 3. 软件测试技术

- ✓ 白盒和黑盒测试技术

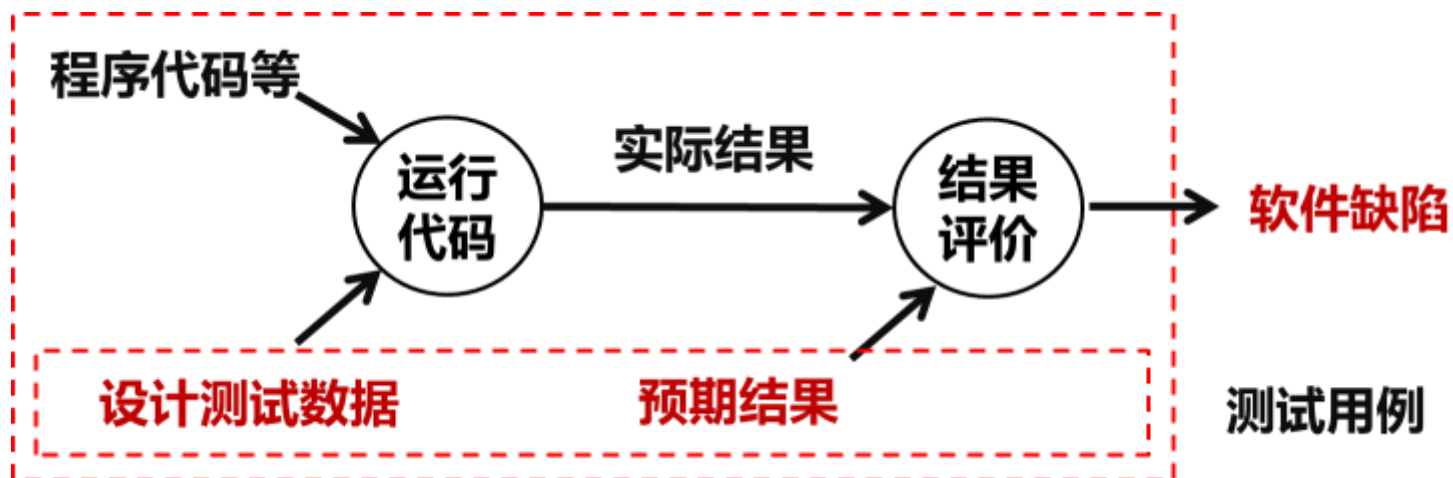
## 4. 软件测试计划及输出

- ✓ 测试计划制定及测试结果



# 思考和讨论

- 如何来设计测试用例？
- 要设计多少个测试用例？
- 如何才能做到充分的软件测试？即通过设计足够多的测试用例来发现软件尽可能多的缺陷

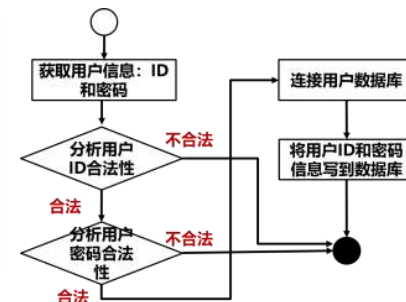
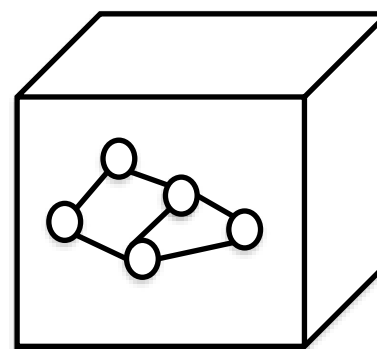


## 3.3 软件测试技术

### □如何设计和运行测试用例

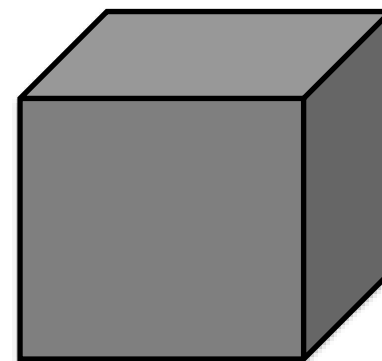
### □白盒测试技术

- ✓基于程序内部的执行流程来设计测试用例
- ✓单元测试主要采用白盒测试



### □黑盒测试技术

- ✓基于程序的外在功能和接口来设计测试用例
- ✓集成测试、确认测试大多采用黑盒测试



程序功能

盒子对应于基本模块单元（如类方法、函数、过程等），测试基本对象

## 3.4 软件测试技术-白盒测试

### □设计测试用例思想

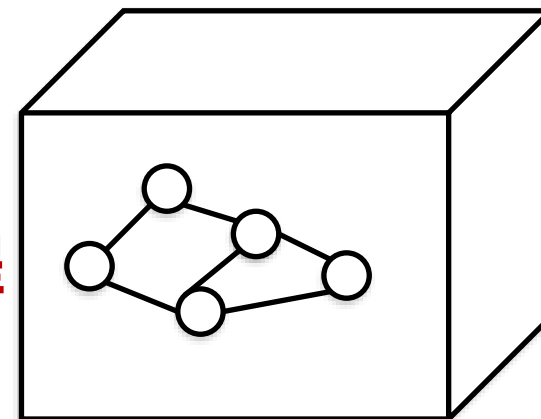
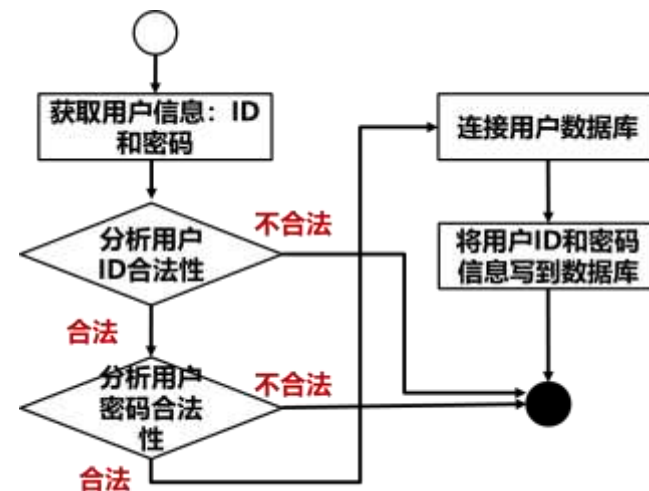
- ✓根据**程序单元内部工作流程**来设计测试用例

### □发现程序单元缺陷

- ✓运行待测试的程序，**检验程序是否按内部工作流程来运行的**，如果不是则存在缺陷

### □特点

- ✓必须了解程序的内部工作流程才能设计测试用例



**模块内部流程**



# 谁负责开展程序单元测试？

- 单元测试必须由最熟悉代码的人来写
- 程序单元代码的编写者完成程序单元测试
- 程序员必须保证自己编写程序的质量

谁写的代码谁自己来负责单元测试

# 单元测试的结果是什么？

## □程序单元测试报告

- ✓测试用例的设计
- ✓程序单元的运行
- ✓运行结果情况
- ✓是否与预期相符一致

## □谁负责撰写该报告

- ✓程序员

**程序单元测试报告**

# 什么时候实施单元测试？

□根据设计文档，在程序代码编写之前设计测试用例

✓程序单元测试计划

□程序代码写完之后开展程序单元测试

**设计测试用例、制定测试计划与实际开展测试可在不同的阶段来开展！**

## 3.7 单元测试的基本原则

---

- 由程序设计人员来完成
- 应在基本功能/参数上验证程序的正确性
- 应该产生可重复、一致的结果
- 具有独立性，单元测试运行/通过/失败不依赖于别的测试
- 应该覆盖所有代码路径
- 应该集成到自动测试的框架中
- 必须和产品代码一起保存和维护

