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

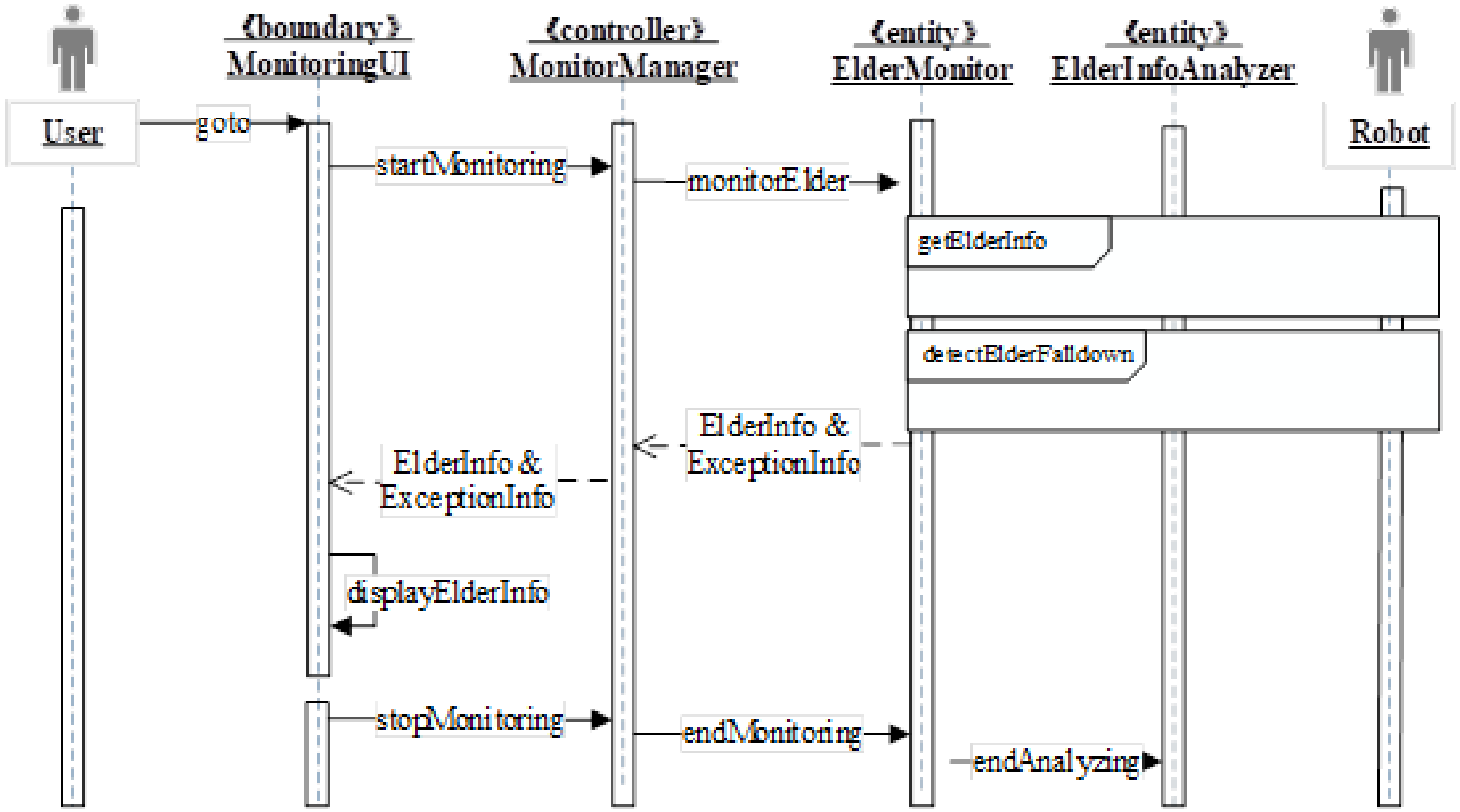
4. 软件测试计划及输出

✓测试计划制定及测试结果

1.1 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

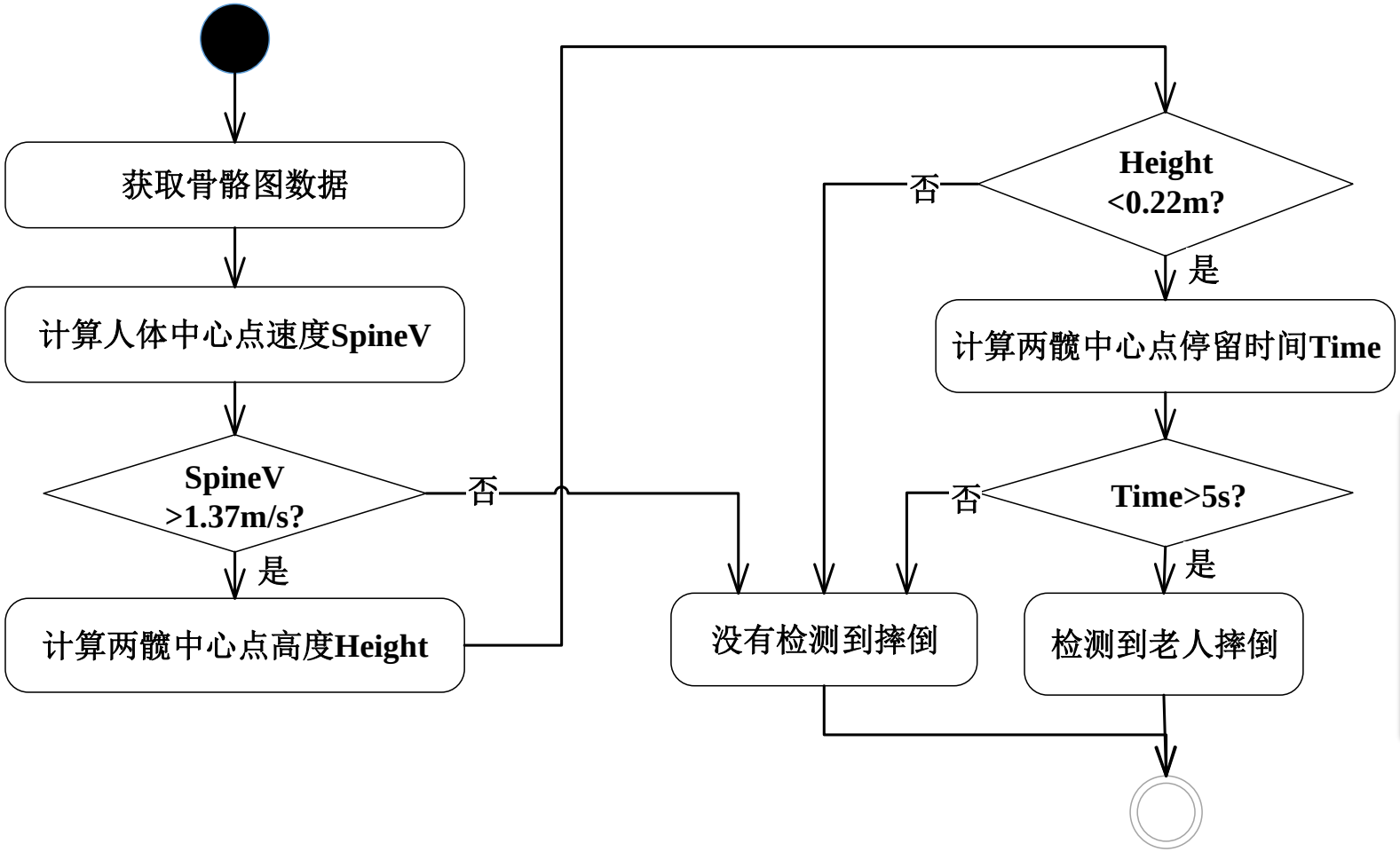


软件需求模型

软件需求缺陷：

- **没有正确描述用户需求？**
- **没有准确地描述需求？**
- **……**

示例：软件设计中的潜在问题



软件设计模型

软件设计存在缺陷

- 没有正确实现软件需求?
- 软件设计存在问题?

示例：程序代码中的潜在问题

程序代码中的缺陷
- 代码没有正确实现功能?
- 代码编写不正确?

```
public static int map (int index) {
    switch (index) {
    case 0:
    case10:
        return -1;
    case 2:
    case 20:
        break;
    default:
        return -2;
    }
    return 0;
}
```

```
public class Library {
        ….
   private static Set  s_books = new HashSet ();
   public static void addBook (String title, String author, int num_pages) {
      int num_chars = num_pages * 80 *50;
      s_books.add (new Book (title, author, new char [num_chars]));
}
```

需求和设计缺陷问题会带到代码中，编码时也会引入问题，导致代码存在缺陷

1.2 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
- ……

ARIANE火箭, 耗资70亿美元，1996年发射37秒后爆炸

```
sensor_get(vertical_veloc_sensor);
sensor_get(horizontal_veloc_sensor);
vertical_veloc_bias := integer(vertical_veloc_sensor);
horizontal_veloc_bias := integer(horizontal_veloc_sensor);
...
exception
  when numeric_error => calculate_vertical_veloc();
  when others => use_irs1();
end;
```

尽可能减少软件缺陷非常重要

软件缺陷的危害

❑发射失败的原因

程序中试图将64位浮点数转换成16位整数时的溢出错误。

```
vertical_veloc_bias := integer(vertical_veloc_sensor);
horizontal_veloc_bias := integer(horizontal_veloc_sensor);
```

如果单看其浮点转换程序，并没有任何问题。

问题在于他们复用了Ariane 4的部分软件需求文档，而软件工程师不知道Ariane 5的水平加速度比Ariane 4快5倍，因此要求额外3位整数存储。

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**
  - ✓复杂软件系统中，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

- 两种策略
  - ✓ 避免和减少缺陷
  - ✓ 发现和修复缺陷

如何有效、快速地发现软件缺陷?

方法：软件测试
(Software Testing)

软件测试方法可以帮助发现软件中潜藏的缺陷

1.3 何为软件测试

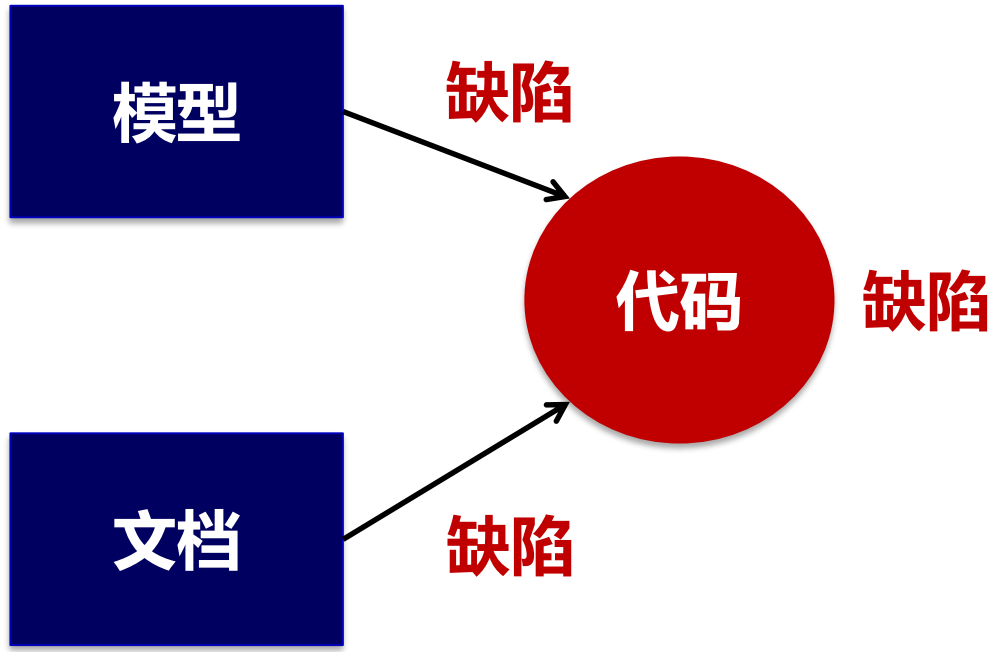
- 软件测试是通过运行程序以发现软件缺陷的过程。

- 软件测试的**目的**是为了**发现软件中的缺陷**,而不是为了证明程序中不存在缺陷。

- **注意点**
  - ✓软件测试是通过**运行程序**的方式来发现潜藏缺陷，这和**代码走查、静态分析**形成鲜明对比。
  - ✓它只负责发现缺陷，不负责修复和纠正缺陷

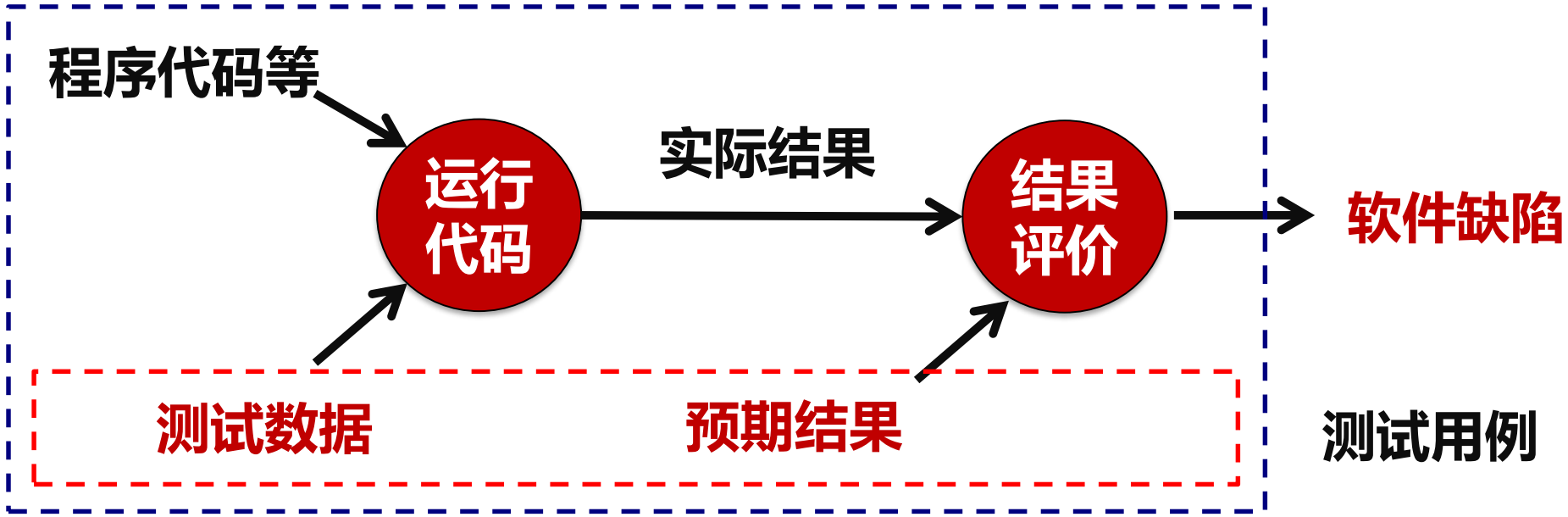
在程序代码中找出软件缺陷

- **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
  - ✓但最终会反映在程序代码上



1.4 软件测试的原理

- **前提认识：** 程序本质上是对数据的处理
- **方法思路：** 设计数据(测试用例) ➔ 运行测试用例 ➔ 判断运行结果(是否符合预期)



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
- 运行程序及测试用例
  - ✓判断运行的实际结果是否与预期结果相一致

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**：执行测试时必须满足的先决条件和环境设置,如用户权限
  - ✓**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

- 多数情况下，一个测试用例只能发现一个缺陷。因此，为了发现尽可能多的缺陷，我们需要多个测试用例

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；

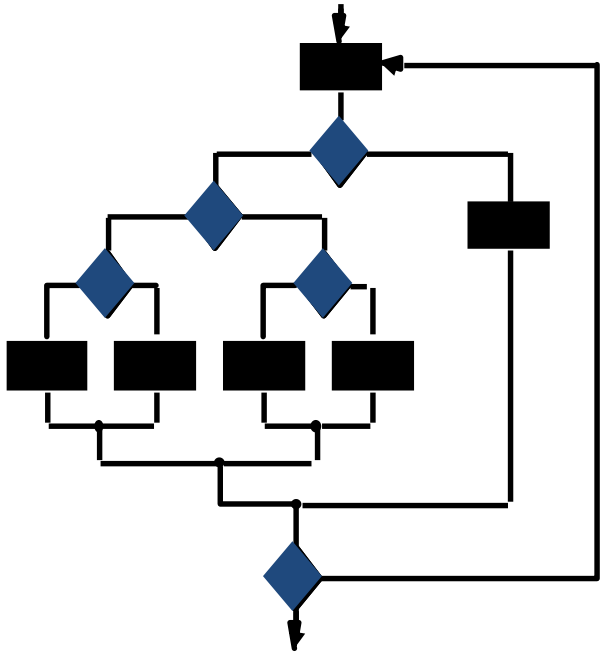
    其次输入“admin”账号和“1234”密码；

    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

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?
- 软件测试方法能够用于**证明软件无缺陷**吗?



