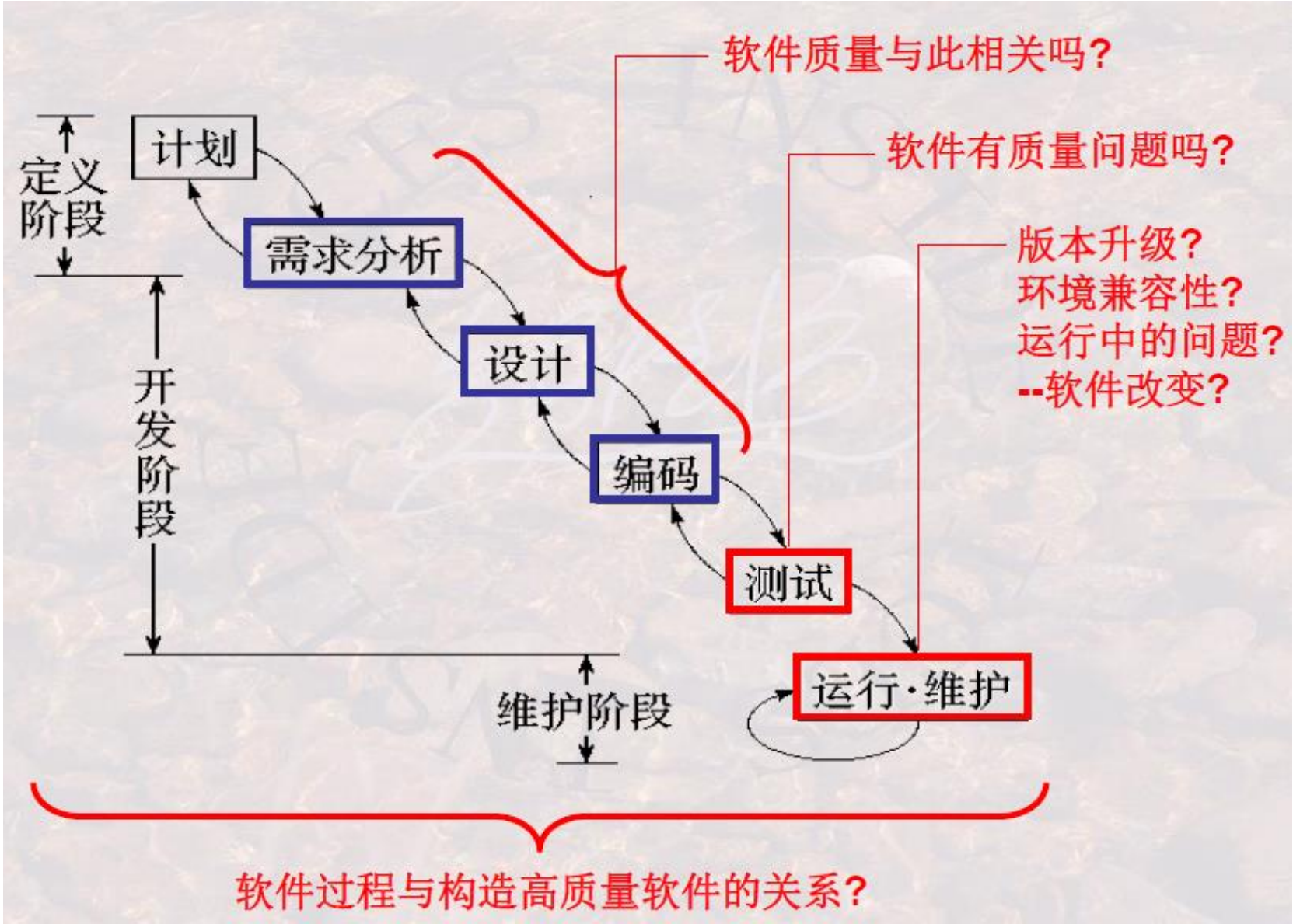
软件工程专业导论

软件测试

学习目标：

- 软件质量的重要性
- 软件测试技术


软件质量与此相关吗?
软件有质量问题吗?
版本升级?
环境兼容性?
运行中的问题?
--软件改变?
定义阶段
计划
需求分析
设计
编码
测试
运行·维护
开发阶段
维护阶段
软件过程与构造高质量软件的关系?

1. 软件质量

- 软件缺陷导致严重后果的典型案例
  - 1996年6月4日，Ariane 5火箭发射后仅仅37秒，火箭偏离它的飞行路径，解体并爆炸了。火箭上载有价值5亿美元的通信卫星。失事调查报告指出，火箭爆炸是因为：
  - **During execution of a data conversion from 64-bit floating point to 16-bit signed integer value, the floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error.**
  - 阿5型的发射系统代码直接重用了阿4型的相应代码，而阿4型的飞行条件和阿5型的飞行条件截然不同

1. 软件质量

- 软件缺陷导致严重后果的典型案例
  - 辐射剂量超标的事故发生在2000年的巴拿马城（巴拿马首都）。从美国Multidata公司引入的治疗规划软件，其（辐射剂量的）预设值有误。有些患者接受了超标剂量的治疗，至少有5人死亡。后续几年中，又有21人死亡，但很难确定这21人中到底有多少人是死于本身的癌症，还是辐射治疗剂量超标引发的不良后果。

1. 软件质量

- 什么是软件质量？
  - 质量是产品的一 组固有特性满足要求的程度 --- ISO 9000
  - 软件质量是软件产品满足规定的和隐含的与需求能力有关的特征和特性的全体 [IEEE标准]。所有描述计算机软件优秀程度的 特性组合。[M.J. Fisher]

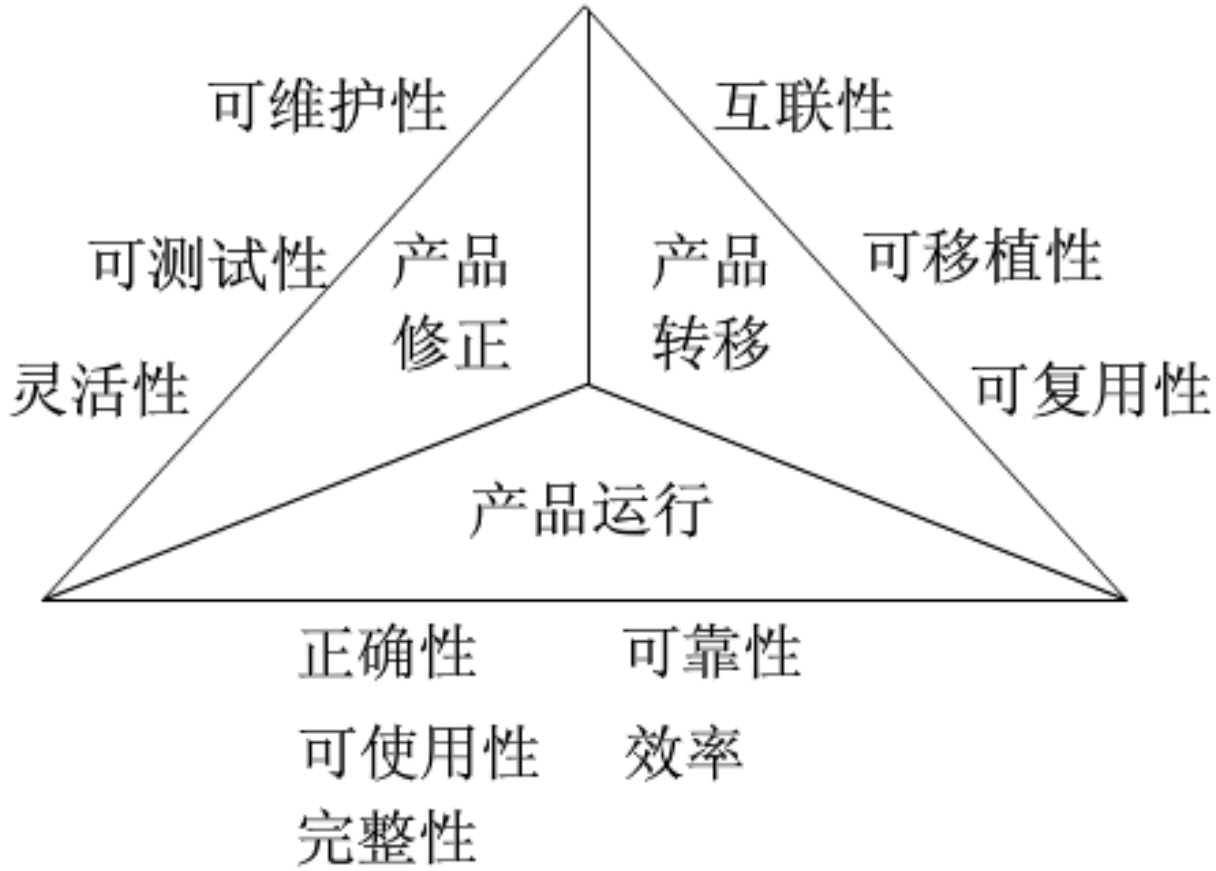
1. 软件质量

- 什么是软件质量？
  - 软件消费者观点
    - 适合使用： 产品或服务应该同被期望的相符
    - 设计质量： 设计的质量特性应包含在产品或服务中
    - 用户满意度 = 合格的产品 + 好的质量 + 按预算和进度安排交付
  - 软件生产者观点
    - 质量的一致性： 确保产品或服务是根据设计制造的
    - 利润： 确保用最少的成本投入获取大利益

1. 软件质量

- 软件质量模型
  - McCall 软件质量模型（1977 年）
    - 起源于美国空军，主要面向的是系统开发人员和系统开发过程。McCall 试图通过一系列的软件质量属性指标来弥补开发人员与最终用户之间的沟壑
    - 从三个视角定义和识别软件产品的质量
      - Product revision（产品修正）：改变的能力
      - Product transition（产品转移）：适应新环境的能力
      - Product operations（产品运行）：基本的操作特性

1. 软件质量

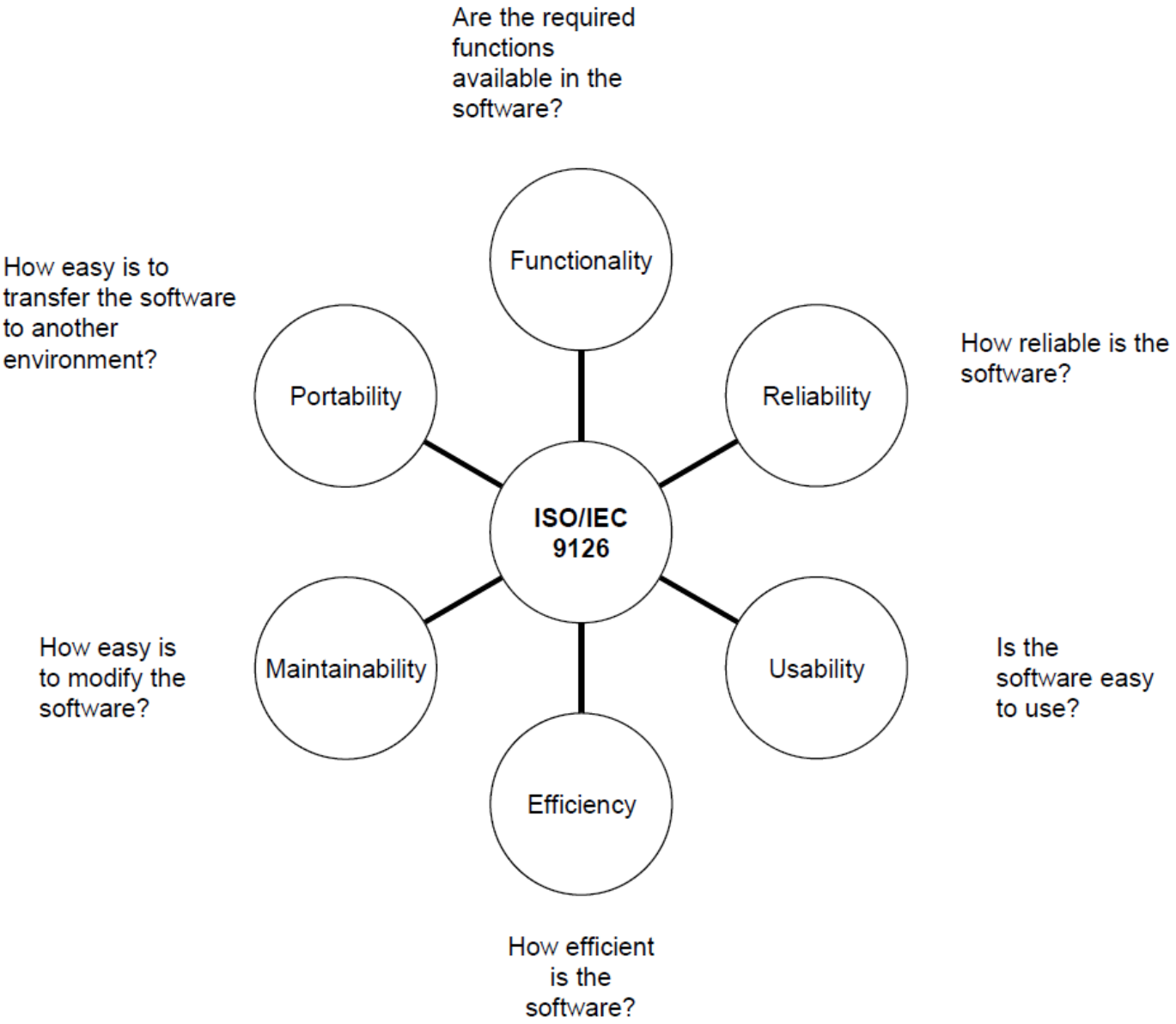


1. 软件质量

- 软件质量模型
  - Boehm 软件质量模型（1978 年）
    - Boehm 软件质量模型试图通过一系列的属性的指标来量化软件质量。Boehm 的质量模型包含了 McCall 模型中没有的硬件属性。Boehm 模型也类似于 McCall 的质量模型，采用层级的质量模型结构，包括高层属性、中层属性和原始属性