# 白盒测试用例设计的指导原则

## □如何设计测试用例？

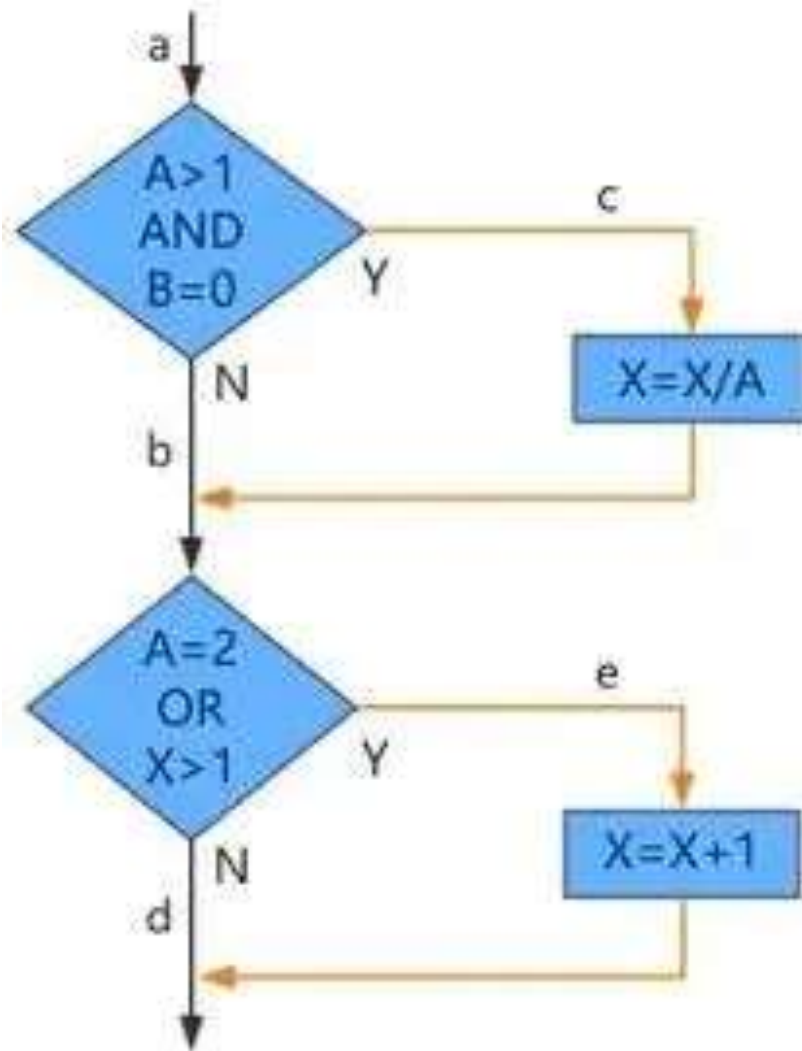
- ✓ 内部执行流程
- ✓ 生成测试数据

## □设计多少测试用例？

- ✓ 遵循覆盖原则

## □测试用例覆盖准则

- ✓ 语句覆盖
- ✓ 分支覆盖
- ✓ 路径覆盖
- ✓ 基本路径覆盖



- ✓ 语句覆盖
- ✓ 分支覆盖
- ✓ 路径覆盖
- ✓ 基本路径覆盖

# 基本路径测试的思想

## □思想

✓ 路径 → 基本路径 → **基本路径测试**

## □基本路径

✓ 至少**引入一个新语句或者新判断**的程序通道

## □前提

✓ 软件模块的逻辑结构（流程图）

✓ 据此可掌握模块的基本路径

## □如何设计测试用例实现基本路径覆盖

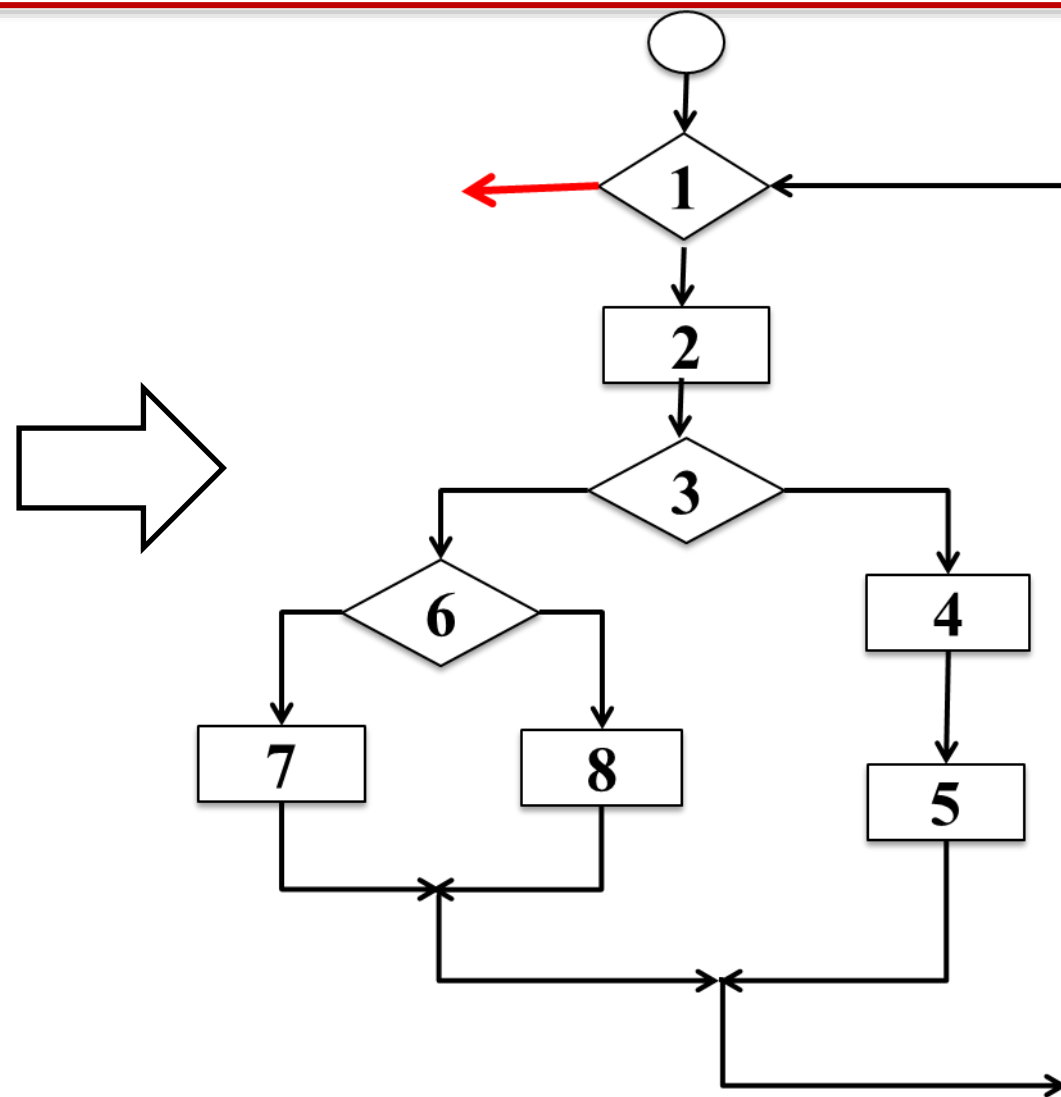
✓ 哪些基本路径 → **流图** → **流程图**

**基本路径测试可实现  
基本路径的测试覆盖**

# 步骤1: 根据程序逻辑画出流程图

```
void Func(int nPosX, int nPosY) {  
    while (nPosX > 0) {  
        int nSum = nPosX + nPosY;  
        if (nSum > 1) {  
            nPosX减一;  
            nPosY减一;  
        }  
        else {  
            if (nSum < -1) nPosX -= 2;  
            else nPosX减四;  
        }  
    } // end of while  
}
```

软件模块设计细节

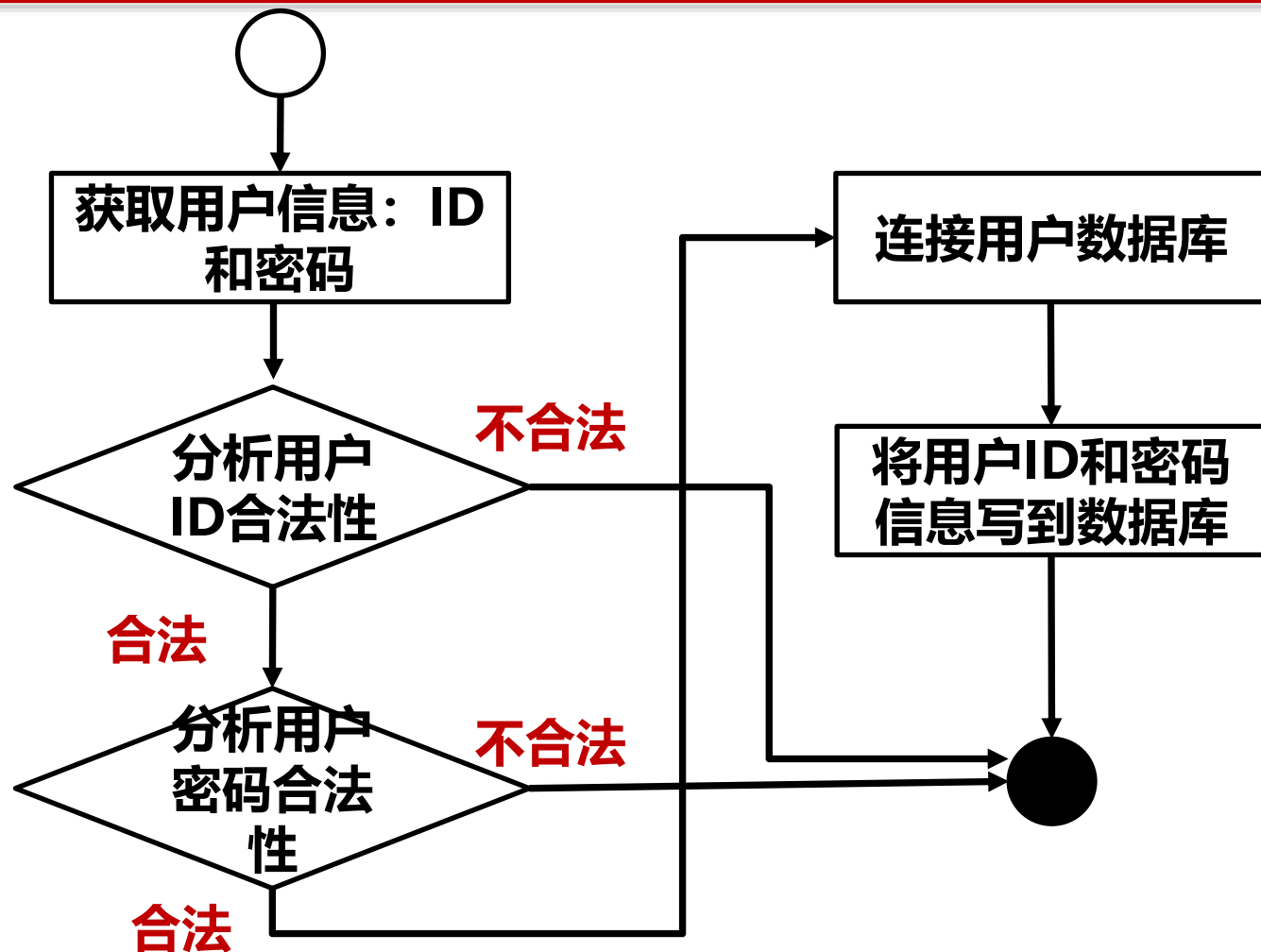


注意流程图的正确画法!

