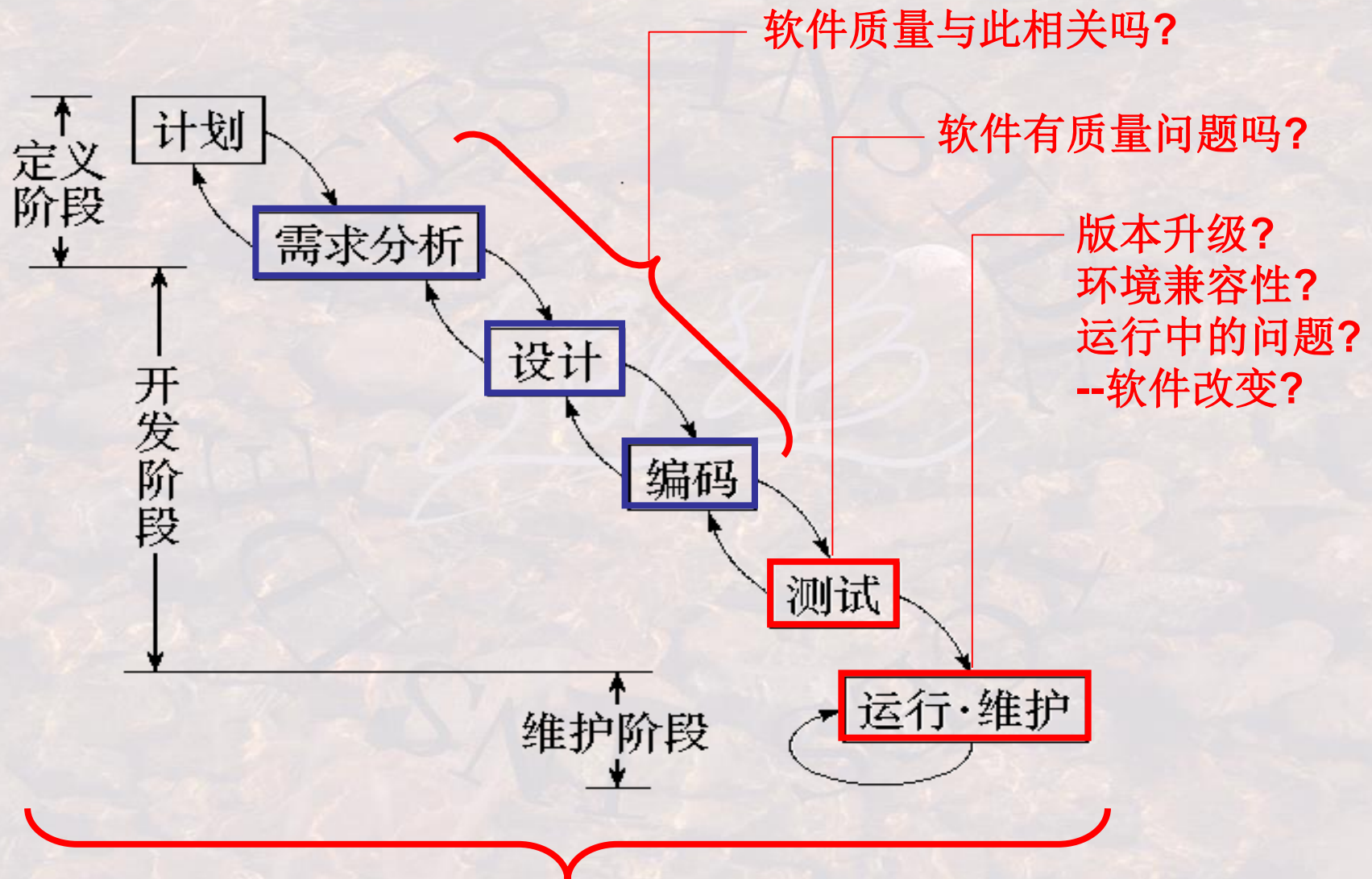


# 软件工程技术-软件测试与维护

徐汉川  
哈尔滨工业大学

Research Center on Intelligent  
Computing for Enterprises & Services,  
Harbin Institute of Technology

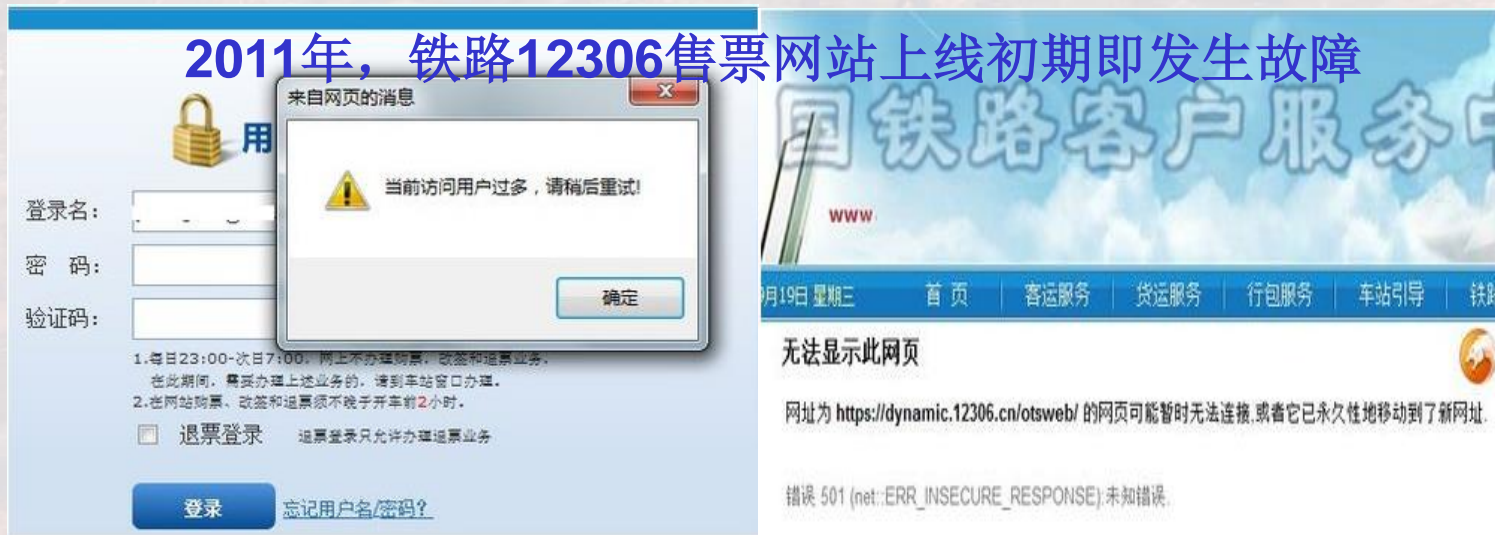
# 软件工程技术-软件测试与维护



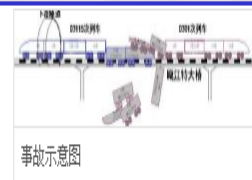
软件过程与构造高质量软件的关系?

# 软件质量概念与内容





经调查认定，导致事故发生的原因是：通号集团所属通号设计院在LKD2—T1型列控中心设备研发中管理混乱，通号集团作为甬温线通信信号集成总承包商履行职责不力，致使为甬温线温州南站提供的LKD2—T1型列控中心设备存在严重设计缺陷和重大安全隐患。国家铁道部在LKD2—T1型列控中心设备招投标、技术审查、上道使用等方面违规操作、把关不严，致使其在温州南站上道使用。当温州南站列控中心采集驱动单元采集电路电源回路中保险管F2遭雷击熔断后，采集数据不再更新，错误地控制轨道电路发码及信号显示，使行车处于不安全状态。



雷击也造成5829AG轨道电路发码器与列控中心通信故障，使从永嘉站出发驶向温州南站的D3115次列车超速防护系统自动制动，在5829AG区段内停车。由于轨道电路发码器故障，导致其向列控中心发送的码序异常，7分40秒后转为目视行车模式以低于20公里/小时的速度向温州南站运行。当列车运行至温州南站D3115次列车在5829AG区段的占用信息，使温州南站列控中心错误的向温州南站区间发送绿灯信号，导致D301次列车发送无车占用码，导致D301次列车驶向D3115次列车并发生追尾。

**信号控制设备在设计上的缺陷，导致本应显示为红灯的区间信号错误显示为绿灯**

## (1)从质量问题谈起—软件引发的问题

- “据推测，(美国)由于软件缺陷而引起的损失额每年高达**595 亿美元**。这一数字相当于美国国内生产总值的 **0.6%**”。  
----2002年6月28日, 美国商务部的国家标准技术研究所(NIST)发布报告.
- 2007年8月14日14时, 美国洛杉矶国际机场电脑发生故障, 60个航班的2万旅客无法入关。原因: 包含旅客姓名和犯罪记录的部分数据系统(海关和边境保护系统)瘫痪。2004年9月发生过类似问题.
- 2000年巴拿马城发生辐射剂量超标事故, 有些患者接受了超标剂量的治疗, 至少有5人死亡。原因: 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。





- “好坏程度”
- 质量是产品的一组**固有特性**满足**要求**的程度---ISO 9000 (2000版)

