第一章 软件测试概述

- 1. 为什么要进行软件测试？——软件缺陷案例
- 2. 软件缺陷：定义、原因、修复
- 3. 初步认识软件测试

第一章　软件测试概述

- 1. 为什么要进行软件测试？——软件缺陷案例
- 2. 软件缺陷：定义、原因、修复
- 3. 初步认识软件测试

1. 软件缺陷案例

1.1 乌龙指——日本瑞穗证券事件

- Fat finger乌龙指/肥手指
- 2005年12月8日，瑞穗证券一名经纪人本应发出“61万日元1股”的出售指令，却错误输成“1日元61万股”。发现错误的瑞穗证券欲取消交易，但“由于东证的系统不完备，‘撤销’指令没有被接受，造成400亿日元的损失”。

瑞穗证券公司总裁福田真（左）在记者招待会上鞠躬道歉

“乌龙指”事件之续集

- 系统开发商：富士通；运营商：东京证券交易所；用户：瑞穗证券
- 如果程序的bug导致了巨大的经济损失，应该由谁来承担？用户？运营商？还是系统开发商？
- 瑞穗证券状告东京证券交易所和富士通
- 东京地方法院判定：程序bug并不能算是重大过失，由这部分导致的损失无需赔偿。
- 重点：“bug是否很容易被检测出来”，是此案判断基准
- 完整事件描述：http://linux.cn/article-7143-1.html

1.2 被击落的空客A300

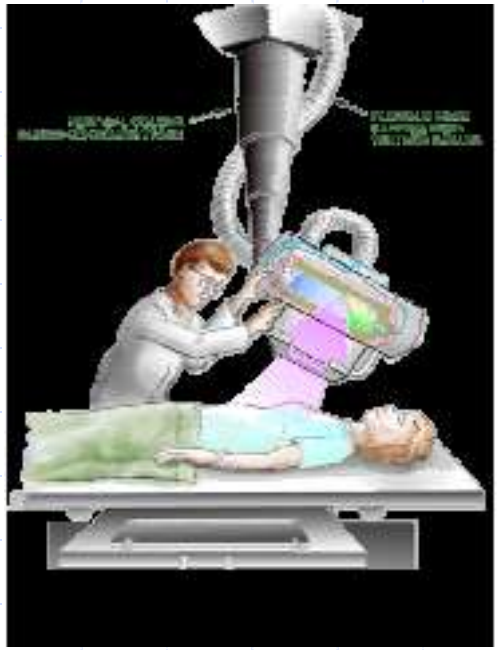
- 1988年
- 两伊战争期间，美国战舰文森斯号当时驻守波斯湾，负责击落擅闯波斯湾的飞机。
- 伊朗航空的A300客机飞行至该处时，文森斯号误以为是F-14战机，将其击落，290人遇难。
- **原因：软件错误**

1.3 爱国者导弹防御系统问题

- 1991年海湾战争中，美国爱国者导弹防御系统用于对抗伊拉克的飞毛腿导弹。
- 但发生几次对抗失利，其中一枚飞毛腿在多哈击毙28名美国士兵。
- 失败原因是软件缺陷：防御系统运行超过14小时后，系统的跟踪功能不再准确
- 多哈战役中，系统已运行100多小时，造成343.3ms的误差，导致687米距离偏差。

1.4 THERAC-25放射治疗仪错误

- THERAC-25是计算机控制的放射治疗仪
- 1986年，两名癌症患者在东Texas癌症治疗中心接受了致命的过量放射治疗
- 原因：软件错误-系统中多个任务同步运行时，系统处理出现错误。

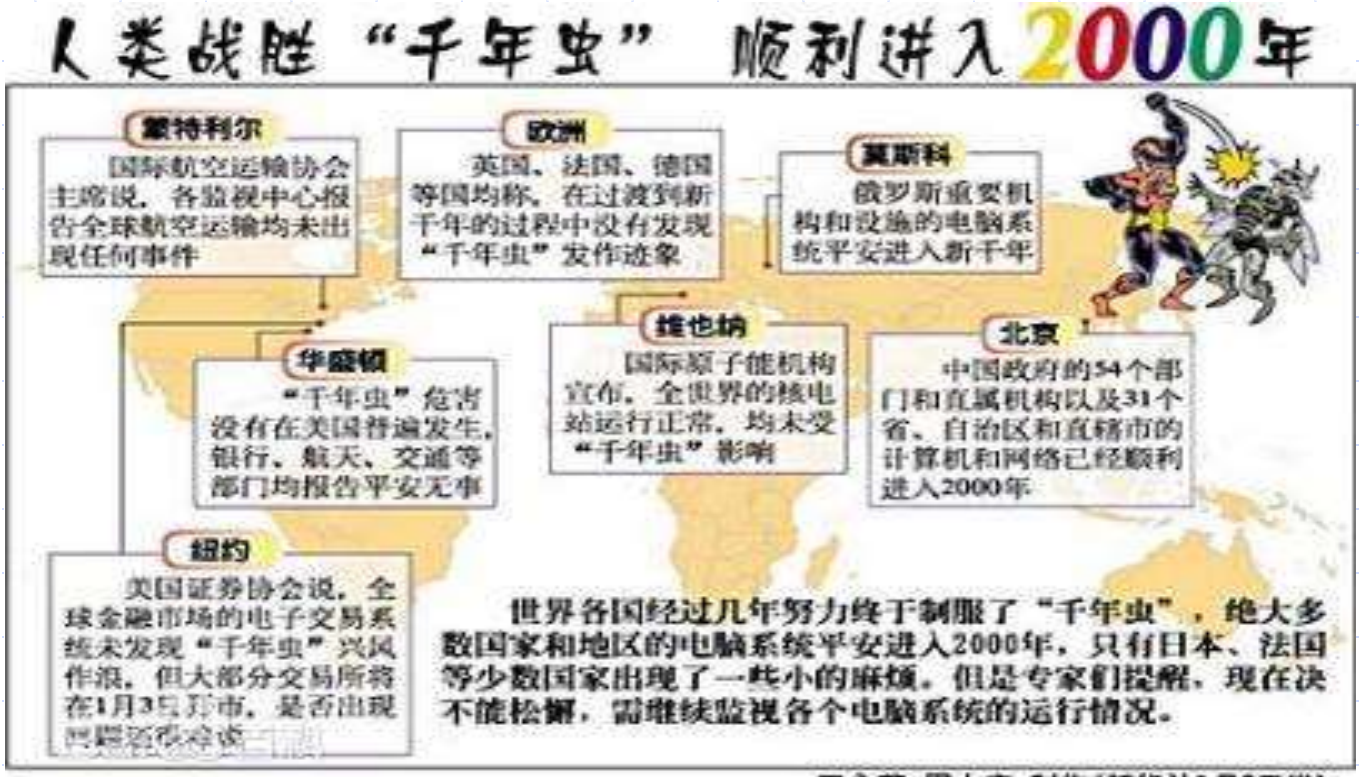


1.5 伦敦救护车服务调度系统

- 伦敦救护车服务计算机辅助调度系统（LASCAD）
- 计算机自动分配各救护车服务需求
- 1992年11月26、27日，系统调度发生严重错误
  - 多辆救护车被分配到同一事故地点
  - 没有将最近的救护车分配到事故地点
- 造成至少20人死亡