# 示例：程序流程图

✓ 数字和字符组合  
✓ 不与已有的重复

✓ 数字和字符组合  
✓ 不少于4个字符



用户注册模块的流程图



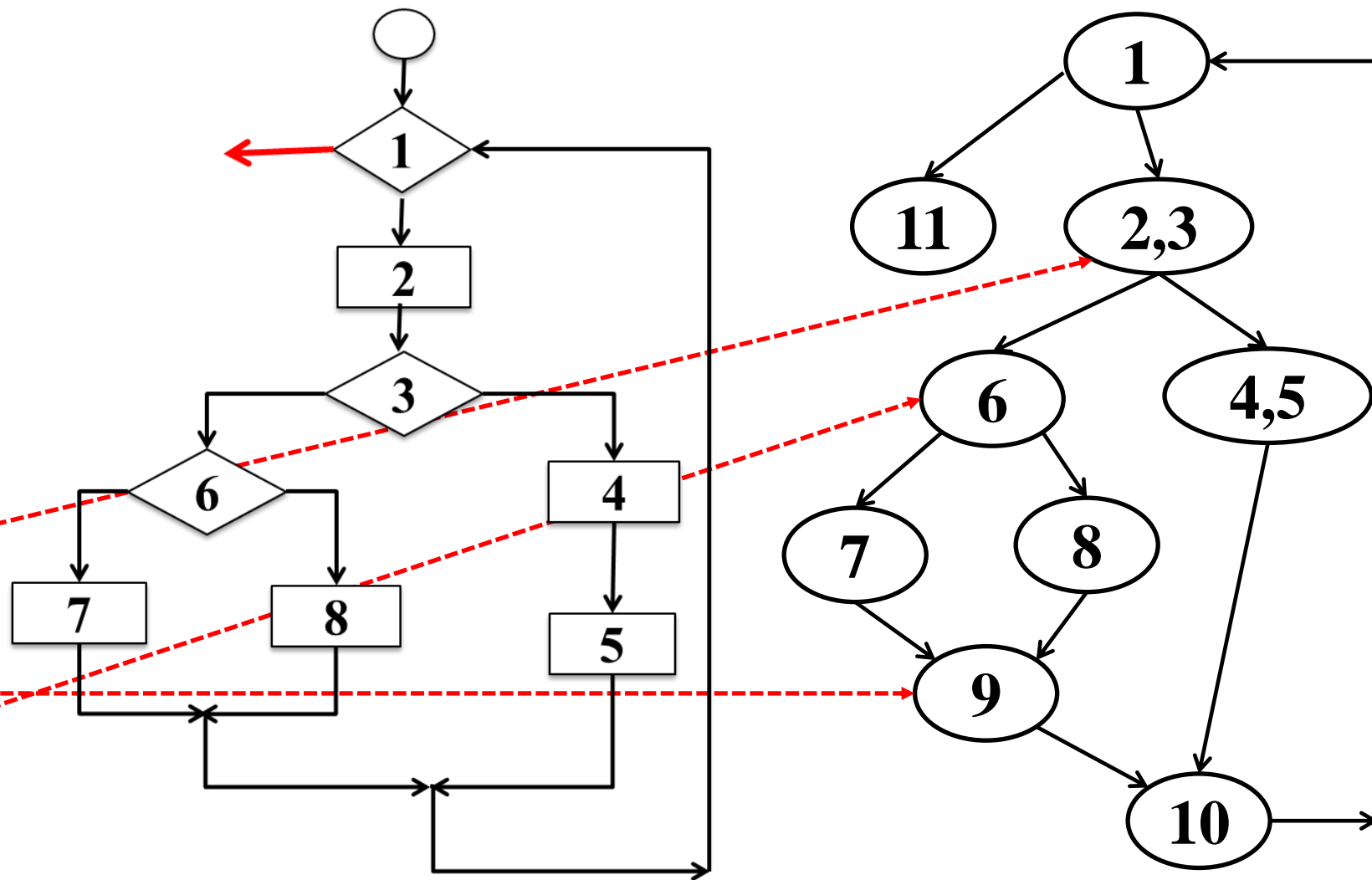
## 步骤2: 将流程图转换为流图 (1/2)

### □何为流图

- ✓ 流图刻画程序控制结构但不涉及程序过程性细节

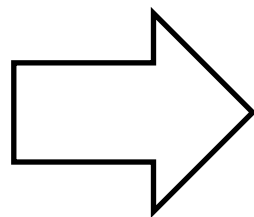
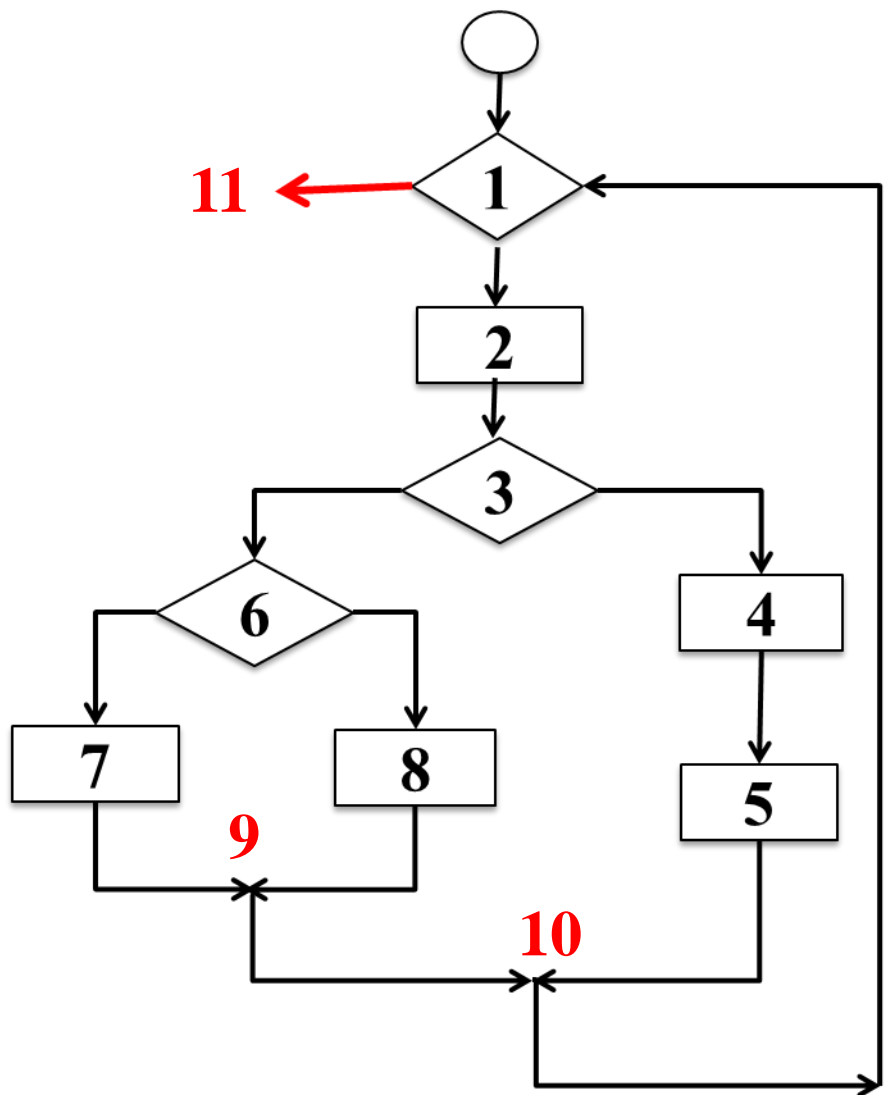
### □节点形式