**特性**: 可区分的特征, 如物理特征--机械运动、温度、电流等, 化学特征--成分组合、合成、分解等;

**固有特性**: 某事物中本来就有的、持久的特征, 如硬度、高度等;

**赋予特性**: 对事物增加的特性, 如价格、位置等;

**显式要求**: 有明确规定的要求(行业标准或用户指定), 如屏幕尺寸;

**隐式要求**: 约定俗成的要求, 如大楼要有楼梯;

**产品质量特性**: 内部特性、外部特性  
满足的 程度 **vs.** 满足的 成本!

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的组合。[M.J. Fisher]

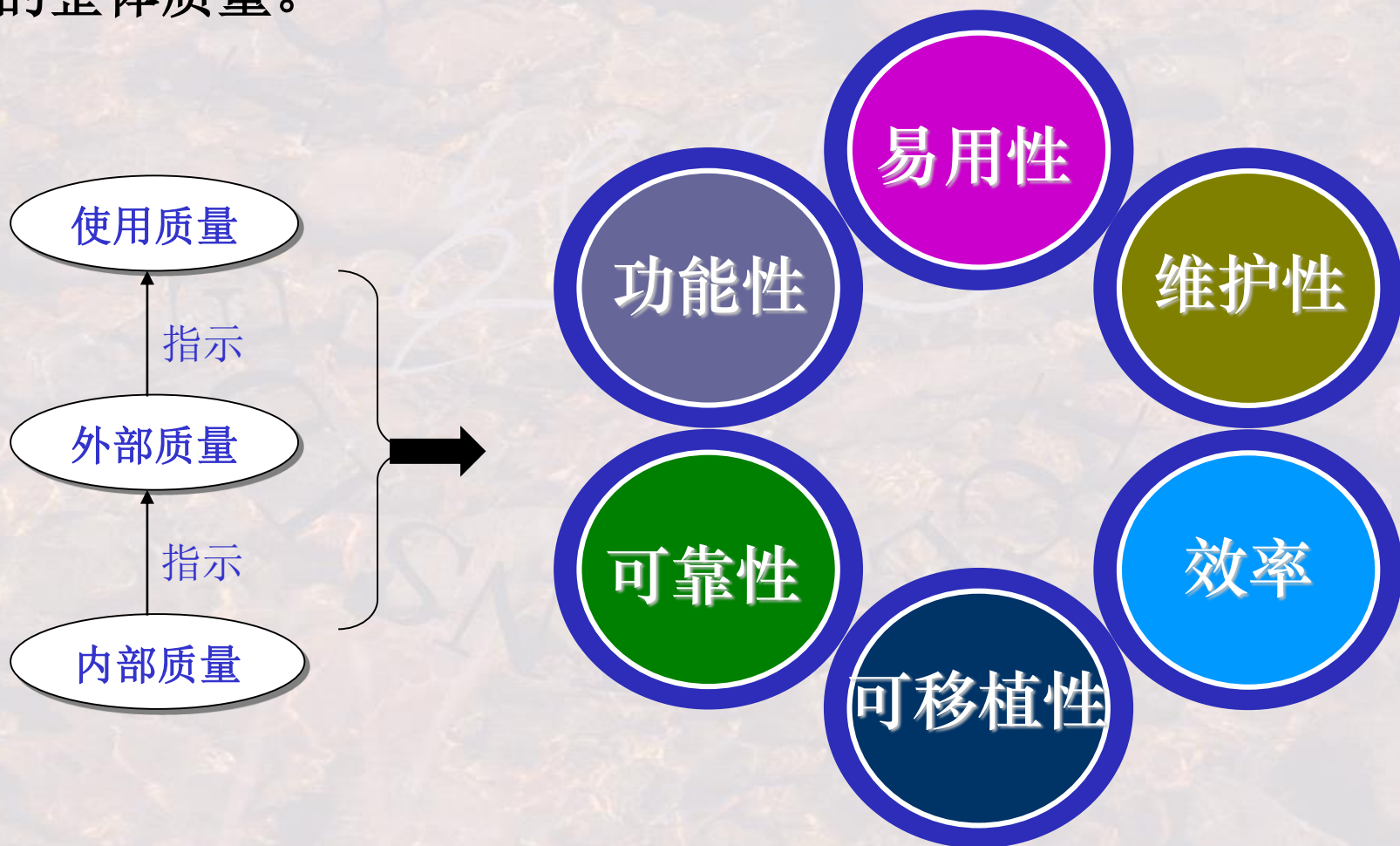
### ■ 软件消费者观点

- **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

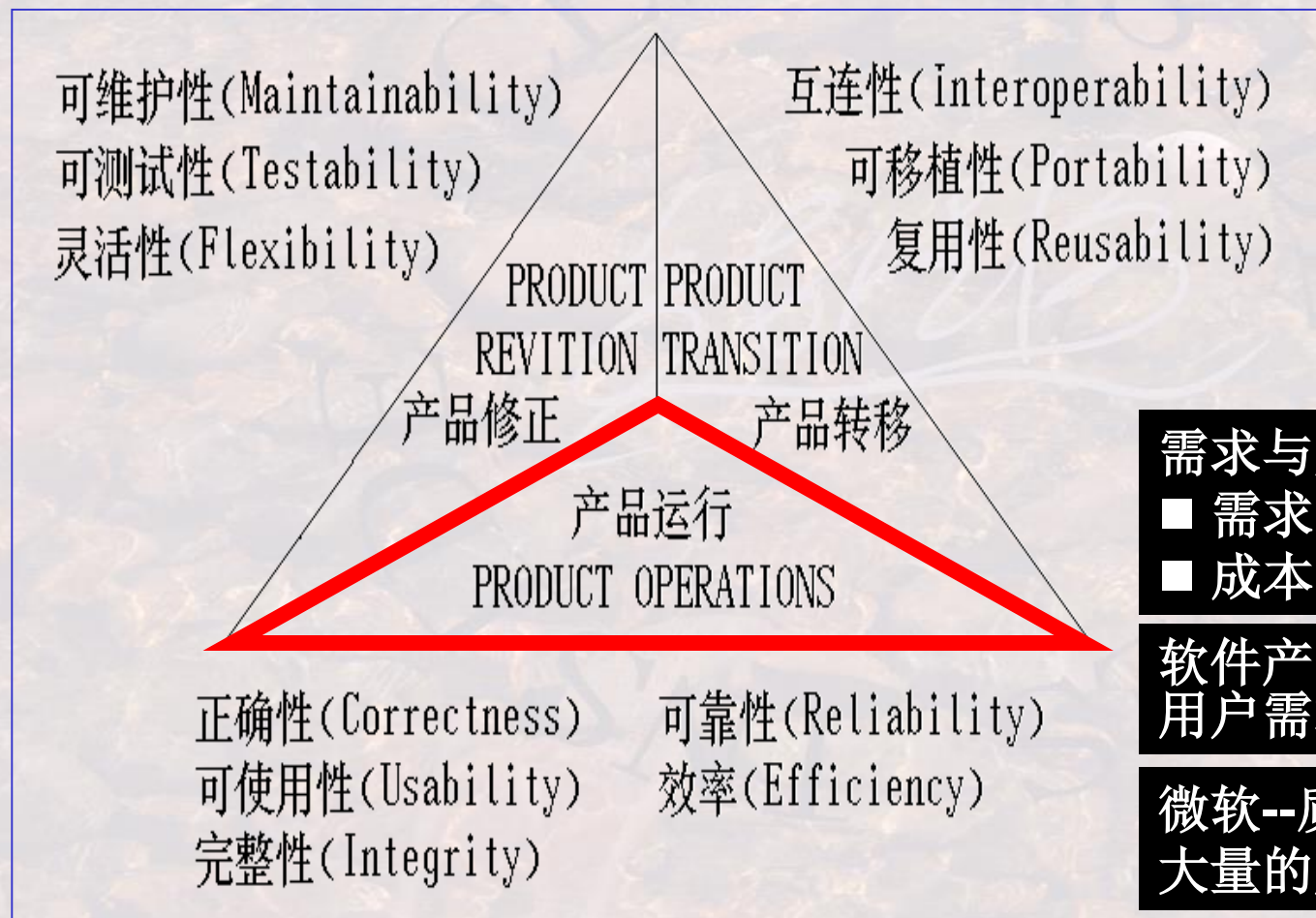
- **质量的一致性**: 确保产品或服务是根据设计制造的
- **利润**: 确保用最少的成本投入获取最大利益

软件质量是**许多质量属性的综合体现**，各种质量属性反映了软件质量的方方面面。人们通过改善软件的各种质量属性，从而提高软件的整体质量。





### 软件质量模型[Barry Boehm]



需求与成本之间的矛盾:

