

软件质量保证与测试

Software Quality Assurance and Testing

第 1 章 绪论

1.2 软件缺陷、软件错误、软件故障



金陵科技学院

第一个bug

9/9

First Computer Bug!

0800 Antan started
 1000 " stopped - antan ✓ { 1.2700 9.037 847 025
 1300 (032) HP-AC ~~1.30476415~~ 9.037 846 895 console
 (033) PRO 2 2.130476415 4.615925059(-2)
 console 2.130676415
 Relays 6-2 in 033 failed special speed test
 in relay. remove test.
 Relays changed
 1100 Started Cosine Tape (Sine check)
 1525 Started Multi Adder Test.
 1545  Relay #70 Panel F
 (moth) in relay.
 First actual case of bug being found.
 1600 Antan started.
 1700 closed down.

Relay 2145
 bug 2375

Grace Hopper
 Photo Courtesy of Hagley Museum and Library



Grace Hopper



软件缺陷



- ❖ 缺陷是存在于软件（文档、数据、程序）之中的那些不希望或不可接受的偏差。缺陷的存在会导致软件产品在某种程度上不能满足用户的需要。
- ❖ **IEEE729-1983**对缺陷有一个标准的定义：从产品内部看，缺陷是软件产品开发或维护过程中存在的错误、毛病等各种问题；从产品外部看，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

- ❖ 软件出现了产品说明书指明不会出现的错误
- ❖ 软件未达到产品说明书的功能
- ❖ 软件功能超出产品说明书指明范围
- ❖ 软件未达到产品说明书虽未指出但应达到的目标
- ❖ 软件难以理解、不易使用、运行速度缓慢，最终用户认为不好



软件缺陷的多种情况

Soft



Soft'

软件产品要求

实际软件

软件缺陷的多种情况

Soft

软件产品要求



^{1错误}
S0 t'

实际软件

软件缺陷的多种情况

Soft

软件产品要求



1错误
Soft'
2缺少

实际软件

软件缺陷的多种情况

Soft

软件产品要求



1错误
S0 t'
2缺少
3多余

实际软件

软件缺陷的多种情况

Soft

软件产品要求



4未指出
但应有

1错误

SO t'

