软件测试概述

课程概览

- ✓ 软件测试基本思想
- ✓ 软件静态测试技术
- ✓ 软件动态测试技术
- ✓ Web测试技术
- ✓ 性能测试技术
- ✓ 自动化测试技术

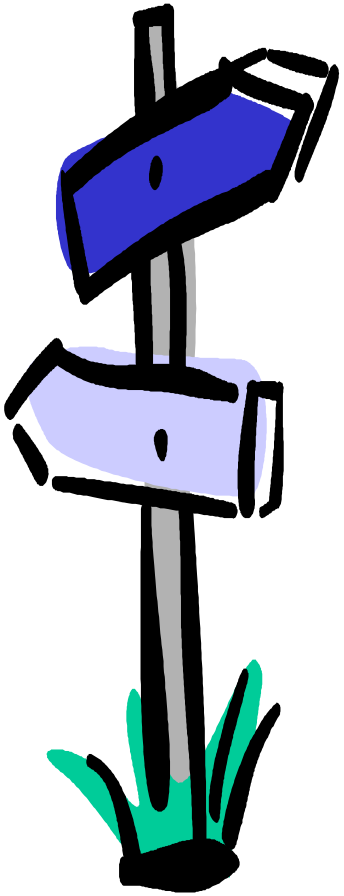
课程目标

- ✓ 掌握软件测试的基本概念、基本思想
- ✓ 掌握白盒测试、黑盒测试和方法
- ✓ 掌握单元测试框架
- ✓ 掌握Web测试方法和工具
- ✓ 掌握性能测试工具

课程考核说明

- ✓笔试成绩70%
- ✓实验成绩30%
  - 项目实战测试
  - 竞赛测试（软件测试大赛）

本章内容



- ✓ **BUG**
- ✓ 软件质量
- ✓ 三个基本术语（PIE模型）
- ✓ 软件测试基本概念
- ✓ 软件测试反思

软件缺陷（bug）

- ✓软件缺陷引发的问题
- ✓软件缺陷是什么
- ✓软件缺陷的含义

大家猜猜

- 100万美元
- 1000万美元
- 2000万美元
- 5000万美元
- 1亿美元
- 2亿美元

?

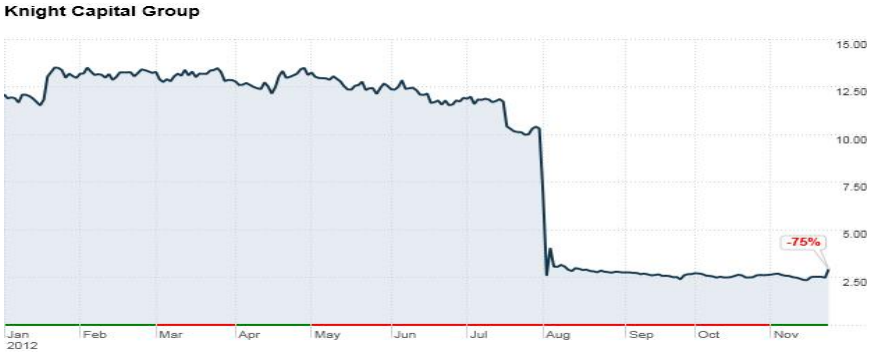
一个最严重的Bug带来的损失有多大?

让我们来看看经典案例-1

- 1993年，Intel奔腾CPU出现浮点计算问题，最后不得不召回所有发售的CPU，造成**4亿多美元损失**
- 1996年，Ariane 5型运载火箭的首航，就因软件引发的问题导致火箭在发射39秒后偏轨，最终灰飞烟灭，造成**3.7亿**美元损失
- 1999年，火星探测器因为一个Bug在距离火星表面130英尺的高度垂直坠毁，此项工程成本耗费**3.27亿**美元
- …… ……

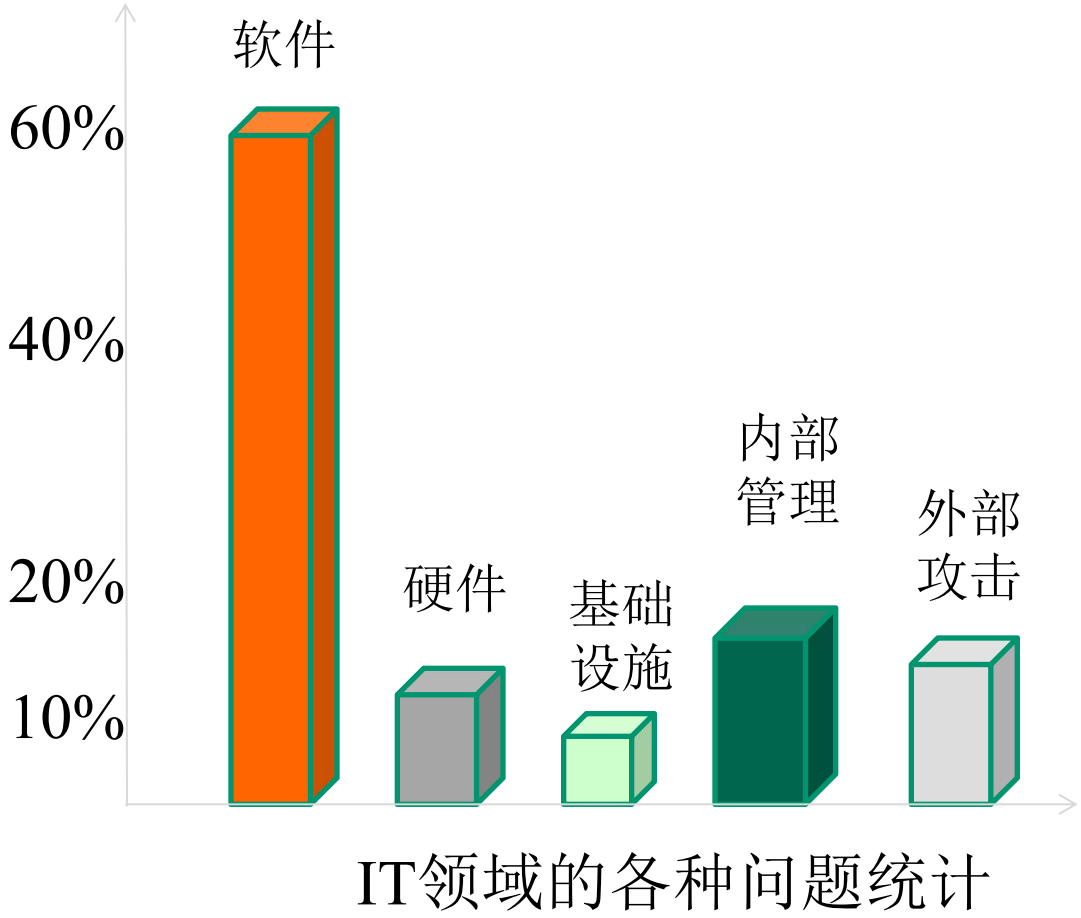
让我们来看看经典案例 - 2

- 2012年8月，美国股市最大经纪商之一骑士资本(KCG)，由于电子交易系统问题导致该公司对148支不同的股票高价购进、低价抛出。这次软件问题直接给KCG带来了税前**4.4亿美元**的损失，当天该公司的股票下跌32%
- 无独有偶，2013年8月国内光大证券公司的套利策略系统的订单生成子系统存在严重缺陷，没有头寸控制机制，生成巨额订单，导致股市大幅波动，事后遭证监会重罚**5.23亿人民币**