- ✓ 过程块
- ✓ 结合点
- ✓ 判定点

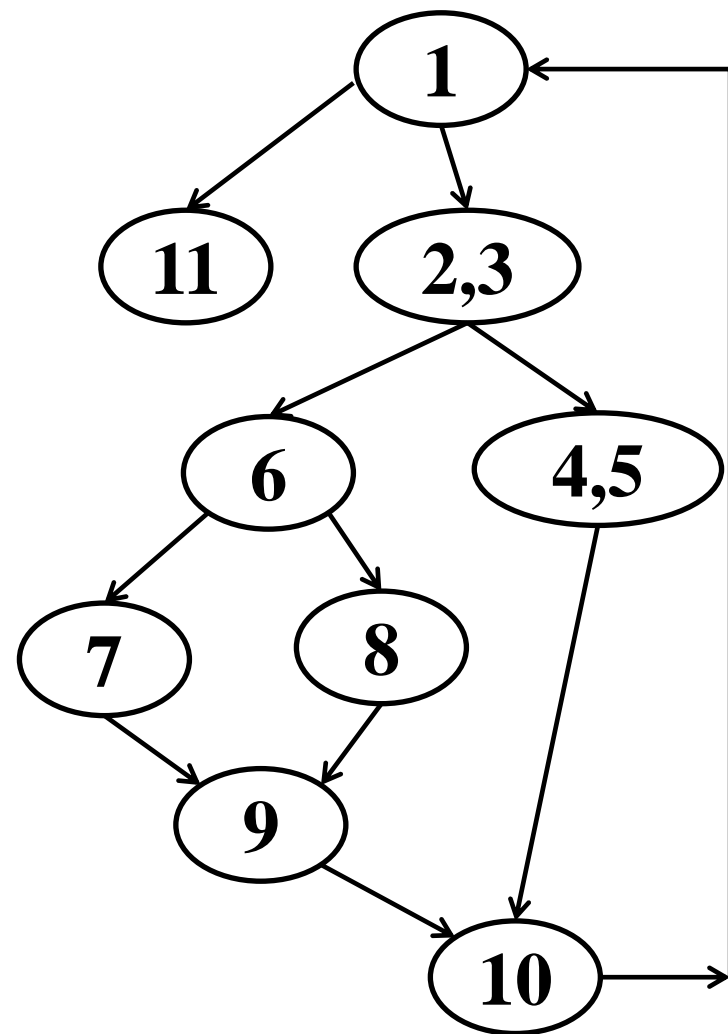


注意流图正确画法!

## 步骤2: 将流程图转换为流图 (2/2)



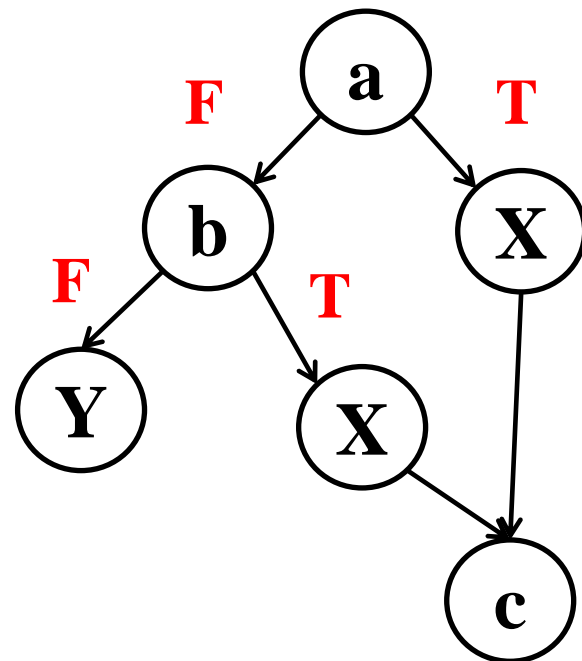
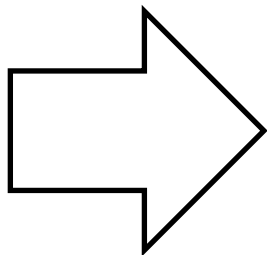
**判定点：转换  
为1个节点**  
**过程块：合并  
为一个节点：**  
**比如2,3**  
**结合点：条件  
语句的汇聚点**  
**， 比如9和10**



# 流图中的判定点不应含复合条件

□ 否则按下列方式增加判定点

If **a or b**  
Then X  
Else Y  
End If



## 步骤3: 确定基本路径集合

### □基本路径的数量

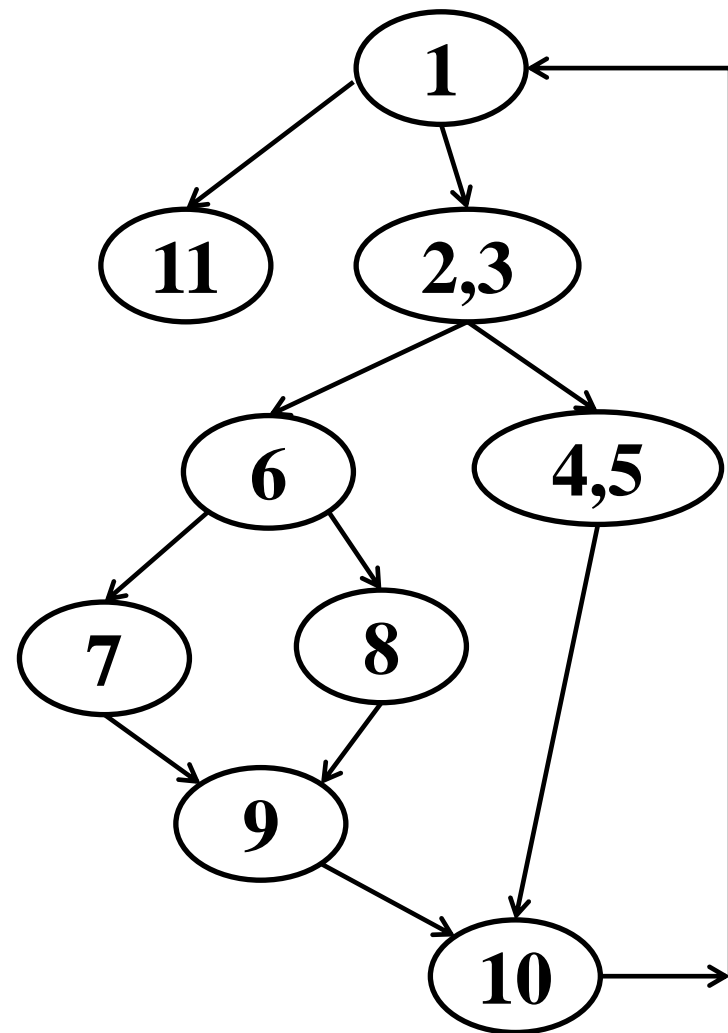
✓ 流图 **Cyclomatic** 复杂度

✓  $V(G) = E(\text{dges}) - N(\text{odes}) + 2$

✓  $V(G) = 11 - 9 + 2 = 4$

- ① 1 - 11
- ② 1 - 2, 3 - 6 - 7 - 9 - 10 - 1 - 11
- ③ 1 - 2, 3 - 4, 5 - 10 - 1 - 11
- ④ 1 - 2, 3 - 6 - 8 - 9 - 10 - 1 - 11