2缺少

3多余

实际软件

软件缺陷的多种情况

Soft



1错误
S O t'
2缺少
3多余

软件产品要求

4未指出
但应有

实际软件

5用户不满意！

软件缺陷产生的原因

- ◆ 软件自身的特点
- ◆ 团队合作
- ◆ 技术问题
- ◆ 管理问题



PIE 模型

缺陷 **Fault:**

指静态存在于程序中的错误代码行

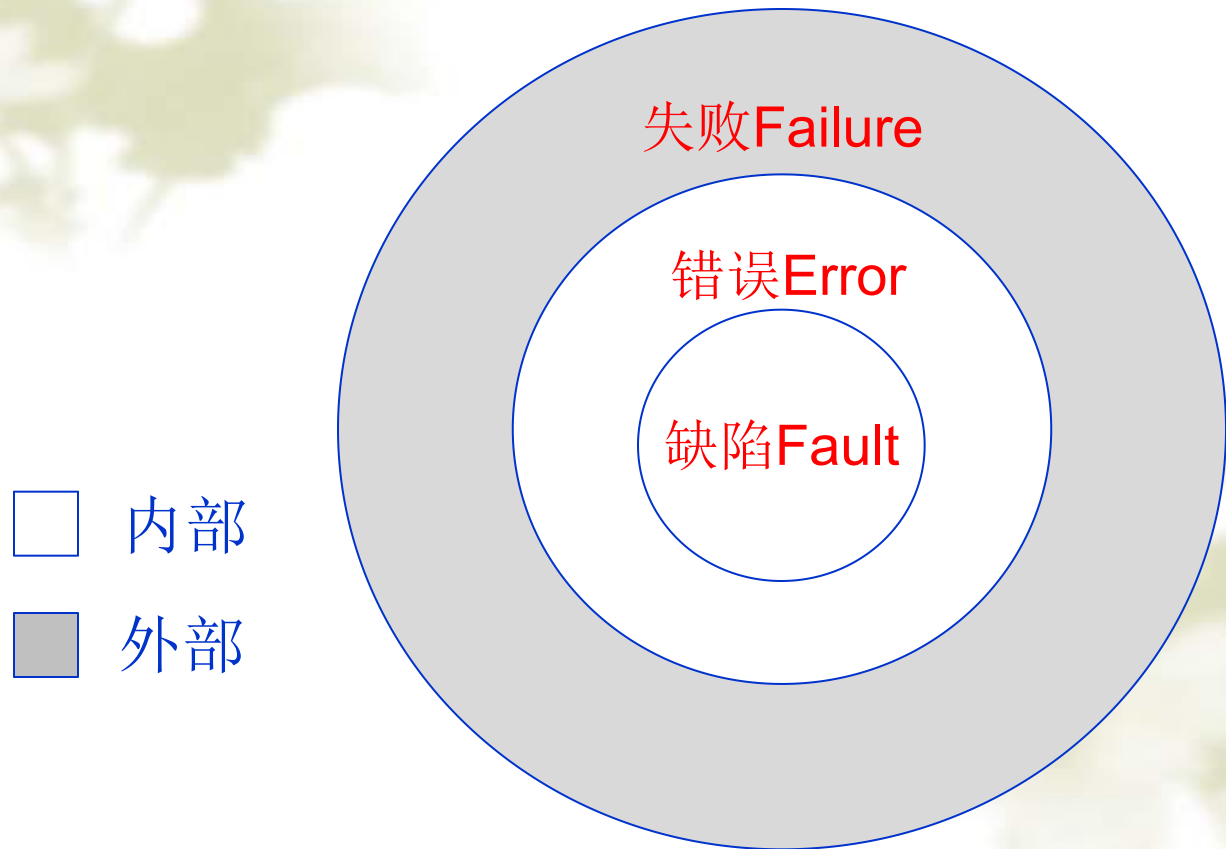
错误 **Error:**

指执行错误代码后导致的内部错误状态

失败 **Failure:**

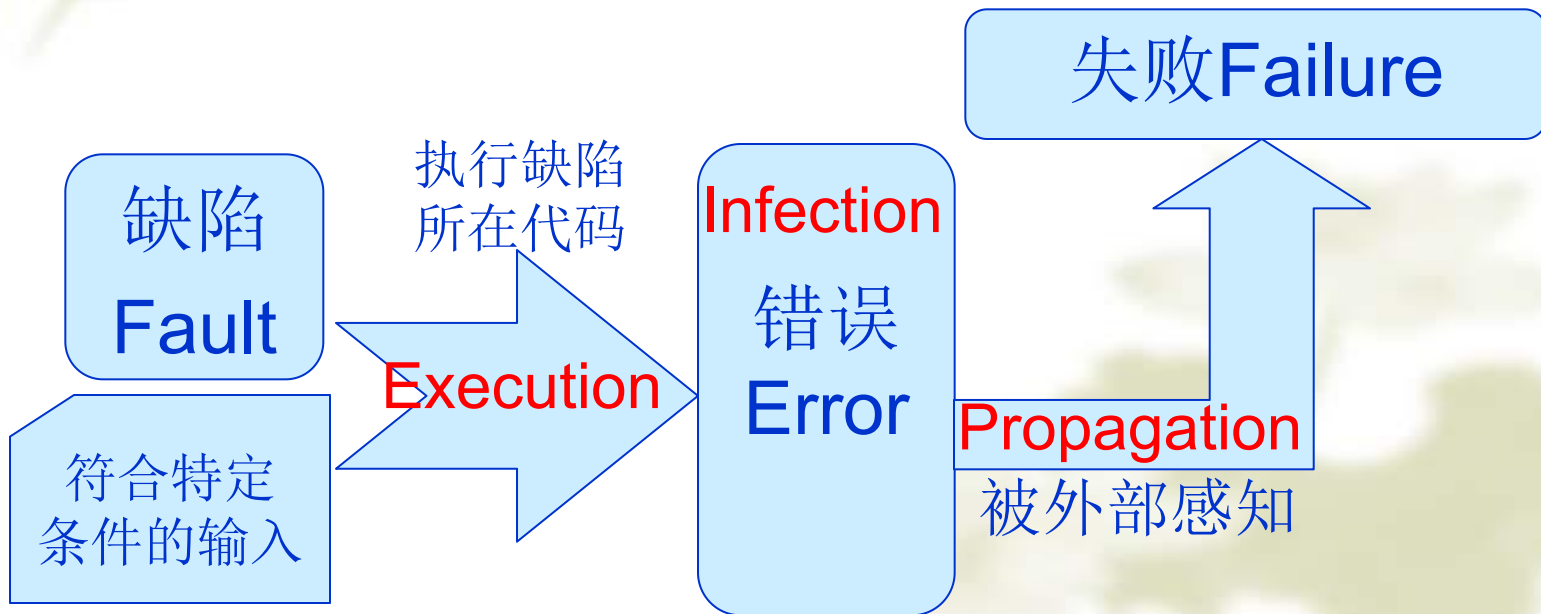
指错误状态传播到软件外部被外部感知

PIE 模型



PIE 模型

PIE = Propagation + Infection + Execution



PIE 模型：案例

```
public static void MY_AVG (int [ ] numbers)
{
    int length = numbers.length;
    double V_avg, V_sum;

    V_sum = 0.0;
    for (int i = 1; i < length; i++)
    {
        V_sum += numbers [ i ];
    }

    V_avg = V_sum / (double) length;
    System.out.println ("V_avg: " + V_avg);
}
```

PIE 模型：案例

```
public static void MY_AVG (int [ ] numbers)
{
    int length = numbers.length;
    double V_avg, V_sum;

    V_sum = 0.0;
    for (int i = 1; i < length; i++) //缺陷Fault
    {
        V_sum += numbers [ i ];
    }

    V_avg = V_sum / (double) length;
    System.out.println ("V_avg: " + V_avg);
}
```

PIE 模型：案例

```
public static void MY_AVG (int [ ] numbers)
{