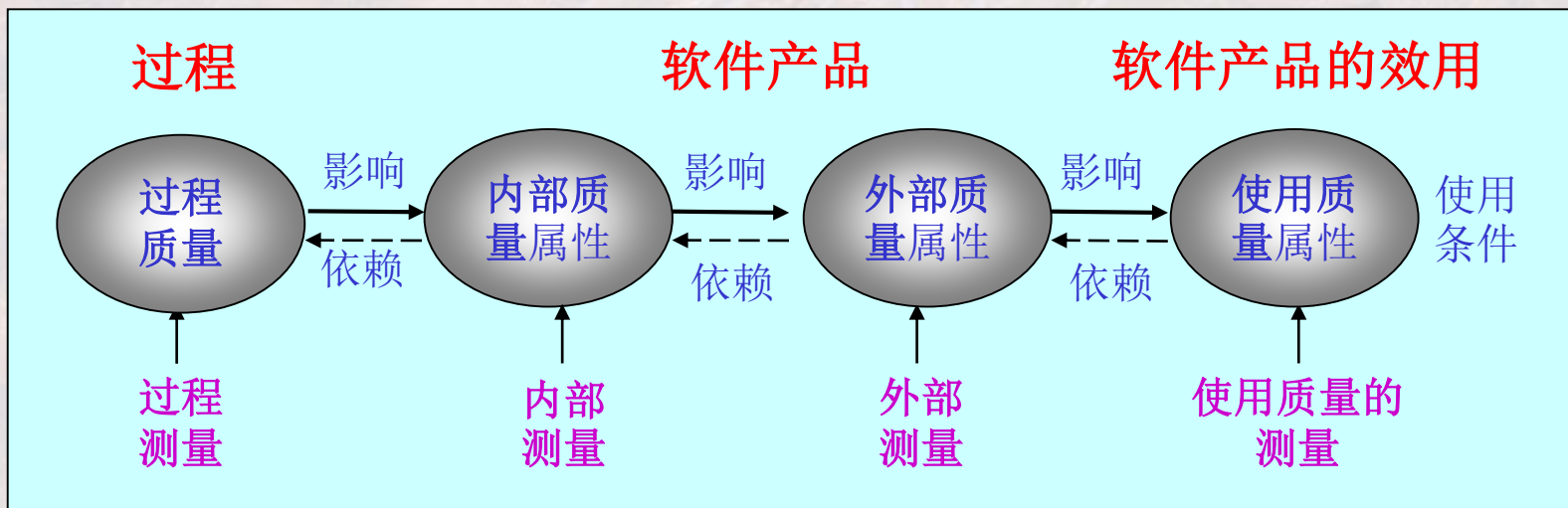
- 需求是永无止境的
- 成本是永远有限的

软件产品属性完全满足  
用户需求是不现实的

微软--质量只要好到能使  
大量的产品卖给客户

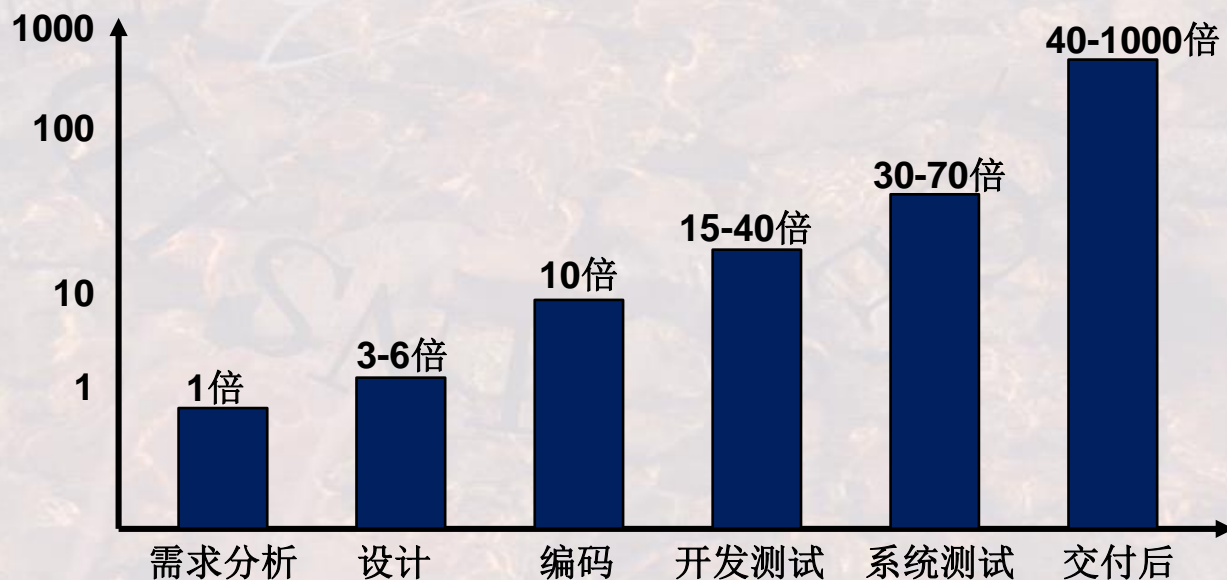
软件组织在软件产品生产中的**质量策划、质量控制、质量度量与验证、质量改进**等与质量有关的相互协调的活动---**软件质量管理**

### ISO-9126的软件质量模型框架



过程质量	有助于提高	产品质量
产品质量	有助于提高	使用质量

- 软件质量保证的思想：“全面质量管理及事先预防”的思想，“缺陷越早发现越早修改越经济”的原则
- 提高软件质量最好的办法是：不断地提高技术水平和规范化水平（软件过程改进），在开发过程中有效地防止工作成果产生缺陷，将高质量内建于开发过程之中。
- 提高软件质量有效的方法是：软件中的缺陷是无法完全避免的，及早进行质量检查（技术评审、软件测试和过程检查），及时找出并消除工作成果中的缺陷。
- 提高软件质量的一般方法是：出现问题，发现缺陷后，及时纠正。



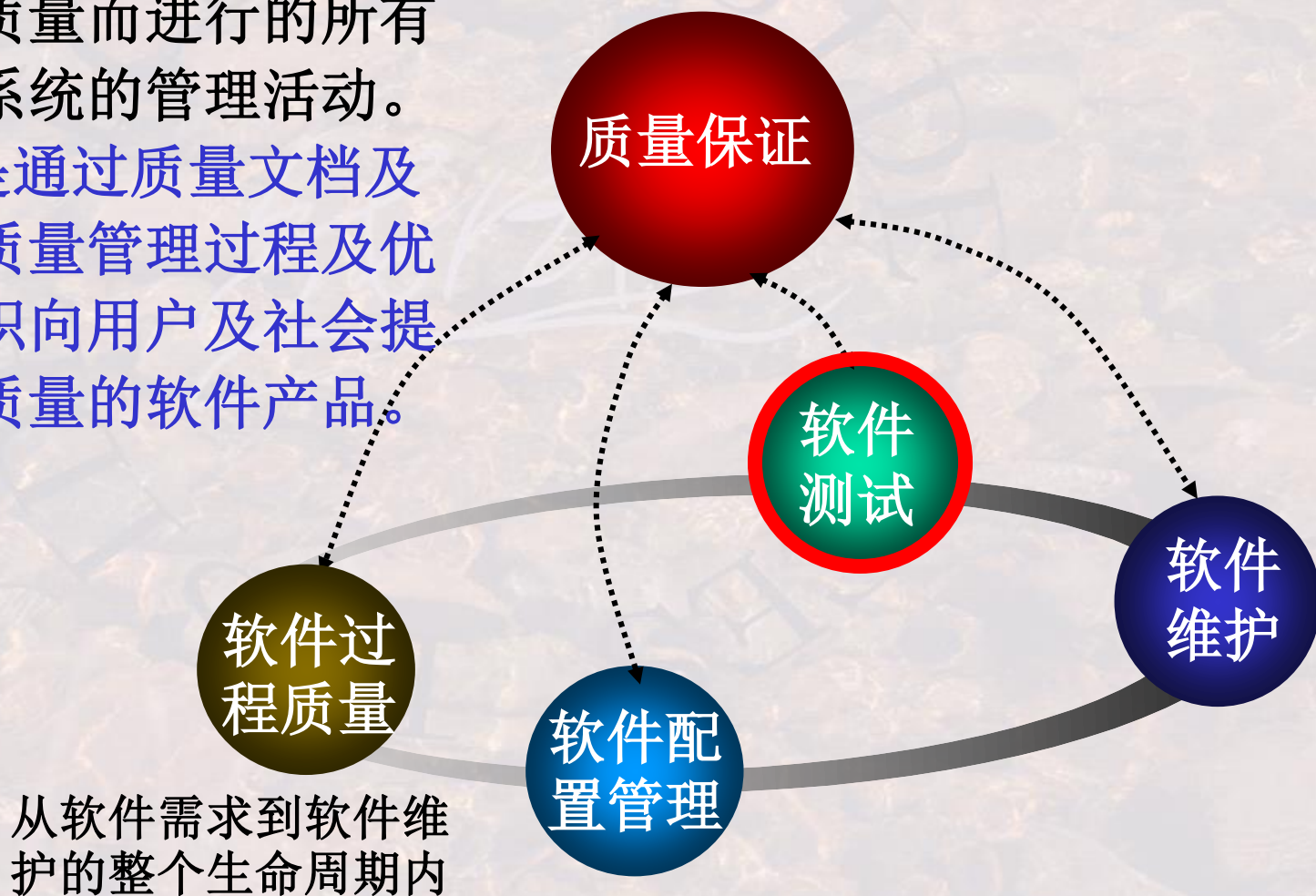
改正一个错误的相对成本[Boehm]



## (6)软件质量保证及其相关活动？

软件质量保证，是软件生命周期内，为了确定、达到和维护需要的软件质量而进行的所有有计划、有系统的管理活动。

■根本目的是通过质量文档及评测评审、质量管理过程及优化，保证组织向用户及社会提供满意的高质量的软件产品。



- 软件测试概念

# 软件测试概念

菲波那切数列计算：已知 $F_i$ 的计算公式如下，求 $F_n$ 取模17的结果。

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

**Main()**

```
{ int fn, n;
// 输入n的语句;
fn = f(n);
fn = fn%17;
// 输出n,fn的语句;
}

int f(int n)
{ int sum;
  if(n==0 || n==1) return 1;
  sum=f(n-1)+f(n-2);
  return sum;
}
```