**4条基本路径**



# 步骤4: 针对测试路径设计测试用例

## □ 1-11

✓ nPosX 取-1, nPosY取任意值

## □ 1 - 2, 3 - 4, 5 - 10 - 1 - 11

✓ nPosX 取1, nPosY取1

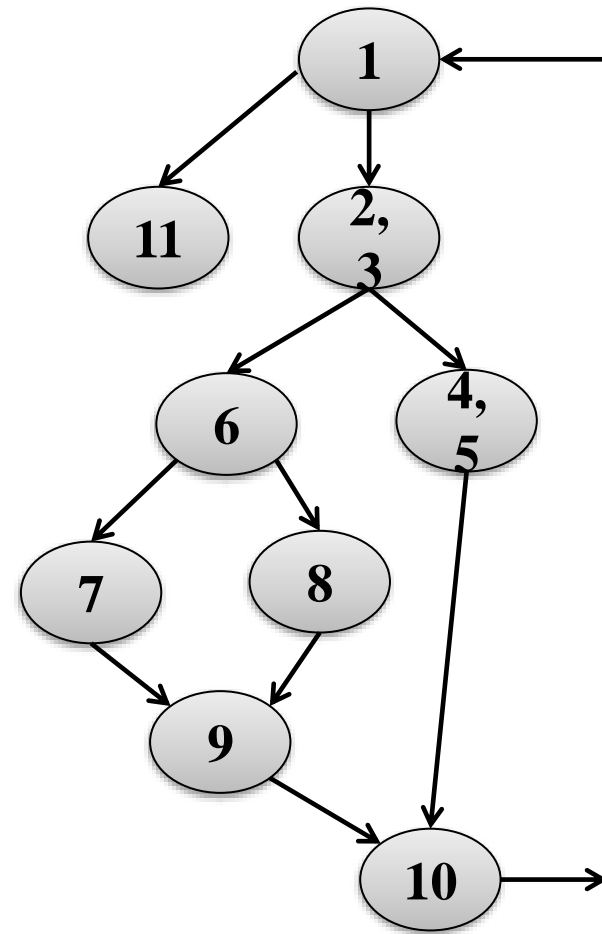
## □ 1- 2, 3- 6-7 - 9 - 10-1- 11

✓ nPosX 取1, nPosY取-1

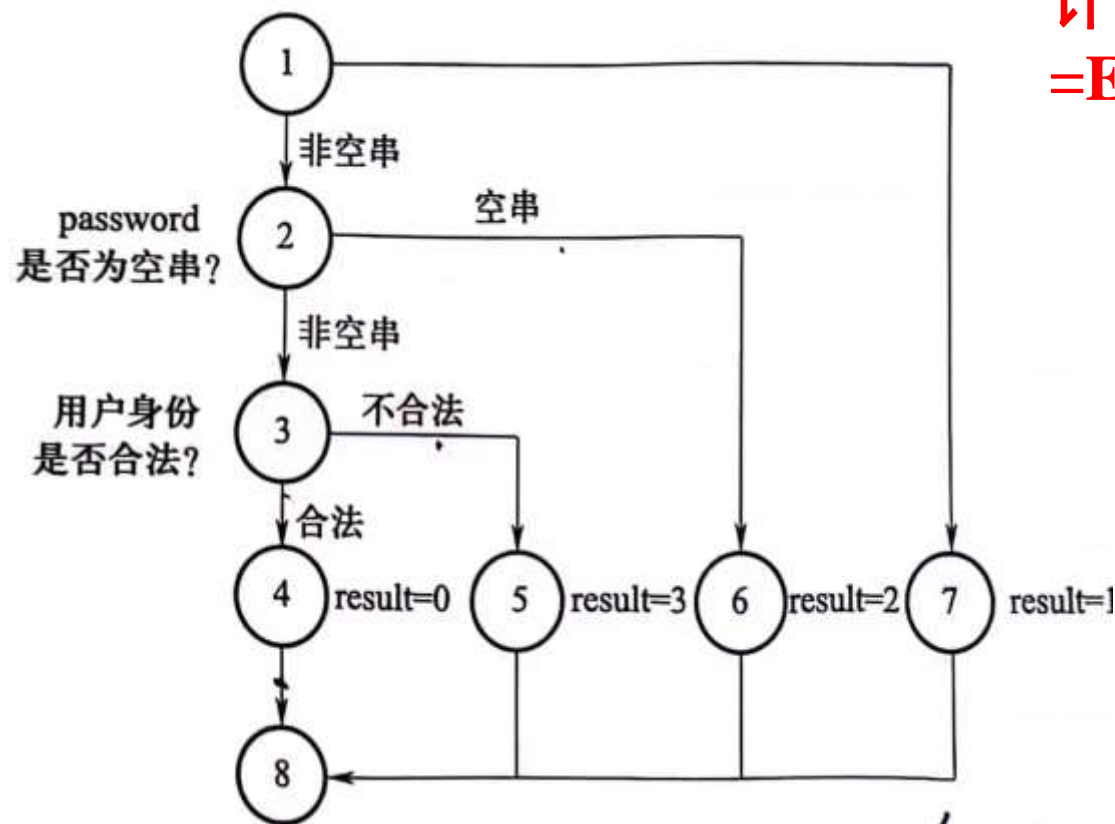
## □ 1- 2, 3 - 6 - 8 - 9 - 10-1- 11

✓ nPosX 取1, nPosY取-3

```
void Func(int nPosX, int nPosY) {  
    while (nPosX > 0) {  
        int nSum = nPosX + nPosY;  
        if (nSum > 1) {  
            nPosX减一;  
            nPosY减一;  
        }  
        else {  
            if (nSum < -1) nPosX -= 2;  
            else nPosX减四;  
        }  
    }  
    // end of while  
}
```



## □为loginManager类的login () 方法设计测试用例



计算流图的cyclomatic复杂度  
 $= \text{Edge (D)} - \text{Node (D)} + 2 = 4$

设计4条基本路径:

1:1-7-8

2:1-2-6-8;

3:1-2-3-5-8;

4:1-2-3-5-8

# 描述每一个测试用例

<b><u>用例标识:</u></b>	对该测试用例赋予一个唯一标识
<b><u>用例开发者:</u></b>	谁编写的本用例
<b><u>用例开发日期:</u></b>	编写用例的日期
<b><u>测试项:</u></b>	描述将被测试的具体特征、代码模块等对象
<b><u>测试输入:</u></b>	测试时为程序提供的输入数据
<b><u>前提条件:</u></b>	执行测试时系统应处于的状态或要满足的条件等
<b><u>环境要求:</u></b>	执行测试所需的软硬件环境、测试工具、人员等
<b><u>测试步骤:</u></b>	(1)...; (如点击“文件”菜单中的“新建”菜单项) (2)...; (如在“test case”目录下选择“test5.dat”文件) .....
<b><u>预期输出:</u></b>	希望程序运行得到的结果
<b><u>用例间的依赖性:</u></b>	该测试用例依赖或受影响的其它测试用例

# 步骤5: 运行程序检验测试用例

---

- 运行待测试的程序代码
- 逐个输入测试用例
- 分析程序的运行路径
- 如果运行路径与期望路径不一样，则存在缺陷



# 程序单元测试的运行环境

测试  
结果

驱动程序

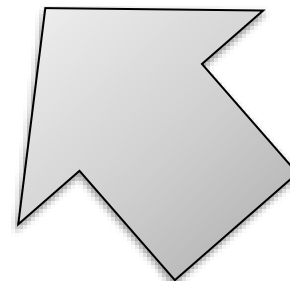
测试用例

<Data, Result>

.....