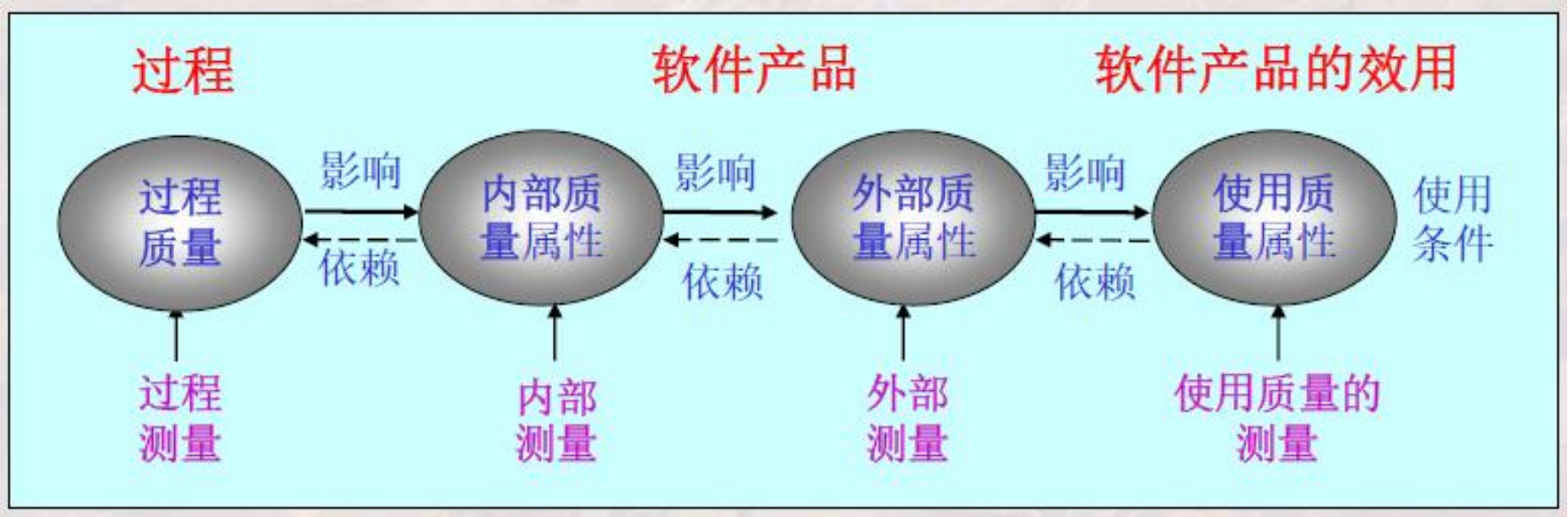
1. 软件质量

- 软件质量模型
  - ISO/IEC 9126 软件质量模型（1993 年）
    - ISO/IEC 9126 模型是建立在 McCall 和 Boehm 模型之上的，同时加入了功能性要求，还包括识别软件产品的内部和外部质量属性


Are the required functions available in the software?
Functionality
How easy is to transfer the software to another environment?
Portability
Reliability
How reliable is the software?
ISO/IEC 9126
How easy is to modify the software?
Maintainability
Usability
Is the software easy to use?
Efficiency
How efficient is the software?

1. 软件质量

- 软件质量的生命周期



1. 软件质量

- 软件质量保证
  - 软件质量保证，是生命周期内，为了确定、达到和维护需要的软件质量而进行所有有计划、有系统的管理活动
  - 提高软件质量的一般方法是：尽早发现、及时纠正

2. 软件测试

- Software is buggy
  - 据统计美国每年由于bug造成的损失高达600亿美元
  - 平均每1000行代码有1-5个bug
  - 100%正确的软件产品是不可能的

2. 软件测试

失败（failure）：可观测到的不正确行为

故障（fault、bug）：不正确的代码

错误（error）：导致故障的原因

2. 软件测试

```
1.  int doubleValue(int i) {
2.      int result;
3.      result = i * i;
4.      return result;
5.  }
```

- 调用doubleValue(3); 返回9，是一个____?

  - ☐ a failure
  - ☐ a fault
  - ☐ an error

2. 软件测试

```
1.  int doubleValue(int i) {
2.      int result;
3.      result = i * i;
4.      return result;
5.  }
```

- 调用doubleValue(3); 返回9，是一个____?

  - [√] a failure
  - [ ] a fault
  - [ ] an error

2. 软件测试

```
1.  int doubleValue(int i) {
2.      int result;
3.      result = i * i;
4.      return result;
5.  }
```

- 导致failure的fault在程序第____行

2. 软件测试

```
1.  int doubleValue(int i) {
2.      int result;
3.      result = i * i;
4.      return result;
5.  }
```

- 导致failure的fault在程序第__3__行

2. 软件测试

```
1.  int doubleValue(int i) {
2.      int result;
3.      result = i * i;
4.      return result;
5.  }
```

- 导致fault的error是什么？

2. 软件测试

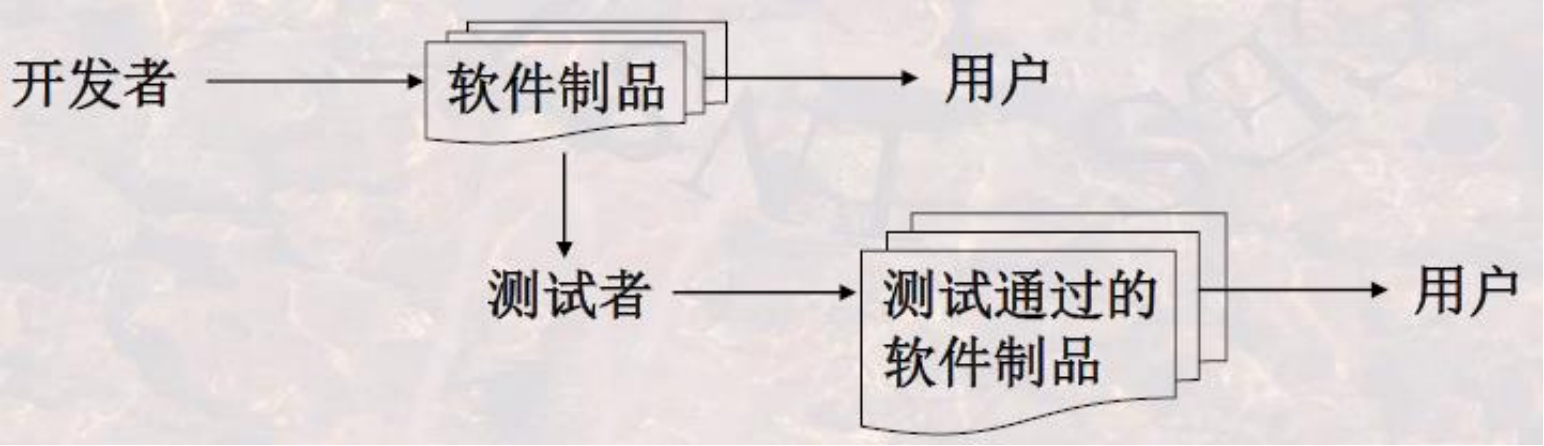
```
1.  int doubleValue(int i) {
2.      int result;
3.      result = i * i;
4.      return result;
5.  }
```