There are 10^{14} possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

loop < 20 X

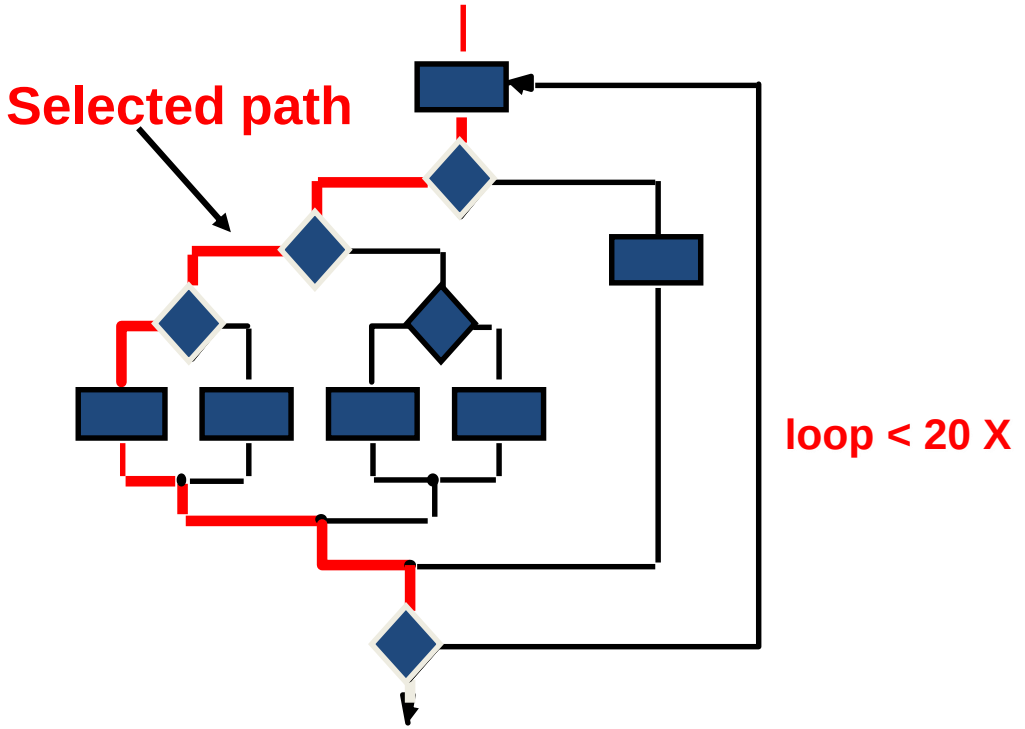
1.5 软件测试面临的挑战

❑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？如全面性、非冗余性

❑发现程序缺陷

- ✓C3：如何运行程序和用例来发现软件缺陷？
- ✓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

1.6 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
- 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%的错误在大约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自己编写的代码
- 尽量避免程序开发组织测试自己开发的程序

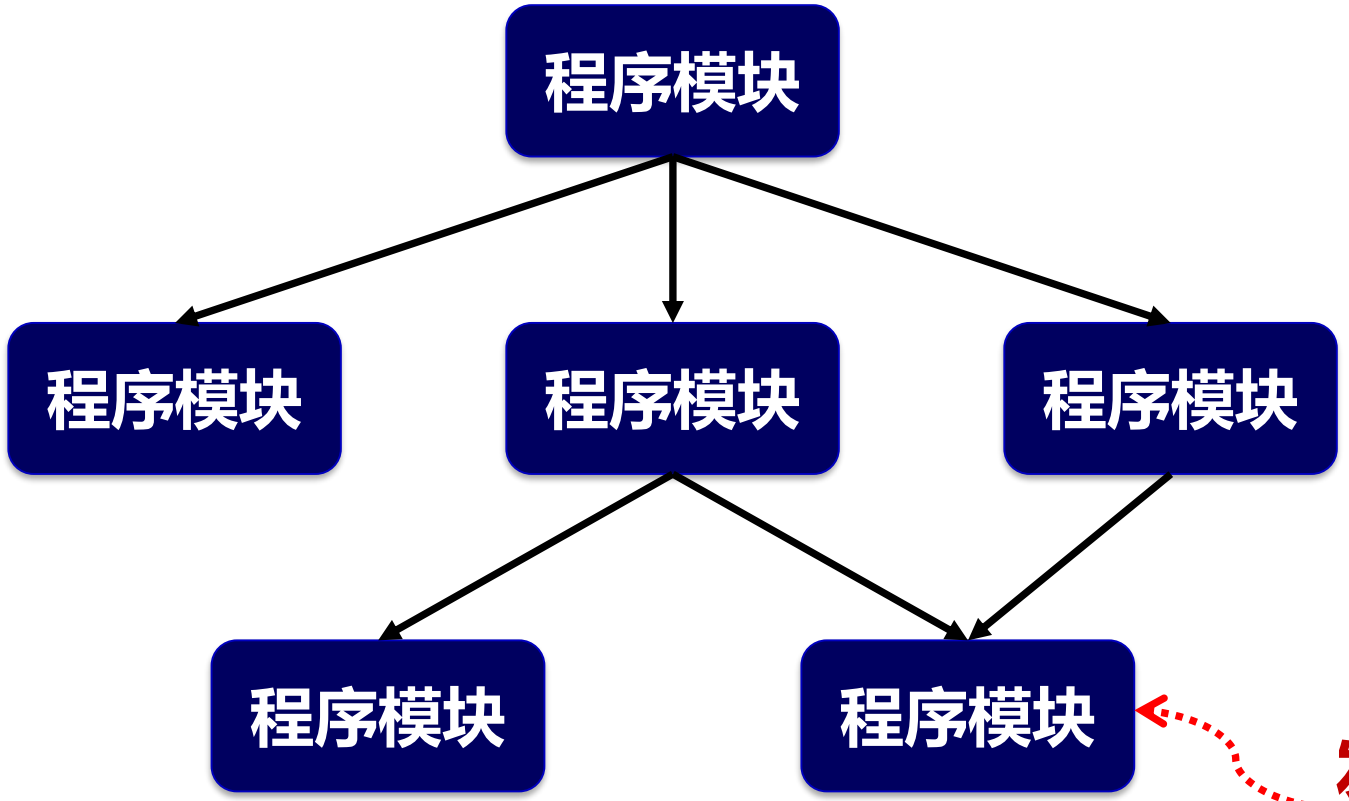
软件测试原则(2/2)

- 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
- 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来制定测试计划
- 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

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**
   - ✓软件测试活动及实施方法
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