1.6 千年虫 (Y2K)

- 20世纪70年代，程序员为了节约宝贵的内存资源和硬盘空间，在存储日期时，只保留年份的后两位。当2000年到来的时候，问题就会出现。
  - 如银行存款程序对利息的计算
- 为了这样一个简单的设计缺陷，全世界付出几十亿美元。



1.7 可以自动吐钞的ATM机

- 2010年，西雅图信息安全专家杰克在“黑帽”(Black Hat)计算机安全会议演示破解ATM机，让其自动吐出钞票
- 通过ATM机的接口连接并运行他编写的破解程序，就能让ATM轻松吐钞
- 或通过网络联机，入侵厂商的远程管理软件，遥控命令让ATM机吐出钞票

1.8 其它安全攸关软件故障

- 2012年11月，美国福特公司召回8.9万辆汽车，原因是汽车冷却系统的监控软件存在故障，会导致发动机在运行时产生过热现象，并已至少发生了9起由此引起的汽车失火事故；
- 2011年7月23日，“甬温线”动车追尾事故造成40人死亡、172人受伤。事故的主要原因之一是列控中心设备中的自检模块软件存在严重设计缺陷，未将系统导向安全侧；
- 2007年10月16日，德国勒奇山基底隧道附近的列车脱轨事故，是由于ETCS-2级列控系统无线闭塞中心（RBC）接入时一个与列车移动授权延伸的相关软件错误而引起的。
- 2004年9月，由于空管软件中的时钟管理缺陷，美国洛杉矶机场400余架飞机与机场指挥一度失去联系，给几万名旅客的生命安全造成严重威胁；

1.9 怎么办？

- 所以，需要**测试**这些产品，可以解决部分问题
- 任何产品都会有缺陷，软件产品也不例外
- 再好的设计也不能保证有完美的产品

第一章　软件测试概述

- 1. 为什么要进行软件测试？——软件缺陷案例
- 2. 软件缺陷：定义、原因、修复
- 3. 初步认识软件测试

2. 软件缺陷

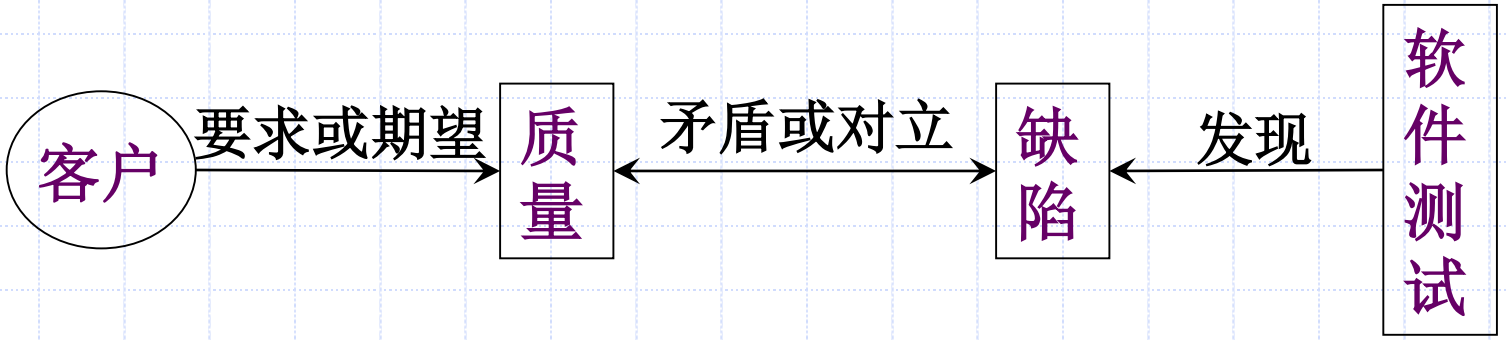
2.1 定义

2.2 原因

2.3 修复

2.1 软件缺陷定义

客户、质量、缺陷和测试的关系



简单来说，没有满足质量要求、和质量冲突的东西就是缺陷，缺陷是质量的对立面。

2.1 软件缺陷定义

- 质量：1986年ISO 8492定义

质量是产品或服务所满足明示或暗示需求能力的固有特性和特征的集合。

2.1 软件缺陷定义

- 软件质量：IEEE Std 729-1983定义
  - The totality of features and characteristics of a software product that bear on its ability to satisfy given needs.软件产品的全部特征和特性，它们具有满足规定需求的能力
  - The degree to which software possesses a desired combination of attributes.软件各种属性的组合程度
  - The degree to which a customer or user perceives that software meets his or her composite expectations.软件满足用户综合期望的程度
  - The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer.软件的综合特征可以在使用过程中满足用户期望的程度

ISO9126 软件质量模型

2.1 软件缺陷定义

- 描述软件缺陷的术语
  - bug(缺陷)
  - fault(故障)
  - problem(问题)
  - error(错误)
  - anomaly(异常)
  - defect(缺点)
  - failure(失败)
  - inconsistency(矛盾)
  - incident(毛病)
  - variance(偏差)