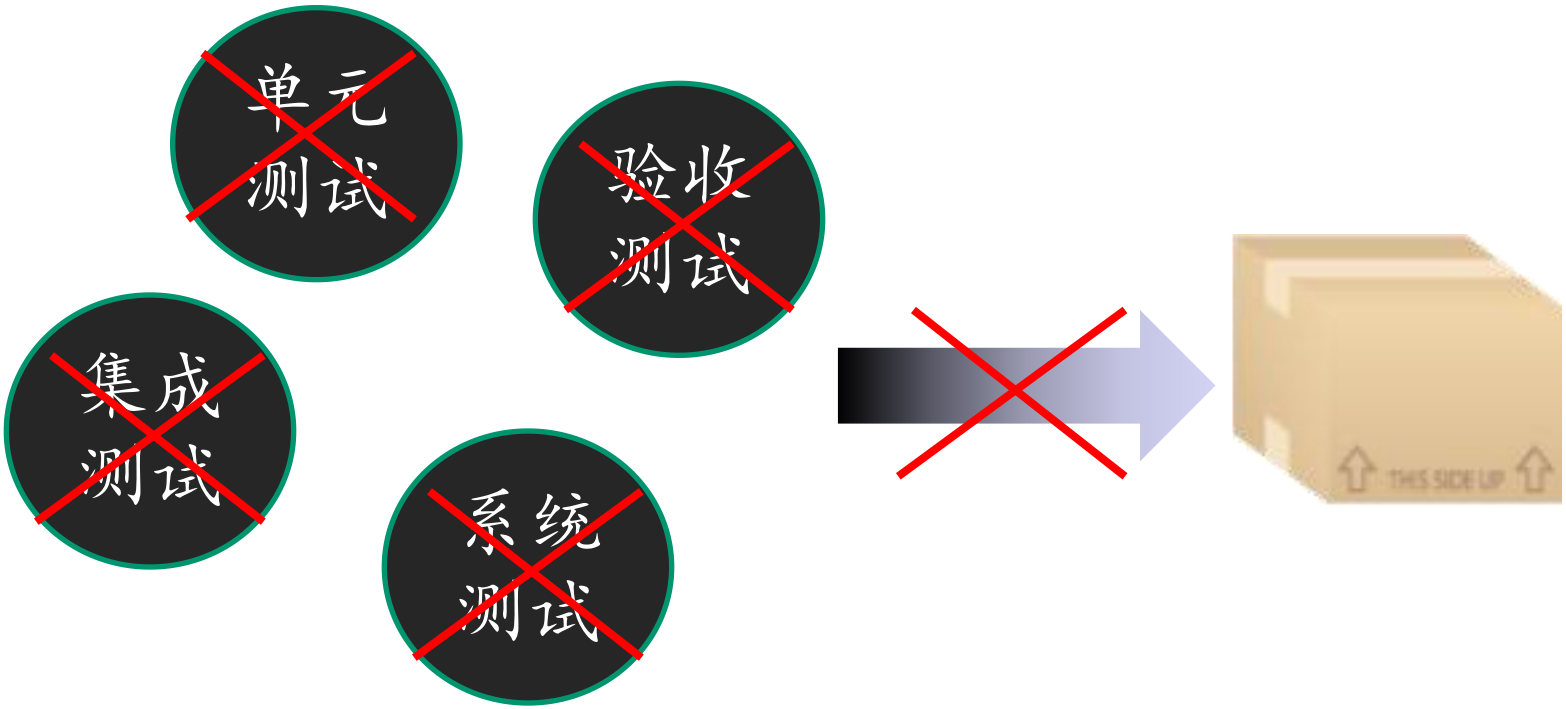


缺陷带来的损失有时难以估量

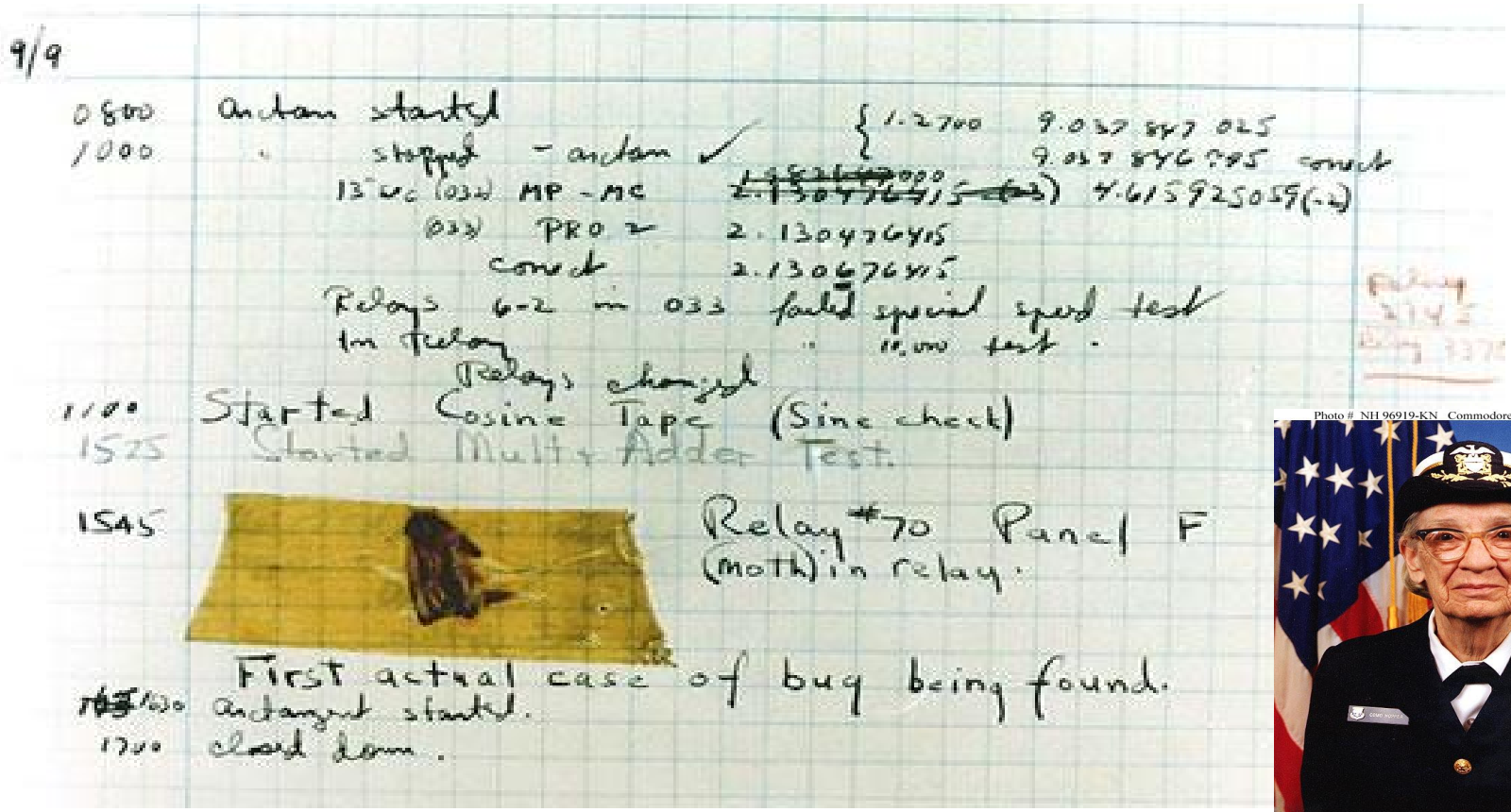
软件缺陷远远高于硬件缺陷



No test， No Release



First Bug



http://en.wikipedia.org/wiki/Grace_Hopper

Grace Hopper

- ✓第一个Bug
- ✓世界上第一个编译器A0系统
- ✓第一个商业程序语言 COBOL(Common Business Oted Language)
- ✓The Queen of Code

缺陷 - Bug

缺点（defect）　　异常（anomy）

故障（fault）　　失效 （failure）

问题（problem）　　矛盾（inconsistency）

错误（error ）　　毛病 （incident ）

软件缺陷含义

IEEE（1983）729 软件缺陷一个标准的定义：

- 从产品内部看，软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题；
- 从外部看，软件缺陷是系统所需要实现的某种功能的失效或违背。

软件缺陷的主要类型/现象：

- 功能、特性没有实现或部分实现
- 设计不合理，存在缺陷
- 实际结果和预期结果不一致
- 运行出错，包括运行中断、系统崩溃、界面混乱
- 数据结果不正确、精度不够
- 用户不能接受的其他问题，如存取时间过长、界面不美观