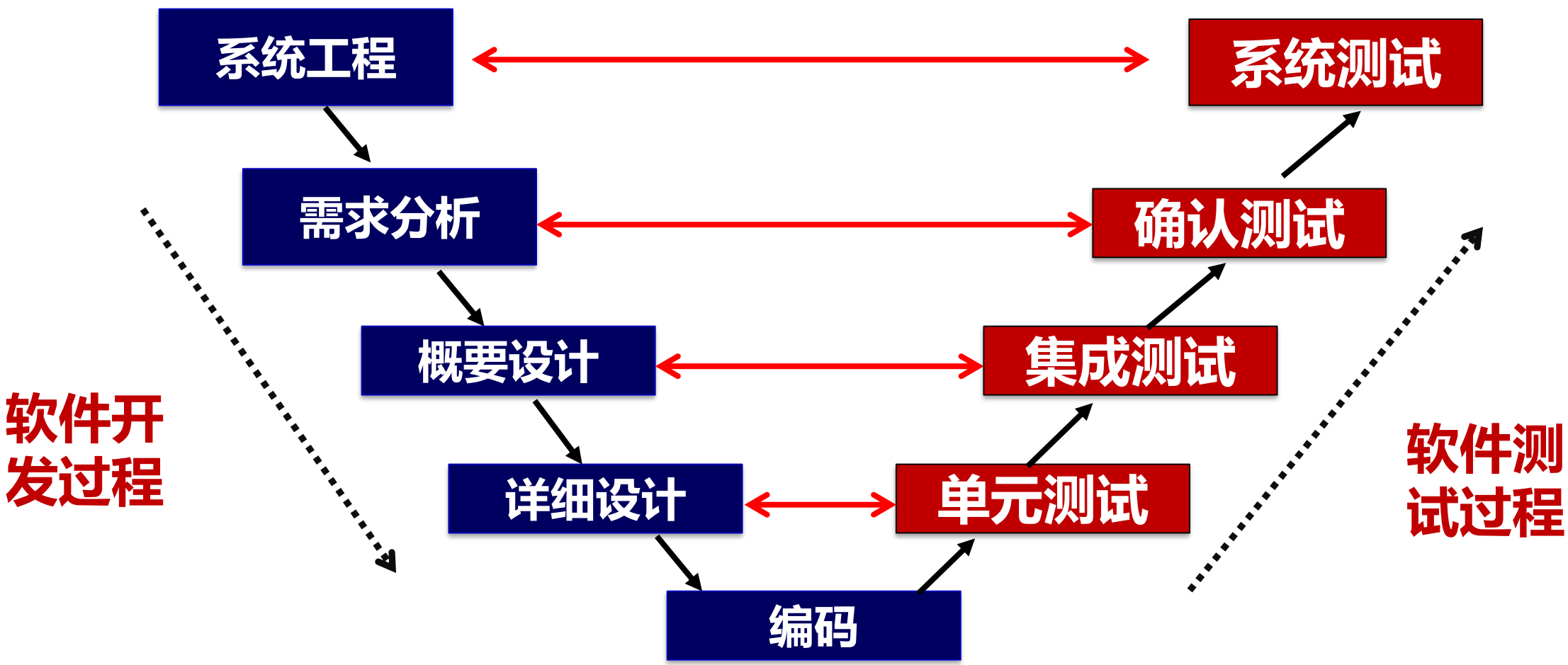
2.1 缺陷的潜在位置



- 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

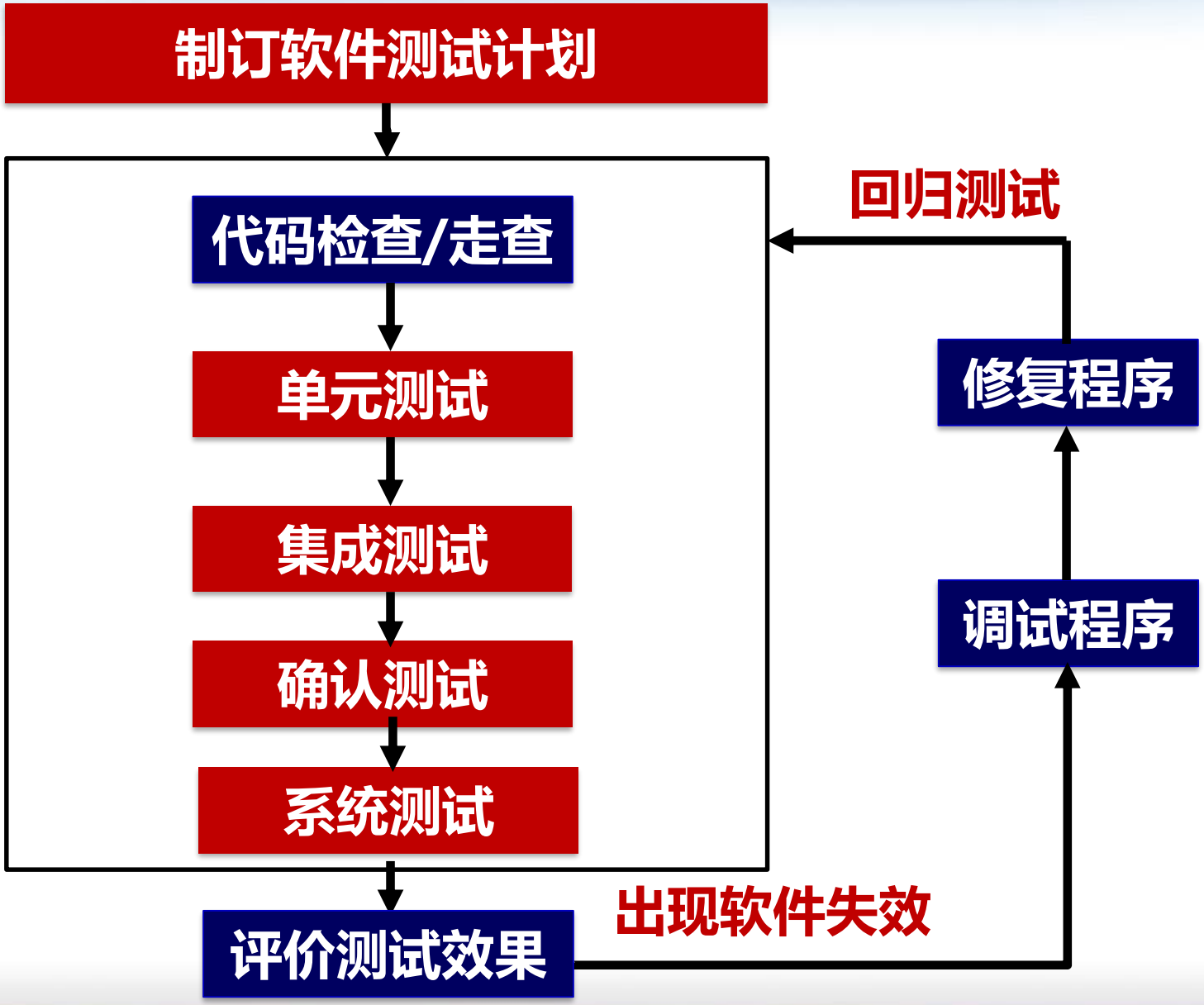
发现和消除基本模块单元中缺陷

2.2 软件测试活动



要对软件进行系统性、全方位的测试

2.4 软件测试过程

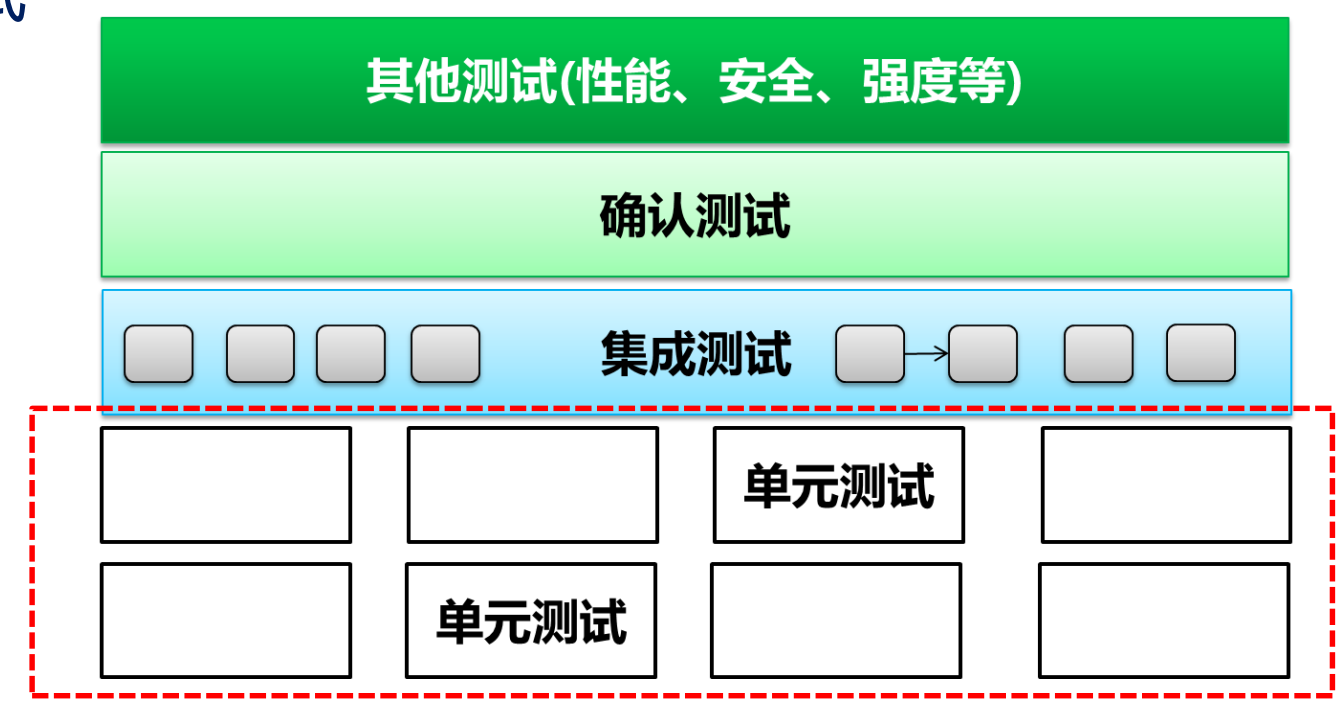

制订软件测试计划
代码检查/走查
单元测试
集成测试
确认测试
系统测试
评价测试效果
回归测试
修复程序
调试程序
出现软件失效

回归测试

- **修改程序可能会引入新的错误**
  - ✓原先“正常”的程序可能变得“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

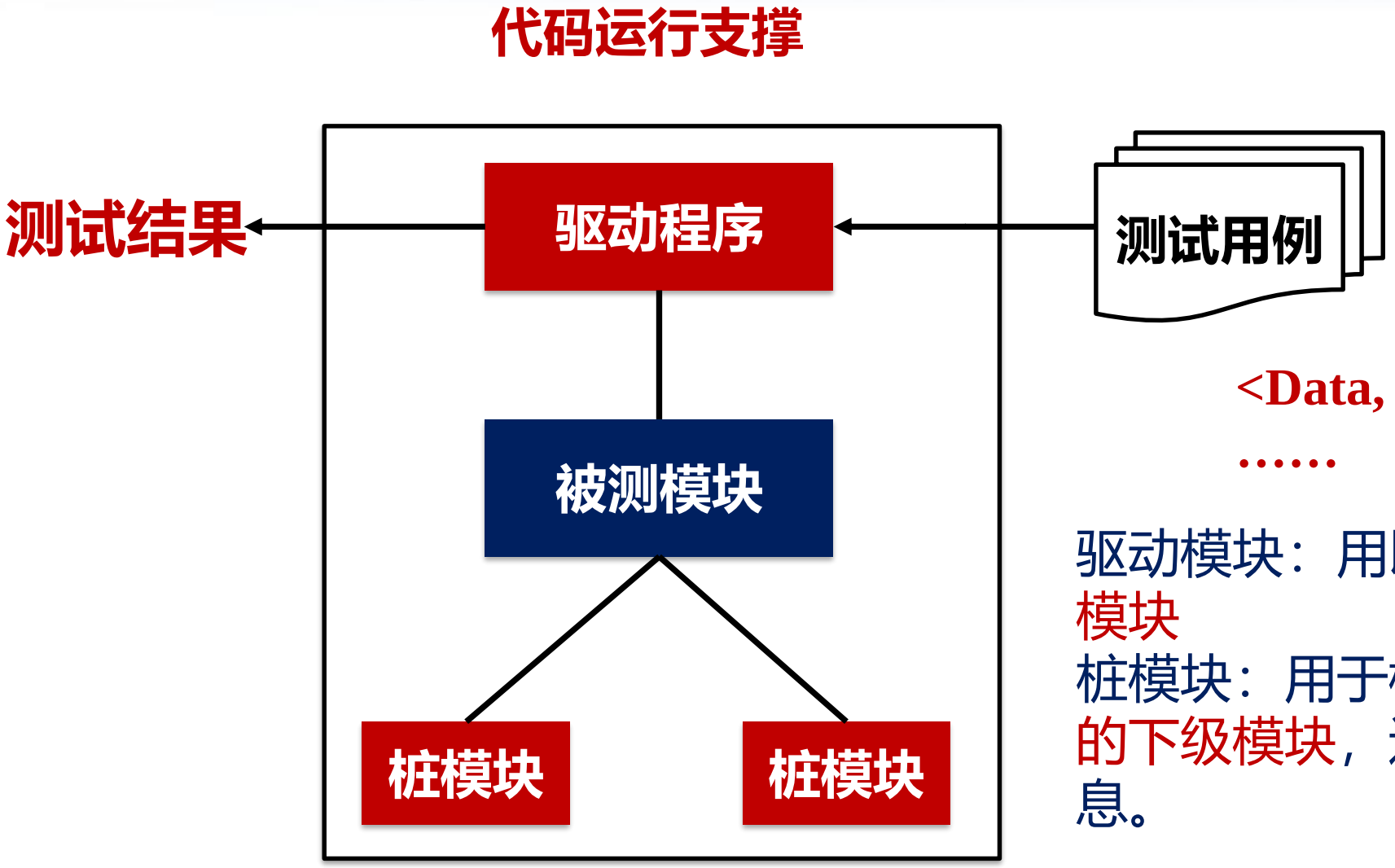
1. 单元测试

- **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



在软件详细设计阶段就需要设计单元测试用例及制定单元测试计划

单元测试的运行环境



<Data, Result>

……

驱动模块：用以模拟被测模块的上级模块

桩模块：用于模拟被测试模块所调用的下级模块，返回被测模块所需的信息。

桩模块的实现

- 实现桩的逻辑
  - ✓对于简单的情况，桩只需要返回固定的数据。
  - ✓对于更复杂的情况，桩可能需要根据输入参数动态生成不同的响应。

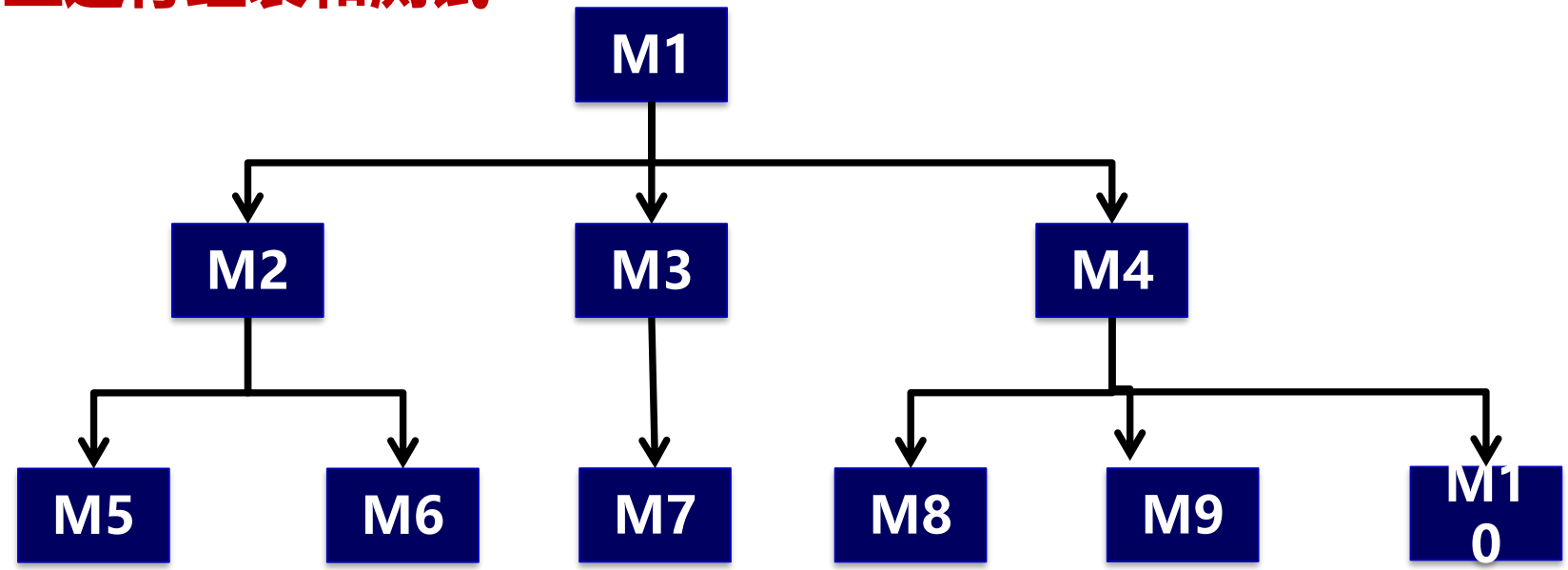
```
//这是待测模块
public void returnBook(int bookid){
      ……..
      int ret=penalty(…. ，  …. );     //超期罚款模块,ret为1表示已经缴纳罚款。
       ……….
 }
//这是编写的桩模块
 int stub_penalty (int uid, int bookid) {  //无须真的完成罚款计算,返回一个预期的结果即可
       if uid=111  then
           return  1;
       else
          return  0;
}
```

2. 集成测试

- **为什么进行集成测试**
  - ✓通过单元测试的模块集成在一起时，仍然可能出错，如调用顺序不当、参数传入或传出错误等
- **集成测试的重点**
  - ✓模块间的接口是否按预期工作，包括数据格式、协议和交互流程等
  - ✓集成后的功能是否达到预期
- **测试技术**
  - ✓采用**黑盒测试技术**
- **集成测试的依据**
  - ✓软件体系结构

集成测试方法

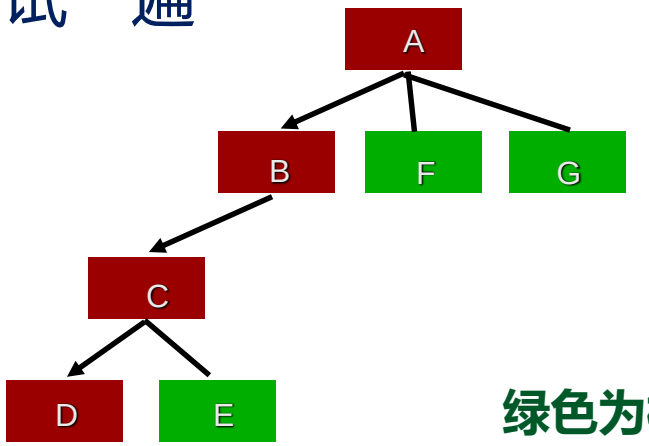
- 集成测试要遵循**循序渐进**的思路
- 自顶向下集成
  - ✓ **深度优先**或者**广度优先**的策略把各个模块进行组装和测试
- 自底向上集成
  - ✓ **自底向上进行组装和测试**



在概要设计阶段就需要设计集成测试用例及制定集成测试计划

自顶向下集成测试过程

- 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

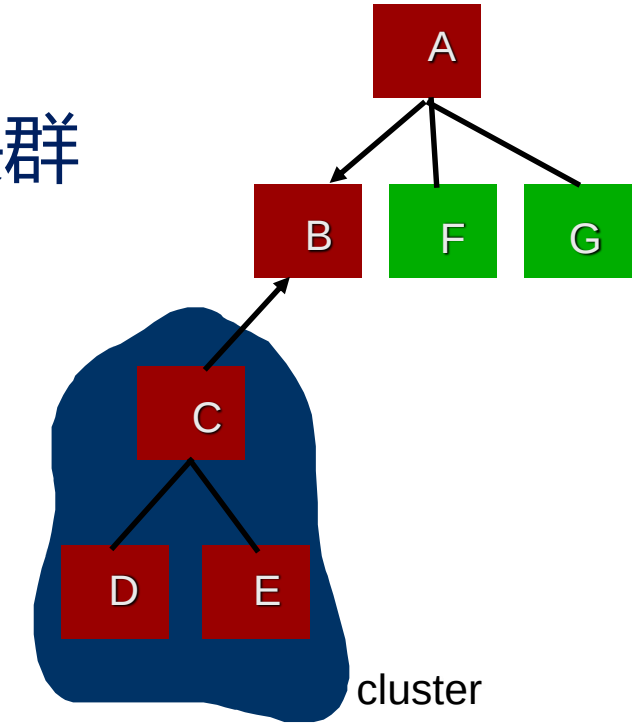


绿色为桩模块

- 循环执行上述步骤（2），直至整个程序结构集成完毕
- 为避免引入新的错误，须不断进行回归测试

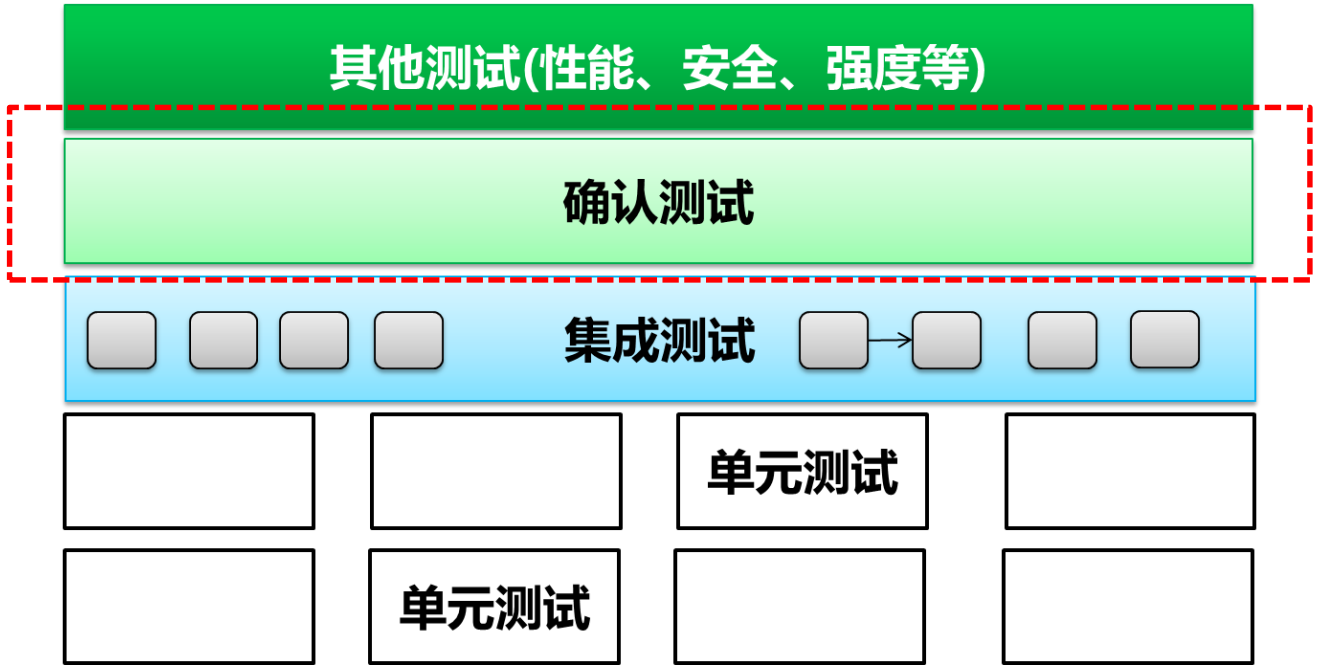
自底向上集成测试过程

- 先把低层模块集成起来，形成实现某模块群
- 开发一个测试驱动模块，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
  - ✓用实际的高层模块代替测试驱动，形成新的模块群
- 重复执行步骤（2）