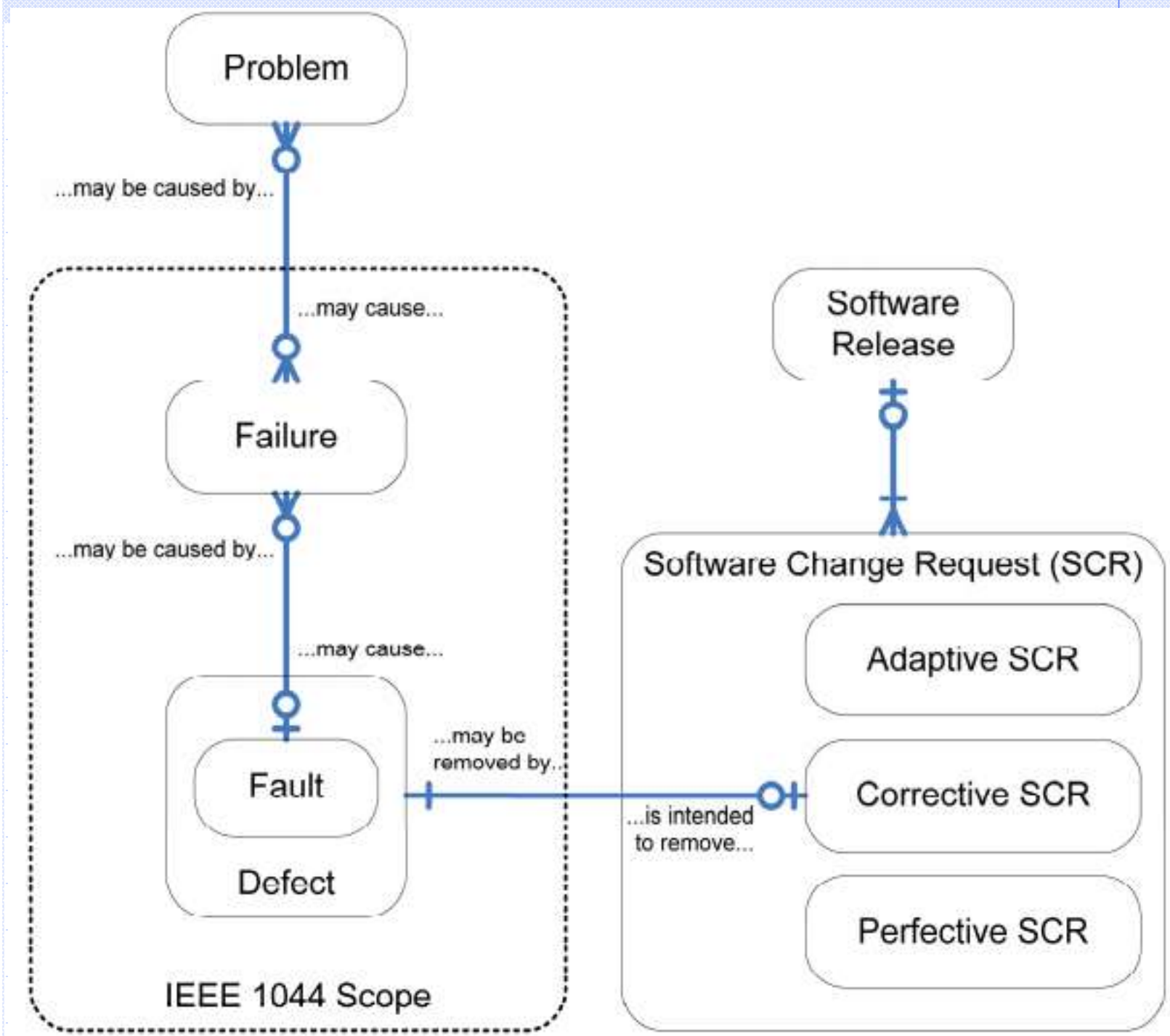
2.1 软件缺陷定义

- 软件缺陷的定义 IEEE Std 610.12-1990
  - bug. see: error; fault.
  - The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. ERROR
  - An incorrect step, process, or data definition. FAULT
  - An incorrect result. FAILURE
  - A human action that produces an incorrect result. MISTAKE
- IEEE Std 729-1983: A manifestation of an error(如错误理解了用户需求) in software is a fault. A fault, if encountered, may cause a failure.

IEEE Std 1044-2009 (IEEE Standard Classification for Software Anomalies)

- **defect:** An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.
- **error:** A human action that produces an incorrect result.
- **fault:** A manifestation of an error in software.
- **failure:** (A) Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits. (B) An event in which a system or system component does not perform a required function within specified limits.
- **problem:** (A) Difficulty or uncertainty experienced by one or more persons, resulting from an unsatisfactory encounter with a system in use. (B) A negative situation to overcome.

IEEE Std 1044-2009 (IEEE Standard Classification for Software Anomalies)
几个概念间的关系



IEEE Std 1044-2009几个概念间的关系

Class/entity pair	Relationships
Problem-Failure	A problem may be caused by one or more failures. A failure may cause one or more problems.
Failure-Fault	A failure may be caused by (and thus indicate the presence of) a fault. A fault may cause one or more failures.
Fault-Defect	A fault is a subtype of the supertype defect. Every fault is a defect, but not every defect is a fault. A defect is a fault if it is encountered during software execution (thus causing a failure). A defect is not a fault if it is detected by inspection or static analysis and removed prior to executing the software.
Defect-Change Request	A defect may be removed via completion of a corrective change request. A corrective change request is intended to remove a defect. (A change request may also be initiated to perform adaptive or perfective maintenance.)

Software Faults, Errors & Failures

- Software Fault : A static defect in the software.
- Software Failure : External, incorrect behavior with respect to the requirements or other description of the expected behavior.
- Software Error : An incorrect internal state that is the manifestation of some fault.

几个例子

Fault: Should start searching at 0, not 1

```
public static int meanArr1 (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the mean value in arr
   int sum = 0, mean;
   for (int i = 1; i < arr.length; i++)
   {
      sum+= arr[i];
   }
    mean = sum / arr.length;
   return mean;
}
```

Test 1
[0, 4, 5]
Expected: 3
Actual: 3

Error: not error for sum
Failure: none

Test 2
[3, 4, 5]
Expected: 4
Actual: 3

A test executing the fault may not produce an error, then no failure of course.

hean
atement

几个例子

Fault: should be len= arr.length

```
public static int meanArr2 (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the mean value in arr
   int sum = 0, mean, len= arr.length-1;
   for (int i = 0; i < len; i++)
   {
      sum+= arr[i];
   }
    mean = sum / len;
   return mean;
}
```

Test
[3, 5, 4]
Expected: 4
Actual: 4

Error: error for sum
Error not propagates to the variable mean
Failure: none

An error may not be propagated to the output.

几个例子

```
public static int absv (int  x)
{ // return the absolute value of x
   int ax=x;
   if(x < 0)
   {
      ax = 1- x;
   }
   return ax;
}
```

Fault: Should be 0-x

Error: none
Failure: none

Test
[5]
Expected: 5
Actual: 5

A test may not execute the fault.