- 导致fault的error是什么？

不知道！（键盘输入错误、理解错误、…，只有开发者知道）

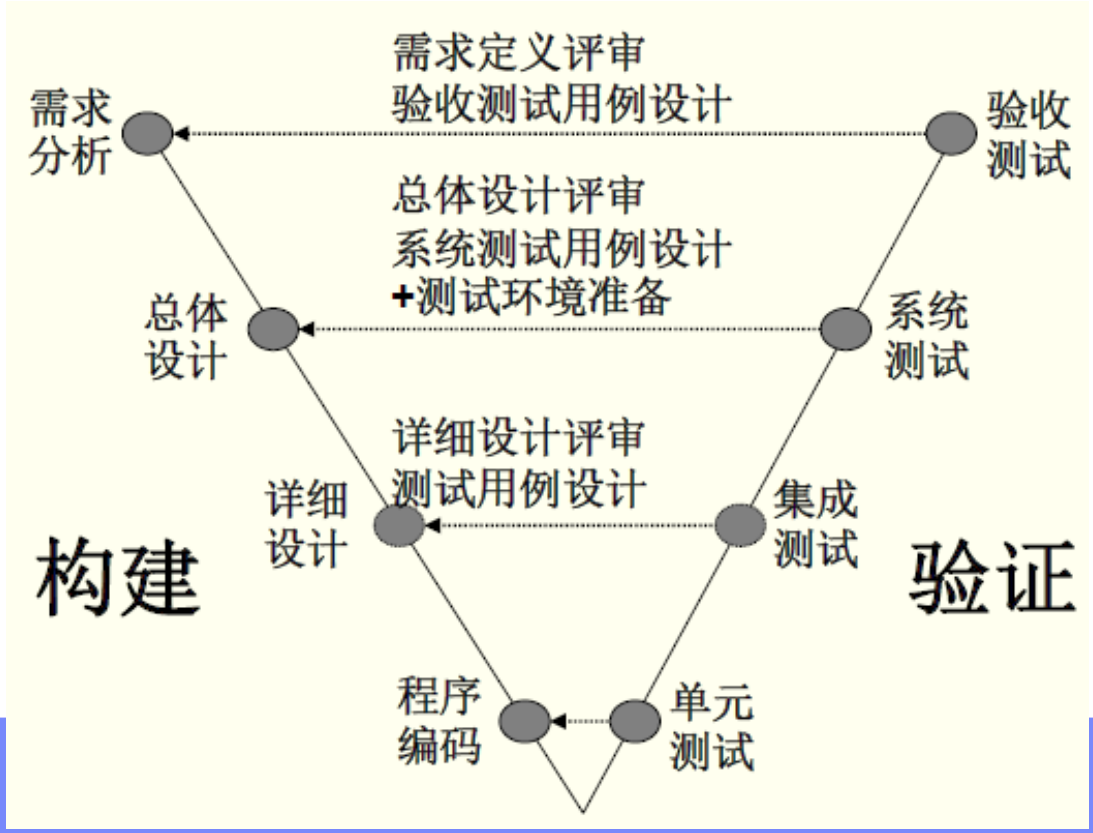
2. 软件测试

- 什么是软件测试？
  - IEEE：测试是使用人工和自动手段来运行或检查某个系统的过程，其目的在于检验系统是否满足规定需求或弄清预期结果与实际结果之间的差别



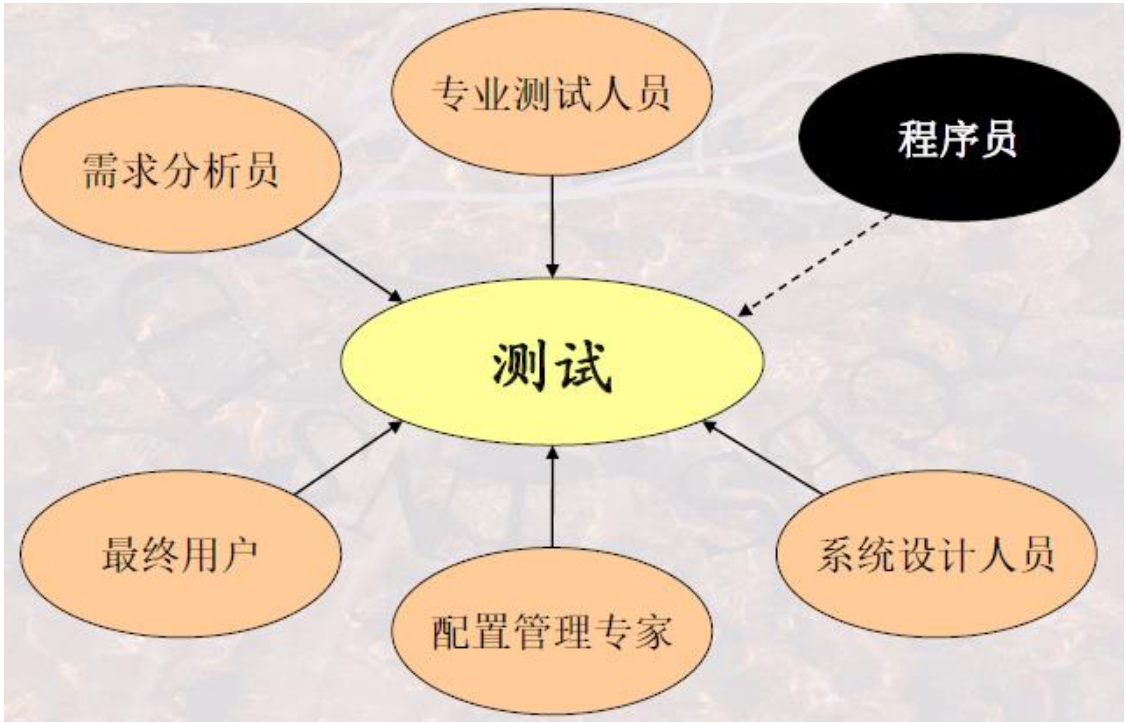
2. 软件测试

- 什么是软件测试？
  - 软件测试并不等于程序测试，应贯穿于开发的各个阶段



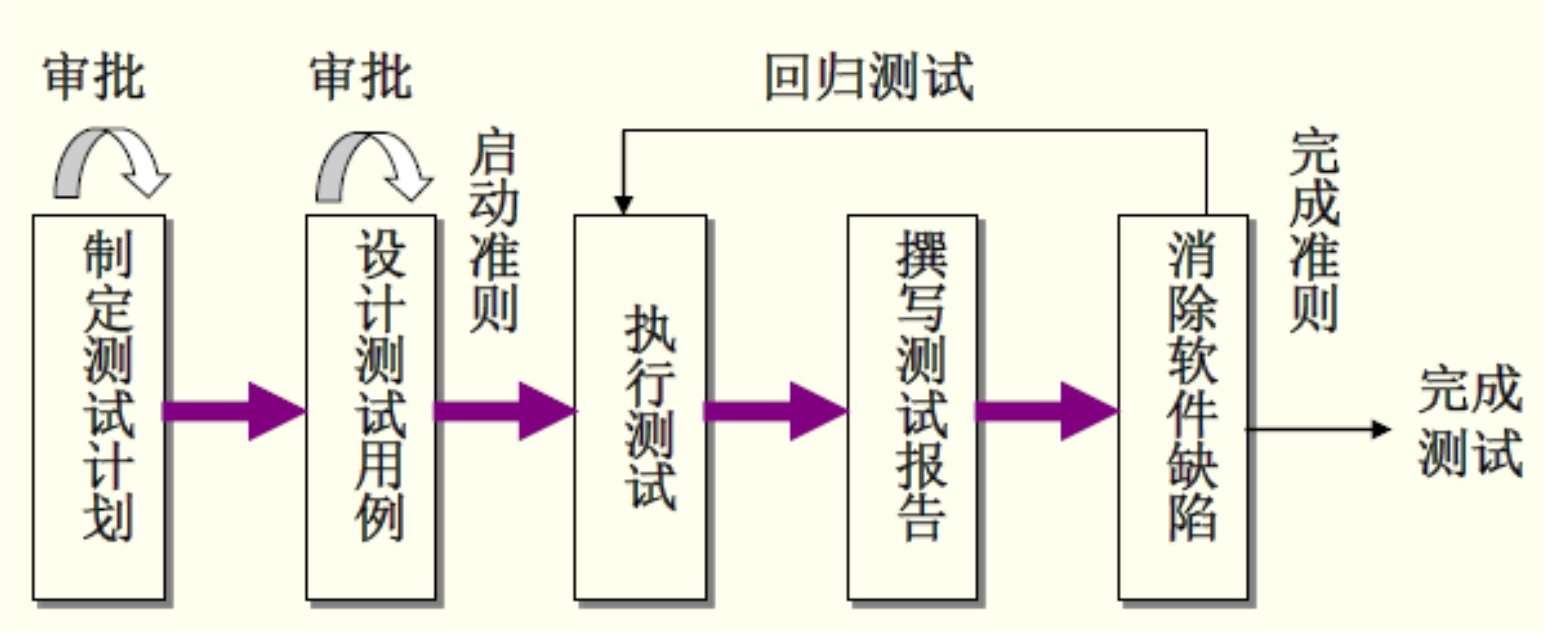
2. 软件测试

- 什么是软件测试？
  - 需要各类人员的参与，不应由程序员独立完成



2. 软件测试

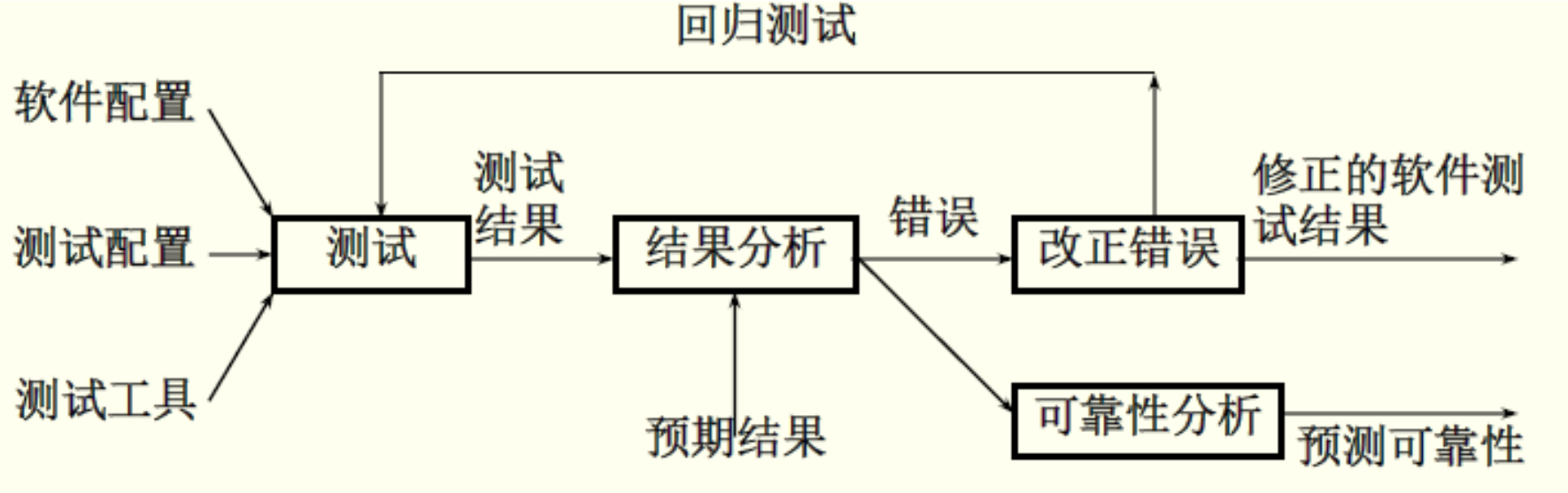
- 测试流程：
  - 制定测试计划、设计测试用例、执行测试、撰写测试报告



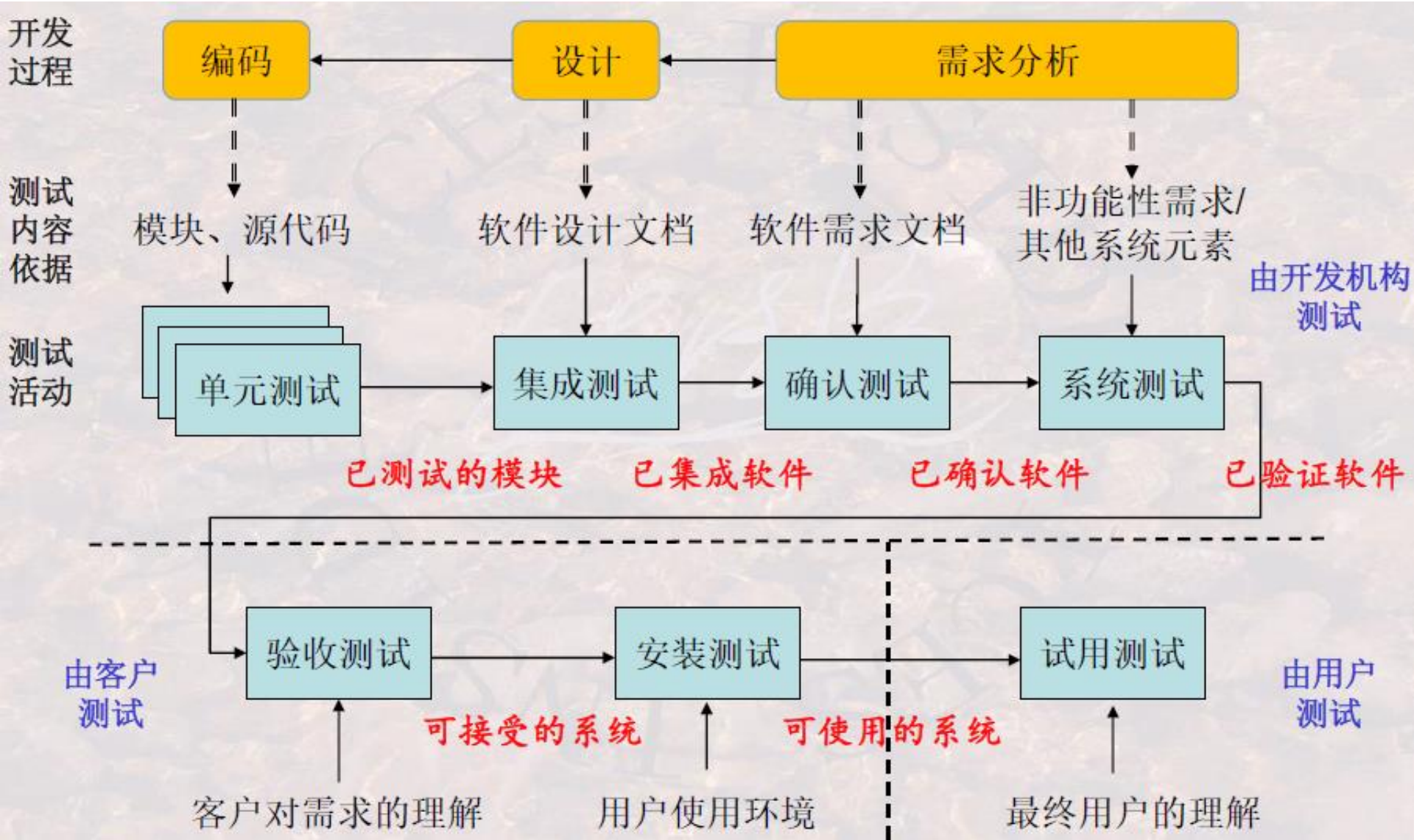
2. 软件测试

- 软件测试输入信息
  - 软件配置：软件需求规格说明、设计规格说明、源代码等
  - 测试配置：测试计划、测试用例、测试驱动程序
  - 测试工具：动静态分析工具、自动化测试工具、测试结果分析工具等
- 软件测试输出信息
  - 测试结果：对其进行分析，同预期结果比较

2. 软件测试

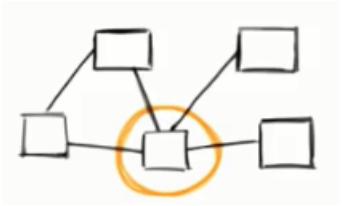

回归测试
软件配置
测试配置
测试工具
测试
测试结果
结果分析
错误
改正错误
修正的软件测试结果
预期结果
可靠性分析
预测可靠性

2. 软件测试

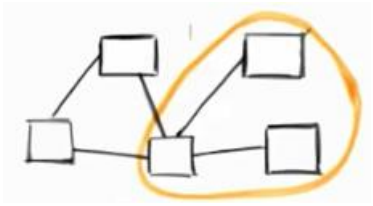

开发过程
编码
设计
需求分析
测试内容依据
模块、源代码
软件设计文档
软件需求文档
非功能性需求/其他系统元素
由开发机构测试
测试活动
单元测试
集成测试
确认测试
系统测试
已测试的模块
已集成软件
已确认软件
已验证软件
由客户测试
验收测试
安装测试
试用测试
由用户测试
可接受的系统
可使用的系统
客户对需求的理解
用户使用环境
最终用户的理解

2. 软件测试

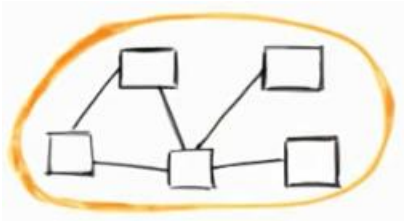
- 测试粒度



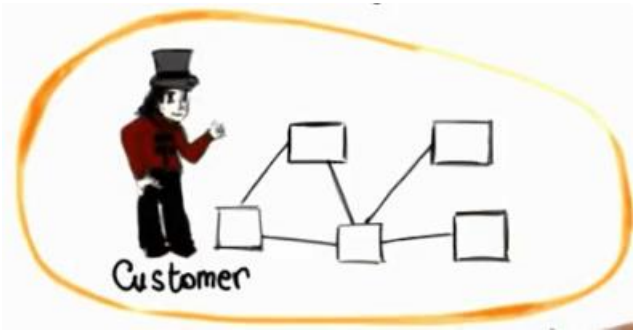
单元测试
Unit Testing



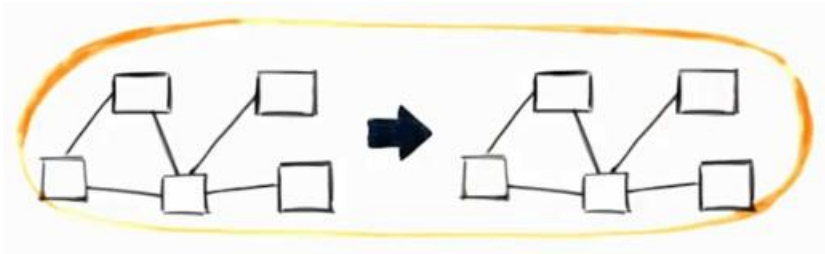
集成测试
Integration Testing



系统测试
System Testing



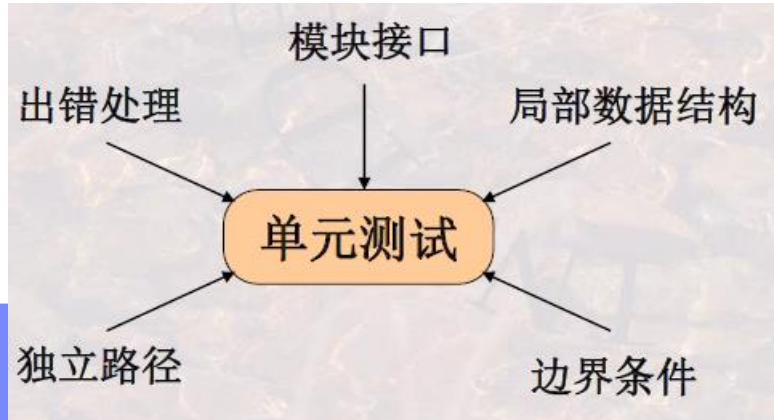
可接受测试
Acceptance Testing



回归测试
Regression Testing

2. 软件测试

- 单元测试
  - 确保产生符合需求的可靠程序单元（结构化为模块，面向对象程序为类）
  - 单元测试需要频繁重复运行，需要工具提高效率
  - 以单元测试工具 Junit 为例：其提供一套完整的测试框架和工具，包括输入测试用例、调用被测程序、判断结果正确性等功能。可做到一次定义，重复使用