2.4.3 确认测试

- **测试对象**
  - ✓对组装完成的整个软件系统的**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**



在需求分析阶段就需要设计确认测试用例及制定确认测试计划

α测试和β测试

❑α测试

- ✓**软件开发公司组织内部人员**模拟各类用户行为对即将面市的软件产品（称为α版本、内部测试版）进行测试
- ✓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α测试并进行修改后的软件产品称为**β版本**（也称外部测试版）

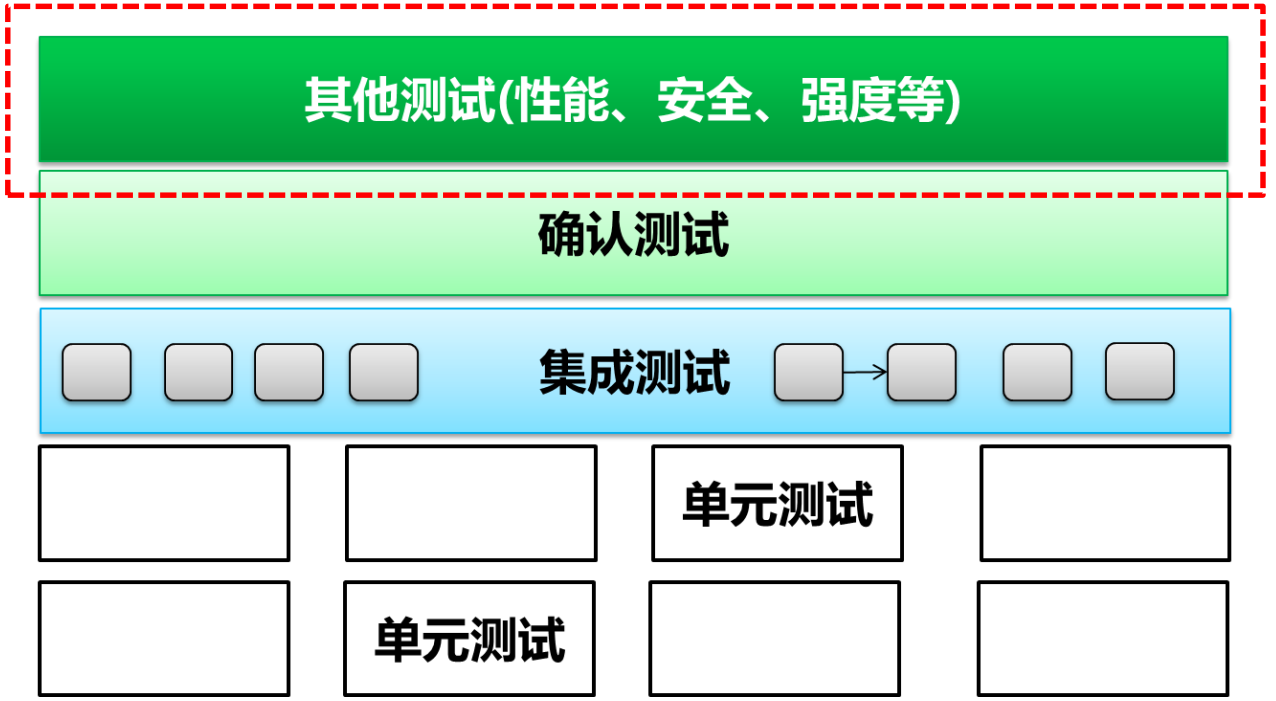
❑β测试

- ✓软件开发公司组织**典型用户**实际使用β版本，或将β版本免费赠送给典型用户，并要求用户报告异常情况、提出批评意见
- ✓β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**

2.4.4 系统测试

- 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
- ……

系统测试就是将软件系统与其它系统进行集成，以发现由硬件、软件、用户所构成的系统整体是否存在缺陷



在需求分析阶段就需要设计非功能测试用例及制定非功能测试计划

性能测试用例示例

用例标识：	12306-NSF-1
测试项：	12306软件的性能能否支持同时有10000个用户登陆使用。并且平均响应时间少于1秒。
测试输入：	由模拟软件模拟10000个用户使用系统。
前提条件：	12306 APP和服务器软件已经完成并且安装成功。
环境要求：	使用n台服务器，配置为××××； 使用20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。
测试步骤：	(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50个用户登陆请求，并模拟查询车次的过程； (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。
预期输出：	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秒。

强度测试(压力测试)

- **强度测试可被视为性能测试一部分**
  - ✓性能测试是测试系统在正常使用时是否能够达到性能指标
  - ✓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**
  - ✓12306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的最大强度**
- **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

强度测试(压力测试)

表 7.19 压力测标准

并发用户数	压测时长	90%用户响应时间	平均响应时间	事务成功率	每秒处理事务	CPU 占用率	内存使用率
5000	15min	<3s	<3s	>99%		<75%	<75%
10000	15min	<4s	<4s	>99%		<75%	<75%
15000	15min	<5s	<5s	>99%		<75%	<76%

表 7.20 压力测结果

场景	并发用户数	压测时长	90%用户响应时间	平均响应时间	事务成功率	每秒处理事务	成功事务数	失败事务数	脚本运行错误数
用户注册	100	15min	4.625	2.391	99.28%	9.371	14358	103	205
	200	15min	6.039	3.753	98.19%	9.125	18770	345	523
	500	15min	12.748	6.452	91.18%	10.621	36134	3493	5261

安全性测试

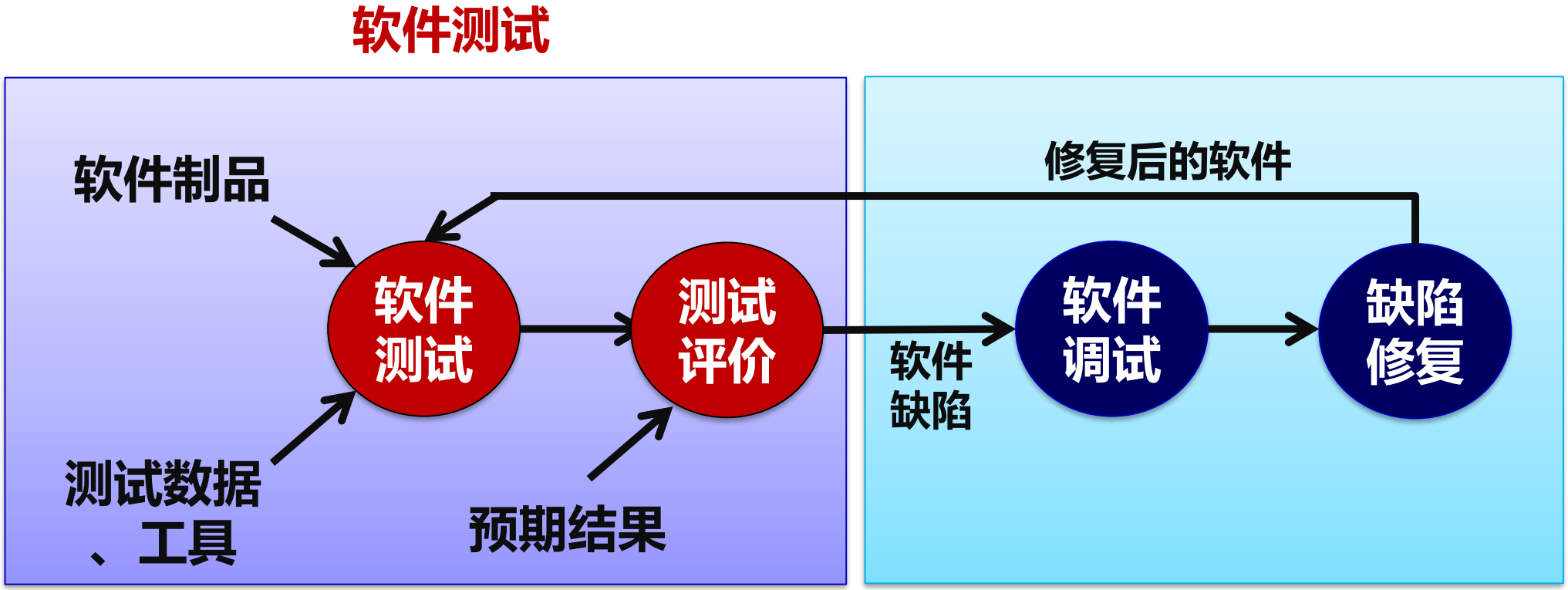
- **设计测试用例模拟攻击，以暴露软件安全漏洞的过程**
  - ✓如设法破坏数据库管理系统的数据安全机制、突破软件系统的用户访问控制等
- **安全性测试方法**
  - ✓ 功能验证：对系统提供的安全控制措施进行验证，如加解密服务、认证服务以及授权服务。
  - ✓漏洞扫描：利用漏洞扫描工具，对主机、网络及功能模块进行扫描，发现系统中存在的安全弱点及漏洞。
  - ✓模拟攻击试验：通过模拟典型的安全攻击来验证系统的防护能力，如模拟SQL注入攻击。
  - ✓侦听测试：如在系统通信过程中，对数据进行截取分析，从而发现系统在防数据窃取方面的防护能力。

兼容性测试

- 兼容性测试是确保应用程序在不同环境、设备和系统中正常运行的一种测试类型。
- 如浏览器、操作系统、数据库、分辨率等兼容性测试

测试方面	测试内容和细节	注意事项
版本兼容性测试	在不同版本的微信应用上测试健康码系统运行情况	确保核心功能正常工作
界面和布局测试	验证健康码页面在不同分辨率和尺寸的微信应用内显示是否正确	适配不同手机屏幕

2.5 软件测试的后续工作



测试是为了发现错误，定位缺陷和纠正错误不属于软件测试的工作范畴

测试、调试和排错

- 目的
  - ✓测试发现缺陷
  - ✓调试定位缺陷
  - ✓排错纠正错误

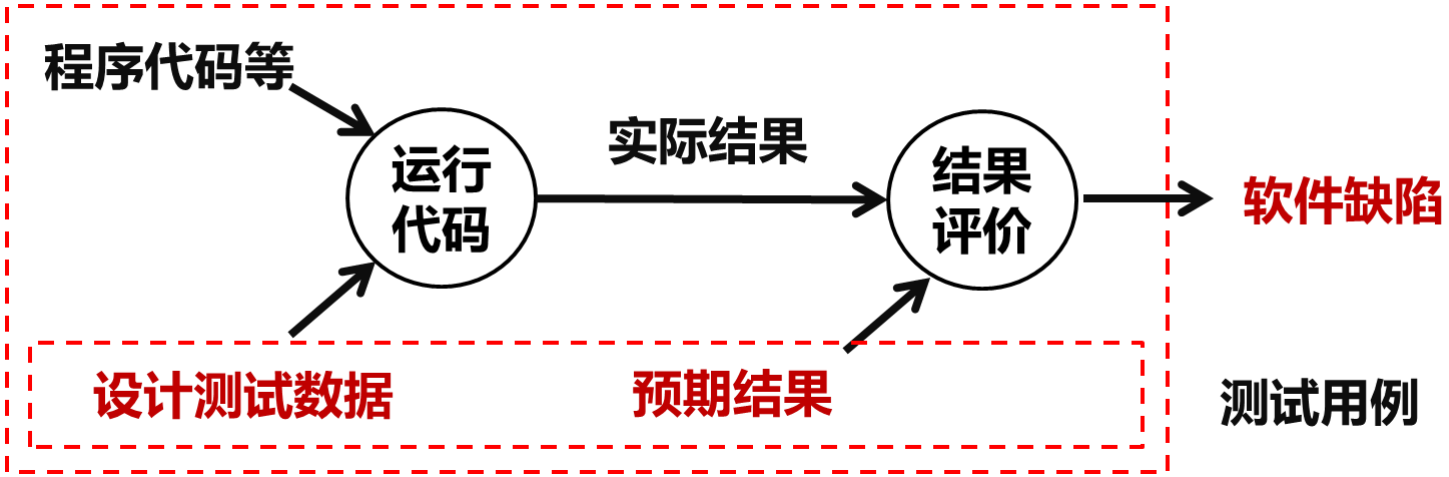
- 测试由独立的测试小组进行
- 调试和排错由开发人员完成

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
   - ✓面向对象软件测试
4. **软件测试计划及输出**
   - ✓测试计划制定及测试结果

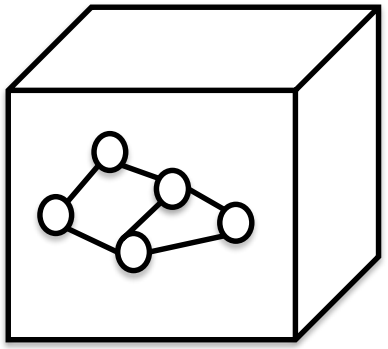
思考和讨论

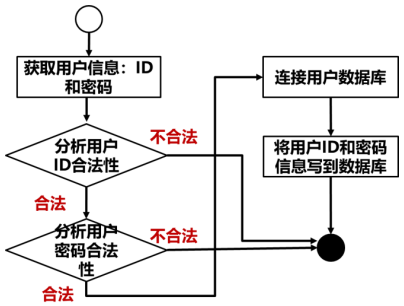
- 如何来**设计测试用例**?
- 要设计**多少个测试用例**?
- 如何才能做到**充分的软件测试**? 即通过设计足够多的测试用例来发现尽可能多的软件缺陷