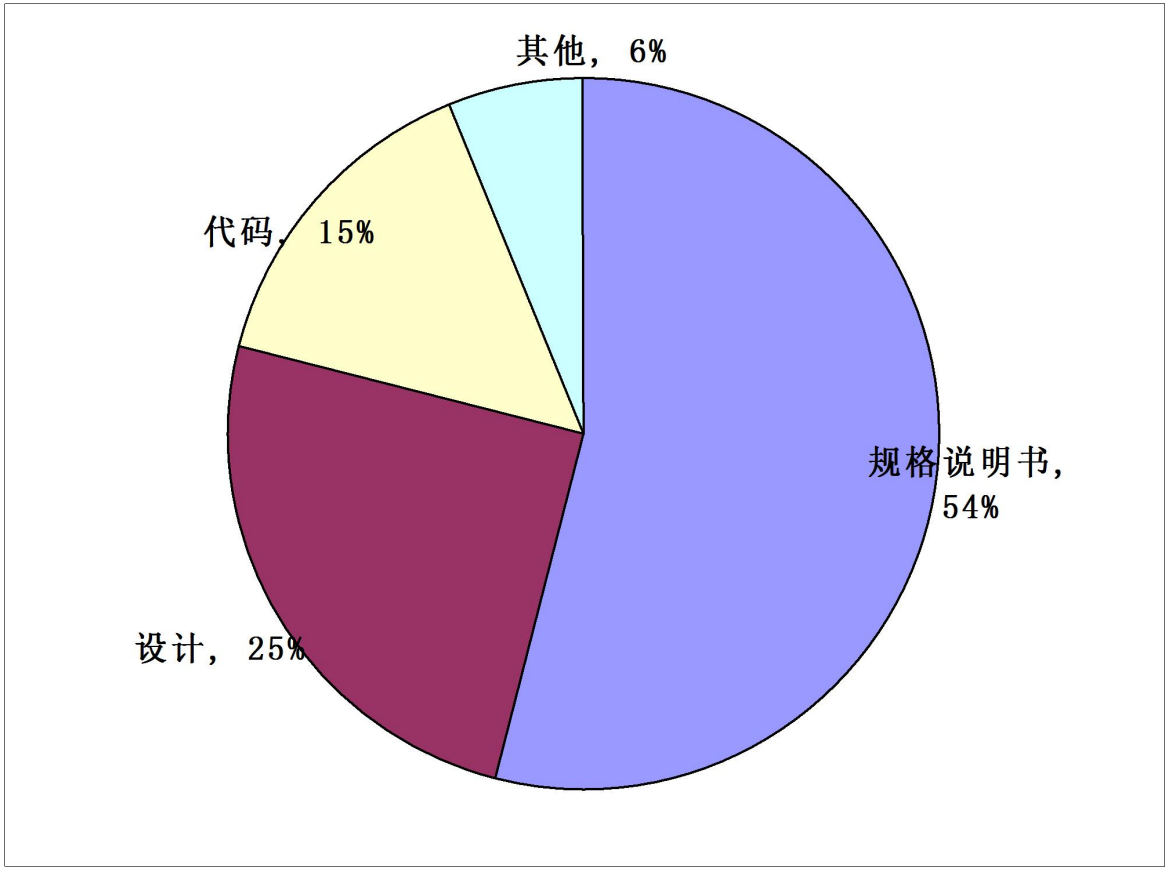
软件缺陷的产生

① **技术问题：**算法错误，语法错误，计算和精度问题，接口参数传递不匹配

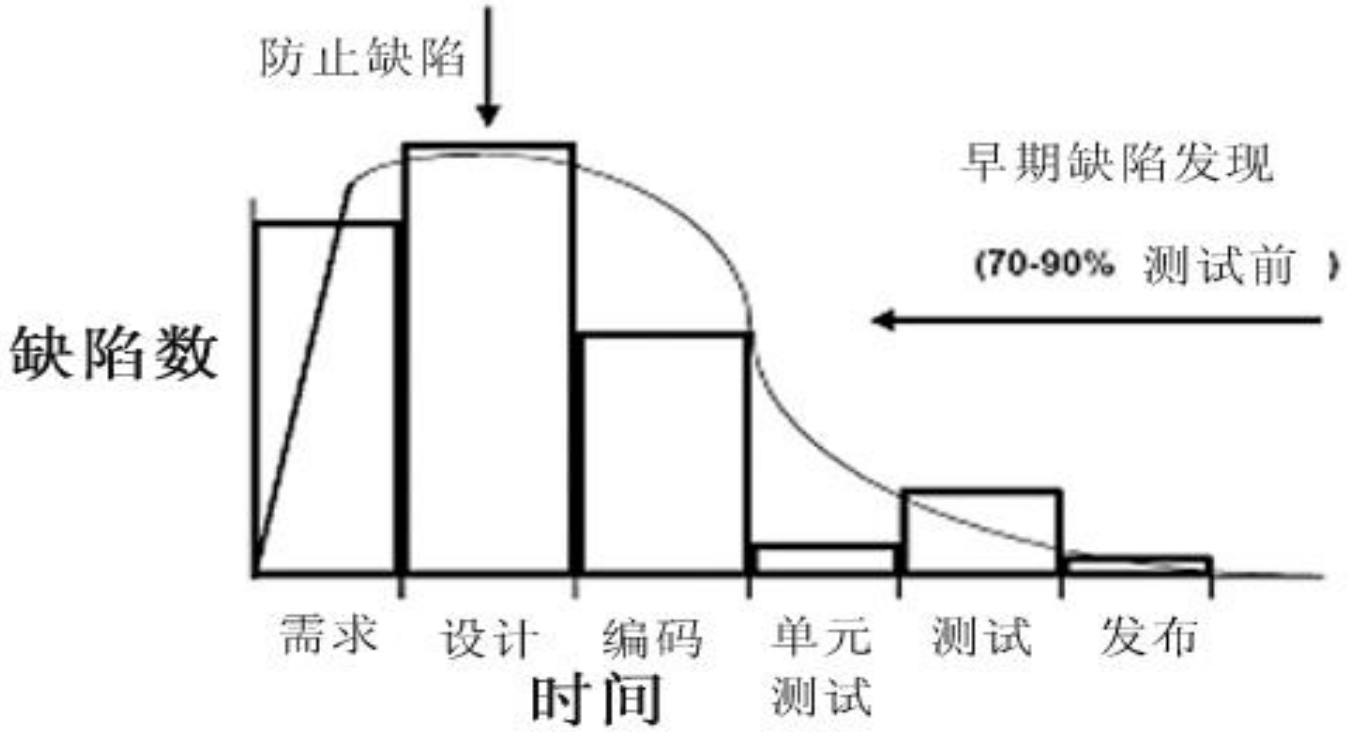
② **团队工作：** 误解、沟通不充分

③ **软件本身：**文档错误、用户使用场景(user scenario)，时间上不协调、或不一致性所带来的问题。系统的自我恢复或数据的异地备份、灾难性恢复等问题