被测模块

- ✓ 产生测试用例
- ✓ 执行测试程序
- ✓ 生成测试报告



桩模块

桩模块

程序代码等

运行  
代码

实际结果

结果  
评价

软件缺陷

设计测试数据

预期结果

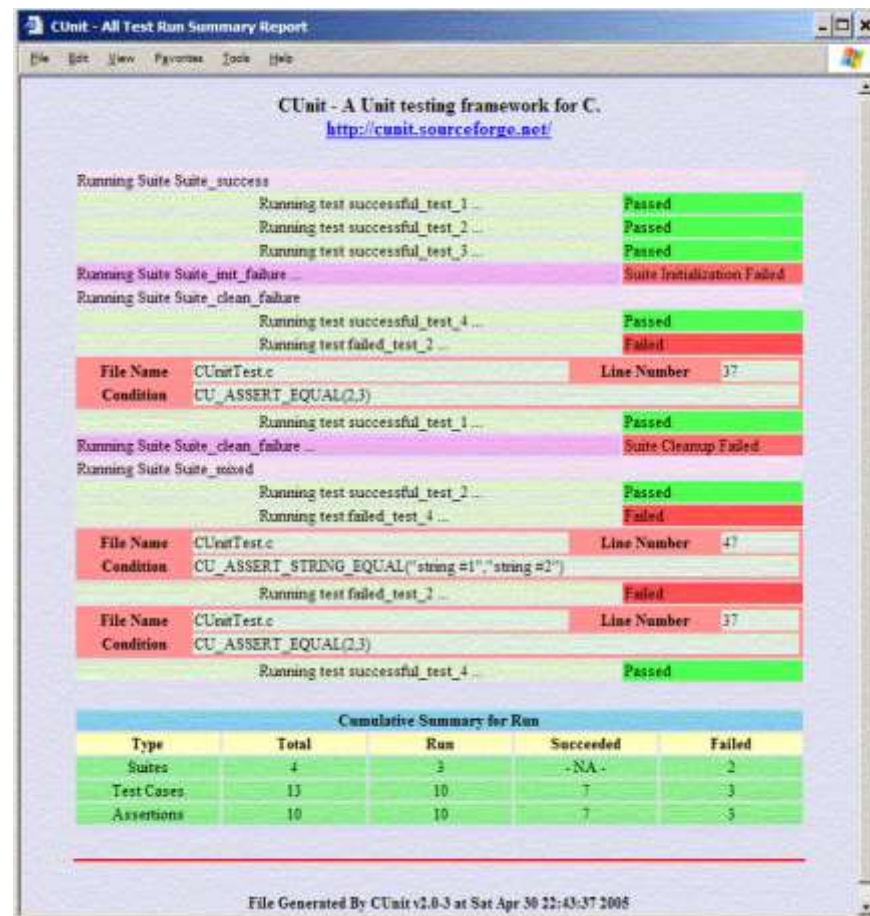
测试用例

□ C/C++语言 - C++Test、Cunit、CppTest

□ .Net开发 - NUnit

□ Java语言 - Junit

□ Python语言 - PyUnit



## 3.5 JUnit

- JUnit是一个Java语言的单元测试框架
- 由Kent Beck和Erich Gamma建立
- 开源软件



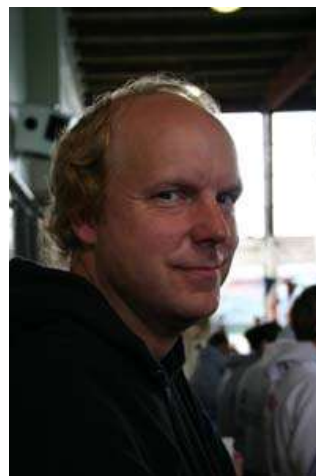
<http://junit.org/>

下载并安装

□ 1997年Erich Gamma和Kent Beck为Java语言创建了一个简单有效的单元测试框架



**Erich Gamma**  
**《设计模式》作者之一**



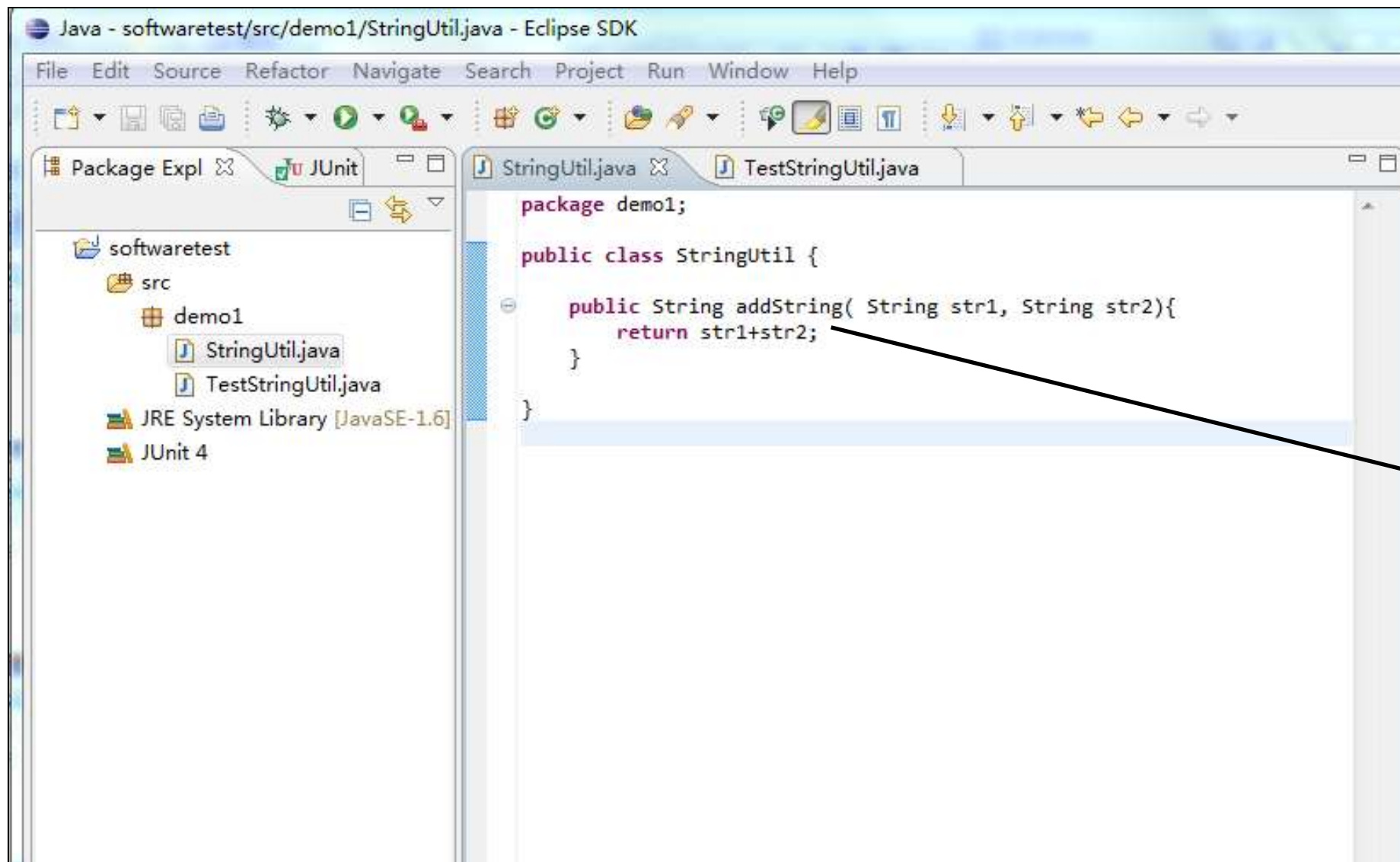
**Kent Beck**  
**提出软件开发方法“极限编程”**  
**《重构:改善既有代码的设计》作者**