3.1 软件测试技术

❑如何设计和运行测试用例





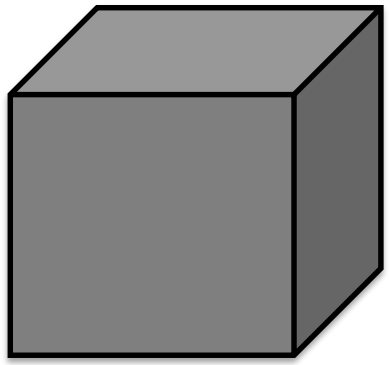
程序内部执行流程

❑白盒测试技术

✓基于程序内部的执行流程来设计测试用例

❑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



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

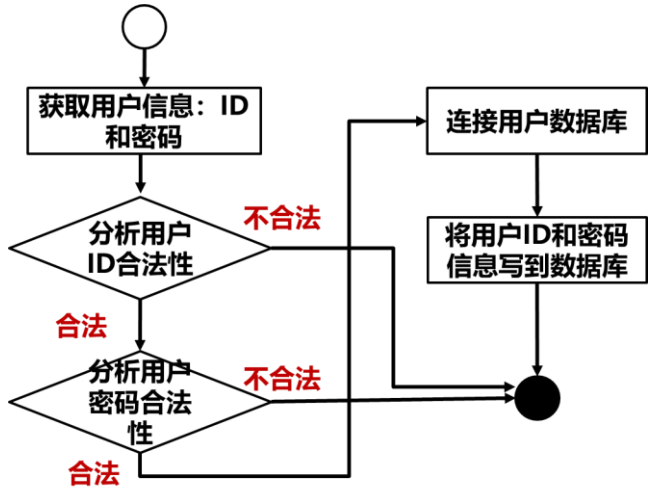
3.2 白盒测试

❑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

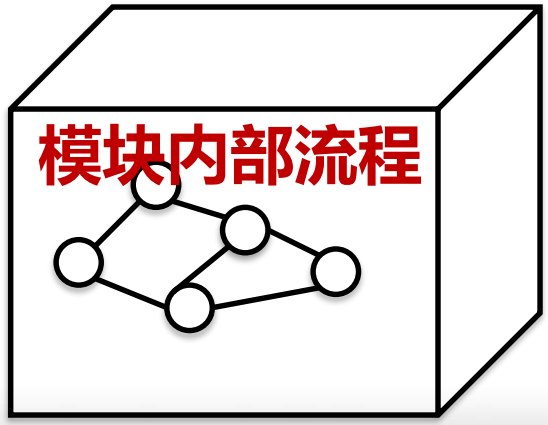
❑发现程序单元缺陷

- ✓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



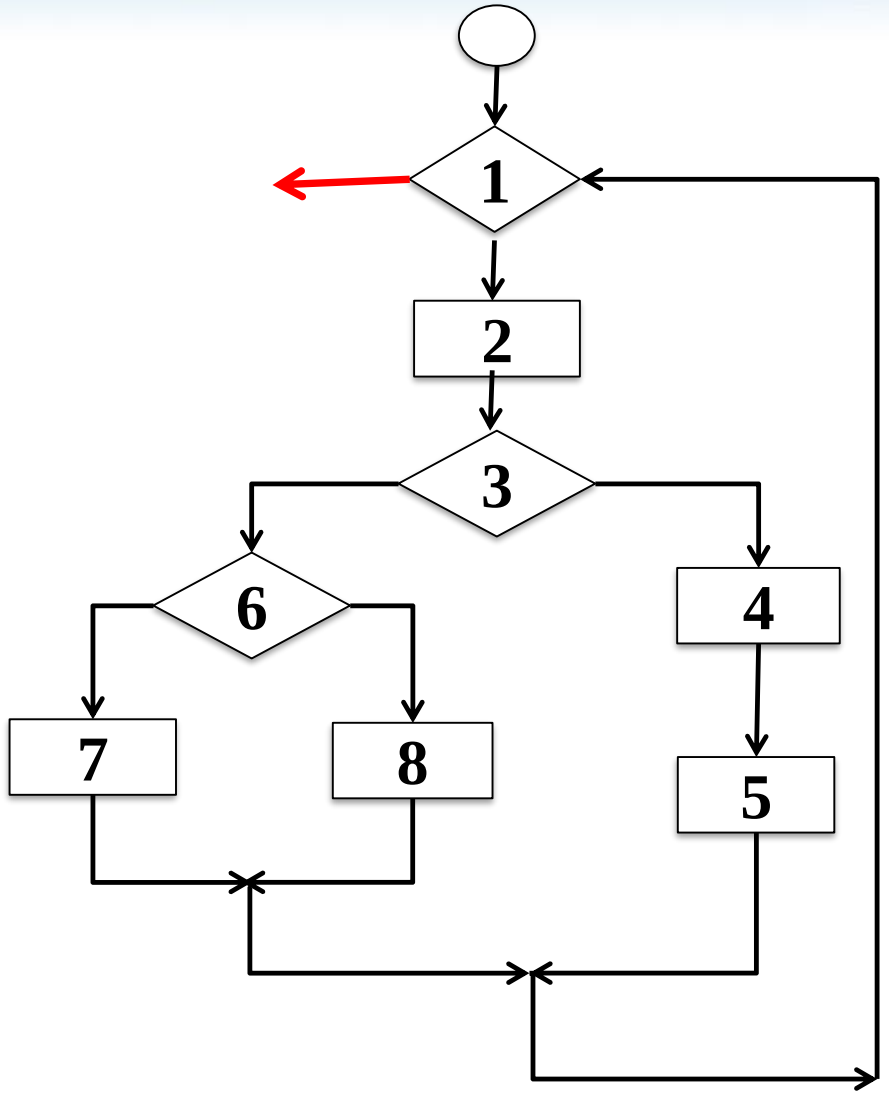
❑特点

- ✓必须了解程序的**内部工作流程**才能设计测试用例



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**

程序的路径可能有无穷条，但是基本路径只有有限条

基本路径测试的思想

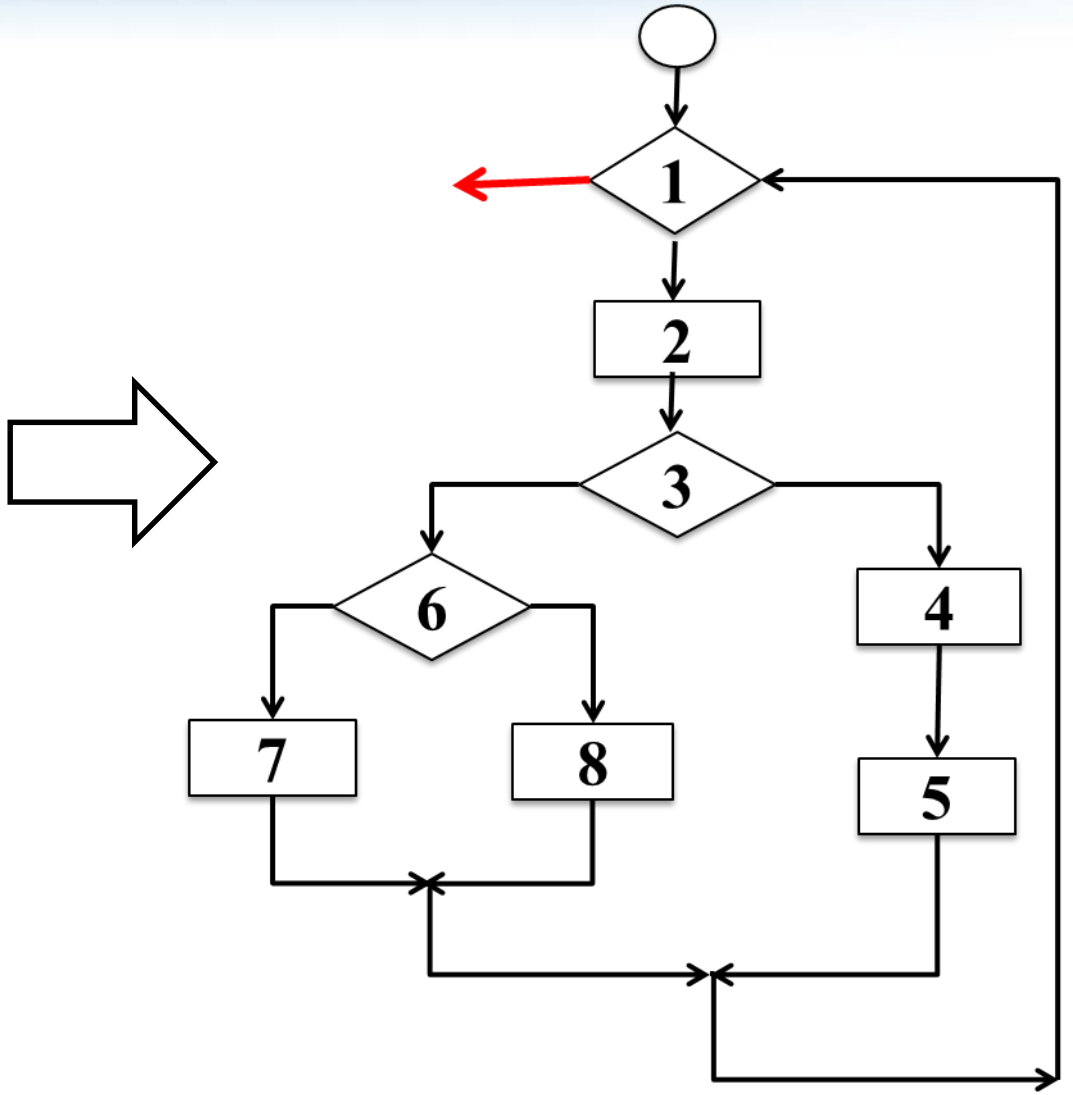
- **思想**
  - ✓路径 → 基本路径 → **基本路径测试**
- **基本路径**
  - ✓至少**引入一个新语句或者新判断**的程序通道
- **前提**
  - ✓软件模块的逻辑结构（流程图）

注意：基本路径不等于完全路径

- **如何设计测试用例实现基本路径覆盖**
  - ✓**程序→ 流程图 → 流图** → 哪些基本路径

步骤1：根据程序逻辑画出流程图

```
void Func(int nPosX, int nPosY) {
    1 while (nPosX > 0) {
    2       int nSum = nPosX + nPosY;
    3       if (nSum > 1) {
    4           nPosX-=1;
    5           nPosY-=1;
            }
            else {
    6           if (nSum < -1)
    7               nPosX -= 2;
    8           else
                    nPosX-=4;
            }
    }       // end of while
}
```



软件模块设计细节

注意流程图的正确画法！

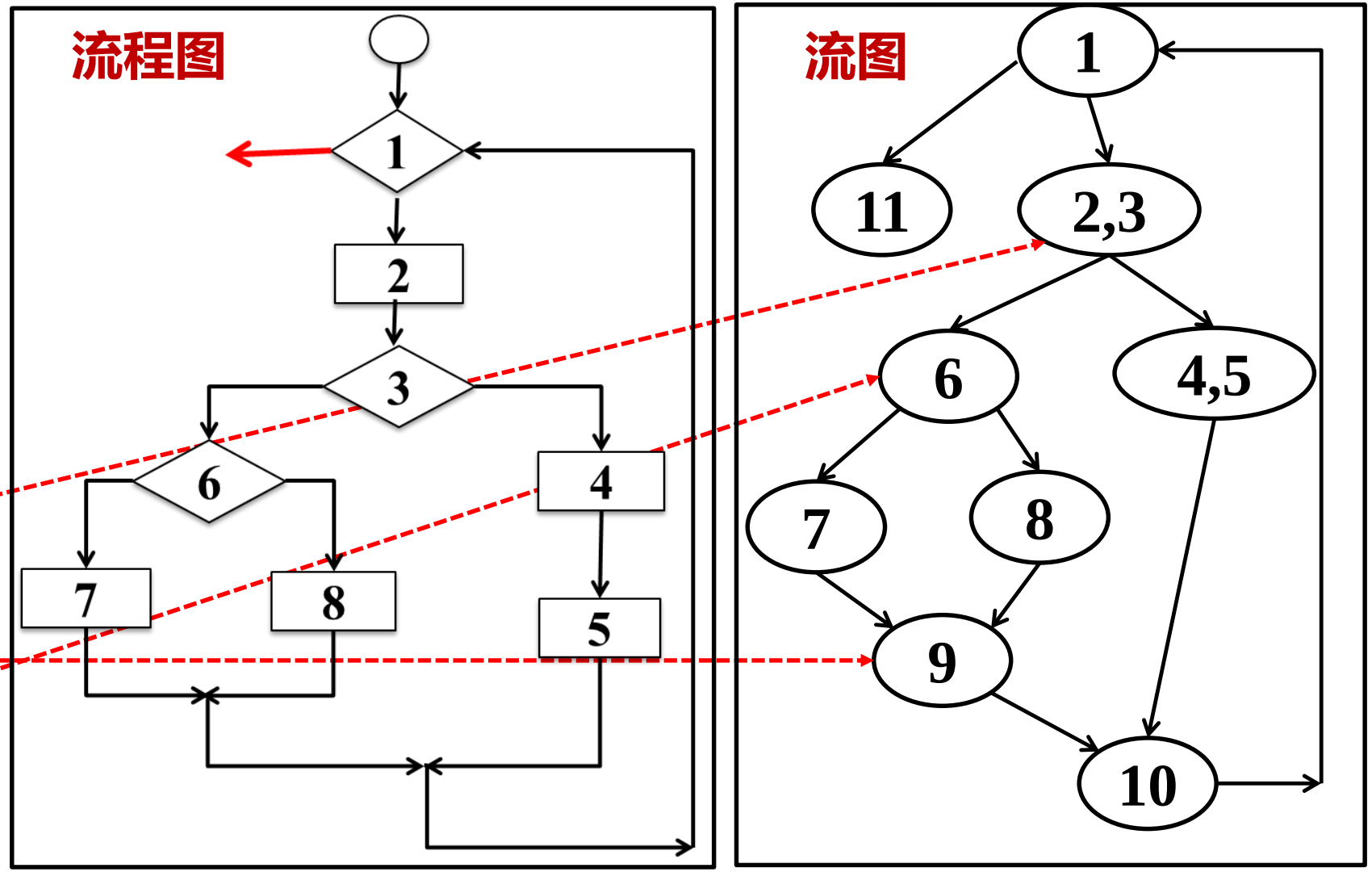
步骤2: 将流程图转换为流图 (1/2)