软件缺陷构成

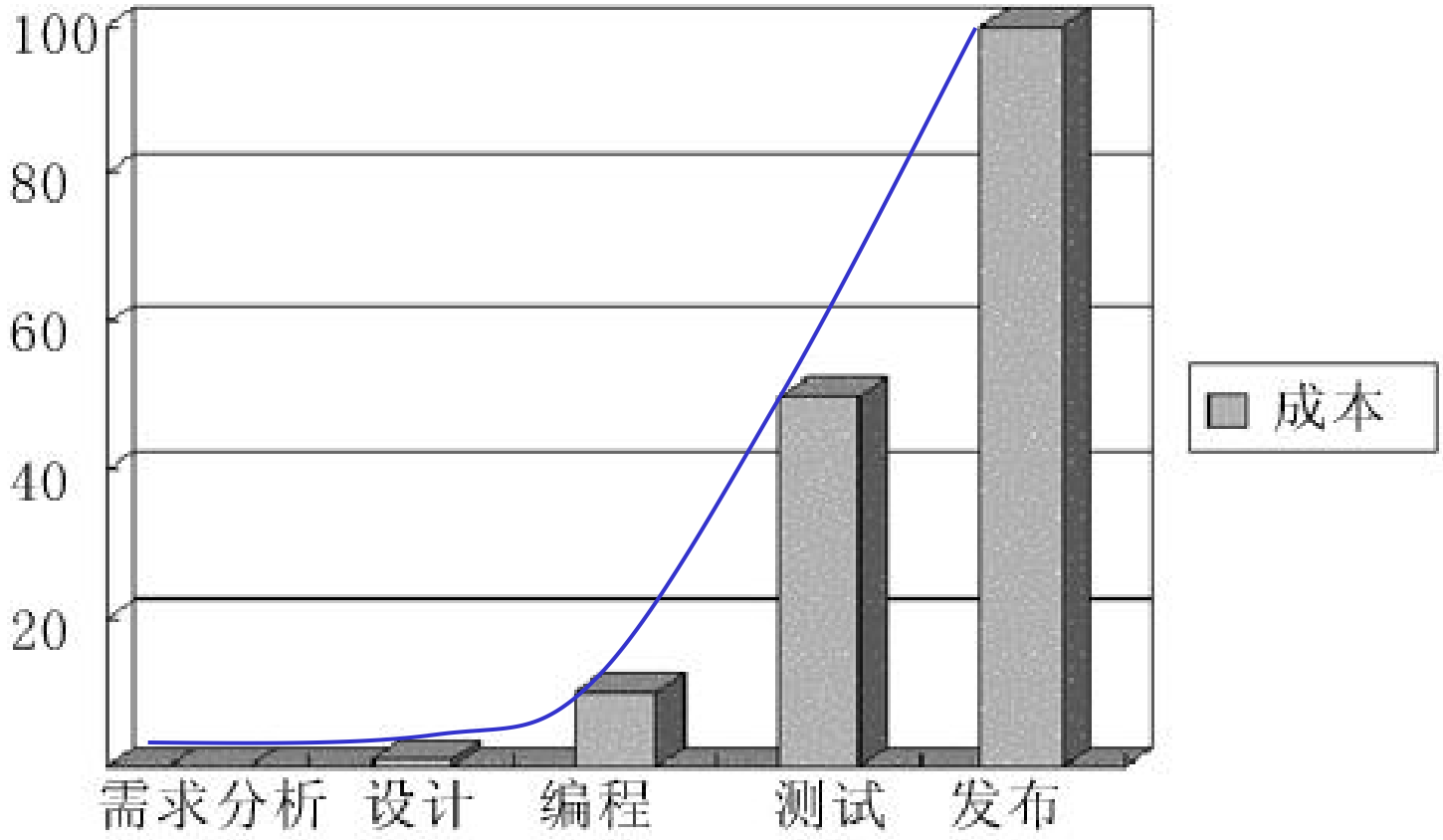


软件缺陷在不同阶段的分布

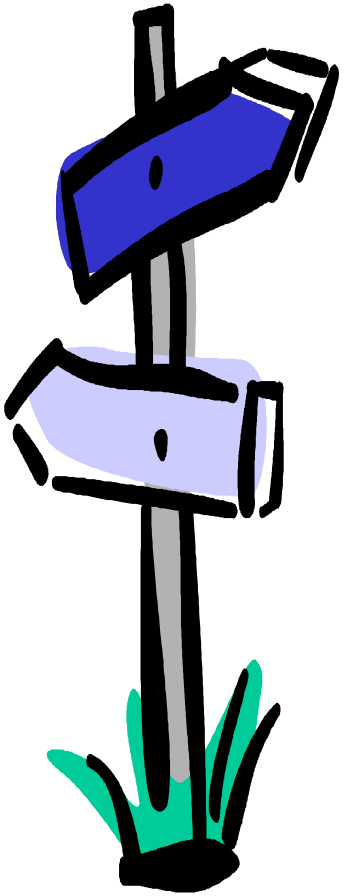


在真正的程序测试之前，通过审查、评审会可以发现更多的缺陷。规格说明书的缺陷会在需求分析审查、设计、编码、测试等过程中会逐步发现，而不能在需求分析一个阶段发现

缺陷成本



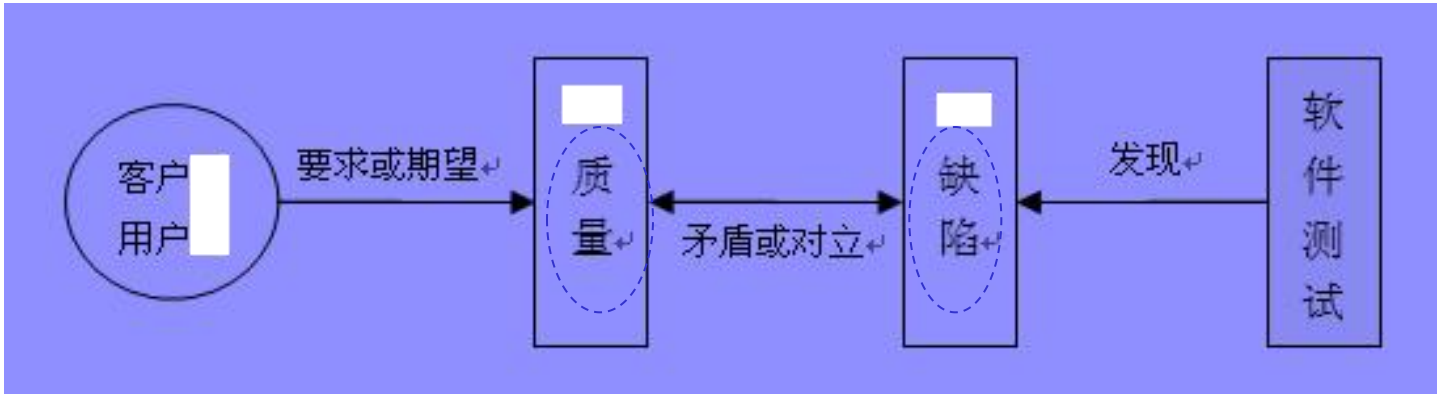
本章内容



- ✓ BUG
- ✓ 软件质量
- ✓ 三个基本术语（PIE模型）
- ✓ 软件测试基本概念
- ✓ 软件测试反思

缺陷是质量的对立面

要了解什么是缺陷(defect)，就必须清楚“质量(Quality)”概念，因为缺陷是相对质量而存在的，违背了质量、违背了客户的意愿，不能满足客户的要求，就会引起缺陷或产生缺陷



软件质量的内涵

IEEE： **质量**是系统、部件或过程满足

① 明确需求

② 客户或用户需要或期望的程度不同

- **软件质量**：软件产品具有满足规定的或隐含要求能力要求有关的特征与特征总和(ISO 8492)
- **软件质量**：软件产品满足使用要求的程度

产品质量的标准

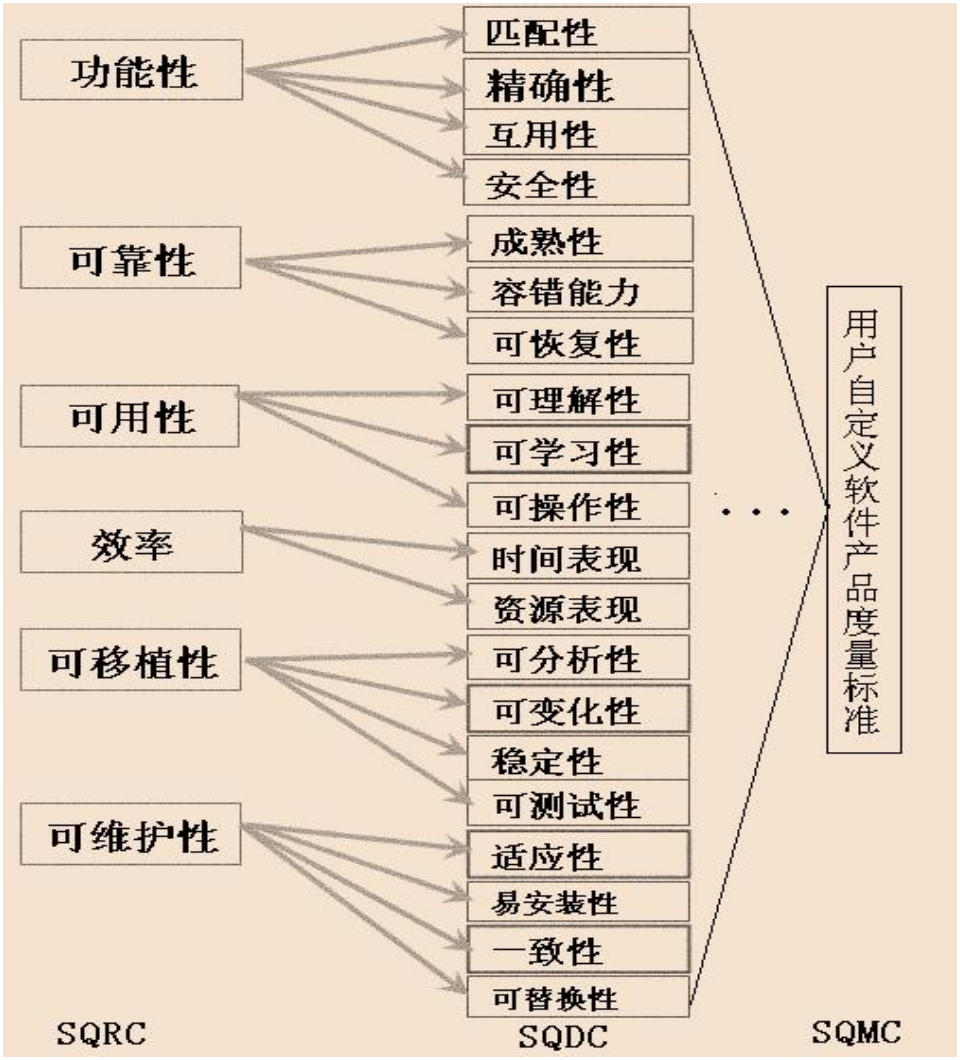
- 功能性 Functionality
- 可用性 Usability
- 可靠性 Reliability
- 性能 Performance
- 容量 Capacity
- 可伸缩性 Scalability
- 可维护性 Service manageability
- 兼容性 Compatibility
- 可扩展性 Extensibility