● $n=1, \dots, 1000$ 时是否正确？  
 ● $n > 10000$ 时是否正确？  
 ● $n$ 取任何值都正确吗？  
 ？什么情况正确？

该图展示了库存流水账的界面和相关的Java代码。界面部分显示了一个表格，列出了库存流水账的记录，包括序号、出入库日期、出入库单号、出入库类型、数量、单价和金额。代码部分展示了BillModel和BillController类的实现。BillModel类包含transfer方法，用于处理库存流水账的转移。BillController类包含iAccount方法，用于处理账户的利息计算。图中用红色箭头标注了代码中的关键部分，如transfer方法的调用和iAccount方法的实现。

有缺陷吗？

性能如何？

怎样验证？

软件测试



- 传统：测试是一种旨在评估一个程序或系统的属性或能力，确定它是否符合其所需结果的活动。

- Myers：测试是为了发现缺陷而执行一个程序或系统的过程。

明确提出了“**在程序中寻找缺陷**”是测试的目的。

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

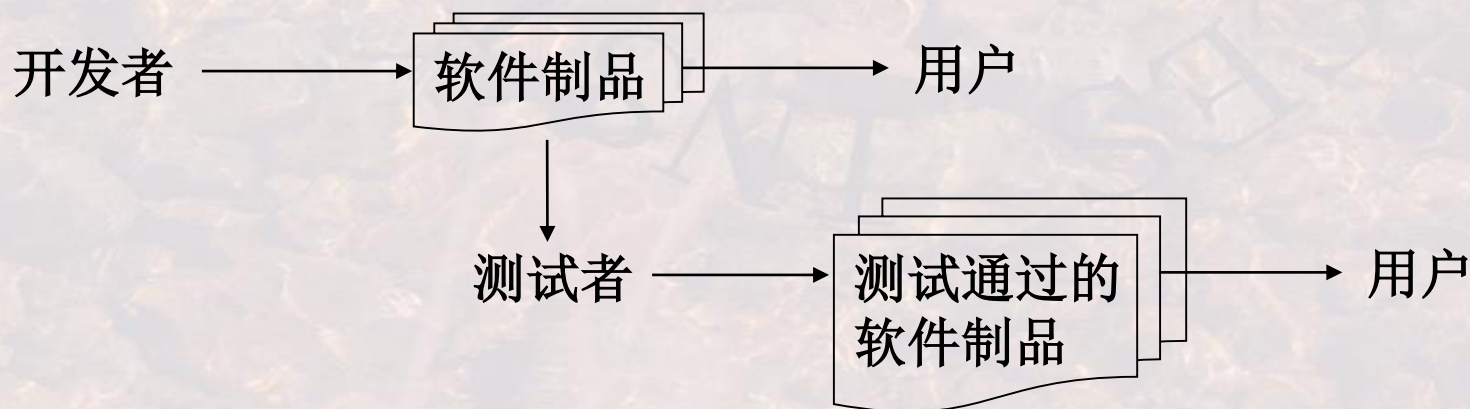
该定义明确提出了软件测试以“**检验是否满足需求**”为目标。

### 软件测试

- 发现软件缺陷;
- 交付给用户合格产品的一种质量保证手段----“未发现软件缺陷的软件产品即为合格产品”。

### 软件缺陷

- 软件出现了产品说明书中指明不会出现的**错误**;
- 软件未达到产品说明书中已经标明的功能;
- 软件未达到产品说明书中虽未指出但应当达到的目标;
- 最终用户认为该软件使用效果不良。



### 软件缺陷及其特征?

“看不到” --缺陷不易被看到

“看到但是抓不到” --发现了缺陷，但不易找到引发问题的原因

设计者的失误，导致系统中留有错误的设计 --故障(**fault**)；这些故障导致系统的错误执行--错误(**error**)；由于错误导致系统的错误输出--失败(**failure**)。

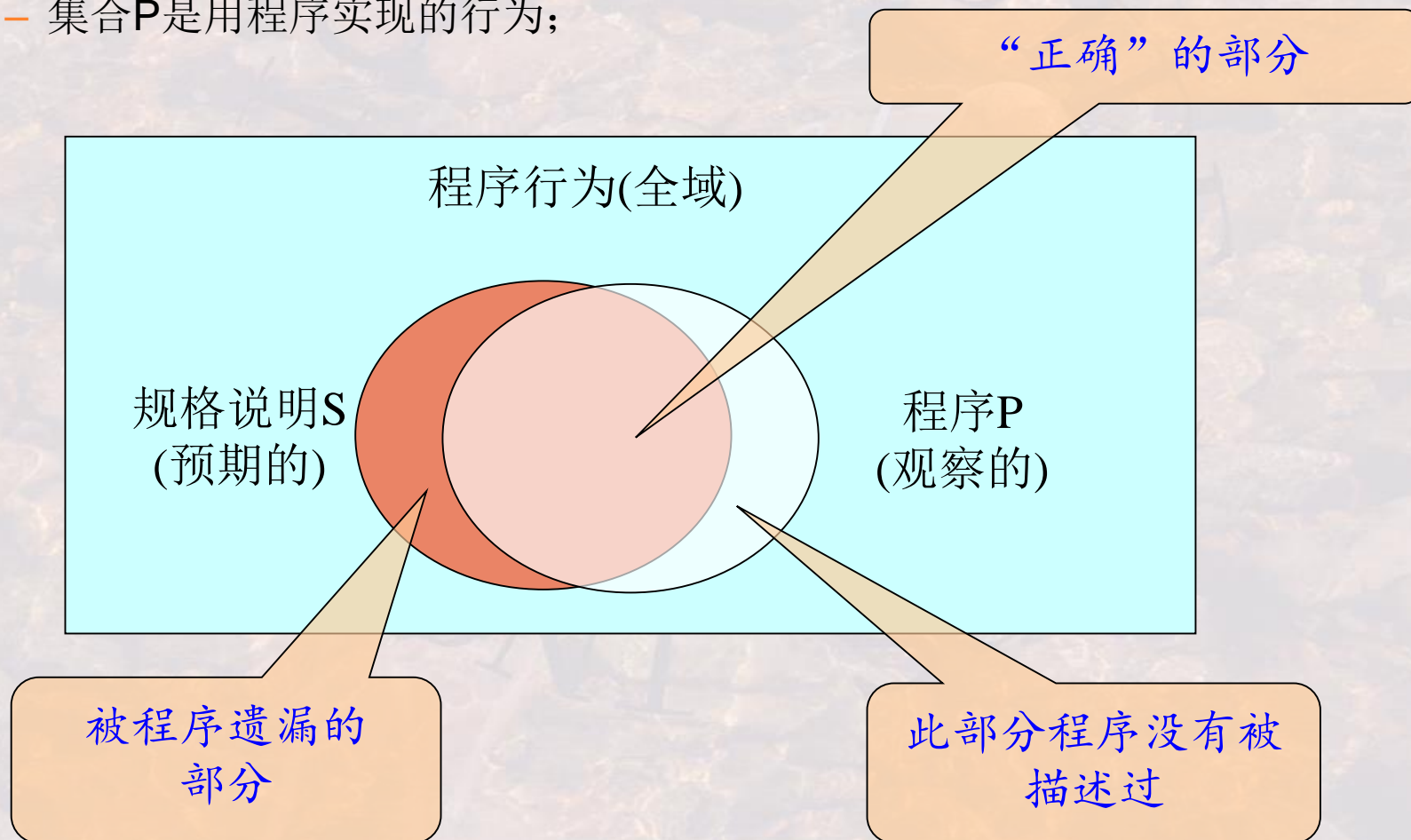
### Prof. Brian Randell:

- A system **failure** occurs when the delivered service is adjudged to have deviated from fulfilling the system function.
- An **error** is that part of the system state which is *liable to lead to subsequent failure*: an error affecting the service is an indication that a failure occurs or has occurred. The *adjudged or hypothesised cause* of an error is a **fault**.  
(Note: errors do not necessarily lead to failures – this may be avoided by chance or design; component failures do not necessarily constitute faults to the surrounding system – this depends on how the surrounding system is relying on the component).
- These three concepts (an **event**, a **state**, and a **cause**) must be distinguished, whatever names you choose to use for them.