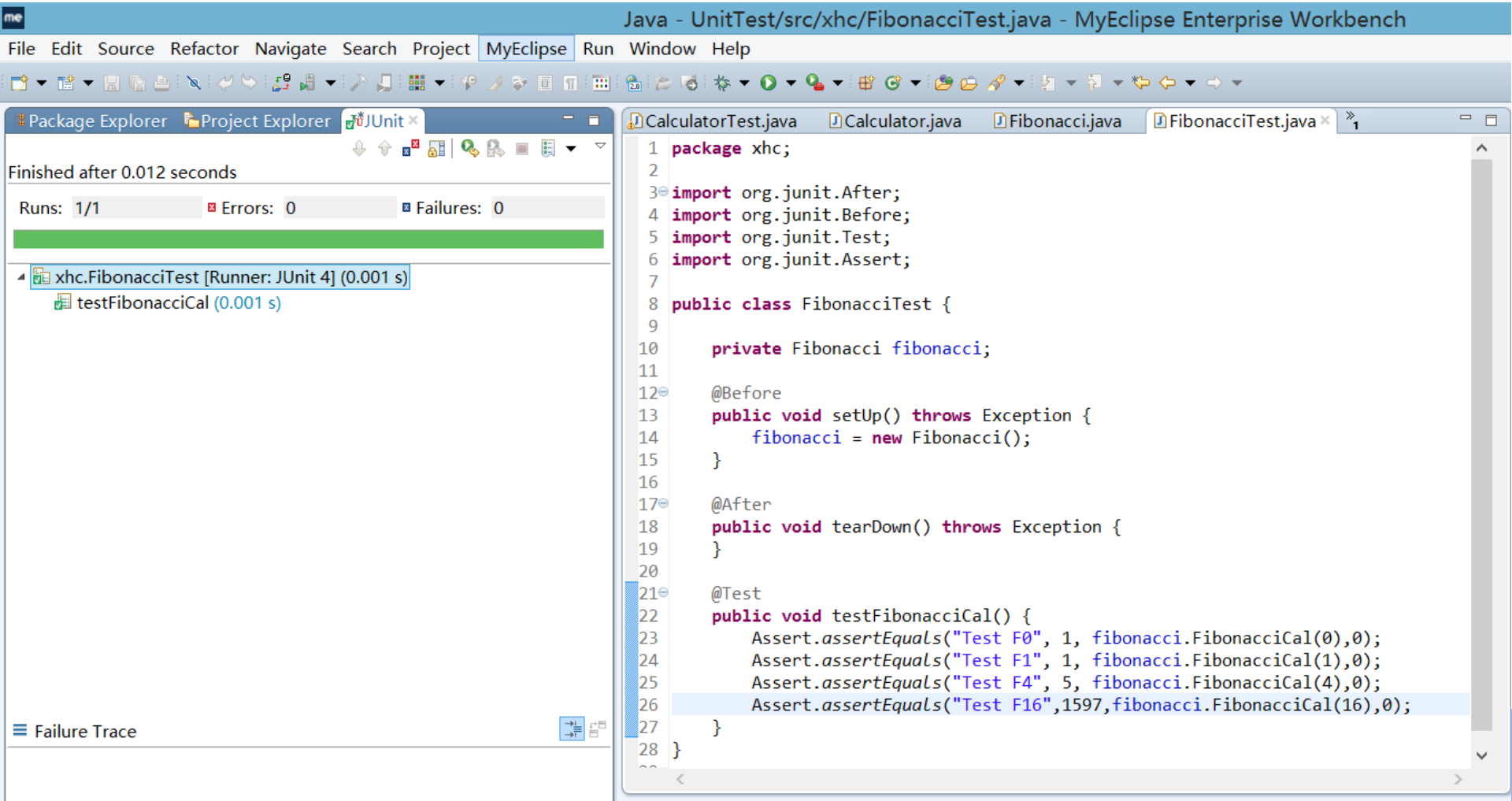


2. 软件测试

- 单元测试
  - 准备测试用例：用测试用例初始化系统，录入输入数据和期望得到的输出，例：菲波那切数列计算，以(0,1),(1,1),(4,5),(16,1597)为测试用例
  - 执行测试程序：调用被测程序中的对象或方法
  - 判断结果：比较调用结果和预期结果，判断是否通过测试

2. 软件测试

- 单元测试


Java - UnitTest/src/xhc/FibonacciTest.java - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
Package Explorer
Project Explorer
JUnit
Finished after 0.012 seconds
Runs: 1/1
Errors: 0
Failures: 0
xhc.FibonacciTest [Runner: JUnit 4] (0.001 s)
testFibonacciCal (0.001 s)
Failure Trace
CalculatorTest.java
Calculator.java
Fibonacci.java
FibonacciTest.java
1 package xhc;
2
3 import org.junit.After;
4 import org.junit.Before;
5 import org.junit.Test;
6 import org.junit.Assert;
7
8 public class FibonacciTest {
9
10 private Fibonacci fibonacci;
11
12 @Before
13 public void setUp() throws Exception {
14 fibonacci = new Fibonacci();
15 }
16
17 @After
18 public void tearDown() throws Exception {
19 }
20
21 @Test
22 public void testFibonacciCal() {
23 Assert.assertEquals("Test F0", 1, fibonacci.FibonacciCal(0),0);
24 Assert.assertEquals("Test F1", 1, fibonacci.FibonacciCal(1),0);
25 Assert.assertEquals("Test F4", 5, fibonacci.FibonacciCal(4),0);
26 Assert.assertEquals("Test F16",1597,fibonacci.FibonacciCal(16),0);
27 }
28 }

2. 软件测试

- 集成测试
  - 在单元测试的基础上，将所有模块按照总体设计的要求组装成为子系统或系统进行测试，旨在发现与接口相关的错误
  - 结构化集成测试针对调用关系测试，面向对象集成测试针对依赖关系测试

2. 软件测试

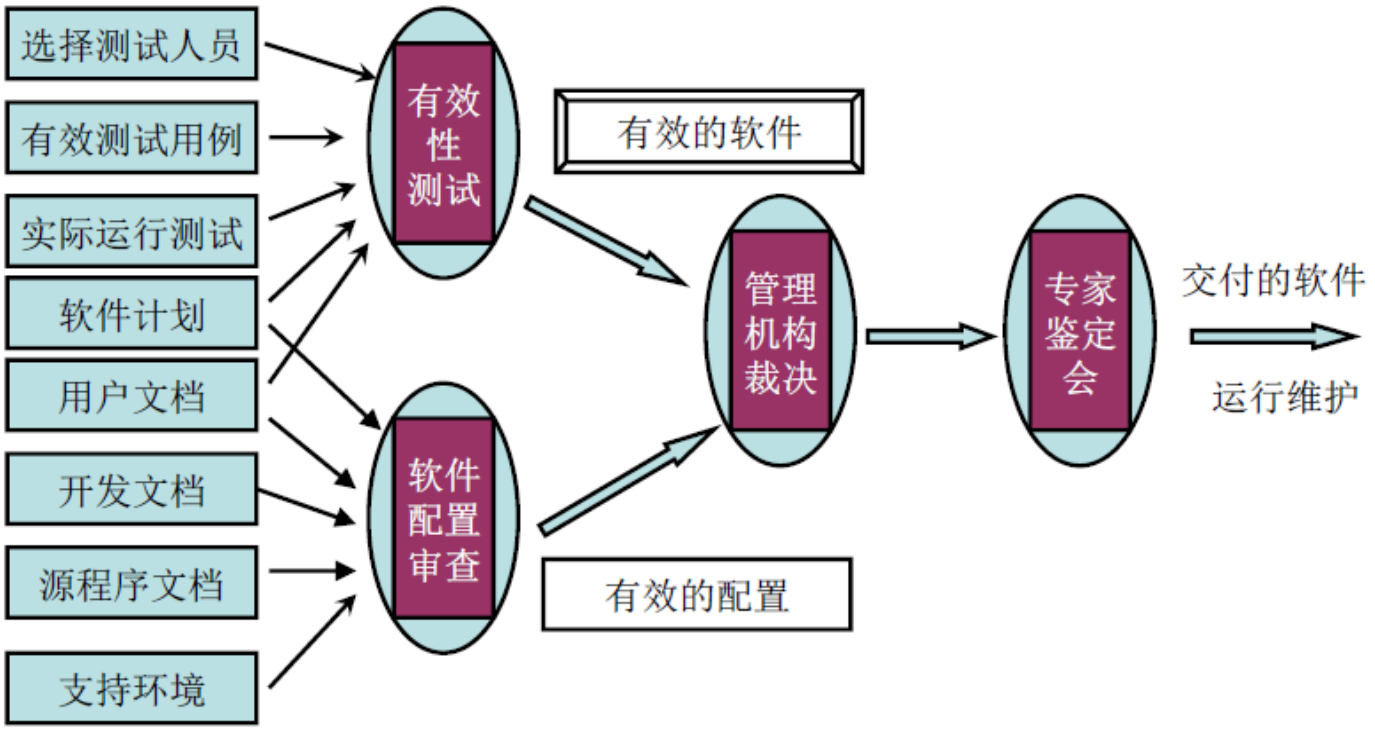
- 集成测试
  - 模块连接后，模块接口的数据是否会丢失？
  - 子功能组合后，能否达到预期要求的父功能？
  - 模块的功能是否会相互产生不利的影响？
  - 全局数据结构是否有问题？
  - 模块的误差累积是否会放大？

2. 软件测试

- 集成测试
  - 整体集成测试：把所有模块按设计要求一次全部组装起来
    - 优点：效率高，工作量低，简单，易行
    - 缺点：难以进行错误定位和修改
  - 增量式集成测试：逐步将新模块加入并测试
    - 优点：不易遗漏错误
    - 缺点：工作量大

2. 软件测试

- 确认测试
  - 验证软件的功能和性能及其他特性是否与用户的要求一致，即是否满足软件需求说明书中的确认标准



2. 软件测试

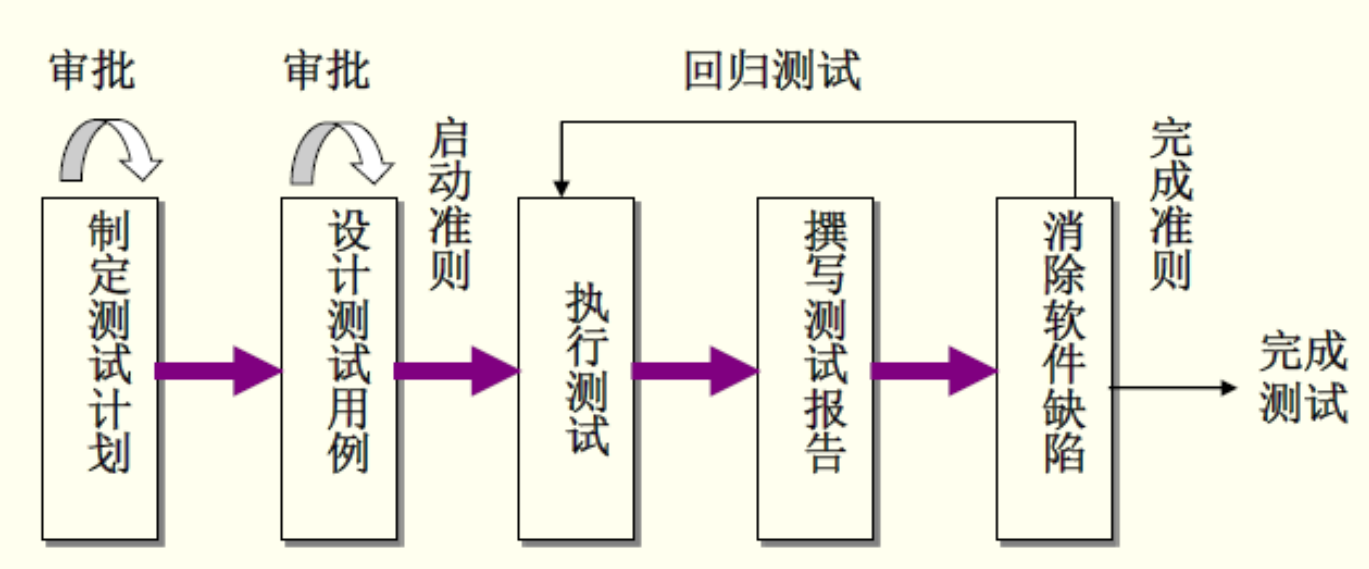
- 系统测试
  - 将集成好的软件系统作为一个整体进行测试，测试软件是否满足功能性和非功能性需求
    - 功能测试、协议一致性测试
    - 性能测试、压力测试、容量测试、安全性测试、恢复性测试
    - 备份测试、GUI 测试、健壮性测试、兼容性测试、可用性测试
    - 可安装性测试、文档测试、在线帮助测试、数据转换测试

2. 软件测试

- 可接受性测试（验收测试）
  - 是以用户为主的测试 ，一般使用用户环境中的实际数据进行测试
  - 在测试过程中，除了考虑软件的功能和性能外，还应对软件的兼容性、可维护性、错误的恢复功能等进行确认