非功能特性

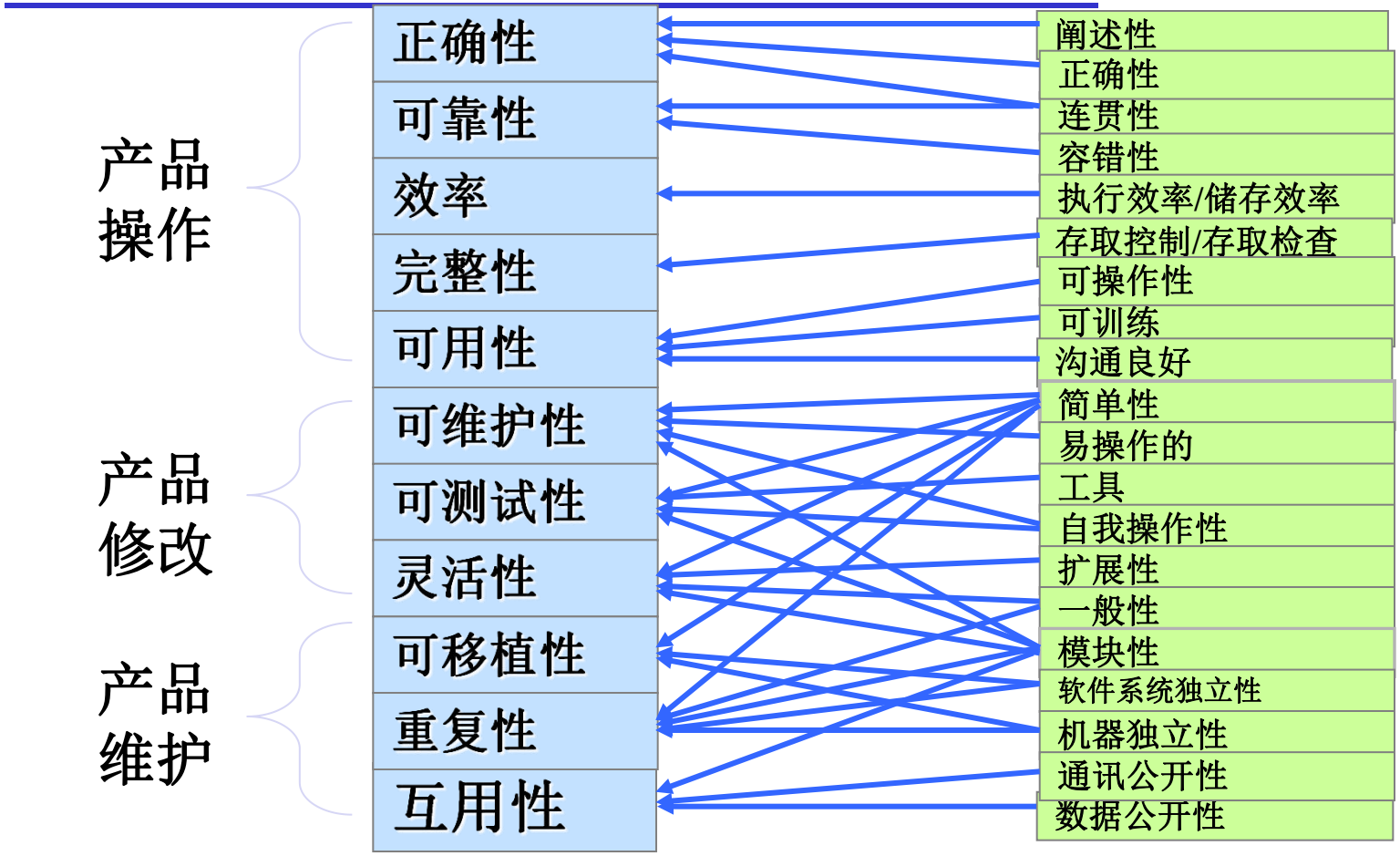
软件质量特征 (ISO 9126)

- **功能**：与一组功能及其指定性质有关的一组属性，这里的功能是满足明确或隐含的需求的那些功能。
- **可靠**：在规定的一段时间和条件下，与软件维持其性能水平的能力有关的一组属性。
- **易用**：由一组规定或潜在的用户为使用软件所需作的努力和所作的评价有关的一组属性。
- **效率**：与在规定条件下软件的性能水平与所使用资源量之间关系有关的一组属性。
- **可维护**：与进行指定的修改所需的努力有关的一组属性。
- **可移植**：与软件从一个环境转移到另一个环境的能力有关的一组属性。
  - ✓其中每一个质量特征都分别与若干子特征相对应。