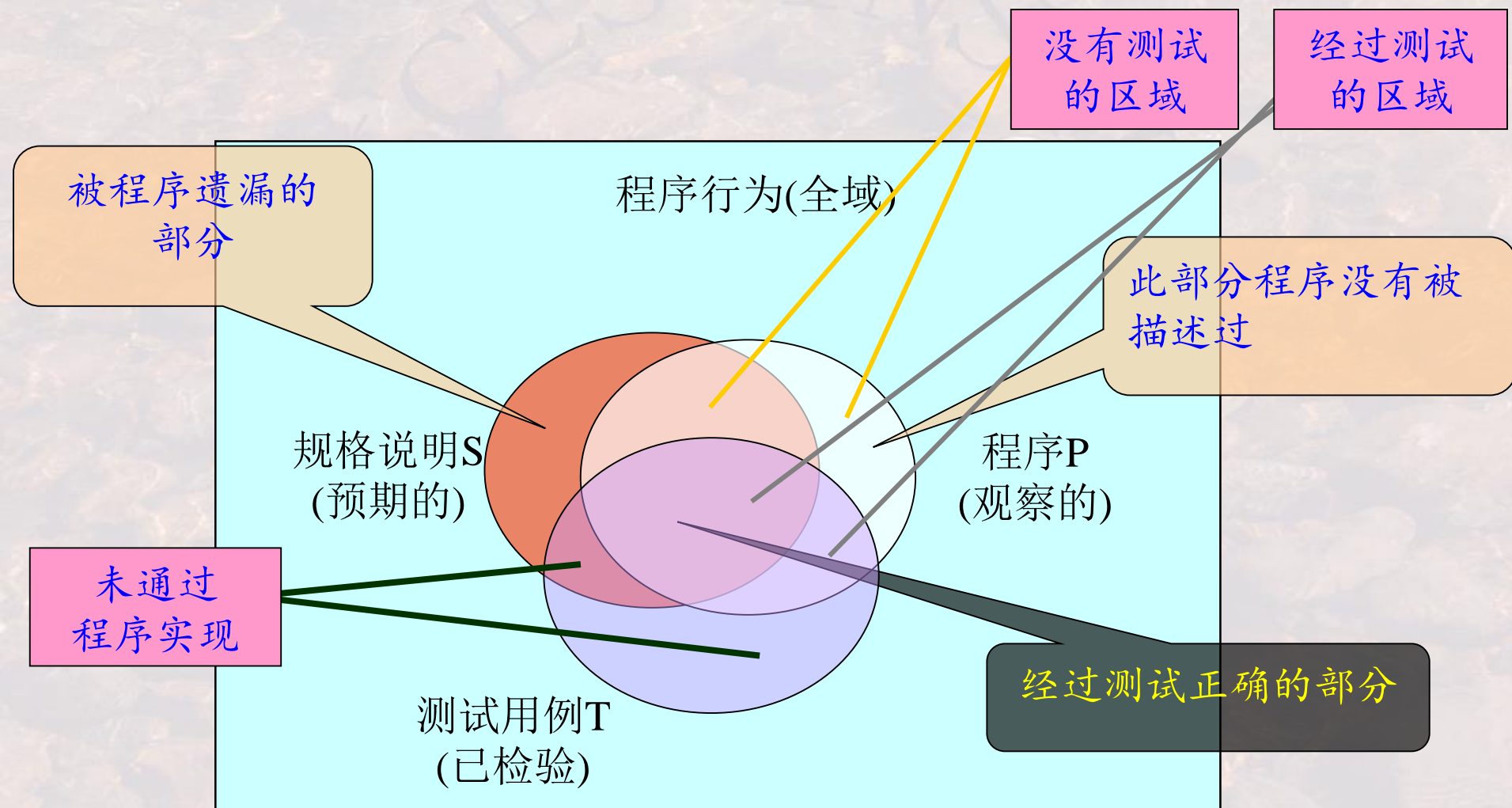


## (4) 用Venn Diagram来理解测试

- 考虑一个程序行为的全域，给定一段程序及其规格说明
  - 集合S是规格说明中所描述的预期实现的行为；
  - 集合P是用程序实现的行为；



- 设计测试用例(测试数据+期望结果)集合T



## (5) 软件测试的意义在哪里？

### 软件测试目的的再理解

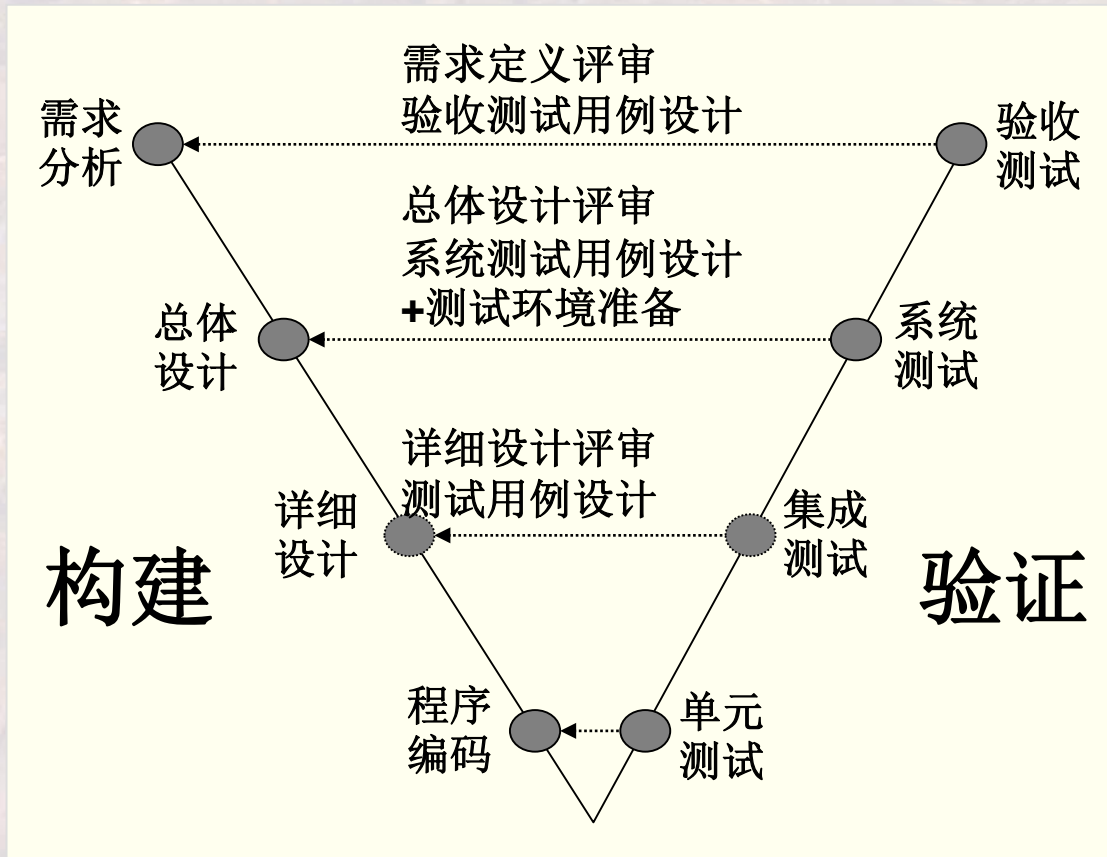
- 测试是为了发现错误而执行程序的过程
  - 测试是为了证明程序有错误，而不是证明程序无错误；
  - 一个好的测试用例是在于它能发现至今未发现的错误；
  - 一个成功的测试是发现了至今未发现的错误的测试。
- 
- ✓ 发现软件缺陷
  - ✓ 发现软件缺陷，尽可能早一些
  - ✓ 发现软件缺陷，尽可能早一些，并确保其得以修复