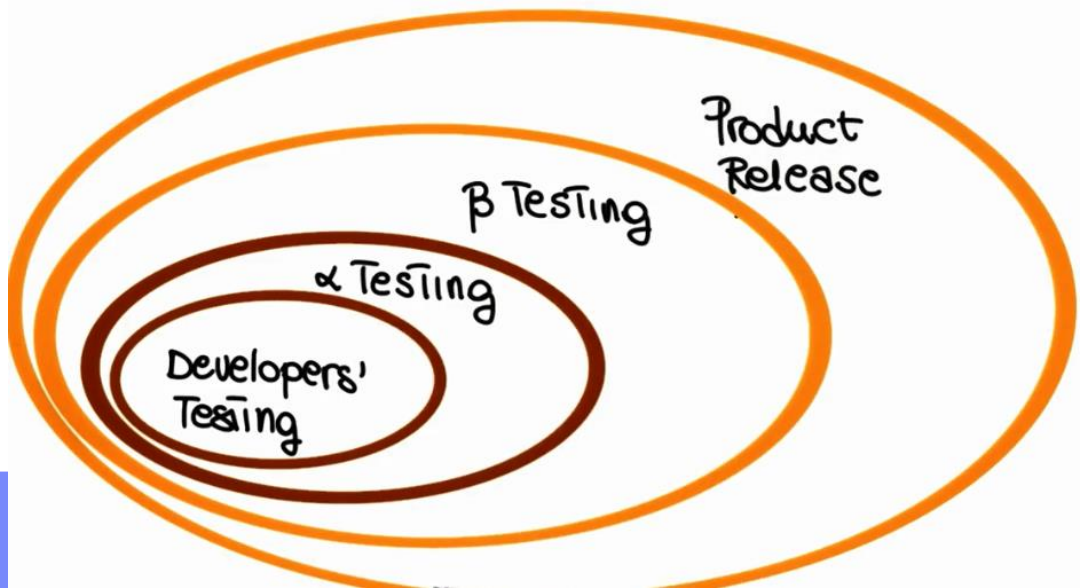
2. 软件测试

- 回归测试
  - 验证对系统的变更没有影响到以前功能， 避免对一个错误的修改引入新的错误，并且保证当前功能的变更是正确（service pack、update）
  - 回归测试可以发生在软件测试的任何阶段



2. 软件测试

- α测试
  - 指软件公司内部人员模拟用户对即将面市的软件产品进行测试
- β测试
  - 软件公司外部的一部分用户在实际使用环境下对产品的测试



2. 软件测试

- 软件测试技术
  - 静态测试： 不实际运行软件，主要是对软件的编程格式、结构等方面进行评估，包括：走查、评审等技术
  - 动态测试 ：计算机必须真正运行被测试的程序，通过输入测试用例，对其运行情况，即输入与输出的对应关系进行分析，以达到检测的目的

2. 软件测试

- 软件测试技术
  - 黑盒测试 ：又称为功能测试、数据驱动测试和基于规格说明的测试。它是一种从用户观点出发的测试，一般被用来确认软件功能的正确性和可操作性，是基于产品功能的软件测试。主要用于集成测试、确认测试和系统测试
  - 白盒测试 ：又称为结构测试、逻辑驱动测试或基于程序的测试 。一般用来分析程序的内部结构、逻辑、循环和路径，是一种基于产品内部结构的软件测试。主要用于单元测试和集成测试

2. 软件测试

- 软件测试技术：黑盒测试举例

说明：输入一个整数并打印

```
1.  void printNumBytes( int param ) {
2.      if(param < 1024) printf(“%d”, param);
3.      else printf(“%d KB”, param  / 124);
4.  }
```

2. 软件测试

- 软件测试技术：白盒测试举例

说明：输入一个整数，当为偶数时返回其值的一半，否则返回其值

```
1.  int fun( int param ) {
2.      int result;
3.      result = param / 2;
4.      return result;
5.  }
```

2. 软件测试

- 黑盒测试

  - 关注软件的输入域，可以确认是否覆盖了软件的重要行为
  - 不需要代码 → 尽早设计测试
  - 可以捕获逻辑上的缺陷
  - 可应用于所有测试级别

2. 软件测试

- 黑盒测试
  - 测试用例是为特定的测试目的而设计的一组测试输入、执行条件和预期结果的组合
  - 测试用例 = { 执行条件 + 测试数据 + 期望结果 }