# 在Eclipse中使用JUnit

---

- 1. 建立一个被JUnit测试的类**
- 2. 建立对应的JUnit Test类**
- 3. 针对自动生成的代码进行修改**
- 4. 执行测试用例**

## 3.5.1 建立一个被JUnit测试的类

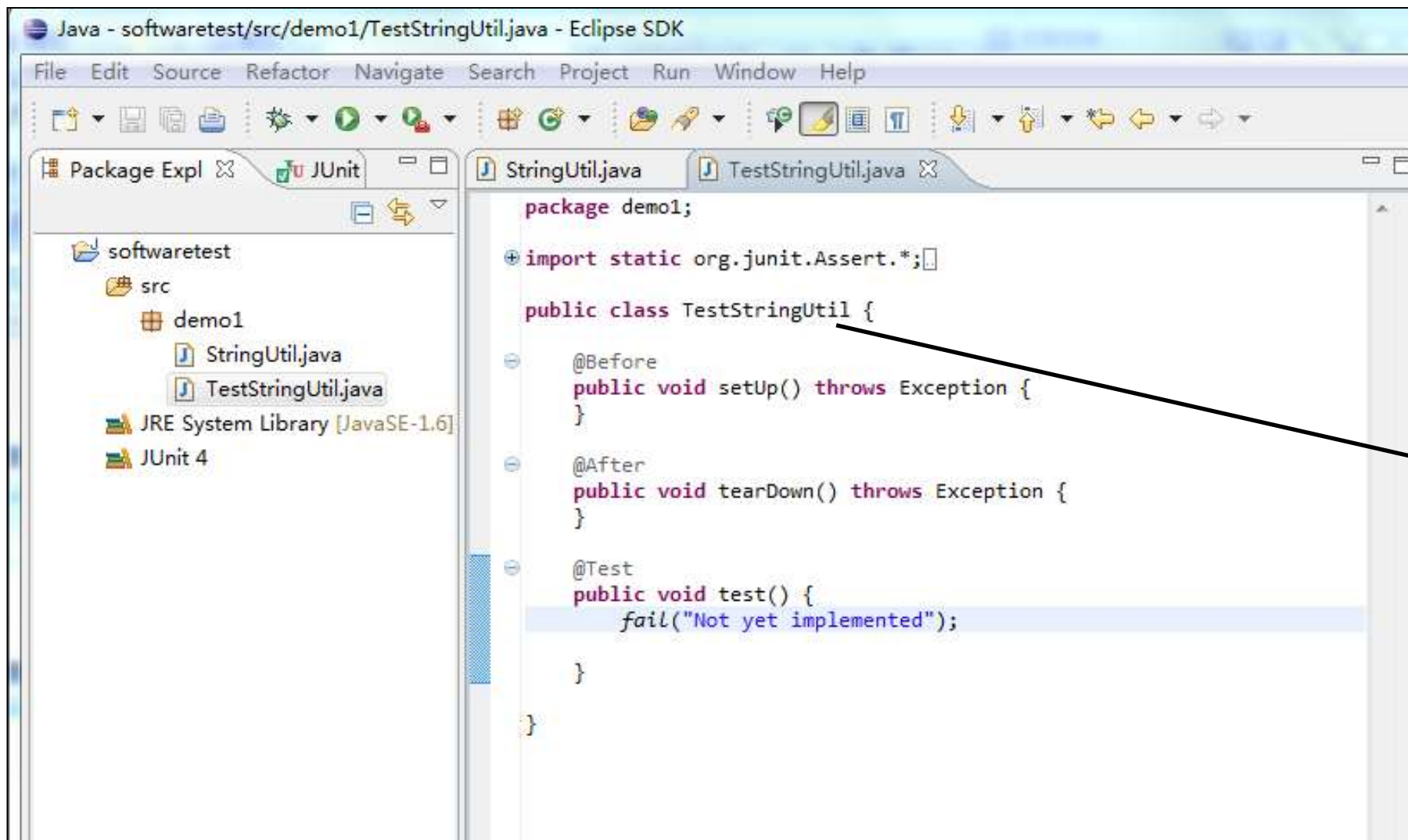


**被测试的对象**

**被测试的程序单元：类方法  
addString ()**



## 3.5.2 建立对应的JUnit Test类



类似于主控模  
块及桩模块

编写测试类

# Assert方法（测试用例预期输出）

## □ **assertArrayEquals**

✓ 判断两个数组是否相等

## □ **assertEquals**

✓ 判断两个对象是否相等

## □ **assertFalse**和**assertTrue**

✓ 判断布尔变量是否为False或True

## □ **assertNotNull**和**assertNull**

✓ 判断一个对象是否为空

## □ **assertNotSame**

✓ 判断两个引用是否指向同一个对象

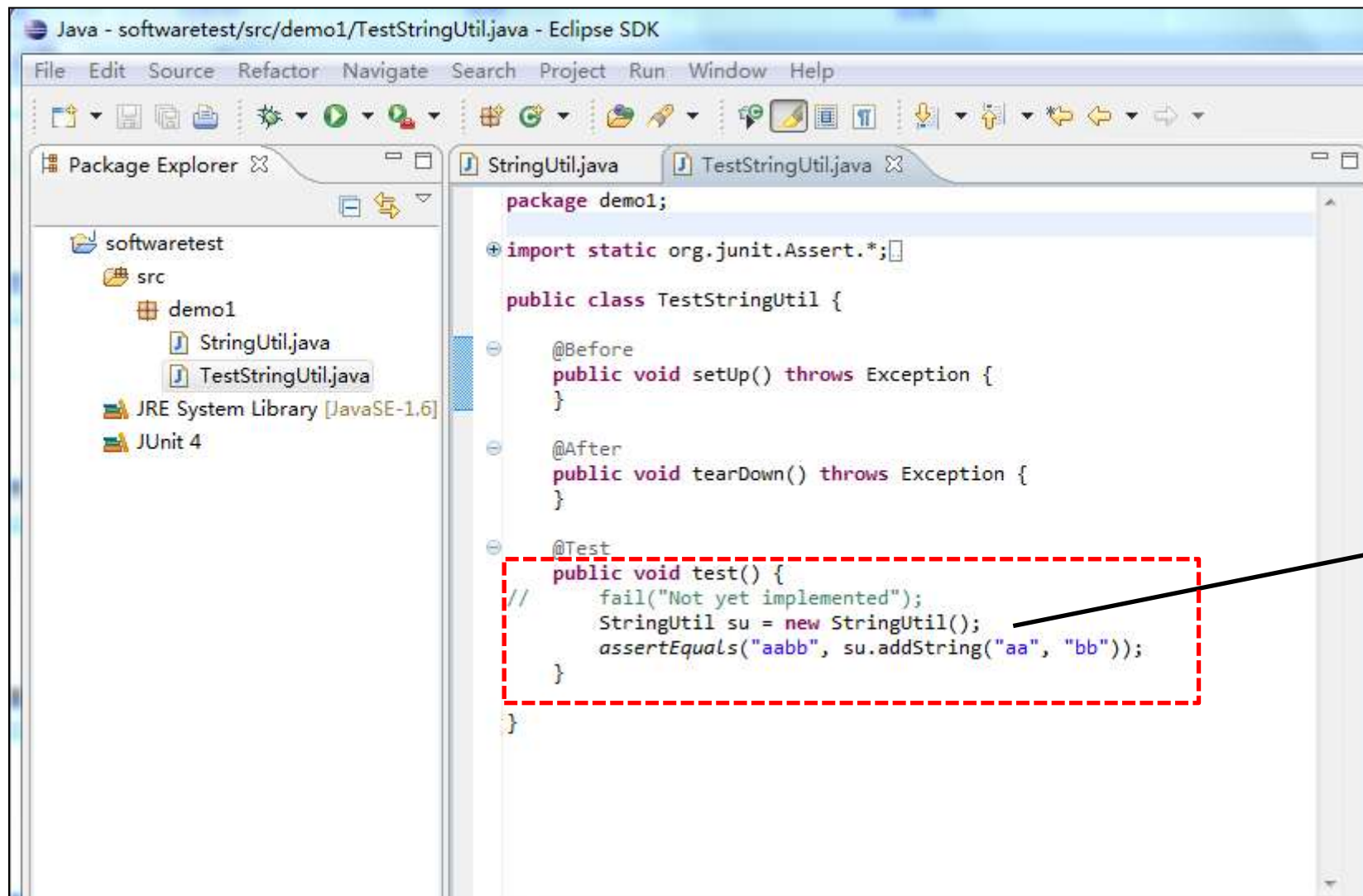
## □ **Fail**

✓ 让测试用例失败

用于判断预期结果与实际结果是否一致



## 3.5.3 针对自动生成的代码进行修改



编写桩模块代码

桩模块代码

## 3.5.4 测试用例执行通过



Java - softwaretest/src/demo1/TestStringUtil.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Expl JUnit

TestStringUtil

Runs: 1/1 Errors: 0 Failures: 0

demo1.TestStringUtil [Runner: JUnit]

test (0.004 s)

**Green Bar**

```
package demo1;

import static org.junit.Assert.*;

public class TestStringUtil {

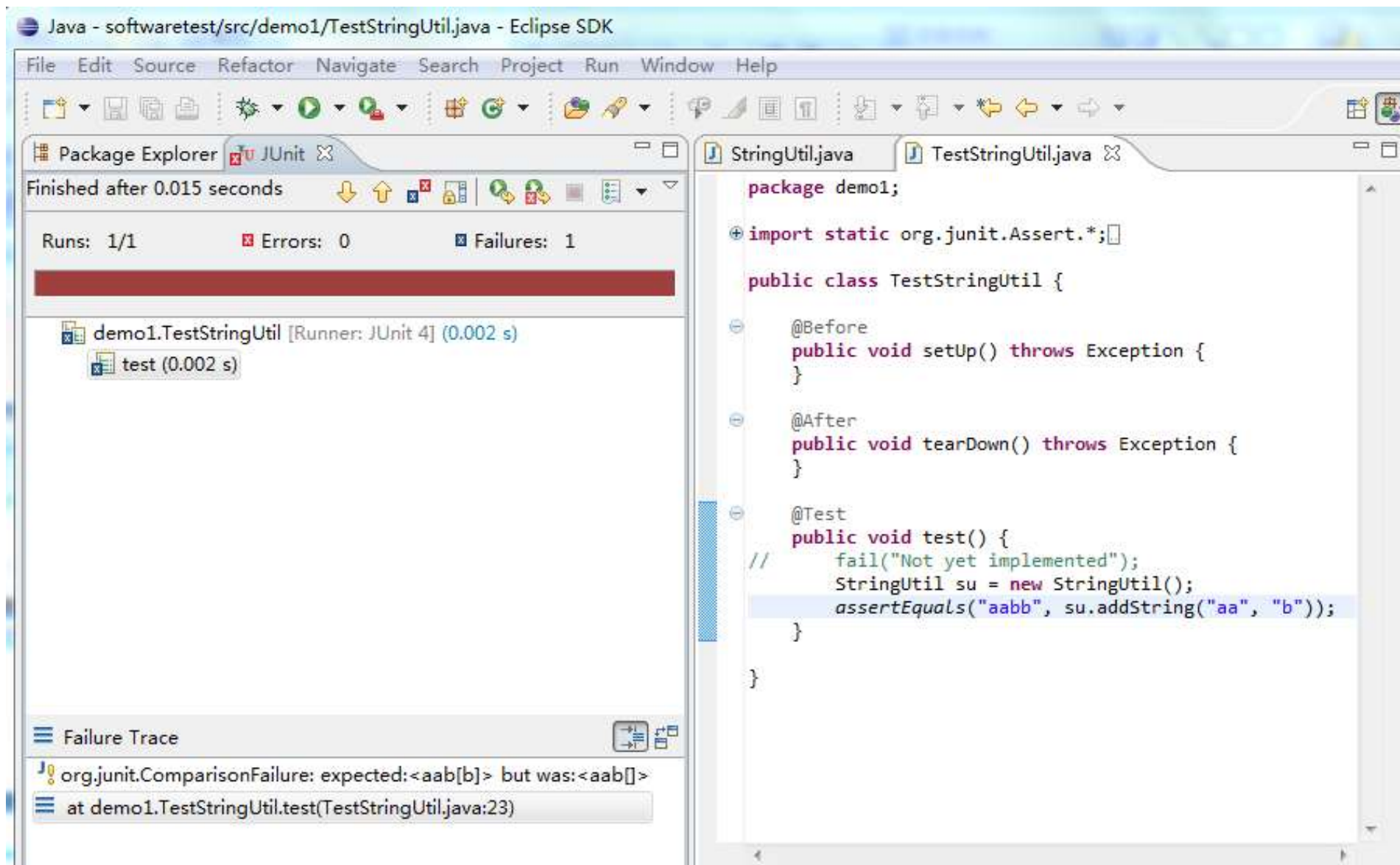
    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void test() {
        // fail("Not yet implemented");
        StringUtil su = new StringUtil();
        assertEquals("aabb", su.addString("aa", "bb"));
    }
}
```

**Keep the bar green to keep the code clean**

# 测试用例执行失败



The screenshot shows the Eclipse IDE interface with the following components:

- Top Bar:** Java - softwaretest/src/demo1/TestStringUtil.java - Eclipse SDK
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Standard Eclipse development icons.
- Package Explorer:** Shows the project structure with 'demo1' and 'TestStringUtil.java'.
- JUnit Console:** Displays the test results. It shows 'Finished after 0.015 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 1'. The test 'demo1.TestStringUtil [Runner: JUnit 4] (0.002 s)' is listed, with a sub-entry 'test (0.002 s)'.
- Failure Trace:** Shows the error message: 'org.junit.ComparisonFailure: expected:<aab[b]> but was:<aab[]>' at 'demo1.TestStringUtil.test(TestStringUtil.java:23)'. The line number 23 is highlighted in the code editor.
- Code Editor:** Displays the source code of 'TestStringUtil.java'. The code is as follows:

```
package demo1;

import static org.junit.Assert.*;

public class TestStringUtil {

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void test() {
        // fail("Not yet implemented");
        StringUtil su = new StringUtil();
        assertEquals("aabb", su.addString("aa", "b"));
    }
}
```

- 使测试代码与产品代码分开
- 针对某一个类的测试代码通过较少的改动便可以应用于另一个类的测试
- 开源软件，可以进行二次开发
- 方便地对JUnit进行扩展
- 与开发环境Eclipse相互集成