❑何为流图

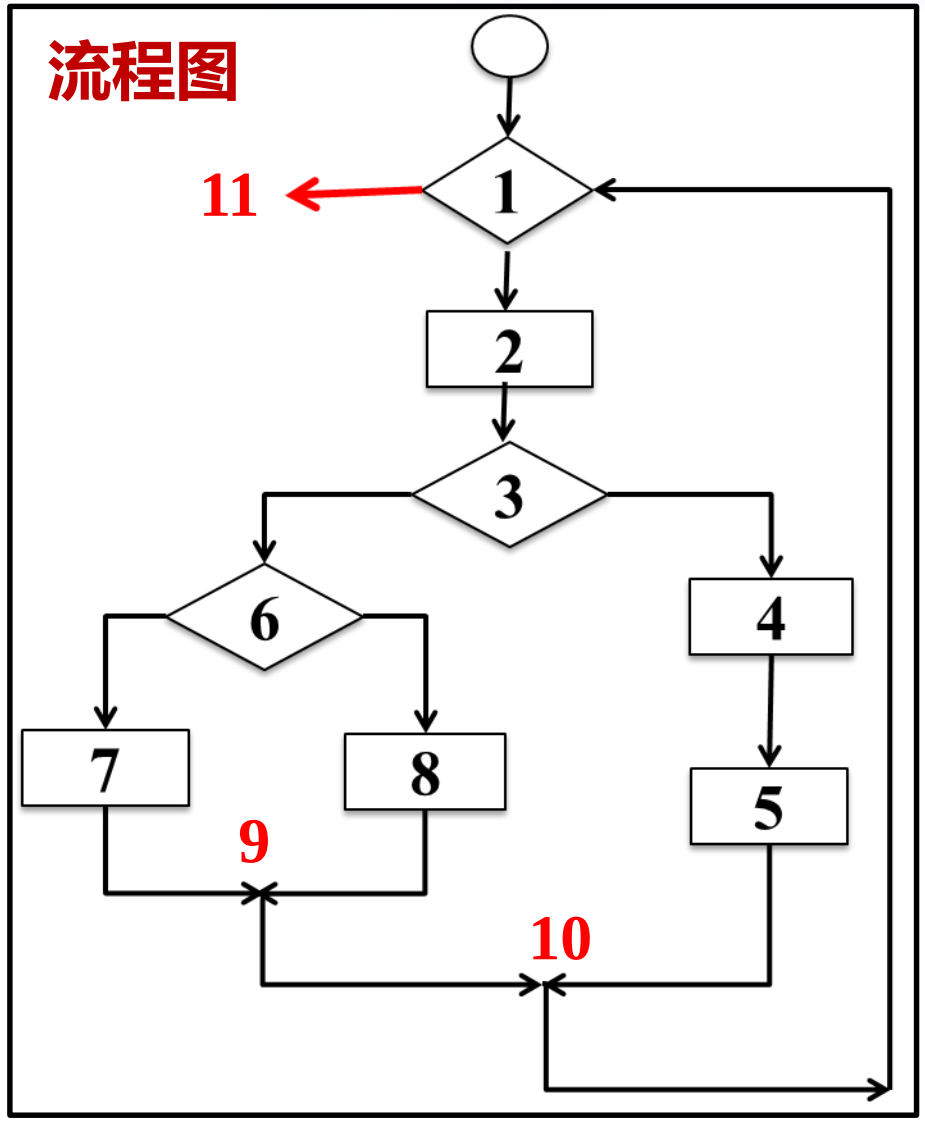
- ✓流图刻画程序控制结构但不涉及程序过程性细节

❑节点形式

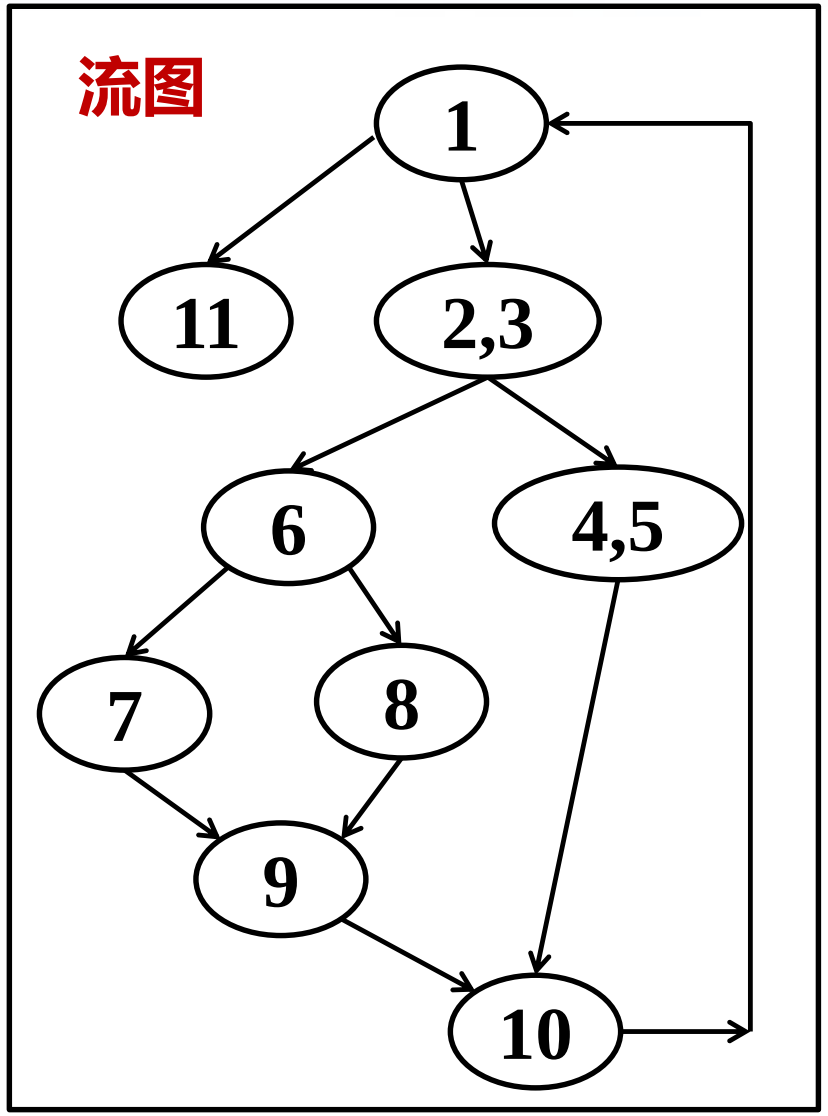
- ✓过程块
- ✓结合点
- ✓判定点



步骤2: 将流程图转换为流图 (2/2)



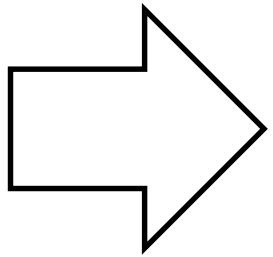
判定点
过程块
结合点

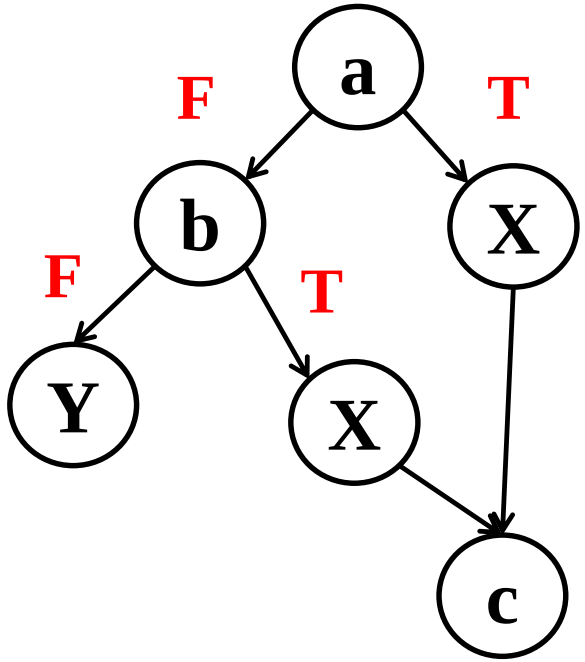


流图中的判定点不应含复合条件

- 否则按下列方式增加判定点

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





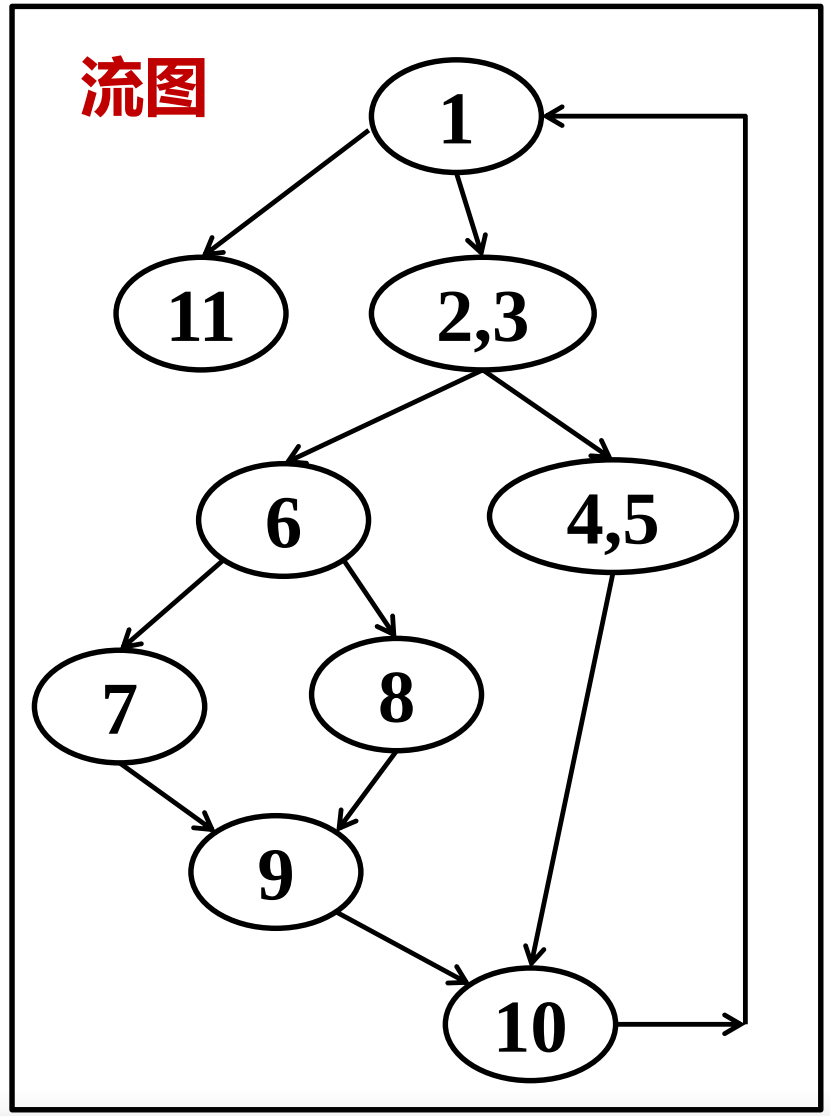
步骤3: 确定基本路径集合

❑基本路径的数量

- ✓流图**Cyclomatic复杂度**
- ✓V(G) = E(dges) – N(odes) + 2
  = 判定节点数+1
- ✓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: 根据基本路径设计测试用例

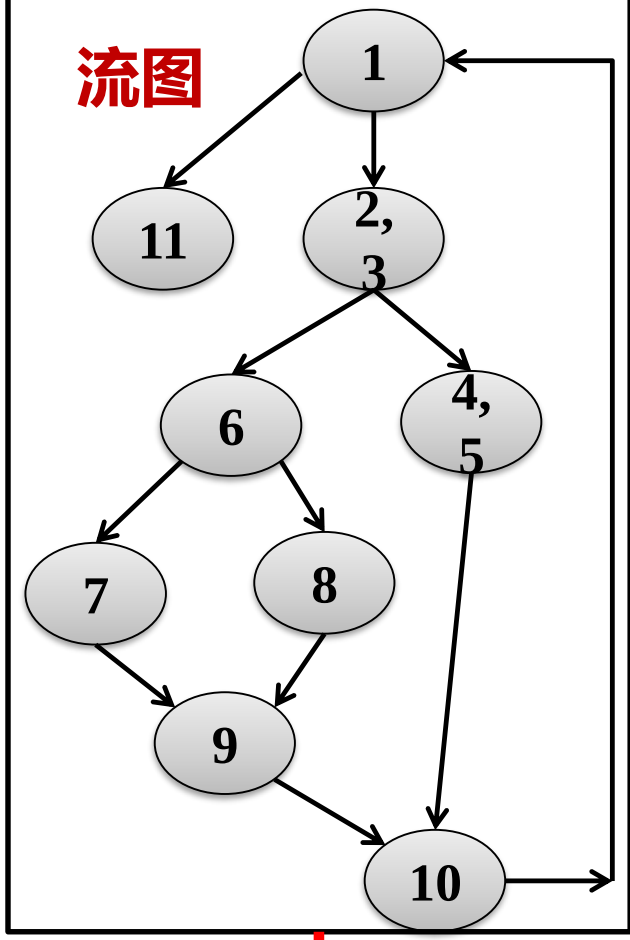
基本路径

- 1-11
  - ✓ nPosX = -1, nPosY取任意值
- 1 - 2, 3 - 4, 5 - 10 - 1 - 11
  - ✓ nPosX =1, nPosY=1
- 1- 2, 3- 6-7 - 9 - 10-1– 11
  - ✓ nPosX =1, nPosY= -1
- 1- 2, 3 - 6 - 8 - 9 - 10-1-11
  - ✓ nPosX =1, nPosY= -3

设计信息

```
void Func(int nPosX, int nPosY) {
    while (nPosX > 0) {
        int nSum = nPosX + nPosY;
        if (nSum > 1) {
            nPosX=-1;
            nPosY=-1;
        }
        else {
            if (nSum < -1) nPosX -= 2;
            else nPosX=-4;
        }
    }   // end of while
}
```

流图



描述每一个测试用例

用例标识：	对该测试用例赋予一个**唯一标识**
用例开发者：	**谁**编写的本用例
用例开发日期：	编写用例的**日期**
测试项：	描述将被测试的具体特征、代码模块等对象
测试输入：	测试时为程序提供的**输入数据**
前提条件：	执行测试时系统应处于的**状态或要满足的条件**等
环境要求：	执行测试所需的**软硬件环境、测试工具、人员**等
测试步骤：	(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) ……
预期输出：	希望程序运行得到的结果
用例间的依赖性：	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

3.3 黑盒测试

❑思想

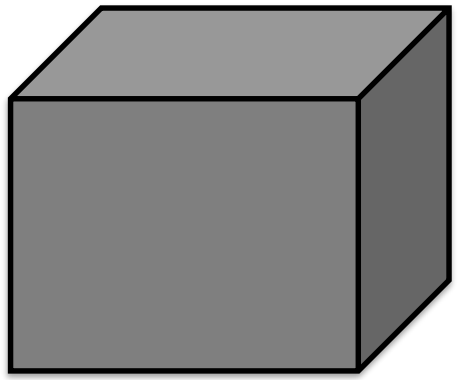
- ✓根据已知的**程序功能和性能**(而非内部细节)，设计测试用例并通过测试检验程序的每个功能和性能是否正常

❑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



只清楚模块的外在功能，不清楚其内部的控制逻辑和算法

黑盒测试发现的缺陷类型

- 测试软件系统是否**满足功能要求**
  - ✓不正确或遗漏的功能
  - ✓模块接口的错误
  - ✓数据结构或外部数据库访问错误
  - ✓性能的错误
  - ✓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