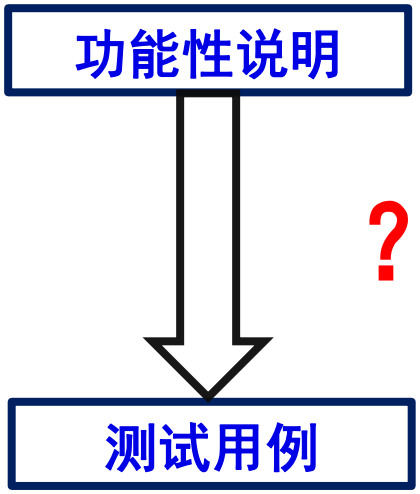
已知F_i的计算公式如下，求F_n取模17的结果。

$$\begin{cases} F_0 = 1 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

例：测试用例(0,1)、(1,1)、(16,16)运行后如果“实际结果=期望结果”，则认为软件在这组测试用例下通过测试，反之不通过。

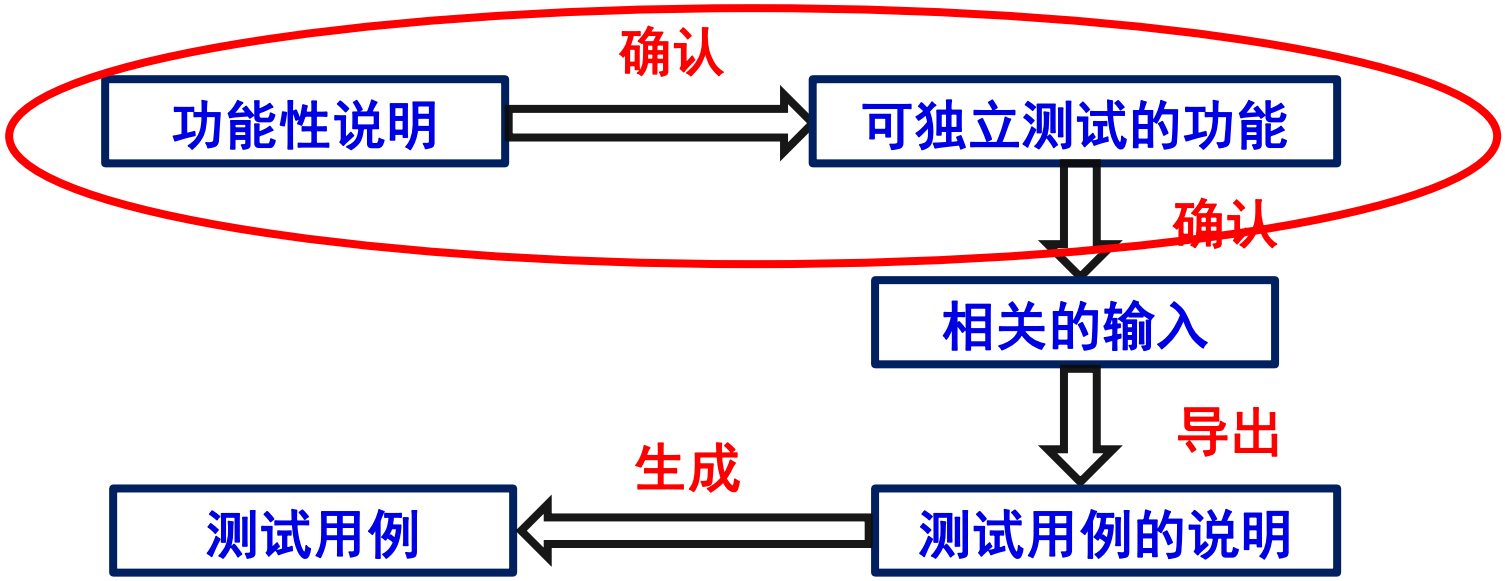
2. 软件测试

- 黑盒测试



2. 软件测试

- 黑盒测试



2. 软件测试

- 确认可测试功能

```
printSum(int a, int b)
```

有多少独立的可测试功能？

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ > 3

2. 软件测试

- 确认可测试功能

```
printSum(int a, int b)
```

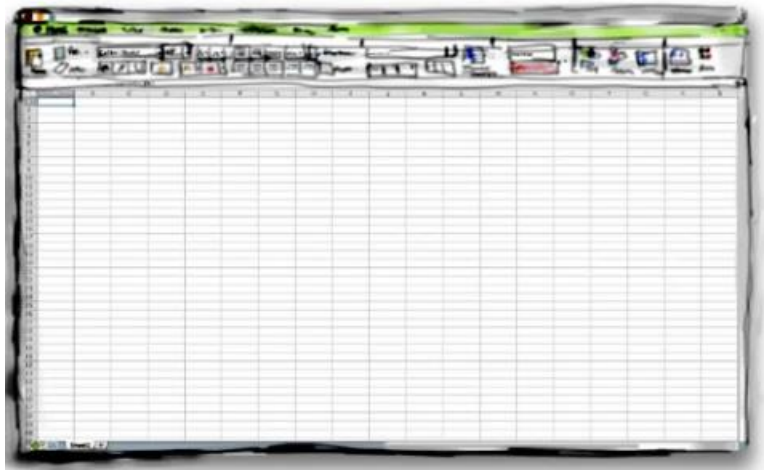
有多少独立的可测试功能？

- [√] 1
- [ ] 2
- [ ] 3
- [ ] > 3

2. 软件测试

- 确认可测试功能

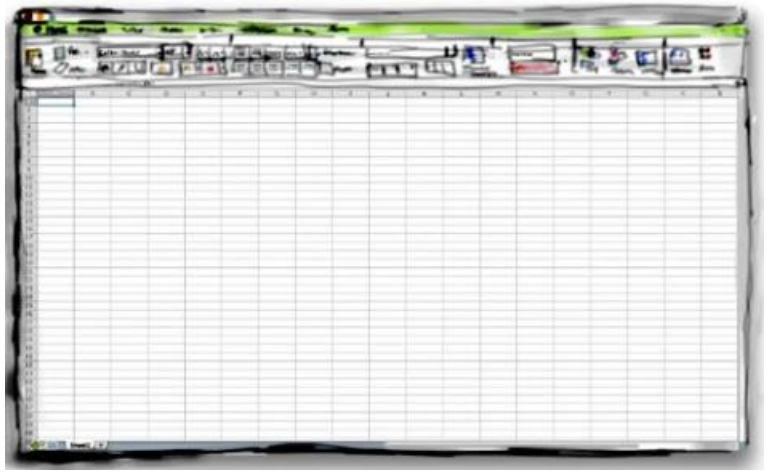
找出三个电子表格可独立测试的功能



2. 软件测试

- 确认可测试功能

找出三个电子表格可独立测试的功能



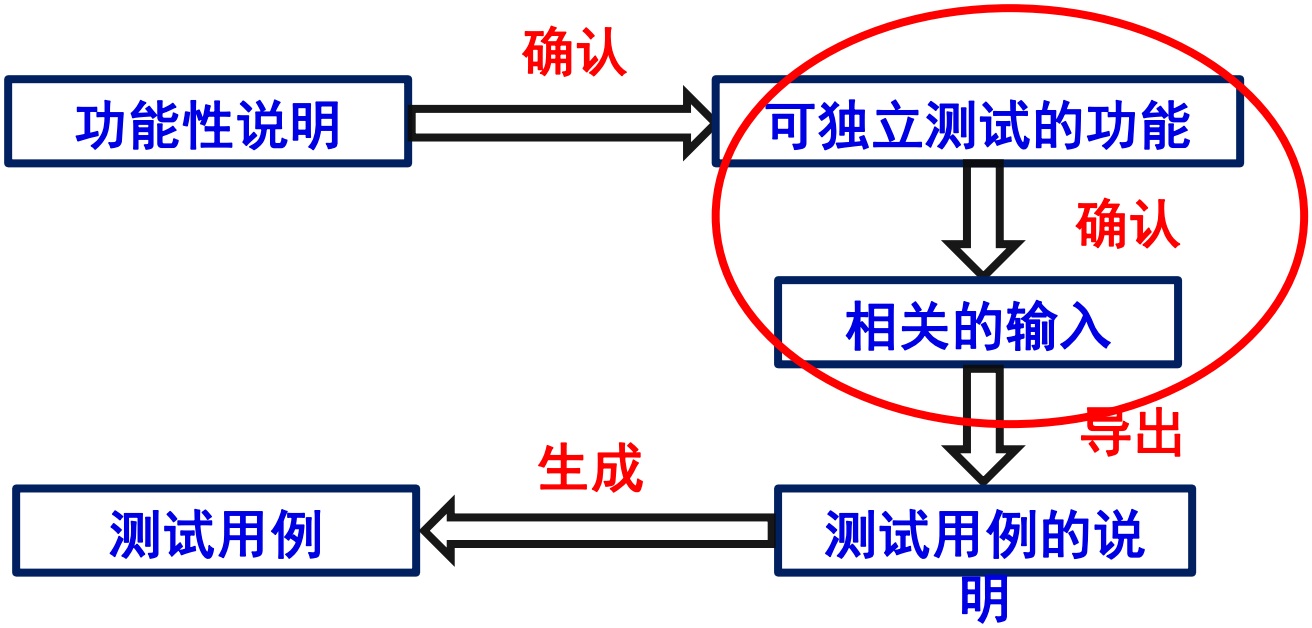
合并单元格

生成图表

统计功能

2. 软件测试

- 黑盒测试

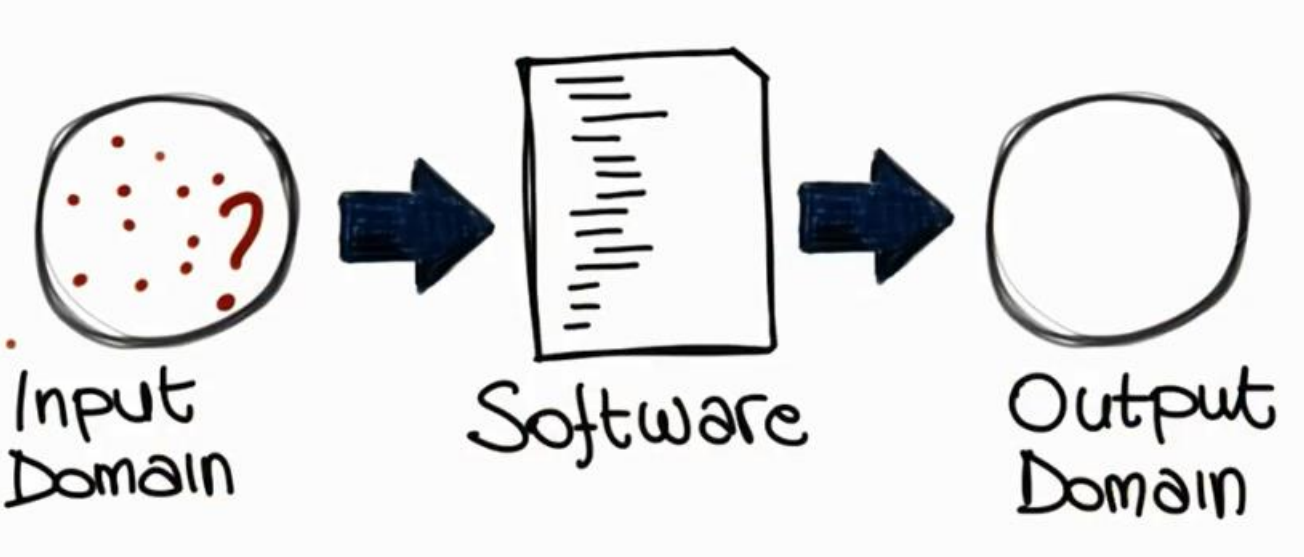


2. 软件测试

- 黑盒测试
  - 选择测试数据

2. 软件测试

- 黑盒测试
  - 选择测试数据



2. 软件测试

- 黑盒测试

```
printSum(int a, int b)
```

进行详尽的（exhaustive）测试需要多长时间？

2. 软件测试

- 黑盒测试

```
printSum(int a, int b)
```

进行详尽的（exhaustive）测试需要多长时间？

$2^{32} * 2^{32} = 2^{64} \approx 10^{19}$ 次测试

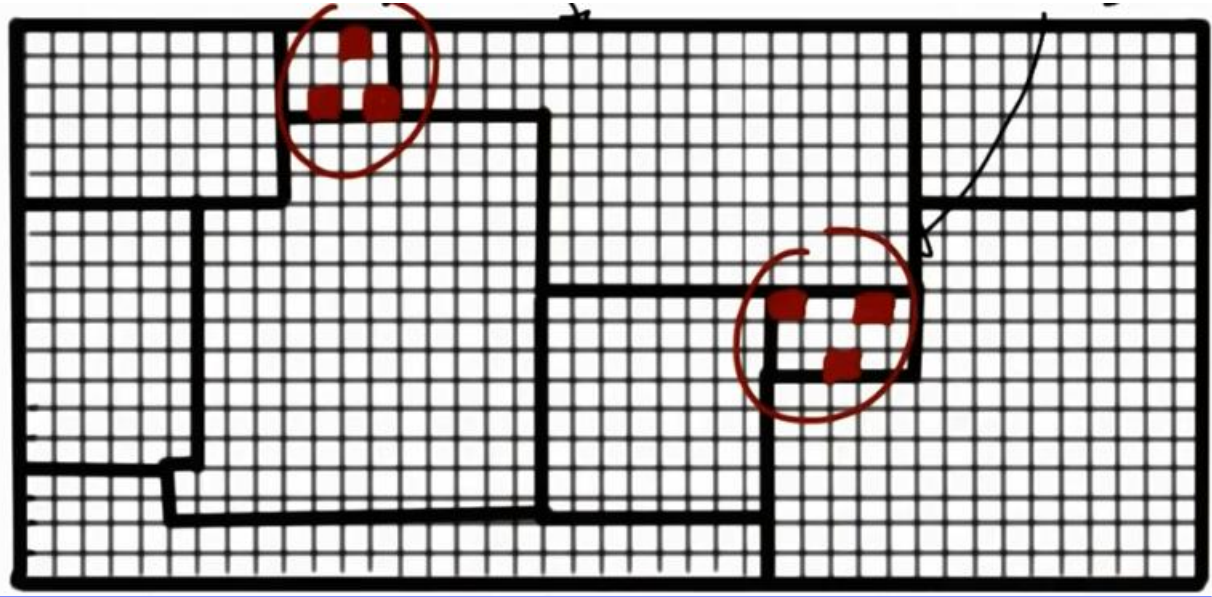
假设1纳秒执行1次测试，大约需要600年

2. 软件测试

- 黑盒测试
  - 随机选择数据？
    - 对所有输入同等对待
    - 均匀的选择输入

2. 软件测试

- 黑盒测试
  - 分区测试
    - 确定输入域的分区
    - 从每个分区中选择数据

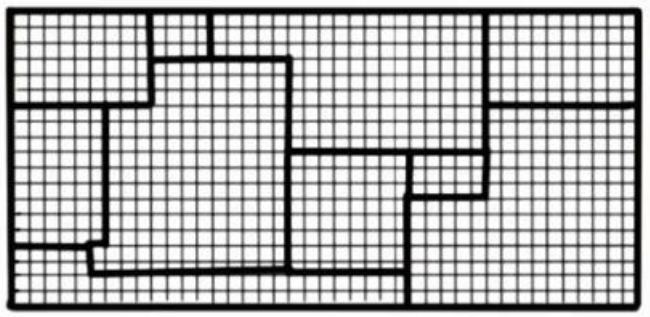


2. 软件测试

- 黑盒测试
  - 分区测试：Split(string str, int size)
    - size < 0
    - size = 0
    - size > 0
    - str：length < size
    - str：length in [size, size * 2]
    - str：length > size * 2

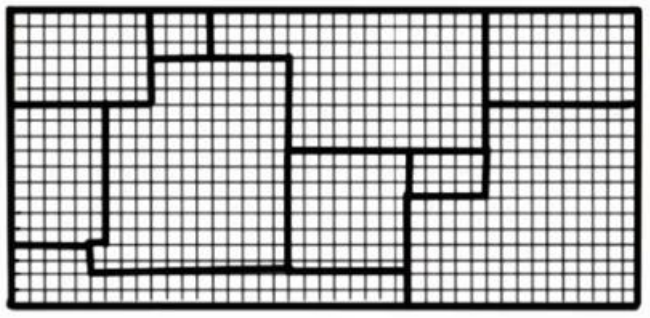
2. 软件测试

- 黑盒测试
  - 边界值
    - 错误容易出现在输入域的边界 --> 选取边界值进行测试
    - 对分区测试的补充



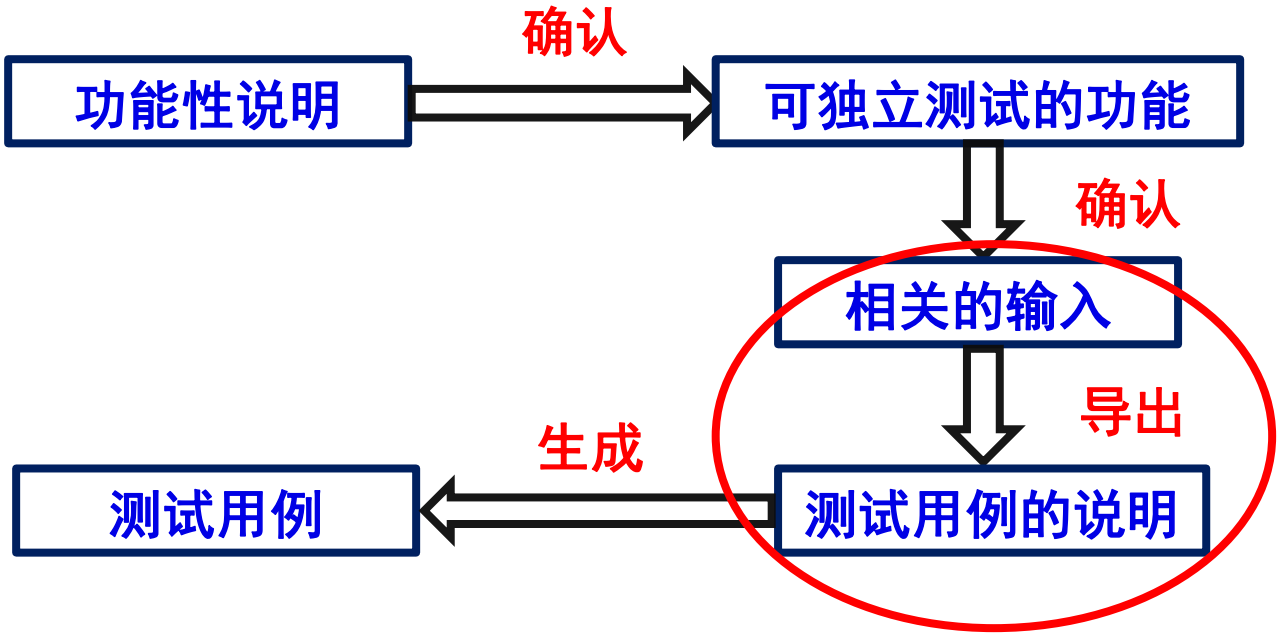
2. 软件测试

- 黑盒测试
  - 边界值
    - size < 0 ： size = -1
    - size = 0
    - size > 0： size = 1, size = MAXINT
    - str： length < size： length = size - 1
    - str： length in [size, size * 2]： length = size…
    - str： length > size * 2



2. 软件测试

- 黑盒测试

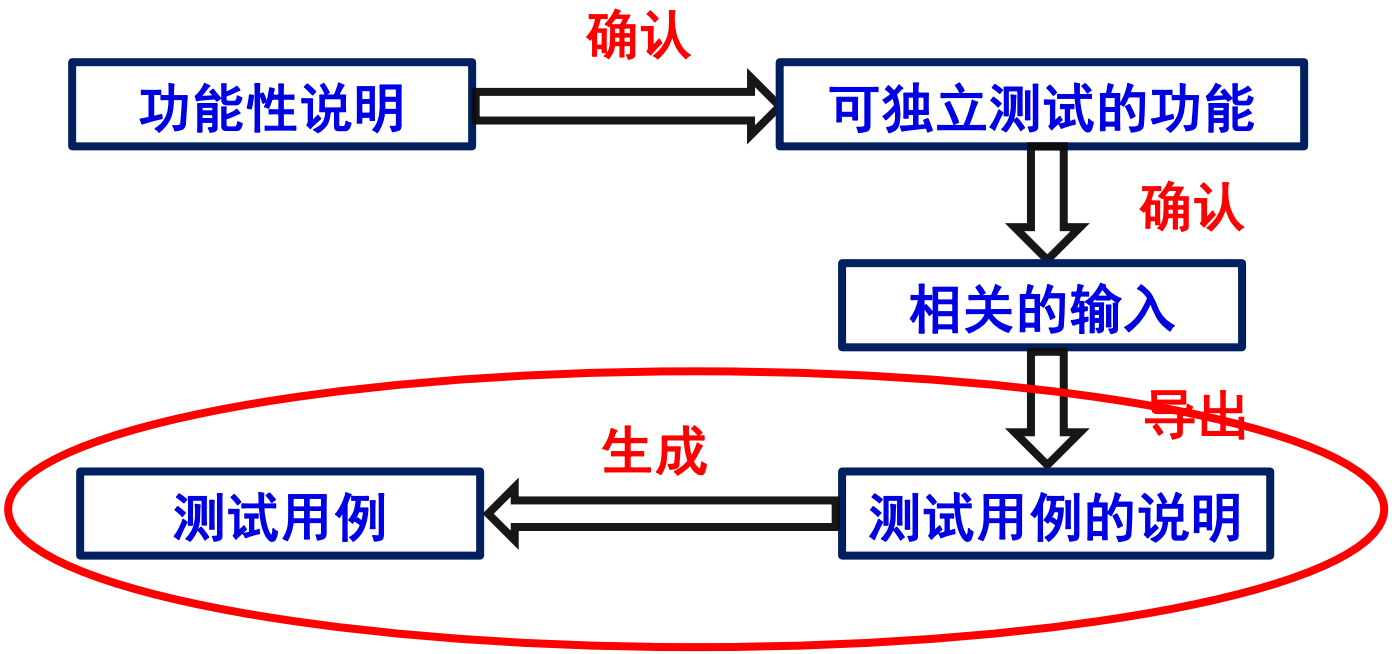


2. 软件测试

- 黑盒测试
  - 测试用例说明
    - 测试输入的组合，如：split(string str, int size)
    - 一些组合是无效的：如：size = -1, str lenth = -2