ISO 9126软件质量三层模型



Boehm软件质量模型



ISO/IEC 9126-1991两个标准

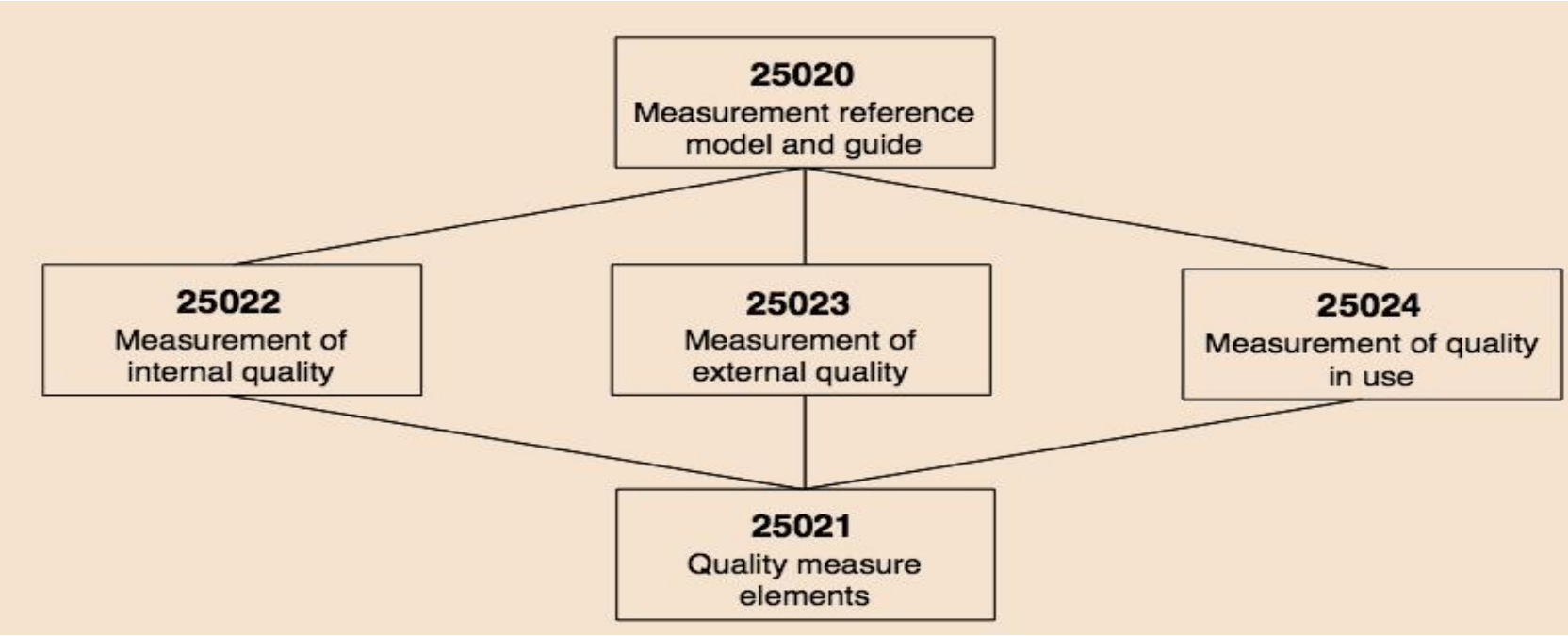
质量模型

- ✓ ISQ/IEC 9126-1:2001 《信息技术-产品质量》的第一部分《质量模型》
- ✓ ISO/IEC TR 9126-2:2003 《IT-产品质量》的第二部分《外部质量》
- ✓ ISO/IEC TR 9126-3:2003 《IT-产品质量》的第三部分《内部质量》
- ✓ ISO/IEC TR 9126-3:2003 《IT-产品质量》的第四部分《使用质量》

软件质量评价方法

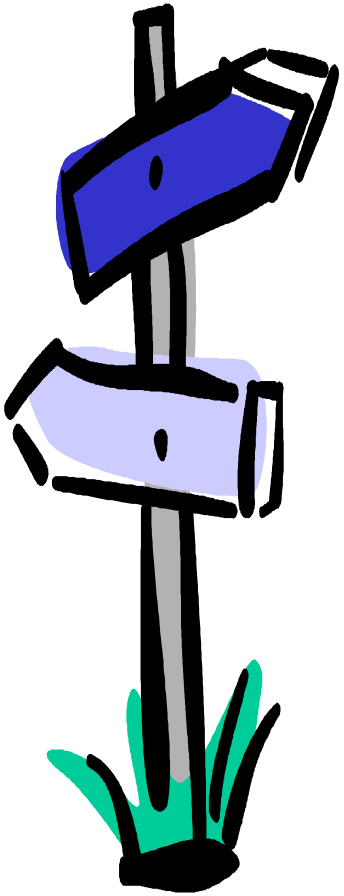
- ✓ ISO/IEC 14598-1:1999 《IT--软件产品评估-- 第一部分：综述》
- ✓ ISO/IEC 14598-2:2000 《IT--产品评估--第二部分：计划和管理》
- ✓ ISO/IEC 14598-3:2000 《IT--产品评估--第三部分：开发者过程》
- ✓ ISO/IEC 14598-4:1999 《IT--产品评估--第四部分：购买方过程》
- ✓ ISO/IEC 14598-5:1998 《IT--软件产品评估--第五部分：评估方过程》
- ✓ ISO/IEC 14598-6:2001 《IT--产品评估--第六部分：评估模型文档》

最新质量标准：ISO25000系列



ISO/IEC25000 SE-软件产品质量要求和评定(SQuaRE)

本章内容



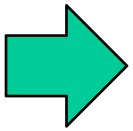
- ✓ BUG
- ✓ 软件质量
- ✓ **三个基本术语（PIE模型）**
- ✓ 软件测试基本概念
- ✓ 软件测试反思

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

Example

```
Public static void Csta(int[] numbers){
  int length=number.length;
Double mean,sum;
Sum=0.0;
For(int i=1;i<length;i++)//i=0
{
    sum+=number[i];
}
   mean=sum/(double)length;
   System.out.println(mean);
}
```



Test Input: 3,4,5

Sum=3+4+5--Error
4+5

Mean = 4 3

PIE模型

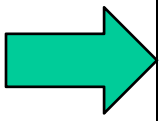
1. Execute/Reachability : The location or locations in the program that contain the fault must be reached
2. Infection : The state of the program must be incorrect
3. Propagation : The infected state must propagate to cause some output of the program to be incorrect

讨论

- ✓Execution/Reachablity : Fault 必须能到执行到
  - 一个测试未必能够执行到Fault位置
- ✓Infection :
  - 一个测试执行到Fault未必产生Error

Example

```
Public static void Csta(int[] numbers){
  int length=number.length;
Double mean,sum;
Sum=0.0;
For(int i=1;i<length;i++)//i=0
{
    sum+=number[i];
}
   mean=sum/(double)length;
   System.out.println(mean);
}
```



Test Input: 0,4,5

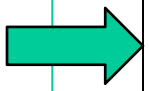
Sum=0+4+5--Error
4+5

Mean = 3 3

讨论

- ✓Propagation：
  - 一个Error未必能够产生Failure

```
Public static void Csta(int[] numbers){
  int length=number.length-1;
Double mean,sum;
Sum=0.0;
For(int i=0;i<length;i++)
{
    sum+=number[i];
}
   mean=sum/(double)length;
   System.out.println(mean);
}
```

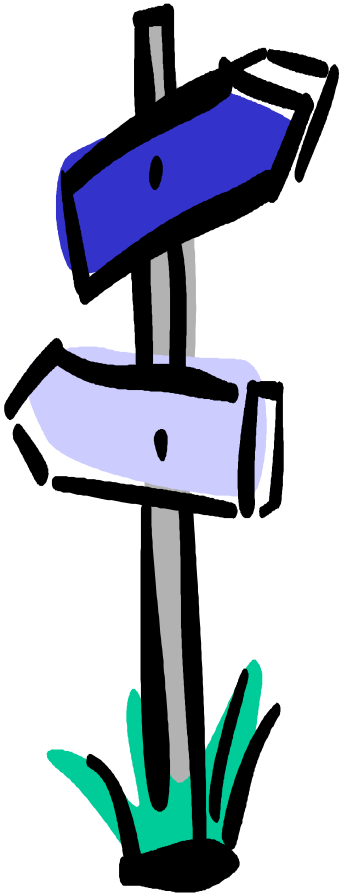


Input：3,5,4

Sum =3+5--Error

Mean=3– Not Failure

本章内容

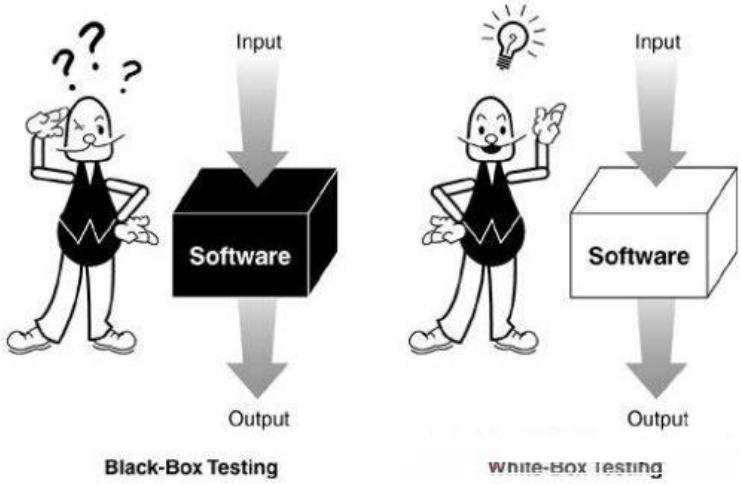


- ✓ BUG
- ✓ 软件质量
- ✓ 三个基本术语（PIE模型）
- ✓ 软件测试基本概念
- ✓ 软件测试反思

黑盒测试和白盒测试

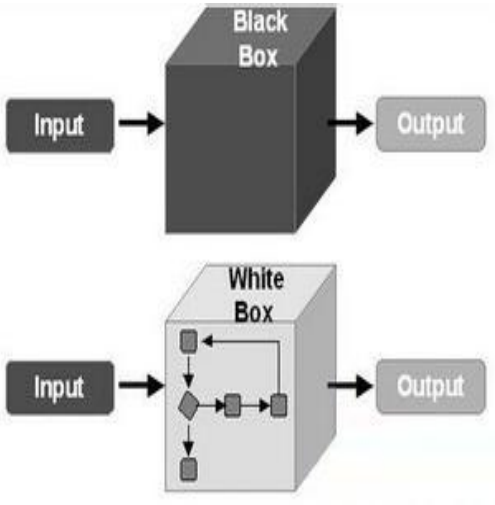
✓什么是黑盒测试

- 又称功能测试或数据驱动测试，是针对软件的功能需求/实现进行测试。穷举输入测试



✓什么是白盒测试

- 也称结构测试或逻辑驱动测试，必须知道软件内部工作过程，通过逻辑覆盖、路径覆盖等方式选择测试用例，可以用测试覆盖率评价测试用例。



动态测试和静态测试

✓动态测试

✓静态测试

Validation & Verification (*IEEE*)