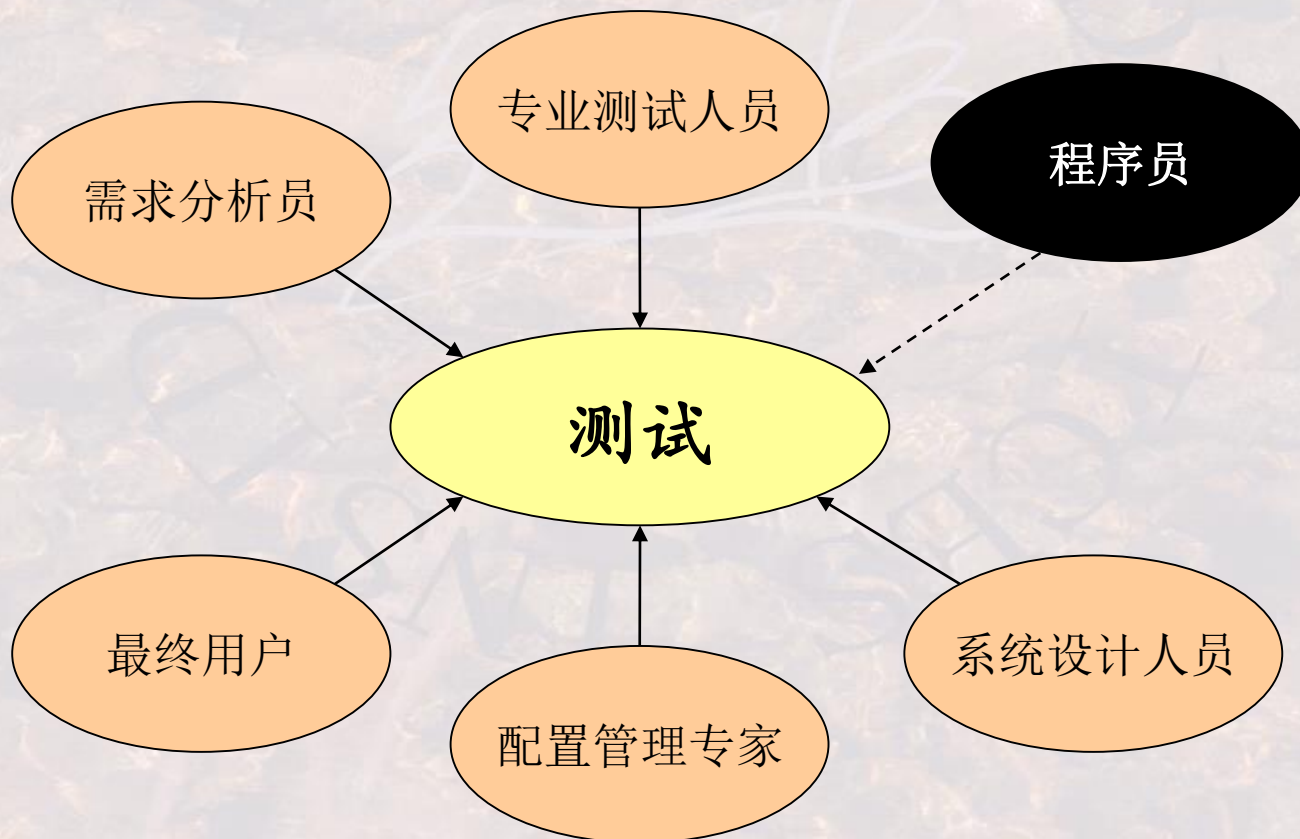
### 跨软件生命周期的软件测试

(狭义的)软件测试，是为了发现错误而执行程序的过程；  
(广义的)软件测试，是将测试延伸到需求评审、设计审查活动中去。

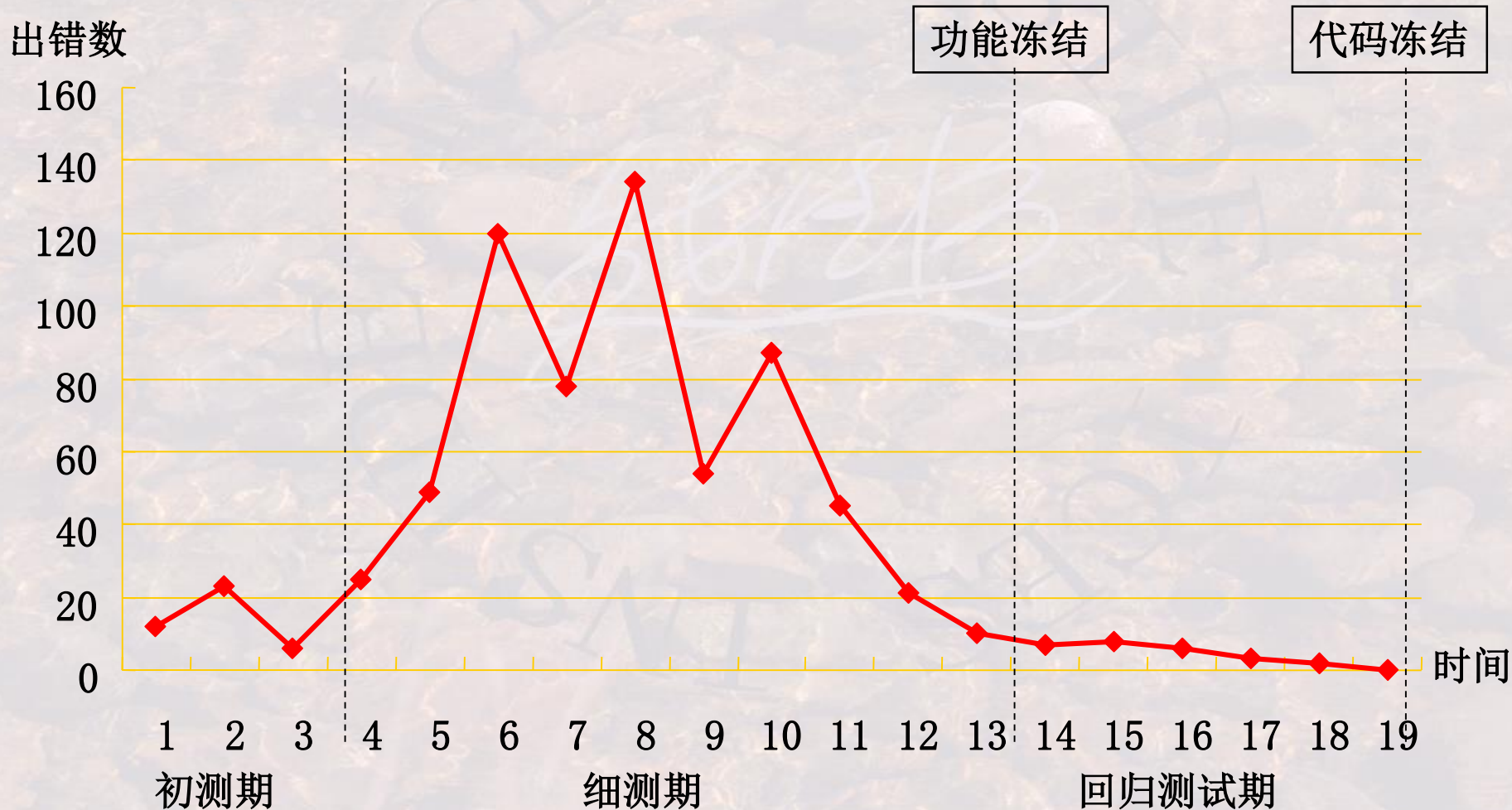
软件测试是贯穿整个软件开发生命周期、对软件产品(包括阶段性产品)进行验证和确认的活动过程，其目的是尽快尽早地发现在软件产品中所存在的各种问题——与用户需求、预先定义的不一致性。