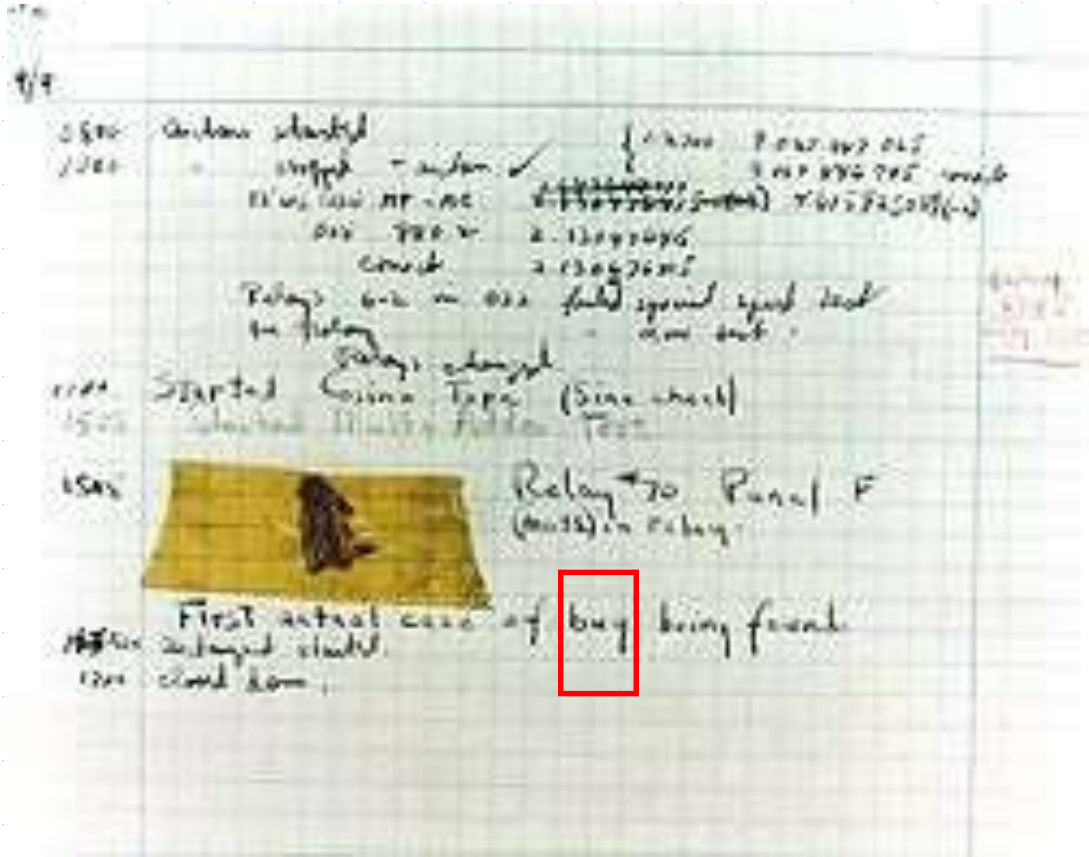
2.1 软件缺陷定义

第一个bug

- **1946年，当Grace Hopper在哈佛的Mark II机器上工作时，发现机器停止工作，结果是一只飞蛾掉在了机器的继电器上，清除后机器工作正常，她在工作日志上粘贴这只蛾子，并写下术语bug.**

Grace Hopper

(1906-1992)



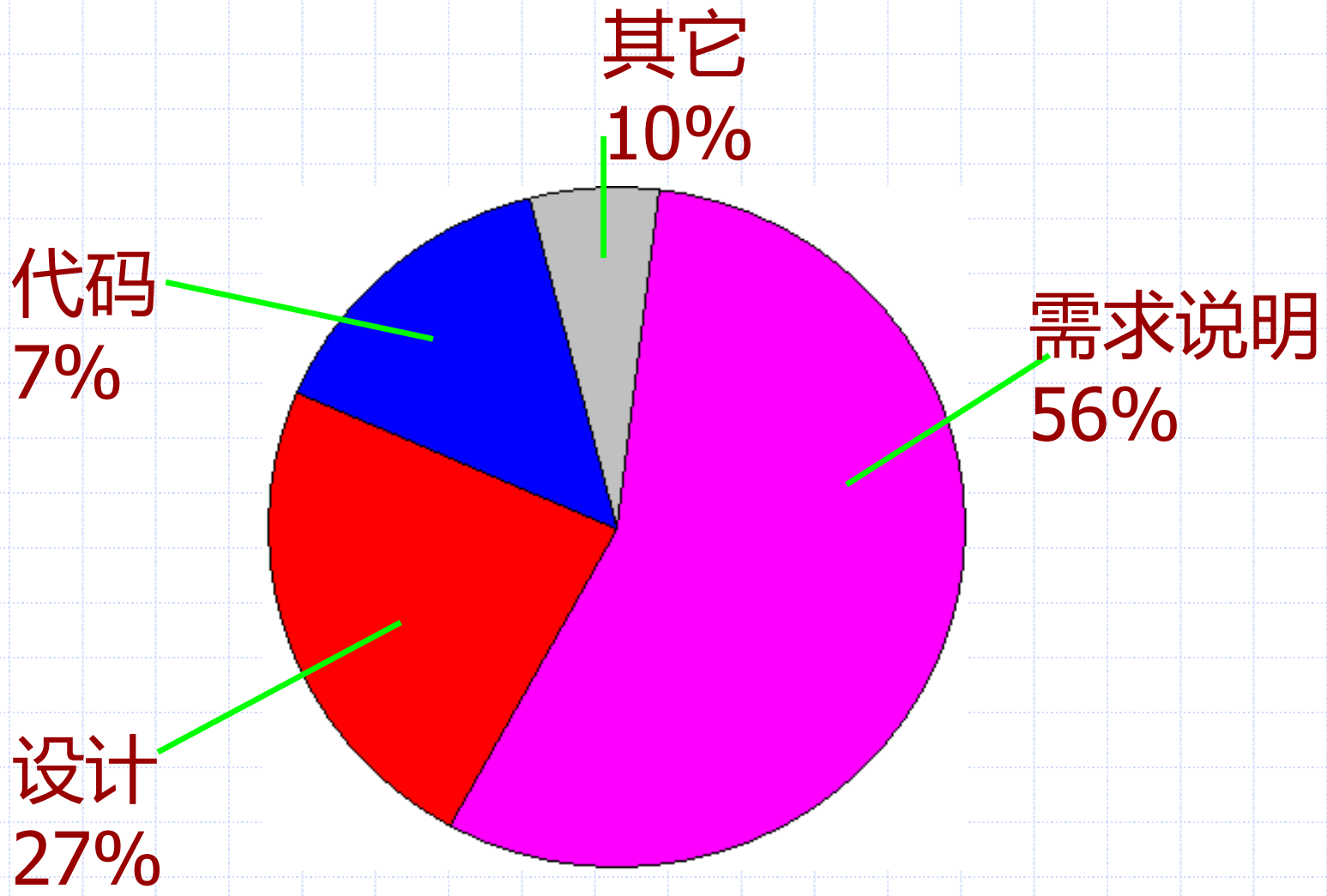
2.1 软件缺陷定义

- *Bug* is used informally
- Sometimes speakers mean fault, sometimes error, sometimes failure … often the speaker doesn't know what it means !
- 简单来说，软件缺陷就是软件产品中所存在的问题，最终表现为用户的需求（功能或性能等）没有完全实现，没有达到用户的需求。