- ✓ Validation : The process of evaluating software at the end of software development to ensure compliance with intended usage

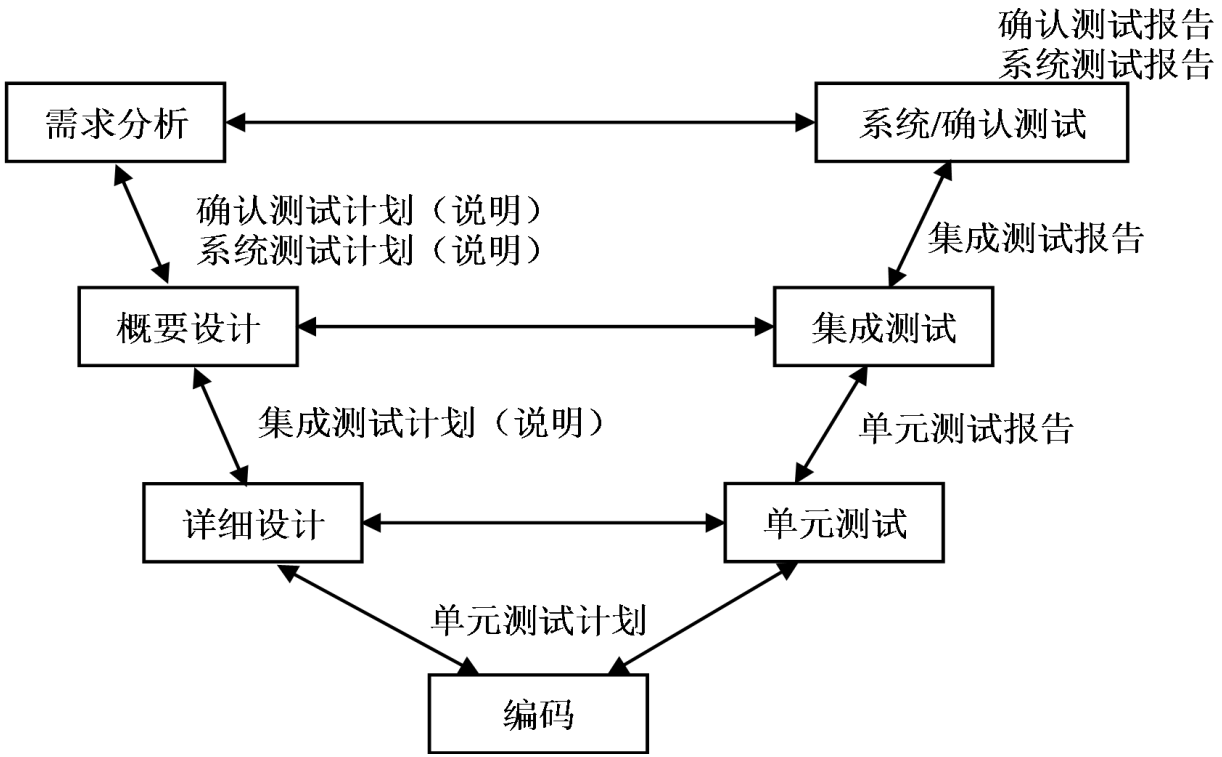
- ✓ Verification : The process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase

Test & Debug

- ✓ Testing : Finding inputs that cause the software to fail

- ✓ Debugging : The process of finding a fault given a failure

V模型示意图


确认测试报告
系统测试报告
需求分析
系统/确认测试
确认测试计划（说明）
系统测试计划（说明）
集成测试报告
概要设计
集成测试
集成测试计划（说明）
单元测试报告
详细设计
单元测试
单元测试计划
编码

单元测试

- ✓ 完成对最小的软件设计单元—模块的验证工作
- ✓ 目标是确保模块被正确地编码
- ✓ 使用过程设计描述作为指南，对重要的控制路径进行测试以发现模块内的错误
- ✓ 通常情况下是面向白盒的
- ✓ 对代码风格和规则、程序设计和结构、业务逻辑等进行静态测试，及早地发现和解决不易显现的错误

单元测试

- ✓ 单元测试的内容
  - 接口测试
  - 内部数据结构
  - 全局数据结构
  - 边界
  - 语句覆盖
  - 错误路径

单元测试

- ✓单元测试的工具
  - – OpenSource: xUnit
    - > Junit --- Java
    - > NUnit --- C#
  - – DevPartner
  - – … …

集成测试

- ✓通过测试发现与模块接口有关的问题
- ✓目标是把通过了单元测试的模块拿来，构造一个在设计中所描述的程序结构
- ✓应当避免一次性的集成（除非软件规模很小），而采用增量集成

集成测试

- ✓集成测试主要内容
  - API
  - API/参数组合
  - ……

系统测试

- ✓根据软件需求规范的要求进行系统测试，确认系统满足需求的要求
- ✓系统测试人员相当于用户代言人
- ✓在需求分析阶段要确定软件的可测性，保证有效完成系统测试工作

系统测试

- ✓系统测试主要内容
  - 所有功能需求得到满足
  - 所有性能需求得到满足
  - 其他需求（例如安全性、容错性、兼容性等）得到满足

用户验收/确认测试

- ✓配置审查
  - 确保已开发软件的所有文件资料均已编写齐全，并分类编目
- ✓Alpha测试
  - 是由用户在开发者的场所来进行的，Alpha测试是在一个受控的环境中进行的

用户验收/确认测试

- ✓Beta测试
  - 由软件的最终用户在一个或多个用户场所来进行的
  - 开发者通常不在现场，用户记录测试中遇到的问题并报告给开发者
  - 开发者对系统进行最后的修改，并开始准备发布最终的软件