2. 软件测试

- 黑盒测试



2. 软件测试

- 白盒测试
  - 基本假设：揭示错误的一个条件是执行带有错误的语句

  - 基于代码 → 可以客观的来衡量，可以自动来测量
  - 可以进行测试用例的比较
  - 可以覆盖代码的行为

2. 软件测试

- 白盒测试
  - 基于控制流的白盒测试
  - 基于数据流的白盒测试
  - 基于错误的白盒测试

2. 软件测试

- 白盒测试

```
printSum(int a, int b)
```

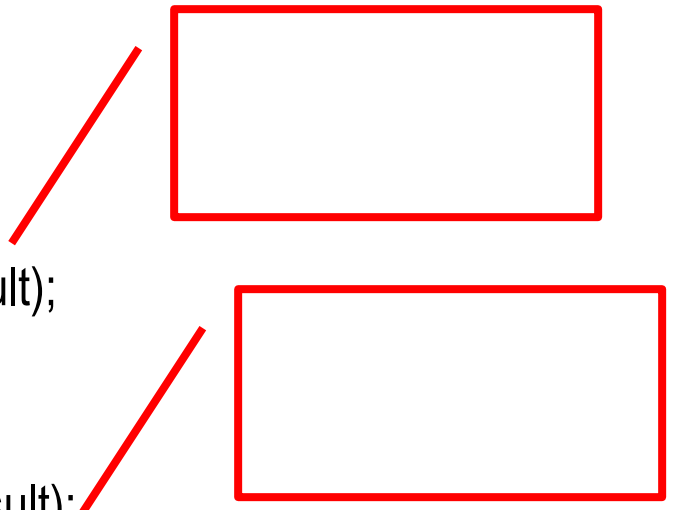
```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.  }
```

2. 软件测试

- 白盒测试

```
printSum(int a, int b)
```

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.  }
```



2. 软件测试

- 白盒测试

```
printSum(int a, int b)
```

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.  }
```

```
#1 a + b > 0
```

```
#2 a + b < 0
```

2. 软件测试

- 白盒测试

```
printSum(int a, int b)
```

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.  }
```

```
#1 a + b > 0
```

```
#2 a + b < 0
```

```
#1 ((a=[] ,b=[] ),(outputColor=[] ,outputValue=[] ))
#2 ((a=[] ,b=[] ),(outputColor=[] ,outputValue=[] ))
```

2. 软件测试

- 白盒测试

```
printSum(int a, int b)
```

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.  }
```

```
#1 a + b > 0
```

```
#2 a + b < 0
```

```
#1 ((a=[2],b=[3]),(outputColor=[red],outputValue=[5]))
#2 ((a=[-2],b=[0]),(outputColor=[blue],outputValue=[-2]))
```

2. 软件测试

- 白盒测试
  - 语句覆盖（statement coverage）

测试需求：程序中的语句

覆盖测量：$$\frac{\text{执行的语句数}}{\text{总的语句数}}$$

2. 软件测试

- 白盒测试
  - printSum: 语句覆盖

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.  }
```

```
TC #1
a = 3
b = 9
```

```
TC #2
a = -5
b = -8
```

2. 软件测试

- 白盒测试
  - 语句覆盖
    - 工业中的常用方法
    - 典型的覆盖率为80 – 90%

为什么覆盖率达不到100% ？

2. 软件测试

- 白盒测试
  - 语句覆盖

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.      [ else do nothing ]
8.  }
```

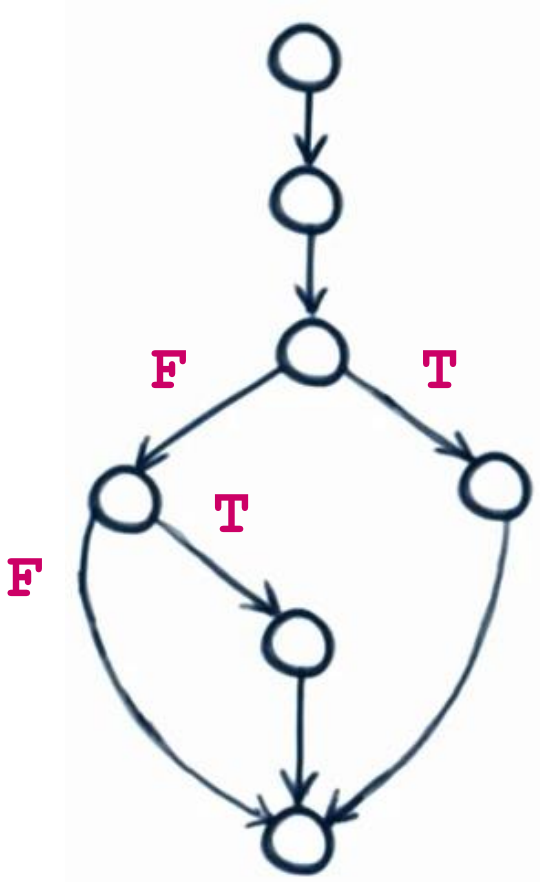
```
TC #1
a = 3
b = 9
```

```
TC #2
a = -5
b = -8
```

2. 软件测试

- 白盒测试
  - 控制流图

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol(“red”, result);
5.      else if(result < 0)
6.          printCol(“blue”, result);
7.      [ else do nothing ]
8.  }
```



2. 软件测试

- 白盒测试
  - 分支覆盖（branch coverage）

测试需求：程序中的分支

覆盖测量：$\frac{\text{执行的分支数}}{\text{总的分支数}}$

2. 软件测试

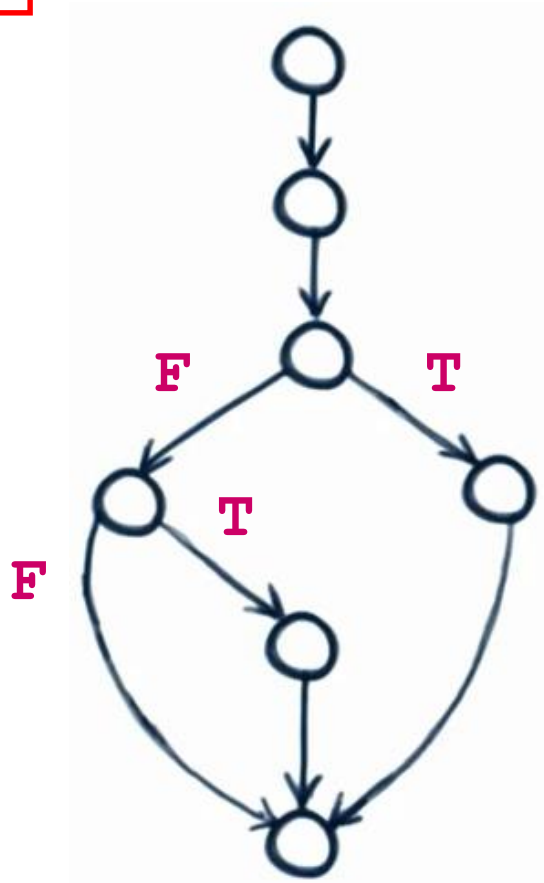
```
a = 3
b = 9
```

```
a = -5
b = -8
```

```
a = 0
b = 0
```

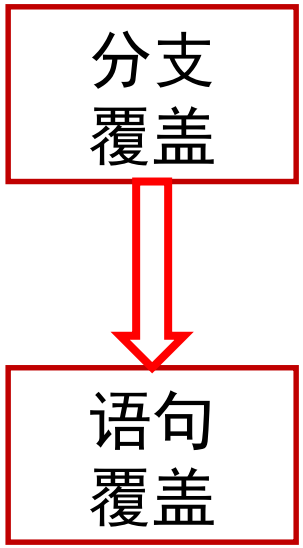
- 白盒测试
  - 分支覆盖（branch coverage）

```
1.  printSum(int a, int b) {
2.      int result = a + b;
3.      if(result > 0)
4.          printCol("red", result);
5.      else if(result < 0)
6.          printCol("blue", result);
7.      [ else do nothing ]
8.  }
```



2. 软件测试

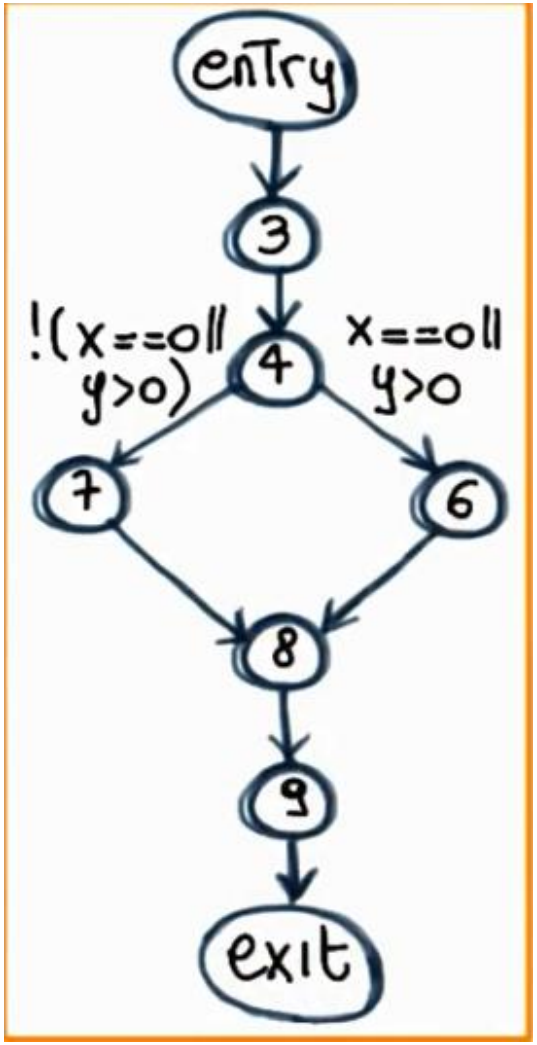
- 白盒测试
  - 测试标准的包容性



2. 软件测试

- 白盒测试

```
1.  void main() {
2.      float x, y;
3.      read(x);
4.      read(y);
5.      if((x == 0) || (y > 0))
6.          y = y / x;
7.      else x = y + 2;
8.      write(x);
9.      write(y);
10. }
```



Tests:(x=5, y=5)
(x=5, y=-5)

分支覆盖：100%

如何更全面的测试？

测试每个条件的 T 和 F

2. 软件测试

- 白盒测试
  - 条件覆盖（condition coverage）

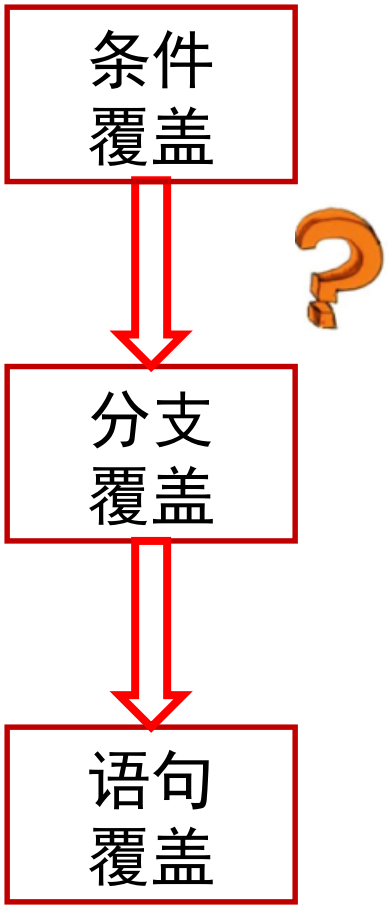
测试需求	程序中的每个条件
覆盖测量	$\frac{\text{条件为T和F的数量}}{\text{总的条件数}}$

2. 软件测试

- 白盒测试
  - 测试标准的包容性

条件覆盖是否意味着分支覆盖？

- ☐ yes
- ☐ no

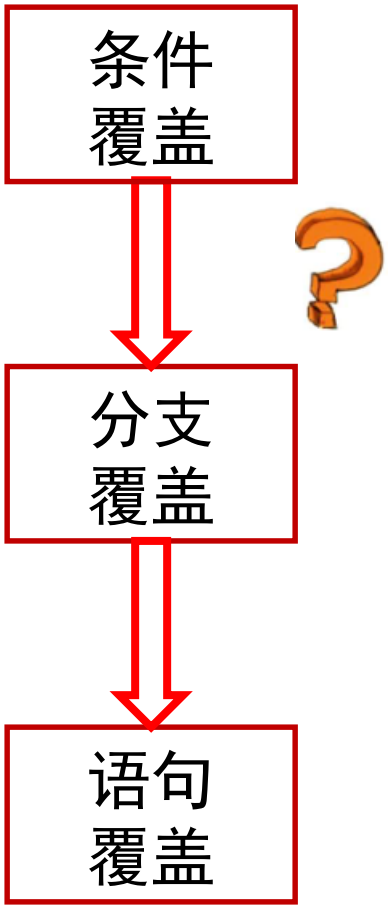


2. 软件测试

- 白盒测试
  - 测试标准的包容性

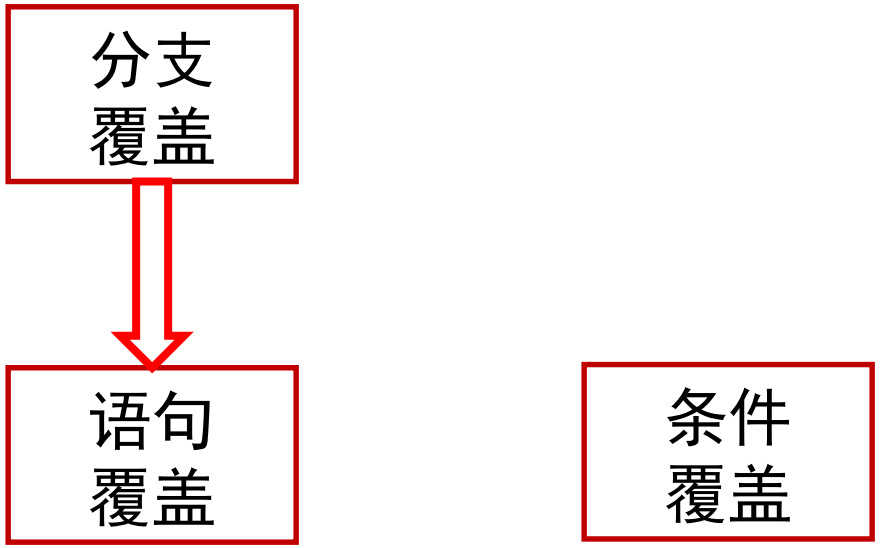
条件覆盖是否意味着分支覆盖？

- ☐ yes
- ☑ no



2. 软件测试

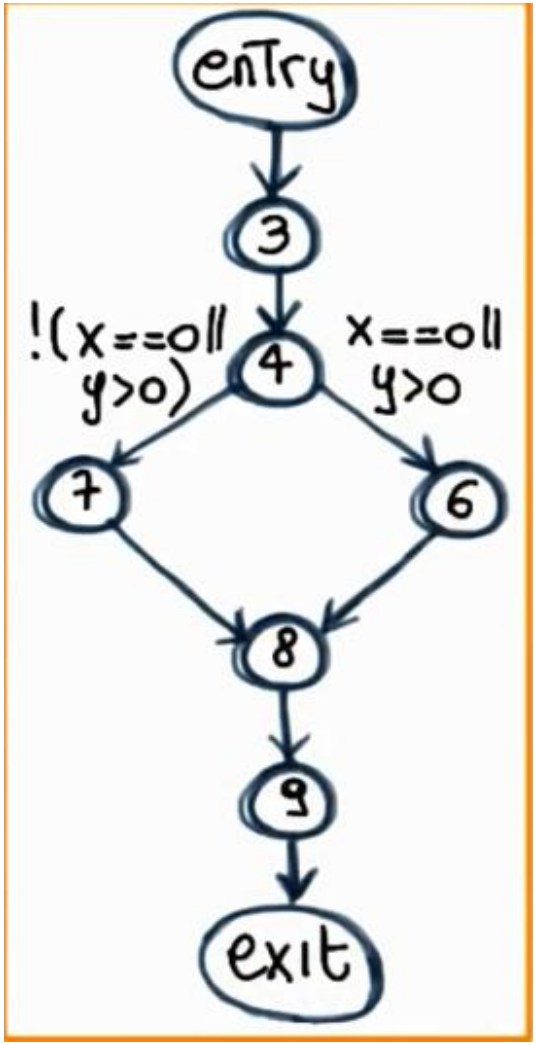
- 白盒测试
  - 测试标准的包容性



2. 软件测试

- 白盒测试