课堂练习

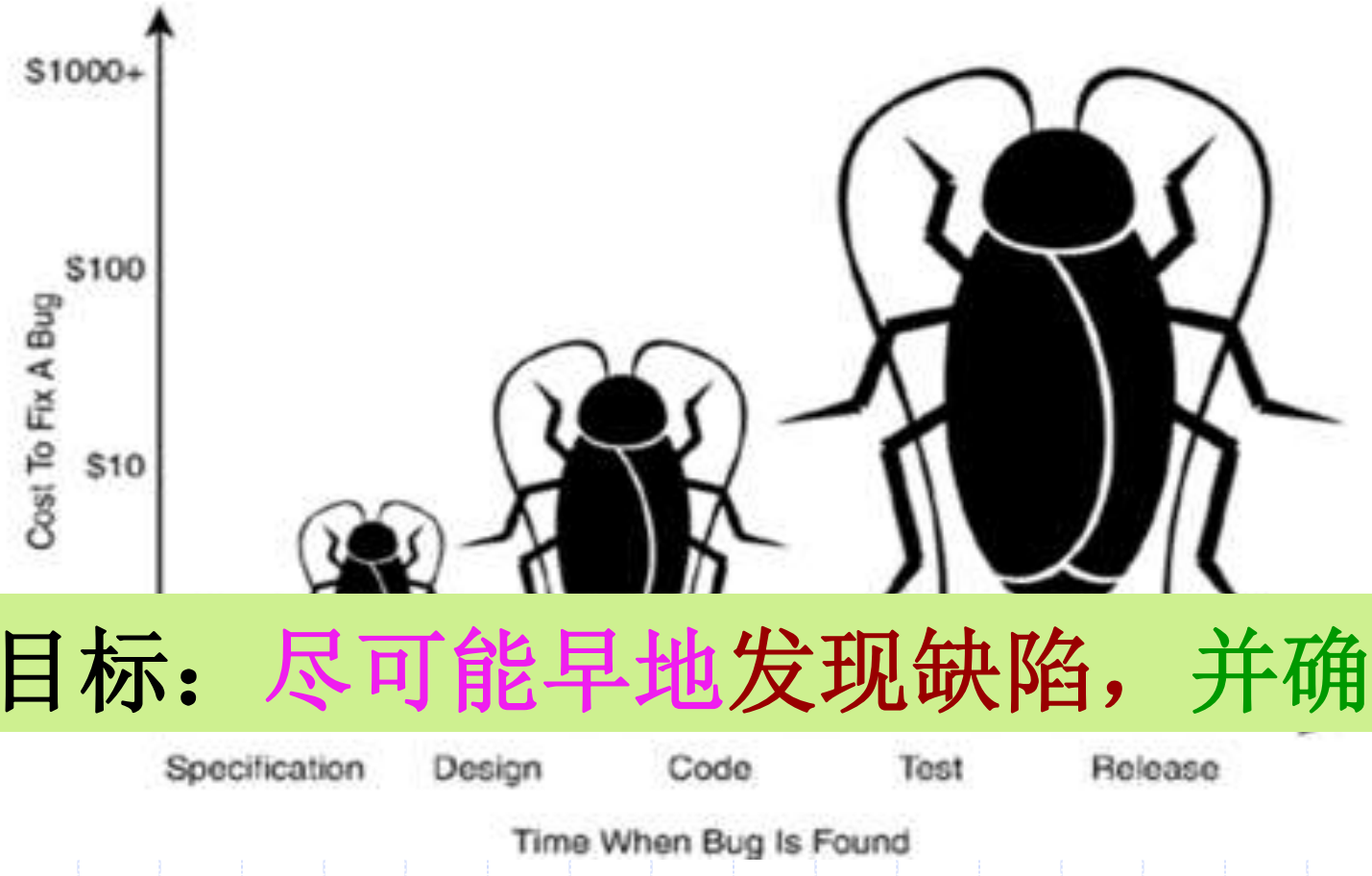
- 要求见课程中心cc.scu.edu.cn
- 15分钟后提交

2.2 软件缺陷出现的原因



2.3 软件缺陷的修复代价

- 软件的修复代价与修复时机相关：随着时间的推移，修复代价成几何数量增长。



软件测试的目标：尽可能早地发现缺陷，并确保其得以修复。

第一章 软件测试概述

- 1. 为什么要进行软件测试？——软件缺陷案例
- 2. 软件缺陷：定义、原因、修复
- 3. 初步认识软件测试

3.1 软件测试的发展历程

- Hetzel将软件测试的历史按照时间分为5个阶段

1. 1956年之前，面向调试的测试
2. 1957 ~ 1978，面向证明的测试
3. 1979 ~ 1982，面向查错的测试
4. 1982 ~ 1987，面向评估的测试
5. 1988 ~ 2000，面向预防的测试

3.1.1面向调试的测试

- 1956年之前，发现软件缺陷和修改缺陷的是同样的编程人员，那时是不区分软件调试和测试的，既没有系统的测试理论出现，也没有专人从事发现软件缺陷的测试工作。
- 软件设计人员仅在机器出现不正常状态时去找问题，然后修复它。

3.1.2面向证明的测试

- 1957-1978，开始出现了软件测试的理论。
- 但是，此时软件测试的主要工作是证明软件可以使用，测试还处于一种软件开发的辅助状态；
- 测试人员主要使用可以正常运行的合法数据来检验软件是否能够正常完成其工作。

例 1 计算银行利息的函数，不加保护的程序

```
double Get_Back_Interest( double fPrincipal, double fBack_Rate, int
    iYear)
{
    double fTotal = fPrincipal * ( 1 + fBack_Rate ) iYear
    return ( fTotal - fPrincipal );
}
```

3.1.3面向查错的测试

- 1979-1982，设计和开发程序的逻辑越来越严密，不仅考虑程序正常状态下的运行情况，也要考虑程序在各种**错误操作和数据**下的承受能力，从这个意义上说，软件测试促进了程序质量的提高；
- 这一阶段对于软件测试的理解并不太成熟，往往**过分强调找到软件中的错误**。

例 2 计算银行利息的函数，加保护的程序

```
double Get_Back_Interest( double fPrincipal, double fBack_Rate, int
    iYear)
{
    if (fPrincipal < 0.0 ) /// 对本金加保护，必须大于等于0
        return ( 0 );
    if (fBack_Rate < 0.0 ) // 对利率加保护，必须大于等于0
        return ( 0 );
    if (iYear <= 0 )        /// 对年加保护，必须大于0
        return ( 0 );
    double fTotal = fPrincipal× ( 1 + fBack_Rate ) iYear
    return ( fTotal - fPrincipal );
}
```

3.1.4面向评估的测试

- 1983-1987，测试的目的不是为了证明软件的对错，而是将可观察到的软件缺陷减少到一个可以接受的程度；
- 测试员为程序员提供软件缺陷的信息，为管理员提供如果发布软件系统会造成负面影响的评估报告；
- 在这个阶段，软件测试不仅得到了蓬勃地发展，而且软件测试的目的变得客观成熟。