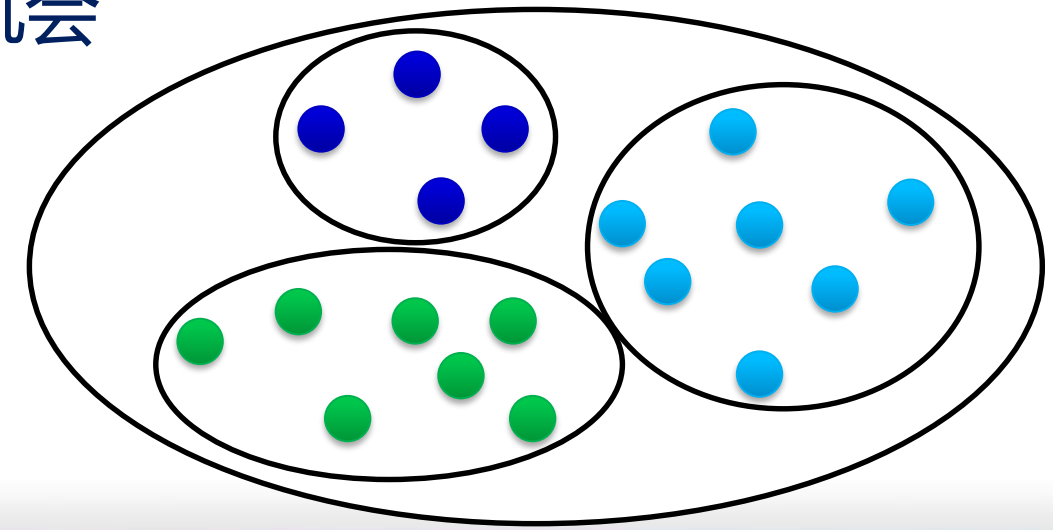
❑思想

- ✓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

- **如果输入条件为一范围**
  - ✓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
- **如果输入条件为一值**
  - ✓划分为三个等价类
  - ✓(1) 有效，(2) 大于，(3) 小于
- **如果输入条件为集合**
  - ✓划分二个等价类
  - ✓(1) 有效(在集合内)，(2) 无效(在集合外)
- **如果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 = 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, +#)

设计的9个测试用例

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=**

黑盒测试-边界值分析法

- **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

软件更容易在输入的边界上出现错误

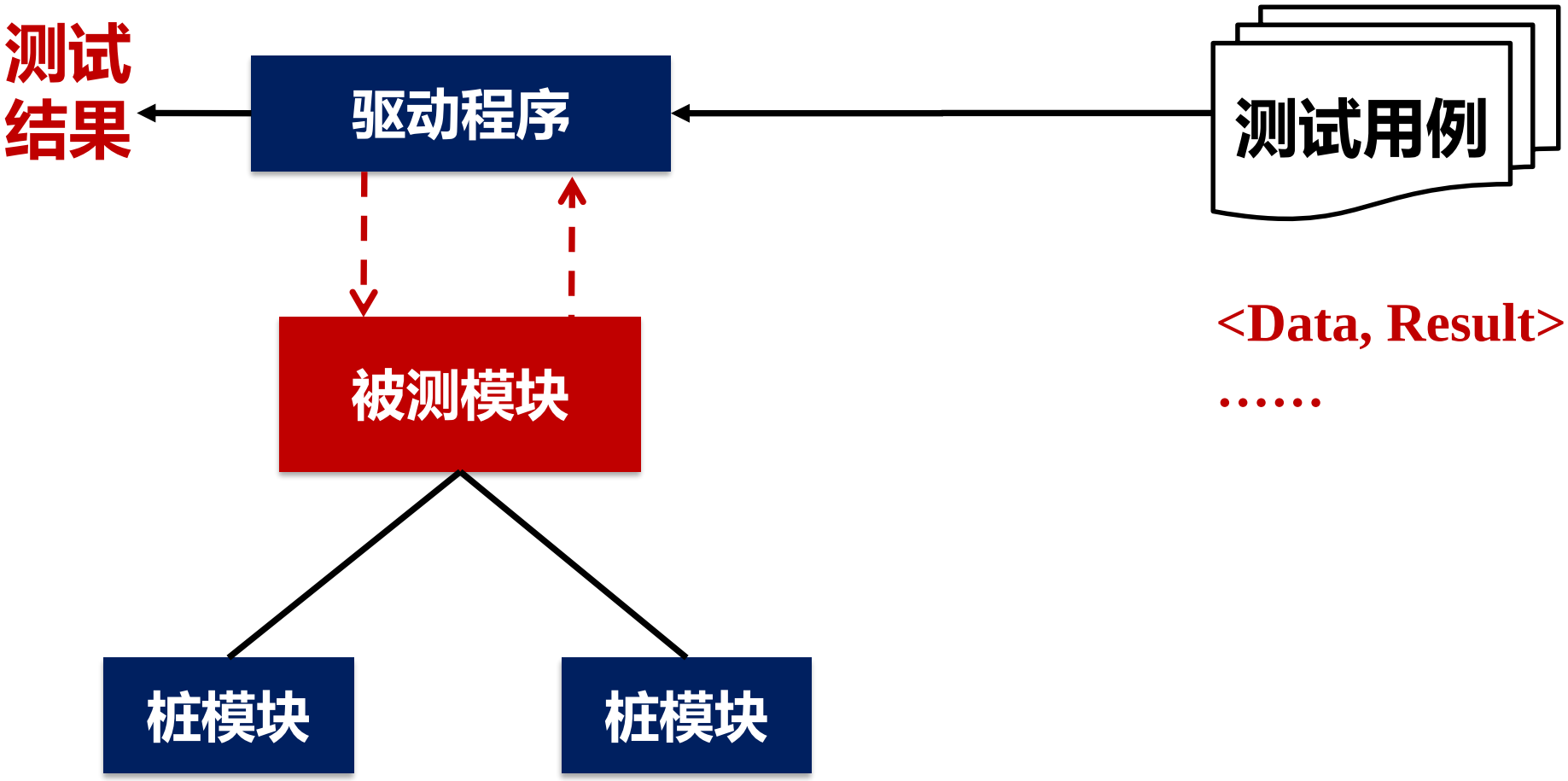
边界值分析法示例

- z = 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 ……

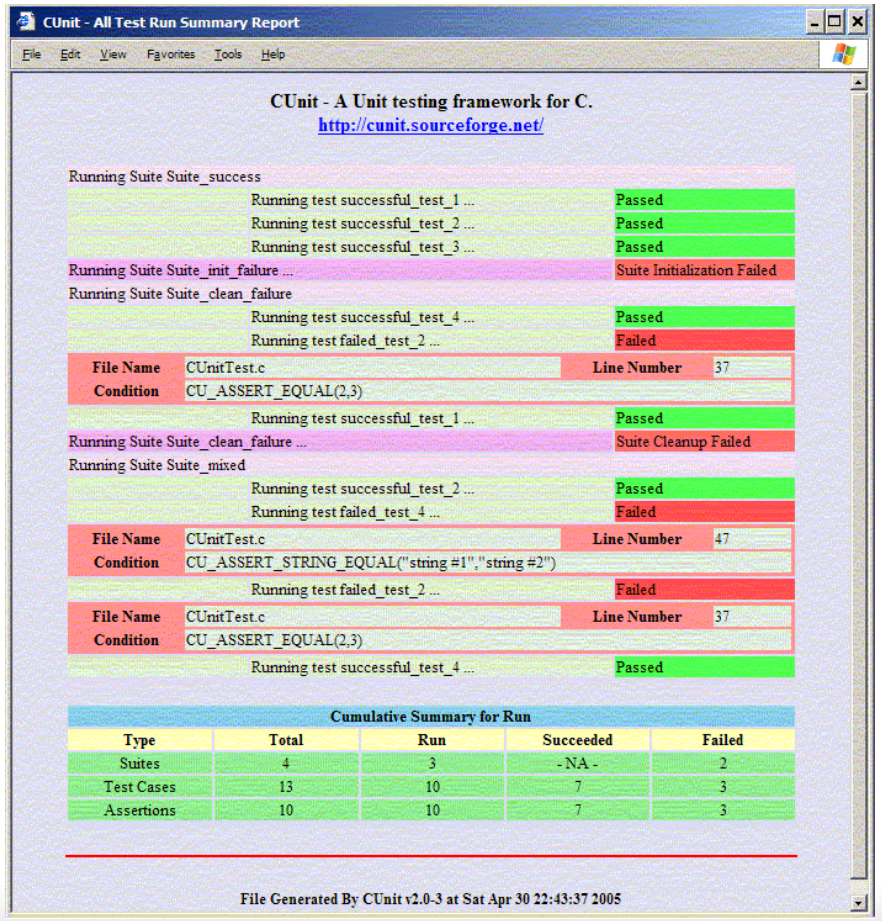
3.4 单元测试执行



单元测试的运行环境

单元测试工具

- C/C++语言 - C++Test、Cunit、CppTest
- .Net开发 - Nunit
- Java语言 –Junit
- Python语言 - PyUnit