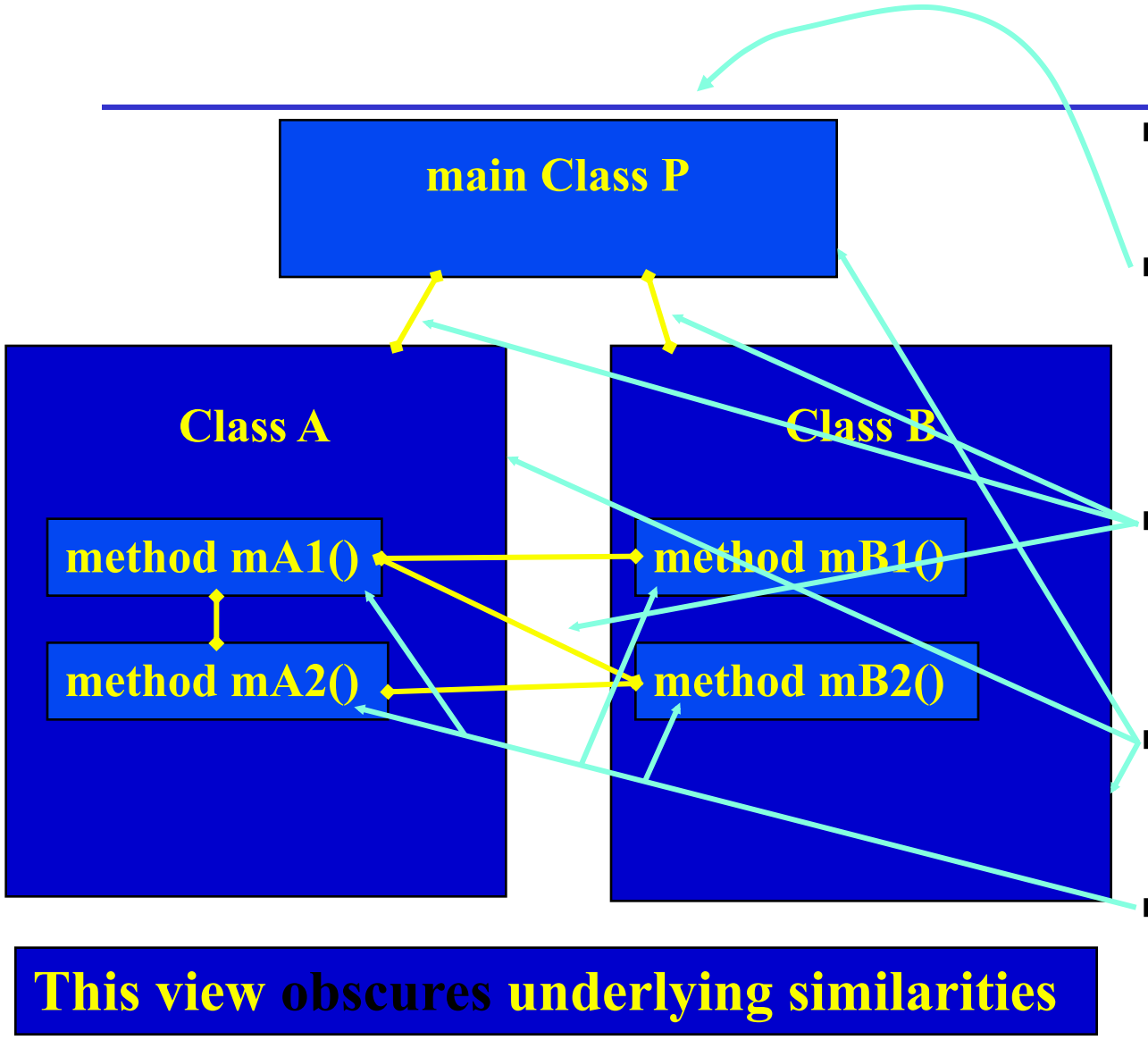
回归测试

- ✓当发现并修改缺陷后，或者在软件中添加新功能后，重新测试，用来检查被发现的缺陷是否被改正，并且所作的修改没有引发新的问题
- ✓回归测试可以通过人工重新执行测试用例，也可以使用自动化的捕获回放工具来进行

回归测试

- ✓回归测试方式
  - 再测试全部用例
    - > 选择基线测试用例库中的全部测试用例组成回归测试包，测试成本最高
  - 基于风险选择测试
    - > 可以基于一定的风险标准来从基线测试用例库中选择回归测试包

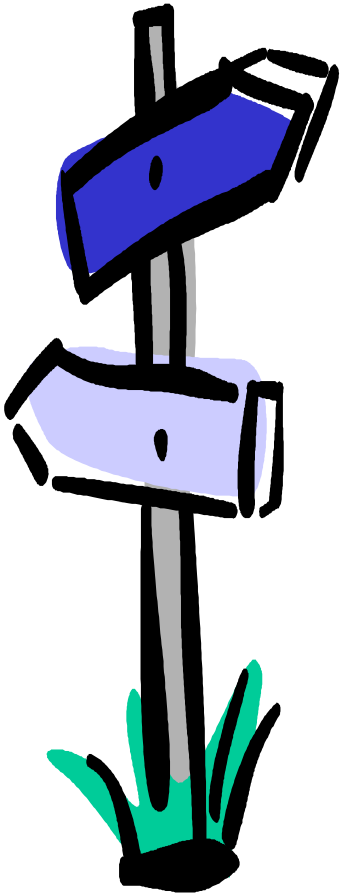
软件测试分类层次



- Acceptance testing: Is the software acceptable to the user?
- System testing: Test the overall functionality of the system
- Integration testing: Test how modules interact with each other
- Module testing: Test each class, file, module or component
- Unit testing: Test each unit (method) individually

This view obscures underlying similarities

本章内容



- ✓ BUG
- ✓ 软件质量
- ✓ 三个基本术语（PIE模型）
- ✓ 软件测试基本概念
- ✓ 软件测试反思

Definitions of Bug

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

Bug

How to confirm a fault?

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

Test
Input: 3, 5
Expected output: 5
Output: **3**
Fail!

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x < y)
5     mx = y;
6  return mx;
7 }
```

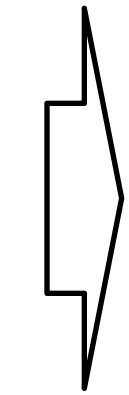
Test
Input: 3, 5
Expected output: 5
Output: **5**
Pass!

Yes, this is a bug!

Fault is defined by fixing!

Which one is a fault?

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```



Minimal Fixing

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x < y)
5     mx = y;
6  return mx;
7 }
```

Test
Input: 3, 5
Expected output: 5
Output: **5**
Pass!

```
1 int max(int x, int y)
2 {
3 int mx = y;
4 if (x > y)
5     mx = x;
6  return mx;
7 }
```

Test
Input: 3, 5
Expected output: 5
Output: **5**
Pass!

Fault is defined by testing!

It is not a Fault!

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

Test 1
Input: 3, 5
Expected output: 5
Output: **3**
Fail!

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = x;
6  return mx;
7 }
```

Test 1
Input: 3, 5
Expected output: 5
Output: **3**
Fail!

It is a fault!

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

Test 2
Input: 5, 3
Expected output: 5
Output: **3**
Fail!

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = x;
6  return mx;
7 }
```

Test 2
Input: 5, 3
Expected output: 5
Output: **5**
Pass!

One or Two?

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

Test 1
Input: 3, 5
Expected output: 5
Output: **3**
Fail!

Two Faults!

```
1 int max(int x, int y)
2 {
3 int mx = y;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

```
1 int max(int x, int y)
2 {
3 int mx = y;
4 if (x > y)
5     mx = x;
6  return mx;
7 }
```

Test 1
Input: 3, 5
Expected output: 5
Output: **5**
Pass!

Test 2
Input: 5, 3
Expected output: 5
Output: **3**
Fail!

Test 2
Input: 5, 3
Expected output: 5
Output: **5**
Pass!

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

Test 1
Input: 5, 3
Expected output: 5
Output: **3**
Fail!

One Fault!

```
1 int max(int x, int y)
2 {
3 int mx = y;
4 if (x > y)
5     mx = y;
6  return mx;
7 }
```

Test 1
Input: 5, 3
Expected output: 5
Output: **3**
Fail!

```
1 int max(int x, int y)
2 {
3 int mx = y;
4 if (x > y)
5     mx = x;
6  return mx;
7 }
```

Test 1
Input: 5, 3
Expected output: 5
Output: **5**
Pass!

Fault Interference

```
1 int max(int x, int y, int z)
2 {
3  if (x > y)
4    y = x;
5  if (y > z)
6   z=y;
7  return z;
8 }
```

Input: 2, 5, 3
Output: **5**
Pass!

Input: 2, 3, 5
Output: **5**
Pass!

```
1 int max(int x, int y, int z)
2 {
3  if (x > y)
4    y = x;
5  if (y < z)
6   z=y;
7  return y;
8 }   -------2Fault
```

Input: 2, 5, 3
Output: **5**
Pass!

Input: 2, 3, 5
Output: **3**
Fail!

```
1 int max(int x, int y, int z)
2 {
3  if (x > y)
4    y = x;
5  if (y < z)
6   z=y;
7  return z;
8 }   ---------1 Fault
```

Input: 2, 5, 3
Output: **3**
Fail!

Input: 2, 3, 5
Output: **3**
Fail!

Fault

Homework

- ✓Please construct a simple program P (with a fault) and 3 tests (t1, t2, t3), s.t.
  - t1 executes the fault, but no error
  - t2 (executes the fault and) produces an error, but no failure
  - t3 produces a failure