## 3.6 基于CASE的软件测试

---

- 测试数据产生器
- 运行测试程序
- 产生测试报告

## 3.8 黑盒测试

### □思想

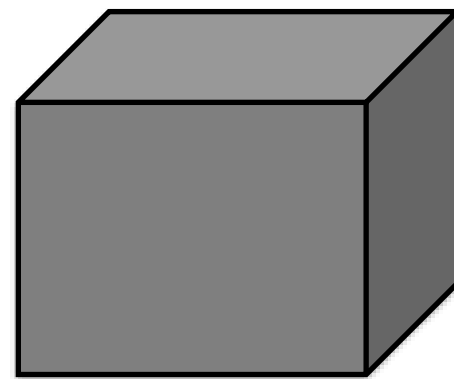
- ✓根据已知的**程序功能和性能**(而非内部细节), 设计测试用例并通过测试检验程序的每个功能和性能是否正常

### □依据

- ✓程序的**功能和性能描述**

### □特点

- ✓知道程序功能和性能, 不必了解程序内部结构和处理细节



**模块外在功能**

## □测试软件系统是否满足功能要求

- ✓不正确或遗漏的功能
- ✓界面错误
- ✓数据结构或外部数据库访问错误
- ✓性能错误
- ✓初始化和终止条件错误

# 黑盒测试的特点

---

- **黑盒测试与软件如何实现无关**，如果软件实现发生变化，测试用例仍然可以使用
- **黑盒测试用例的开发可以与软件实现并行进行**，能够缩短软件开发周期



# 黑盒测试-等价分类法

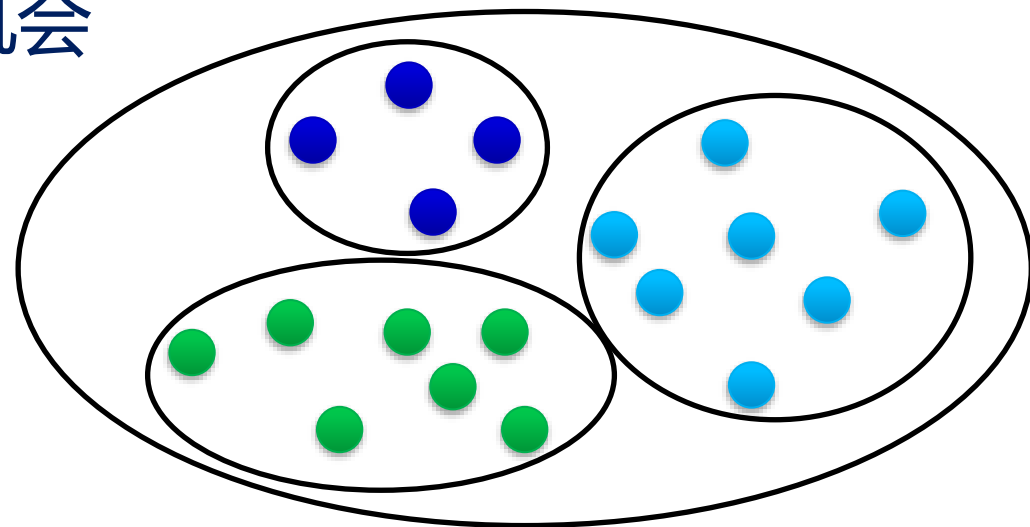
## □思想

- ✓把程序的输入数据集合按输入条件划分为若干个**等价类**
- ✓每一个等价类对于输入条件而言为一组有效或无效的输入
- ✓为每一个等价类设计一个测试用例

## □优点

- ✓减少测试次数，不丢失发现错误的机会

**每个等价类中的  
数据具有相同  
的测试特征**



# 等价分类法的基本原则

## □ 输入条件为一范围

- ✓ 划分出三个等价类
- ✓ (1) 有效等价类(在范围内), (2) 大于输入最大值, (3) 小于输入最少值

## □ 输入条件为一值

- ✓ 划分为三个等价类
- ✓ (1) 有效, (2) 大于, (3) 小于

## □ 输入条件为集合

- ✓ 划分二个等价类
- ✓ (1) 有效(在集合内), (2) 无效(在集合外)

## □ 输入条件为一个布尔量

- ✓ 划分二个等价类
- ✓ (1) 有效(此布尔量), (2) 无效(布尔量的非)

# 等价分类法示例

□  $z = \text{func}(x, y)$ :

✓ 当  $0 < x < 1024$  且  $y = 0$ ,  $z = -1$

✓ 否则,  $z = x * \lg(y)$

□ 关于  $x$  的等价类

✓  $(0, 1024)$

✓  $(-\#, 0]$

✓  $[1024, +\#)$

□ 关于  $y$  的等价类

✓  $\{0\}$

✓  $(-\#, 0)$

✓  $(0, +\#)$

测试用例9个

- ①  $X=1, y=0, z=-1$
- ②  $X=1, y=-1, z=**$
- ③  $X=1, y=1, z=**$
- ④  $X=0, y=1, z=**$
- ⑤  $X=0, y=-1, z=**$
- ⑥  $X=0, y=1, z=**$
- ⑦  $X=2000, y=0, z=**$
- ⑧  $X=2000, y=-100, z=**$
- ⑨  $X=2000, y=200, z=**$

## □输入条件是一范围(a,b)

- ✓ a,b以及紧挨a,b左右的值应作为测试用例

## □输入条件为一组数

- ✓ 选择这组数最大者和最小者，次大和次小者作为测试用例

## □如果程序的内部数据结构是有界的

- ✓ 应设计测试用例使它能够检查该数据结构的边界

# 边界值分析法示例

□  $z = \text{func}(x, y)$ :

✓ 当  $0 < x < 1024$  且  $y = 0$ ,  $z = -1$

✓ 否则,  $z = x * \lg(y)$

□ 关于  $x$  的等价类-6个

✓ -1, 0, 1

✓ 1023, 1024, 1025

□ 关于  $y$  的等价类-3个

✓ -1, 0, 1

测试用例18个

- ①  $X=1, y=0, z=-1$
- ②  $X=1, y=-1, z=**$
- ③  $X=1, y=1, z=**$
- ④  $X=0, y=0, z=**$
- ⑤  $X=0, y=-1, z=**$
- ⑥  $X=0, y=1, z=**$
- ⑦  $X=-1, y=0, z=**$
- ⑧  $X=-1, y=-100, z=**$
- ⑨  $X=-1, y=200, z=**$
- ⑩ .....



# 内容

## 1. 软件测试概述

✓ 软件测试的思想和原理

## 2. 软件测试的过程和策略

✓ 软件测试的活动及实施的方法

## 3. 软件测试技术

✓ 白盒和黑盒测试技术

## 4. 软件测试计划及输出

✓ 测试计划制定及测试结果



## 4.1 成立软件测试组织

---

□ **软件测试是一项独立性的工作**

□ **成立单独的软件测试小组**

- ✓ 成员由各个软件测试工作师组成
- ✓ 高效的开展软件测试，确保软件测试工作的权威性
- ✓ 软件测试小组也不受软件开发小组的管理，以确保软件测试的独立性和结果的客观性

□ **需在软件开发早期就成立软件测试小组**

- ✓ 以便他们尽早地介入到软件测试工作之中，包括开展必要培训、了解软件项目的整体情况、掌握软件需求、制定软件测试计划等



## 4.2 制定和实施软件测试计划 (1/2)

---

- **标识符**：用以标识本测试计划
- **简介**：介绍测试的对象、目标、策略、过程和进程等方面的内容
- **测试项**：描述接受测试的软件元素，包括代码模块或质量属性
- **待测软件特征**：说明接受测试的软件特征，如软件需求等
- **免测软件特征**：说明无需接受测试的软件特征，如软件需求等
- **测试策略和手段**：说明对软件进行测试的策略和手段
- **测试项成败标准**：说明每个测试项通过软件测试的标准
- **测试暂停/停止的标准**：说明软件测试暂停或停止的决策依据





# 制定和实施软件测试计划 (2/2)

---

- **测试交付物**：说明测试过程中或完成之后应该交付的软件测试制品
- **测试任务**：描述测试的主要任务
- **测试环境**：描述测试所需建立的环境
- **职责**：说明每项测试任务的主要负责人或团队及其职责
- **进度安排**：描述测试的过程及时间安排
- **成本预算**：估算软件测试的成本
- **风险与意外**：描述测试过程中可能存在的风险和可能发生的意外

## 4.3 软件测试的输出

---

□ 软件测试计划

□ 软件测试报告，记录软件测试情况以及发现的缺陷

- ✓ 软件单元测试报告
- ✓ 软件集成测试报告
- ✓ 软件确认测试报告
- ✓ 系统测试报告等
- ✓ 每个报告详细

□ 软件测试是为了**发现软件中的缺陷**

✓ 原理是设计和运行测试用例

□ 软件测试方法是**设计测试用例、运行测试代码、发现问题**

✓ 白盒测试、黑盒测试

□ 软件测试的**活动、过程和策略**

✓ 单元测试、集成测试、确认测试

□ 基于测试的结果来进行**纠错**

✓ 测试、调试和纠错