    int length = numbers.length;
    double V_avg, V_sum;

    V_sum = 0.0;
    for (int i = 1; i < length; i++)
    {
        V_sum += numbers [ i ];
    }

    V_avg = V_sum / (double) length;
    System.out.println ("V_avg: " + V_avg);
}
```

情况1:

没有对 **MY_AVG**
的调用，缺陷代码没有
被执行到。

PIE 模型：案例

```
public static void MY_AVG (int [ ] numbers)
{
    int length = numbers.length;
    double V_avg, V_sum;

    V_sum = 0.0;
    for (int i = 1; i < length; i++)
    {
        V_sum += numbers [ i ];
    }

    V_avg = V_sum / (double) length;
    System.out.println ("V_avg: " + V_avg);
}
```

情况2:

测试数据: 3,4,5

缺陷Fault



错误Error



失败Failure

PIE 模型：案例

```
public static void MY_AVG (int [ ] numbers)
{
    int length = numbers.length;
    double V_avg, V_sum;

    V_sum = 0.0;
    for (int i = 1; i < length; i++)
    {
        V_sum += numbers [ i ];
    }

    V_avg = V_sum / (double) length;
    System.out.println ("V_avg: " + V_avg);
}
```

情况3:

测试数据: 0,1,2

缺陷Fault



错误Error



失败Failure

测试设计

测试设计要做的重要工作之一，就是如何恰当的设计测试数据，使得可能存在的软件缺陷**Fault**通过程序执行都尽可能的产生失败**Failure**并被外部观察到。