- 软件测试并不等于程序测试，应贯穿于软件定义与开发的各个阶段。
- 需要各类人员的参与，不应由程序员独立完成



### 基于软件测试的成熟软件产品形成过程





# 软件测试过程

## (1) 软件测试流程是怎样的？

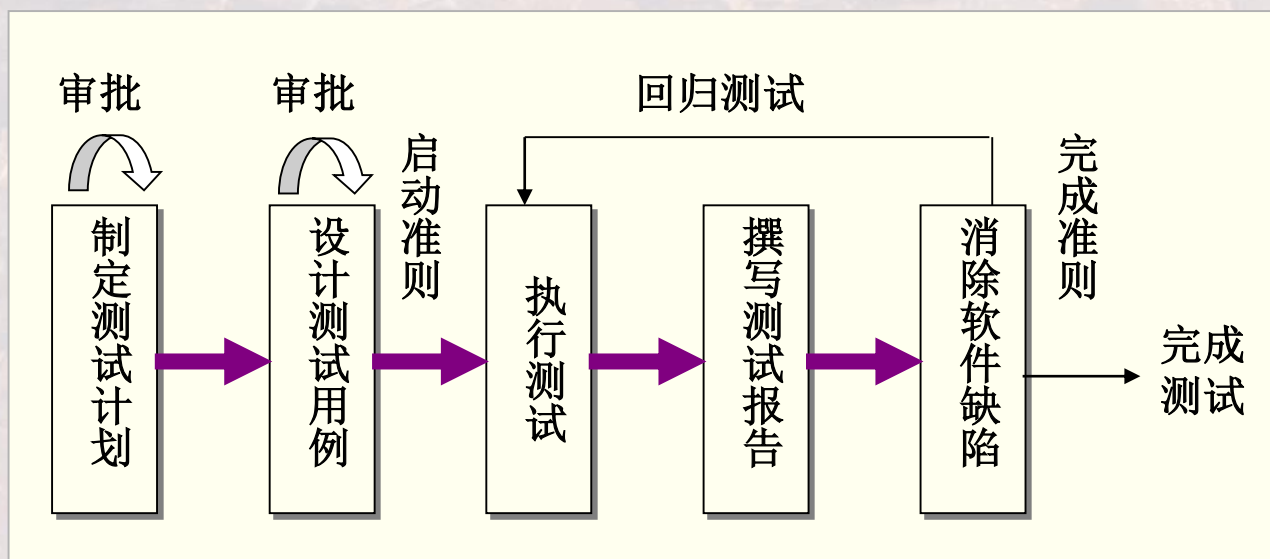
### 怎样保证软件测试是有效的---测试流程

**制定测试计划：**即制定软件测试和验证整体方案；

**设计测试用例：**即设计测试集合及测试步骤等；

**执行测试：**依据测试用例，运行软件进行测试，观察和记录输出结果；

**撰写测试报告：**依据测试用例找出的问题，撰写和提交测试报告，报告测试结果。

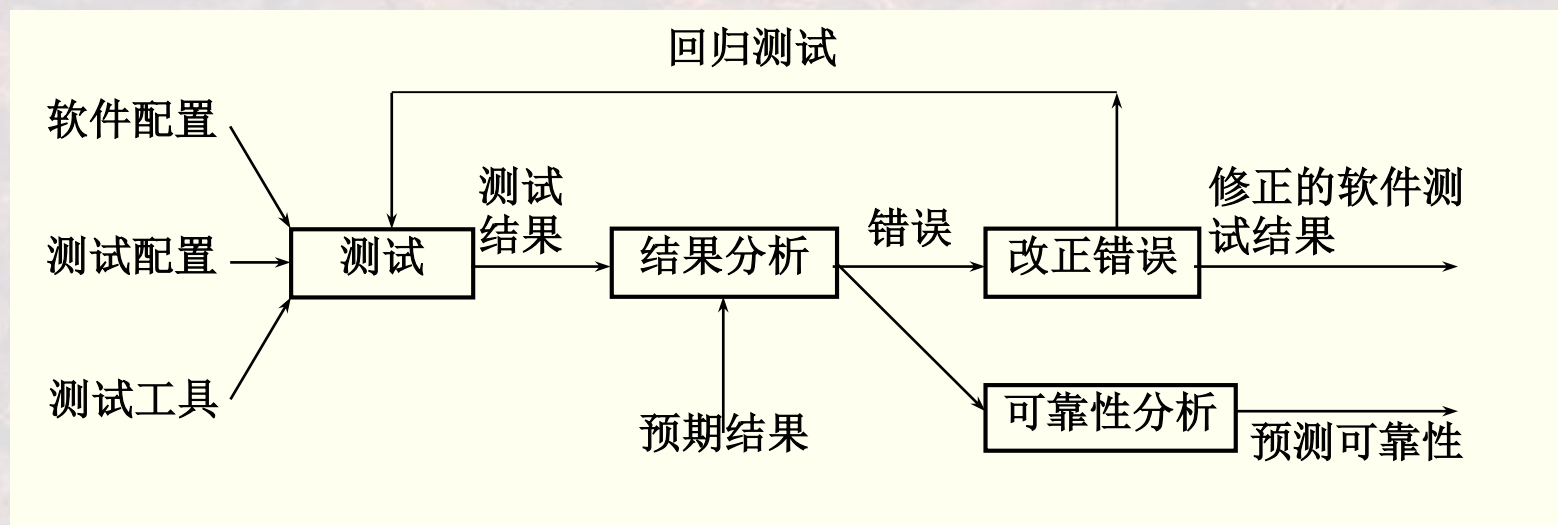


### 软件测试输入信息

- 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、测试驱动程序等
- 测试工具：为提高测试效率提供服务的工具，如动静态分析工具、自动化测试工具、测试结果分析工具等

### 软件测试输出信息

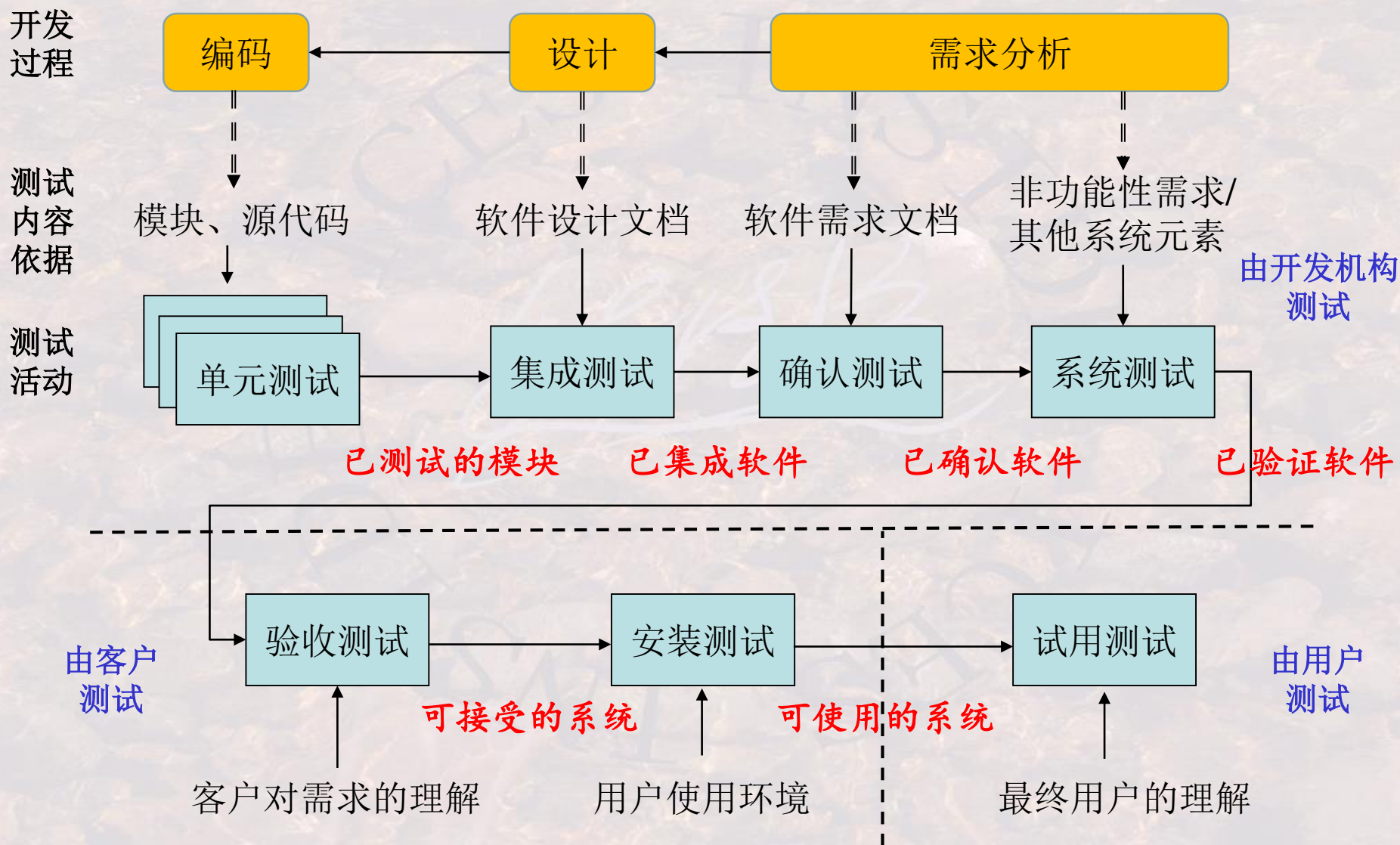
- 测试结果：需要对其分析，将测试结果同预期结果比较





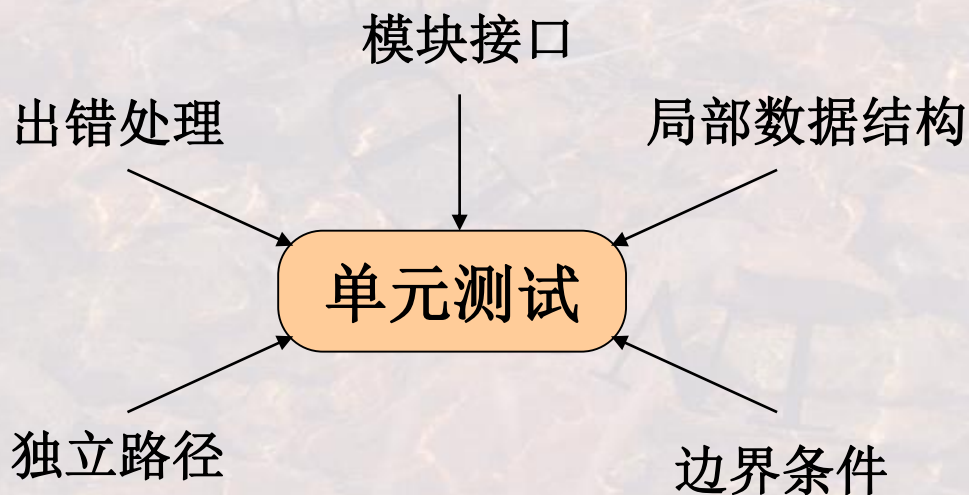
# 软件测试过程

## (3)软件测试活动

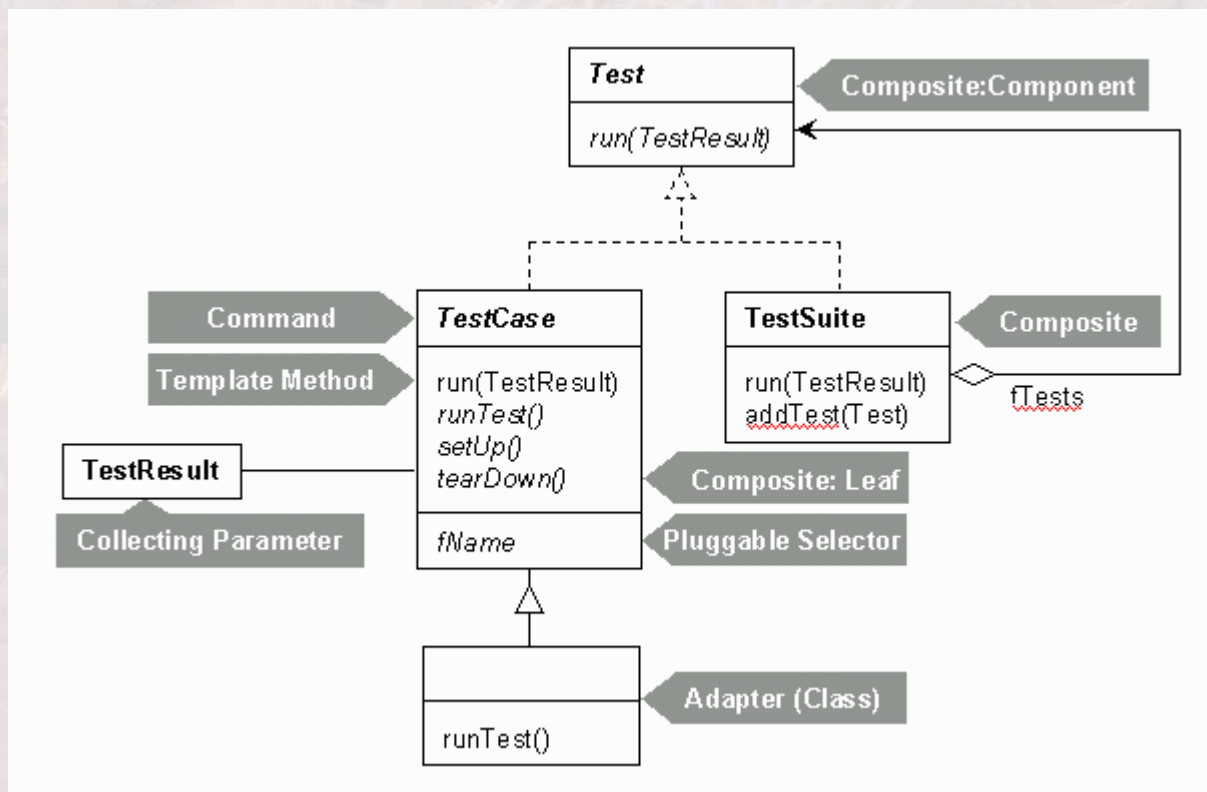


### 单元测试

验证完成的代码是否可以按照其所设想的方式执行而产出符合预期值的结果，**确保产生符合需求的可靠程序单元**（结构化程序为模块，面向对象程序为类），**侧重于测试构件中的内部处理逻辑和数据结构**。