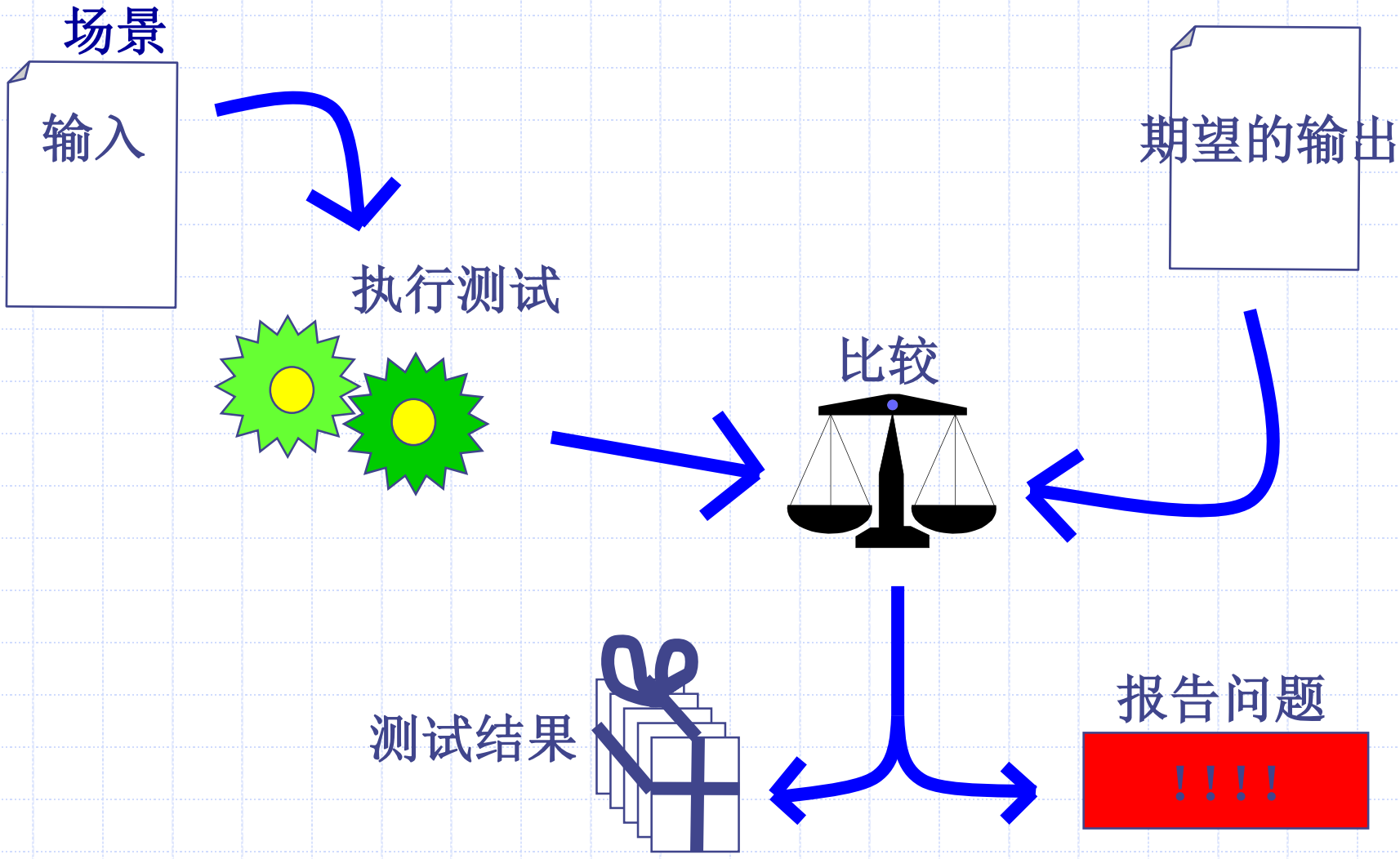
3.1.5面向预防的测试

- 1988-2000，软件测试不仅是操作，而是一种智力训练，它将导致即使进行较少地软件测试也使软件是低风险的；
- 在这种成熟的软件测试情况下，致力于从软件开发初期就是可测试的。

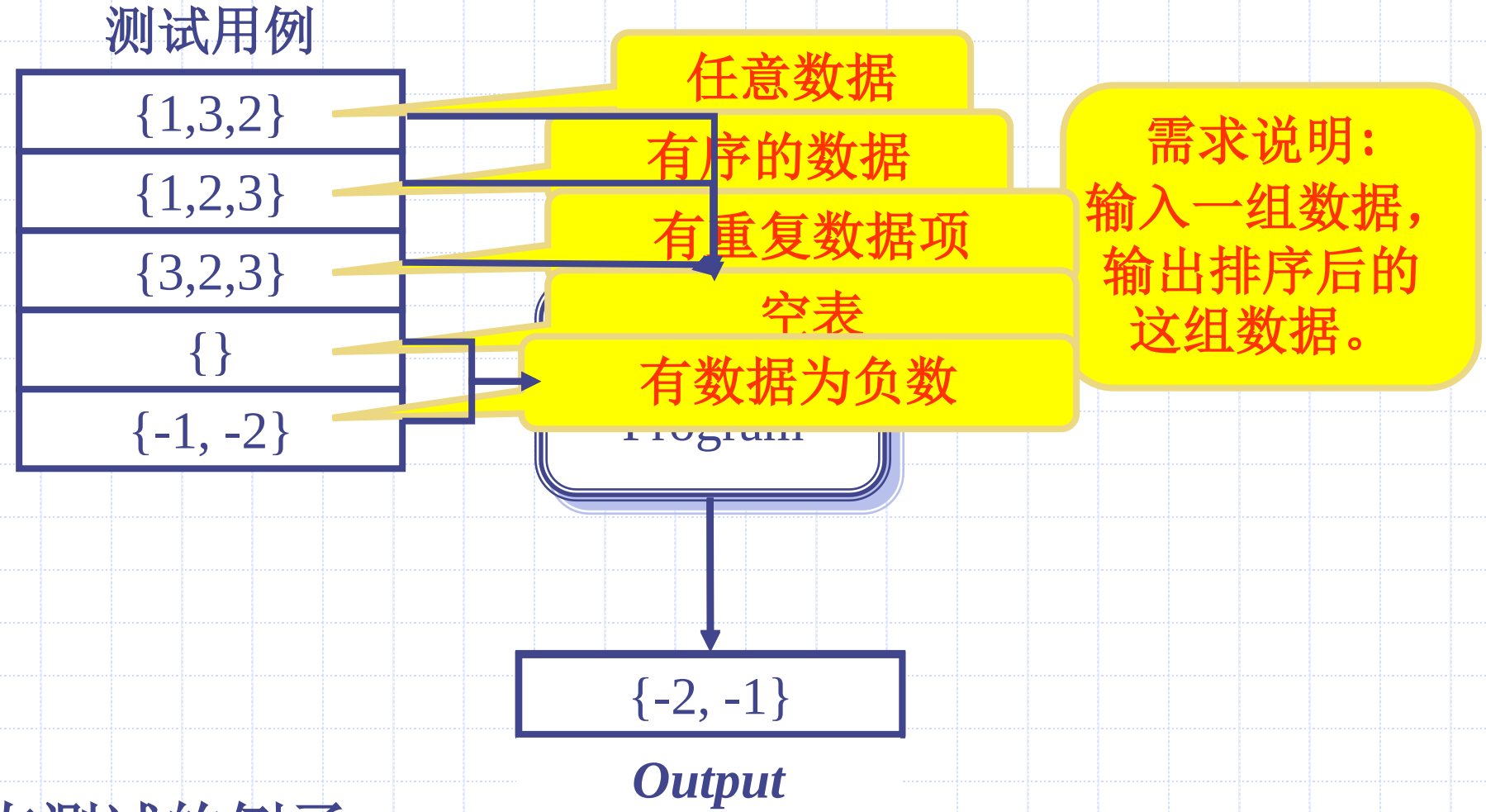
例 3 计算银行利息的函数，具有自诊断能力的代码。

```
double Get_Back_Interest( double fPrincipal, double fBack_Rate, int iYear)
{   if  (fPrincipal < 0.0 )  { // 本金必须大于等于0
        # ifdef _DEBUG /// 自诊断能力的加入，便于错误定位
            printf( " Principal can't less 0. Please reinput! " );
        # endif
        return ( 0 );  }
    if  (fBack_Rate < 0.0 )  { //利率必须大于等于0
        # ifdef _DEBUG
            printf( " Back rate can't less 0. Please reinput! " );
        # endif
        return ( 0 );  }
    if  (iYear <= 0 )  {   //年必须大于0
        # ifdef _DEBUG
            printf( " year can't less 0. Please reinput! " );
        # endif
        return ( 0 );  }
    double fTotal = fPrincipal× ( 1 + fBack_Rate ) iYear;
    return ( fTotal - fPrincipal );
}
```