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

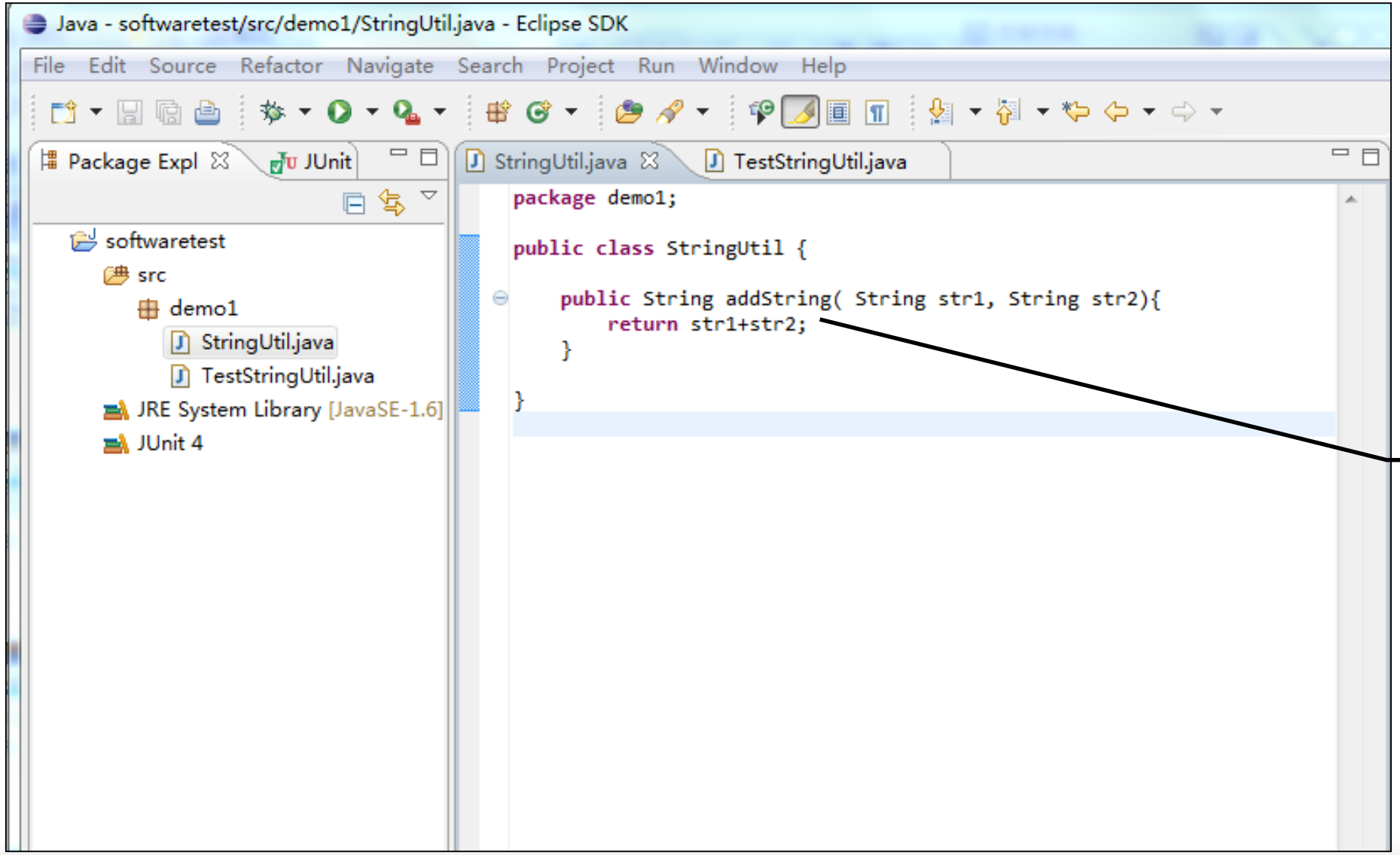
http://junit.org/

下载并安装

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

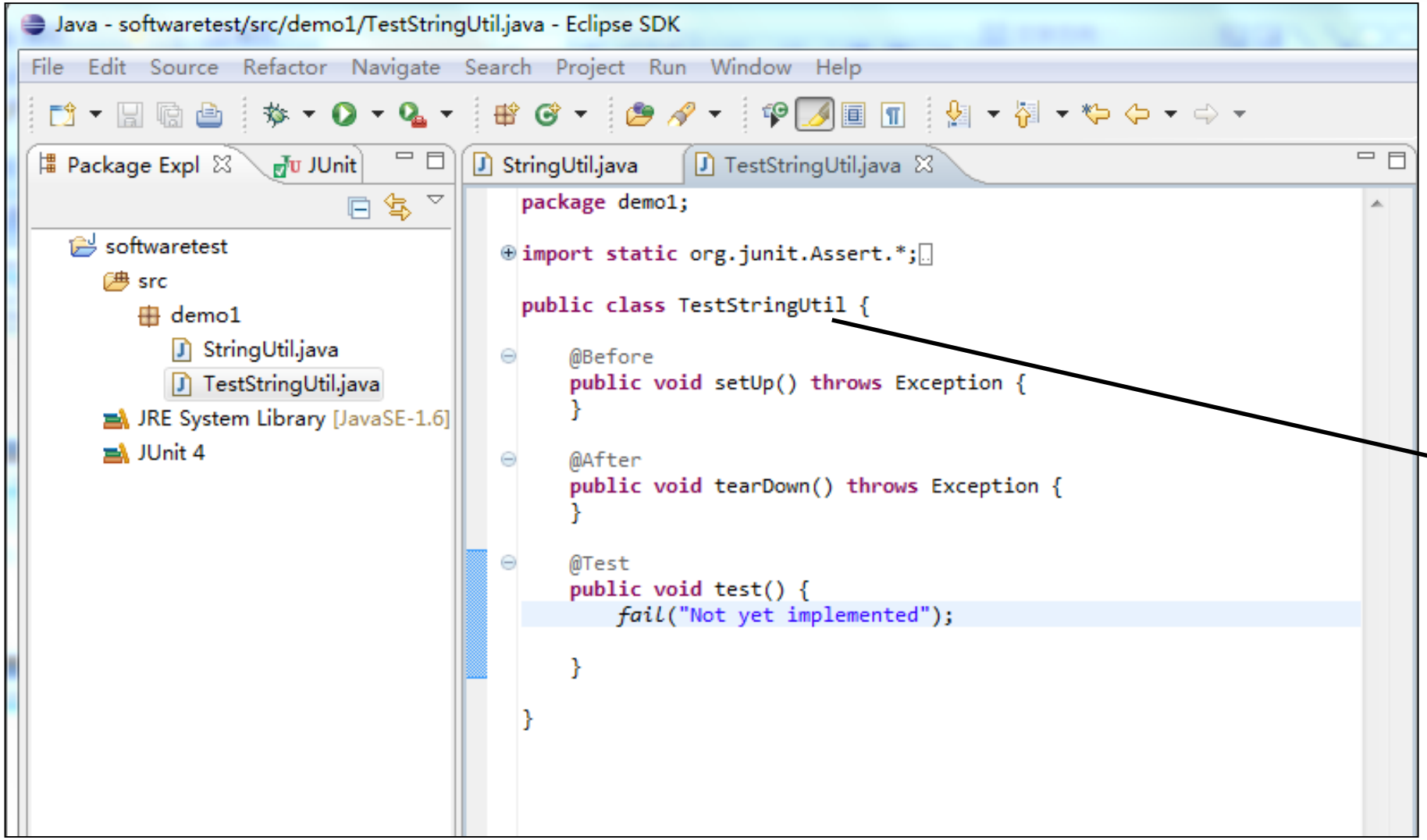
1. 建立一个被JUnit测试的类



被测试的对象

被测试的程序单元：类方法addString（）

2. 建立对应的JUnit Test类



类似于主控模块及桩模块

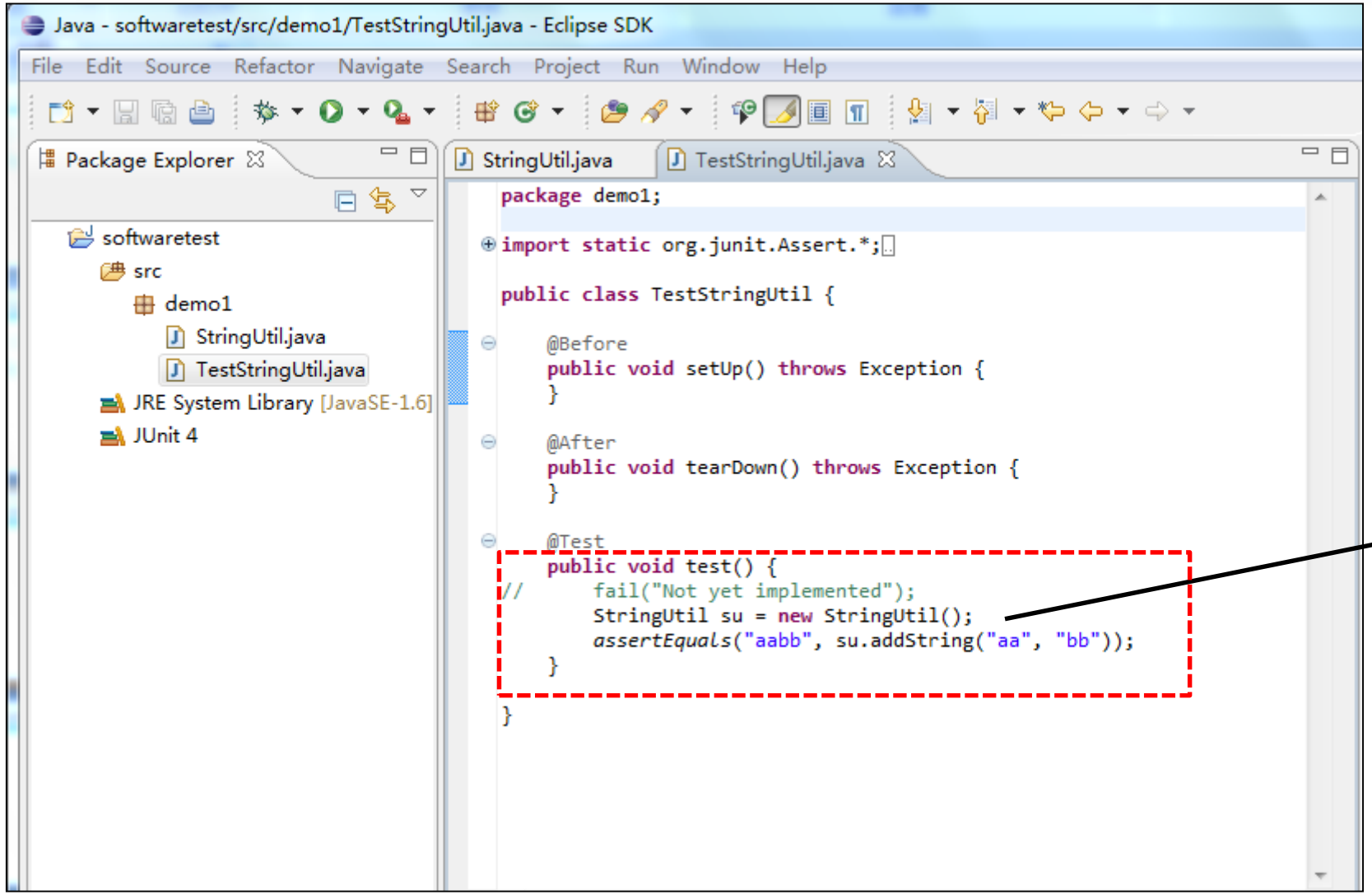
编写测试类

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

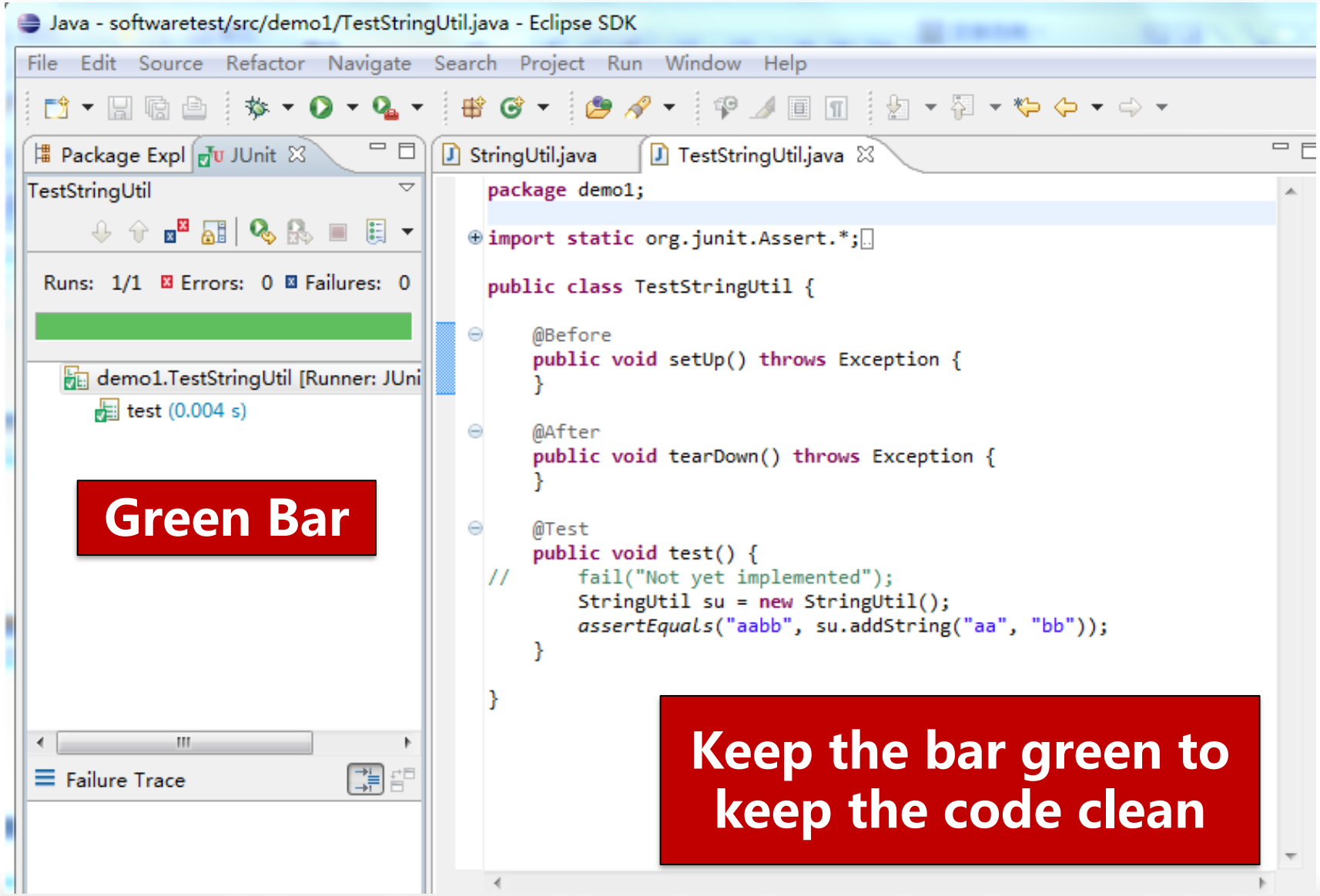
3. 针对自动生成的代码进行修改



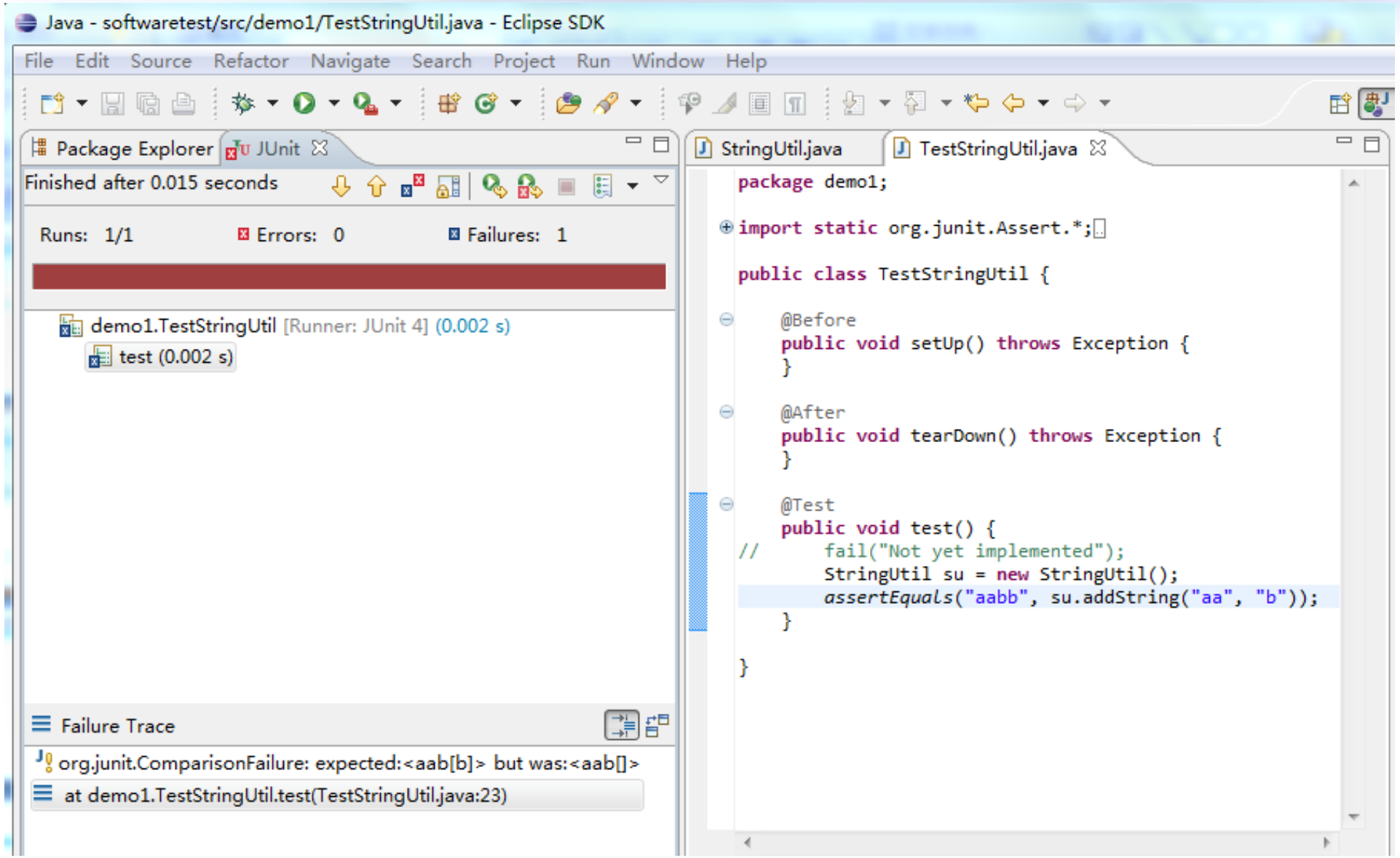
编写桩模块代码

桩模块代码

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: JUni
test (0.004 s)
Green Bar
Failure Trace
StringUtil.java
TestStringUtil.java
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



测试失败会显示红色的信息

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

3.5 面向对象软件测试

❑OO程序与结构化程序不同

- ✓OO单元测试中的**基本单元是类**，而非函数
- ✓OO中存在**继承、多态**等多种机制
- ✓OO中没有控制层级结构，而是通过**消息**传递进行**交互与集成**，因此传统增量集成方式不可行。

需要专门针对面向对象软件的测试方法

类测试

- **需要开发测试驱动程序**
  - ✓创建测试类、运行测试用例、向被测类(对象）发送消息
  - ✓根据响应值、对象状态来判断是否有缺陷
- **对简单的类**
  - ✓单独测试每个方法仍然可行，采用**白盒测试方法**

类测试

❑对于一些较复杂的类（如具有多种状态和较长的生命周期）

- ✓ 需要进一步测试其在生命周期内的方法序列，因为单独的方法测试无法发现缺陷，它与类中方法执行的顺序、接口参数等有关。

- ✓如账户类具有下列方法: open(), setup(), deposit(), withdraw(), balance(), summarize(), creditLimit(), and close()
- ✓测试用例可以设计如下:

Test case r_1: open • setup • deposit • deposit • balance • summarize • withdraw • close

Test case r_2: open • setup • deposit • withdraw • deposit • balance • creditLimit • withdraw • close

类继承的测试

- 可以先测试父类，再测试子类
- 可以重用父类的测试用例来进行子类的测试

类继承的测试

❑继承的成员方法是否不需要测试?

- ✓ 下列两种情况下需要在子类中重新测试
  - （1）继承的成员方法在子类中做了改动
  - （2）成员方法调用了改动过的其它成员方法

如：假设父类Base有两个成员方法：Inherited()和Redefined()，子类Derived只对Redefined()做了改动，子类中的Redefined()就需要重新测试。

如果子类中Inherited() 又调用了Redefined()，它也需要重新测试

OO集成测试

- **基于线程的集成测试**
  - ✓将响应某个输入或事件的一组类进行集成
- **基于使用的集成测试**
  - ✓先测试独立的类
  - ✓再利用独立类测试依赖于它的其它类
- **注意事项**
  - ✓集成测试是通过类间消息传递实现的
  - ✓消息的内容、消息的次序
  - ✓正常的消息序列和不正常的消息序列

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程师组成
  - ✓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，描述软件测试的整体规划情况**
- **软件测试报告，记录软件测试情况及发现的缺陷**
  - ✓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

小结

- **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

综合实践一

- **任务：对编写的代码进行软件测试，发现所维护的代码中存在的缺陷**
- **方法**
  - ✓在维护开源软件过程中，针对所编写的代码，设计测试用例，开展软件集成测试和确认测试
- **要求**
  - ✓针对软件设计文档和需求文档来设计测试用例，确保软件测试覆盖所有的功能
- **结果：反馈通过测试发现的软件缺陷**

综合实践二

- **任务：对编写的程序代码进行软件测试，发现所开发的软件中存在的代码缺陷**
- **方法**
  - ✓针对所编写的程序代码，设计测试用例，开展软件集成测试和确认测试
- **要求**
  - ✓针对软件设计文档和需求文档来设计测试用例，确保软件测试覆盖所有的功能
- **结果：反馈通过测试发现的软件缺陷**