```
1.  void main() {
2.      float x, y;
3.      read(x);
4.      read(y);
5.      if((x == 0) || (y > 0))
6.          y = y / x;
7.      else x = y + 2;
8.      write(x);
9.      write(y);
10. }
```



Tests: (x=0, y=-5)
(x=5, y=5)

条件覆盖：100%

分支覆盖：50%

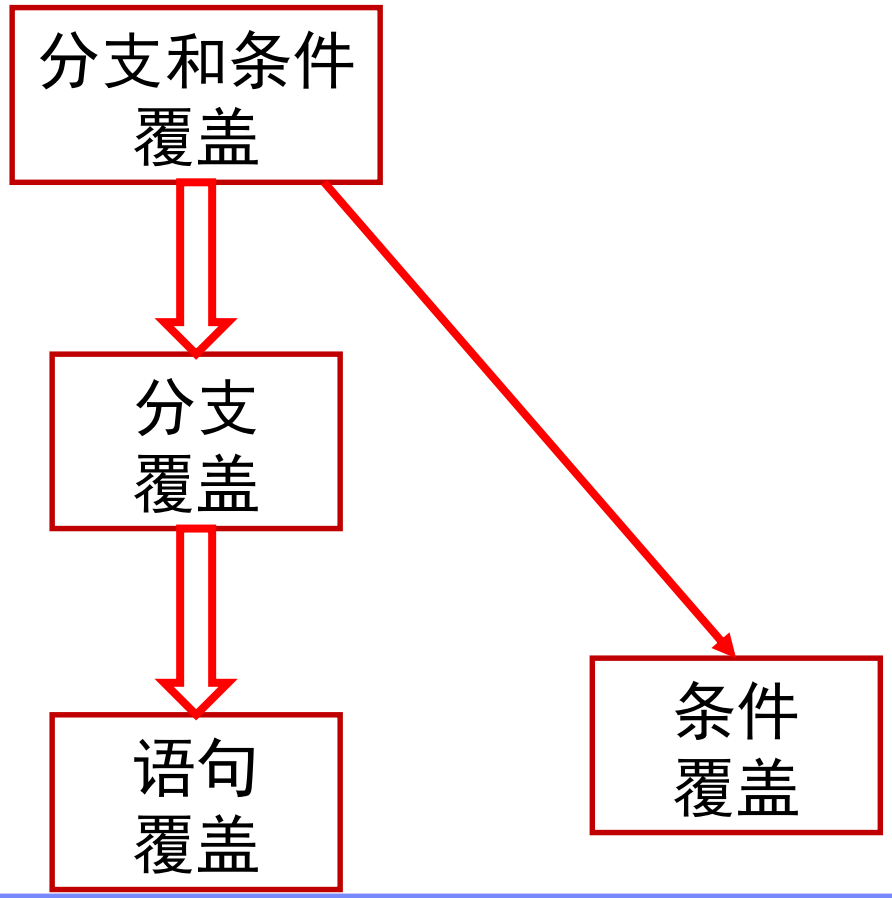
2. 软件测试

- 白盒测试
  - 分支和条件覆盖

测试需求	程序中的分支和每个条件
覆盖测量	计算两个覆盖的测量

2. 软件测试

- 白盒测试
  - 测试标准的包容性



2. 软件测试

- 白盒测试
  - 修正的条件/决策覆盖（Modified Condition/Decision Coverage – MC/DC）
    - 通常是侧重安全性应用所需的测试，如：FAA（Federal Aviation Administration 美国联邦航空管理局）要求所有商用飞机上的软件依据MC/DC进行测试
    - 主要思想：测试重要的条件组合，限制测试成本
    - 测试的条件：每个条件独立影响测试结果

2. 软件测试

- 白盒测试
  - MC/DC举例：a && b && c

Test Case	a	b	c	结果
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

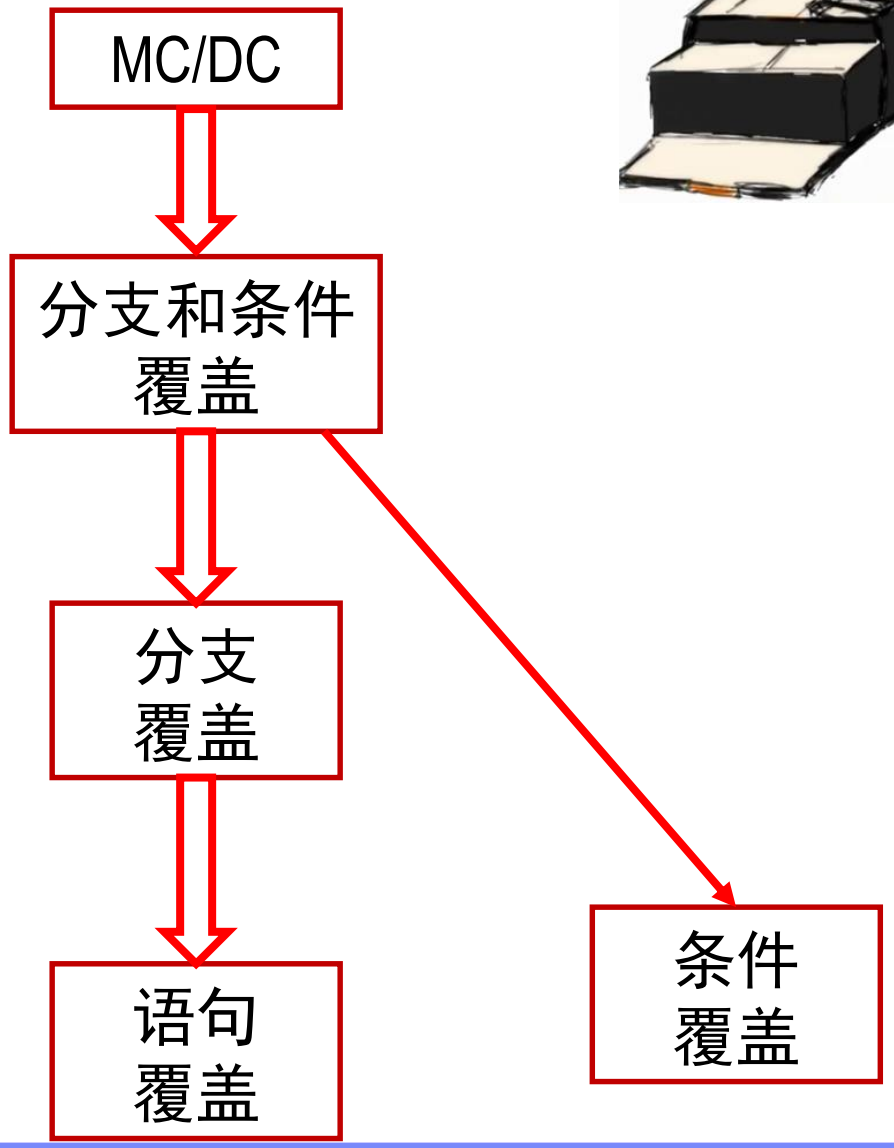
a && b && c

Test Case	a	b	c	结果
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	**True**	**True**	**True**	**True**
5	**False**	**True**	**True**	**False**
3	**True**	**False**	**True**	**False**
2	**True**	**True**	**False**	**False**

2. 软件测试

- 白盒测试
  - 测试标准的包容性



2. 软件测试

- 白盒测试

```
1.    int i = 0;
2.    int j;
3.    read(j);
4.    if((j > 5) && (i > 0))
5.          print(i);
```

你是否能够创建测试用例达到语句覆盖？

☐ yes

☐ no

2. 软件测试

- 白盒测试

```
1.    int i = 0;
2.    int j;
3.    read(j);
4.    if((j > 5) && (i > 0))
5.          print(i);
```

你是否能够创建测试用例达到语句覆盖？

☐ yes

☑ no

每个程序都可能包含不可到达的代码！

小结

- 软件测试贯穿于整个软件生命周期
- 单元测试、集成测试
- 黑盒测试、白盒测试

复习

1.用于判断“新引入的变化没有给现有软件造成破坏”的测试活动是____?

- A.压力测试
- B.回归测试
- C. beta测试
- D. alpha测试

复习

1.用于判断“新引入的变化没有给现有软件造成破坏”的测试活动是____?

- [ ] A.压力测试
- [√] B.回归测试
- [ ] C. beta测试
- [ ] D. alpha测试

复习

2. 进行单元测试时主要依据 ____?

- A.需求规格说明
- B.用户使用说明书
- C.概要设计文档
- D.详细设计文档

复习

2. 进行单元测试时主要依据 ____?

- [ ] A.需求规格说明
- [ ] B.用户使用说明书
- [ ] C.概要设计文档
- [√] D.详细设计文档

复习

3. 集成测试的主要测试对象是____?

- A.局部数据结构
- B.模块间的接口
- C.独立路径
- D.出错处理

复习

3. 集成测试的主要测试对象是 ____?

- [ ] A.局部数据结构
- [√] B.模块间的接口
- [ ] C.独立路径
- [ ] D.出错处理

复习

4.下面关于测试用例的论述，不正确的是____?

- A.测试用例只是一组针对特定程序的输入数据
- B.对同样的测试用例，系统的多次执行结果应当是相同的
- C.测试用例不仅要能够代表并覆盖各种合理的和合法的输入数据、操作和环境设置，也要能够代表并覆盖各种不合理、非法的或越界的情况
- D.测试用例设计时，应该选取最有可能抓住错误的，一组相似测试用例中最有效的

复习

4.下面关于测试用例的论述，不正确的是____？

- [√] A.测试用例只是一组针对特定程序的输入数据
- [ ] B.对同样的测试用例，系统的多次执行结果应当是相同的
- [ ] C.测试用例不仅要能够代表并覆盖各种合理的和合法的输入数据、操作和环境设置，也要能够代表并覆盖各种不合理、非法的或越界的情况
- [ ] D.测试用例设计时，应该选取最有可能抓住错误的，一组相似测试用例中最有效的

复习

5.以下关于软件配置管理的内容和目标的论述，错误的是____?

- A. 标识和控制变更
- B. 确保变更的正确实现
- C. 杜绝软件变更
- D. 向开发组织内各角色报告变更

复习

5.以下关于软件配置管理的内容和目标的论述，错误的是____?

- [ ] A. 标识和控制变更
- [ ] B. 确保变更的正确实现
- [√] C. 杜绝软件变更
- [ ] D. 向开发组织内各角色报告变更

复习

6.下面关于软件测试组织和人员的论述，最准确的是____?

- A.软件测试由专业测试人员独自完成
- B.软件测试主要由程序员完成
- C.软件测试应通过最终用户的使用完成
- D.软件测试需要专业测试人员、程序员、最终用户等各类人员的参与

复习

6.下面关于软件测试组织和人员的论述，最准确的是____?

- [ ] A.软件测试由专业测试人员独自完成
- [ ] B.软件测试主要由程序员完成
- [ ] C.软件测试应通过最终用户的使用完成
- [√] D.软件测试需要专业测试人员、程序员、最终用户等各类人员的参与

复习

7.某程序模块的输入要求为1至8位的数字，下面哪些是针对该输入的无效等价类是 ____？(多选题)

- [ ] A.输入位数小于1
- [ ] B.输入位数大于8
- [ ] C.输入位数在1-8之间，且均为数字
- [ ] D.输入位数在1-8之间，含有非数字的符号

复习

7.某程序模块的输入要求为1至8位的数字，下面哪些是针对该输入的无效等价类是 ____？(多选题)

- [√] A.输入位数小于1
- [√] B.输入位数大于8
- [ ] C.输入位数在1-8之间，且均为数字
- [√] D.输入位数在1-8之间，含有非数字的符号