3.2 什么是测试?



一个软件测试例子



一个动态测试的例子

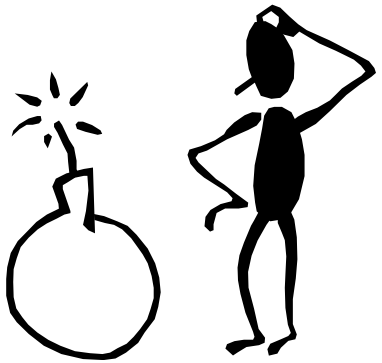
3.3 对软件测试的认识

- 测试是通过运行程序来发现错误的过程——动态测试
- 测试的目的是发现程序中的错误，是为了证明程序有错，而不是证明程序无错；
- 测试可以说明软件存在错误，但不能说明它不存在错误；
- 理想的目标：用相对少的测试尽可能多地找到程序中的缺陷。
- **“好软件是做出来的，不是改出来的。”**

软件测试的目的

1、最终目的

验证用户需求



2、直接目的

发现软件缺陷

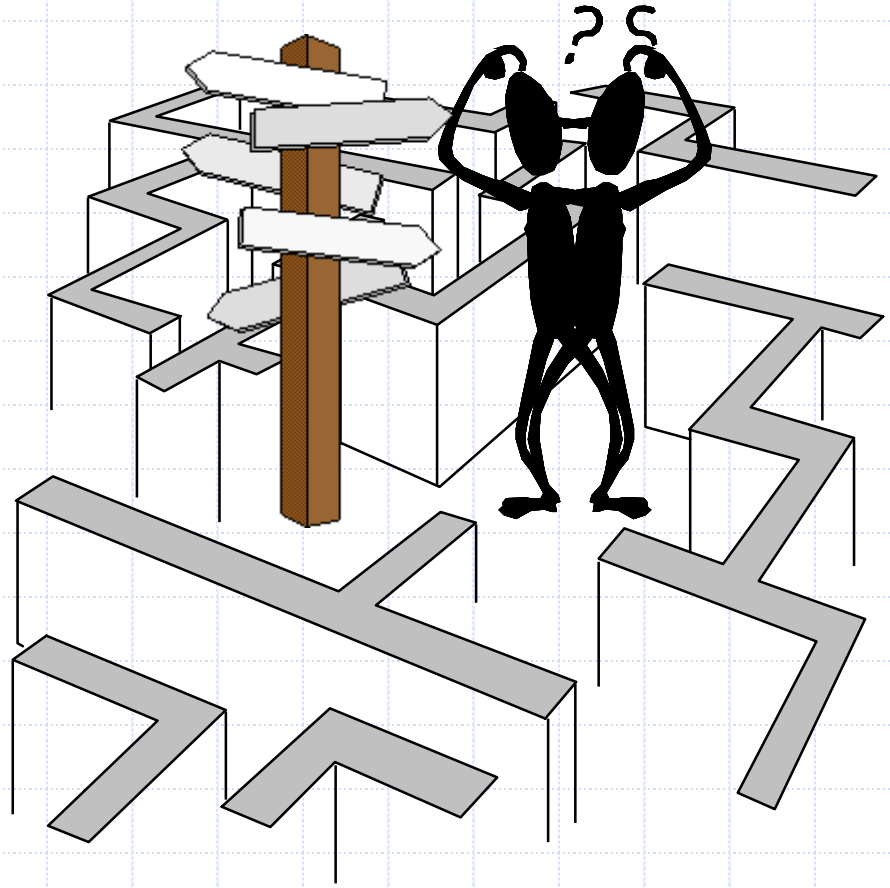
3、附加目的

改进开发过程

3.4 软件测试的对象

- 软件测试并不等于程序测试。软件测试应贯穿于软件定义与开发的整个期间。
- 测试对象：软件开发过程中各阶段的人工制品。
  - 需求分析、概要设计、详细设计以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计说明、详细设计说明以及最终的系统，都是软件测试的对象。

3.5 软件测试的挑战



An Example

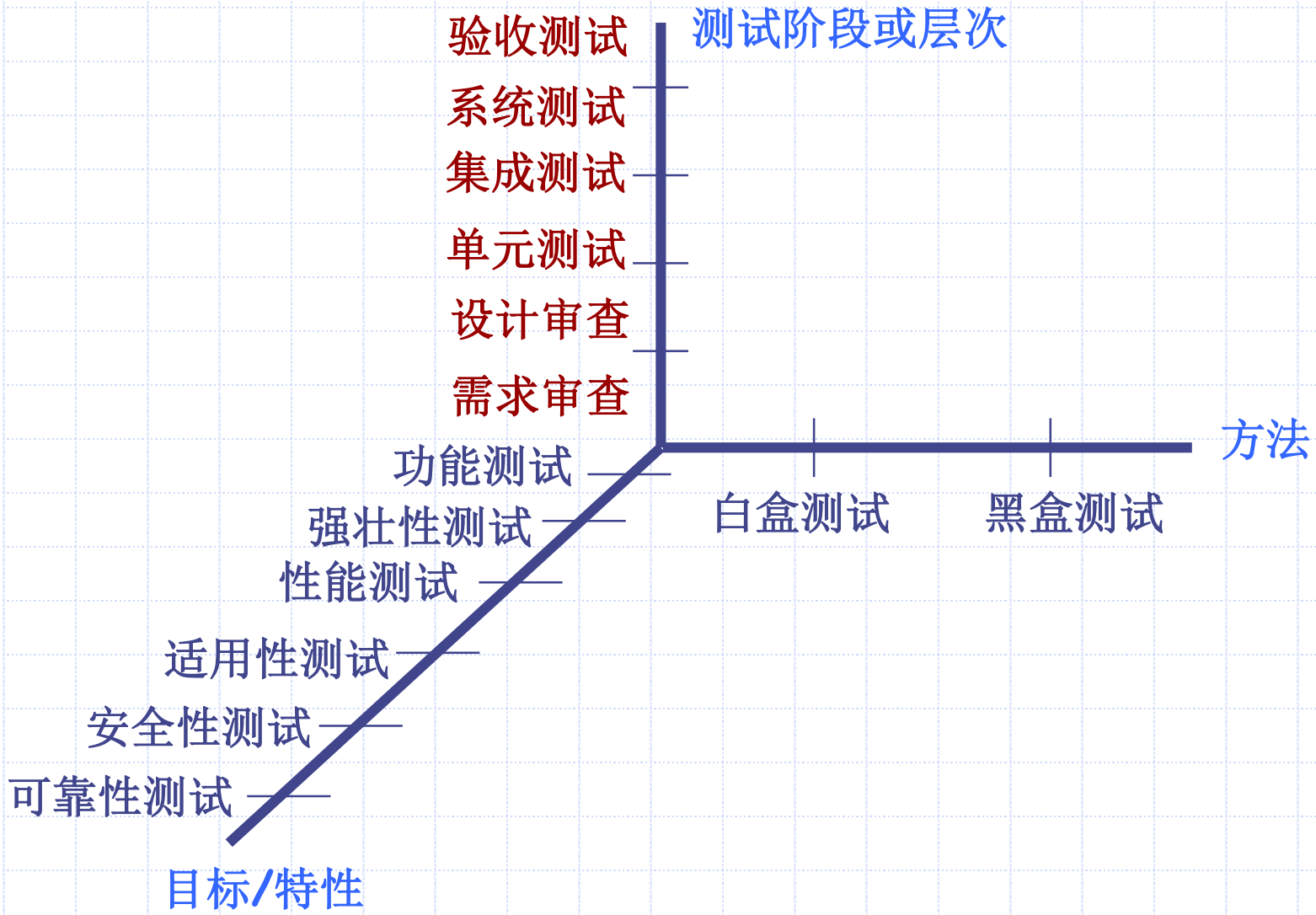
软件行业的发展**为软件测试带来新的挑战**

- **软件结构越来越复杂**
- **团队规模越来越大**
- **开发成本与风险越来越高**
- **用户对应用程序的质量要求越来越严格**

3.5 软件测试的挑战(cont.)

- 软件类型繁多
  - OO软件系统
  - 基于组件的系统
  - 并发系统
  - 分布式系统
  - GUI系统
  - Web系统
- 测试目的不同
  - 可用性、安全性、正确性、性能……

3.6 软件测试分类


测试阶段或层次
验收测试
系统测试
集成测试
单元测试
设计审查
需求审查
方法
白盒测试
黑盒测试
功能测试
强壮性测试
性能测试
适用性测试
安全性测试
可靠性测试
目标/特性

3.6 软件测试分类

自动测试和手工测试

手工模拟用户操作

3.6 软件测试分类

静态测试和动态测试：被测软件是否被执行

运行程序

本章内容回顾

- 1. 为什么要进行软件测试？——软件缺陷案例
- 2. 软件缺陷：定义、原因、修复
- 3. 初步认识软件测试

课堂练习

1. 判断正误：

1)测试人员说：“没有可运行的程序，我无法进行测试工作”。 错

2)在软件开发过程中，若能推迟暴露其中的错误，则为修复和改进错误所花费的代价就会降低。 错

2. 软件测试的目的是（　）　B

A）避免软件开发中出现的缺陷

B）发现软件开发中出现的缺陷

C）证明软件中没有缺陷

D）修改软件中出现的缺陷

课堂练习

3. 下列可以作为软件测试对象的是（　）。

A）需求规格说明书

B）软件设计规格说明

C）源程序

D）以上全部

D

4. 软件调试的目的是（　）。

A) 找出错误所在并改正之

B) 排除存在错误的可能性

C) 对错误性质进行分类

D) 统计出错的次数

A

课后作业：

1、在你的印象中，是否还有其它实例说明软件问题会造成巨大经济损失或带来社会灾害？请至少给出一个实例。

2、需求分析、系统设计阶段存在的问题在软件产品缺陷中占有较大比例，对软件开发和测试工作有什么启发？

课后作业

3. Below are four faulty programs. Each includes test inputs that result in failure. Answer the following questions about each program.

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.

(b) If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.

(c) If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.

(d) If possible give a test case that results in an error state, but **not** a failure. If not, briefly explain why not.

```
/*P1: Find last index of element
* @param x array to search
* @ param y value to look for
* @return last index of y in x; -1 if absent
* @throws NullPointerException if x is null
*/
public int findLast (int[] x, int y)
{
    for (int i=x.length-1; i > 0; i--)
    {
            if (x[i] == y)
            {
                return i;
             }
     }
return -1;
}
// test: x = [2, 3, 5]; y = 2; Expected = 0
```

```
/*P2: Find last index of zero
* @param x array to search
* @return last index of 0 in x; -1 if absent
* @throws NullPointerException if x is null
*/
public static int lastZero (int[] x)
{
    for (int i = 0; i < x.length; i++)
    {
        if (x[i] == 0)
        {
            return i;
         }
    }
return -1;
}
// test: x = [0, 1, 0]; Expected = 2
```

```
/*P3: Count positive elements
* @param x array to search
* @return count of positive elements in x
* @throws NullPointerException if x is null
*/
public int countPositive (int[] x)
{
    int count = 0;
    for (int i=0; i < x.length; i++)
    {
        if (x[i] >= 0)
        {
            count++;
        }
     }
     return count;
}
// test: x = [-4, 2, 0, 2]; Expected = 2
```

```
/*P4: Count odd or positive elements
* @param x array to search
* @return count of odd/positive values in x
* @throws NullPointerException if x is null
*/
public static int oddOrPos(int[] x)
{
    int count = 0;
    for (int i = 0; i < x.length; i++)
    {
        if (x[i]%2 == 1 || x[i] > 0)
        {
            count++;
         }
     }
     return count;
}
// test: x = [-3, -2, 0, 1, 4]; Expected = 3
```

The End

Any Question?