- 单元测试需要频繁重复运行，需要工具提高测试效率
- 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。**





### ■ 使用步骤

- 1.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：斐波那契数列计算，以(0,1),(1,1),(4,5)(16,1597)为测试用例
- 2.执行测试程序：调用被测程序中的对象或方法
- 3.判断结果：比较调用结果和预期结果，判断是否通过测试

The screenshot displays the Eclipse IDE with the following components:

- Left Panel:** Shows the source code of `Fibonacci.java` and `FibonacciTest.java`. The `FibonacciTest` class includes test methods for `fibonacciCal(0)`, `fibonacciCal(1)`, `fibonacciCal(4)`, and `fibonacciCal(16)`.
- Center Panel:** Shows the execution results of the JUnit tests. It indicates that the tests were finished after 0.012 seconds, with 1/1 runs, 0 errors, and 0 failures. A callout box points to the results, asking "是否通过测试" (Did it pass the test?).
- Right Panel:** Shows the source code of `FibonacciTest.java`, which uses JUnit annotations like `@Before`, `@After`, and `@Test` to set up and verify the tests.

Callout boxes highlight specific test cases and their results:

- One callout points to the test results for `fibonacciCal(0)`, `fibonacciCal(1)`, `fibonacciCal(4)`, and `fibonacciCal(16)`, asking "是否正确" (Is it correct?).
- Another callout points to the test results for `fibonacciCal(0)`, `fibonacciCal(1)`, `fibonacciCal(4)`, and `fibonacciCal(16)`, asking "是否正确" (Is it correct?).

(斐波那契数列计算)

单元测试执行结果

(斐波那契数列计算)

### 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的测试。

集成测试是构造软件体系结构的系统化技术，同时也是进行一些**旨在发现与接口相关的错误的测试**。

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

#### ■ 集成测试考虑的主要问题

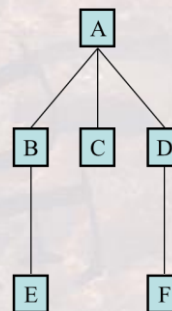
- 模块连接后，模块接口的数据是否会丢失
- 子功能组合后，能否达到预期要求的父功能
- 模块的功能是否会相互产生不利的影响
- 全局数据结构是否有问题
- 模块的误差累积是否会放大



### ■ 集成测试方法

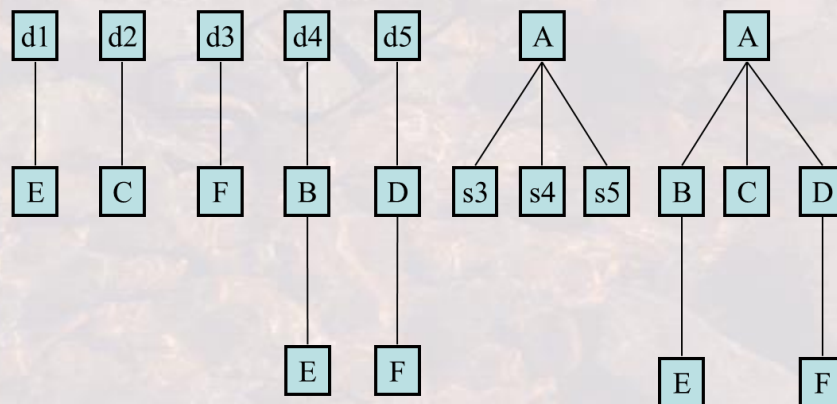
**整体集成方式：**把所有模块按设计要求一次全部组装起来，然后进行整体测试，“一步到位”的集成方式；

- **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；



**增量式集成方式：**逐步将新模块加入并测试，可分为自顶向下、自底向上、上下结合三种增量方式

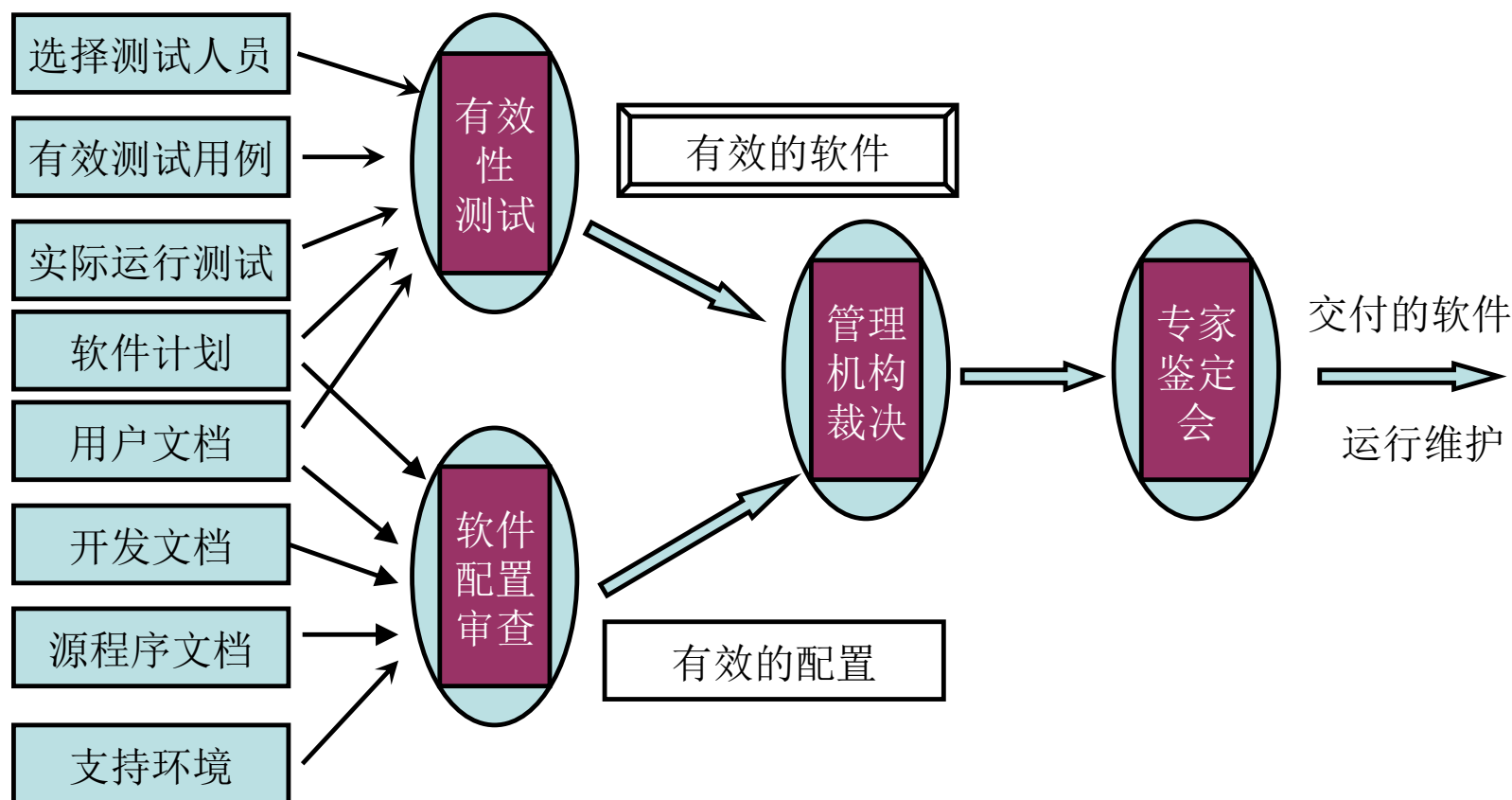
- **优点：**发现错误能力强，不易遗漏错误；
- **缺点：**工作量大





### 确认测试

验证软件的有效性，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。





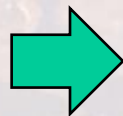
### 验收测试

**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。

#### $\alpha$ 测试

$\alpha$ 测试是指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软件产品(称为 $\alpha$ 版本)进行测试，试图发现错误并修正；经过 $\alpha$ 测试调整的软件产品称为 $\beta$ 版本。



#### $\beta$ 测试

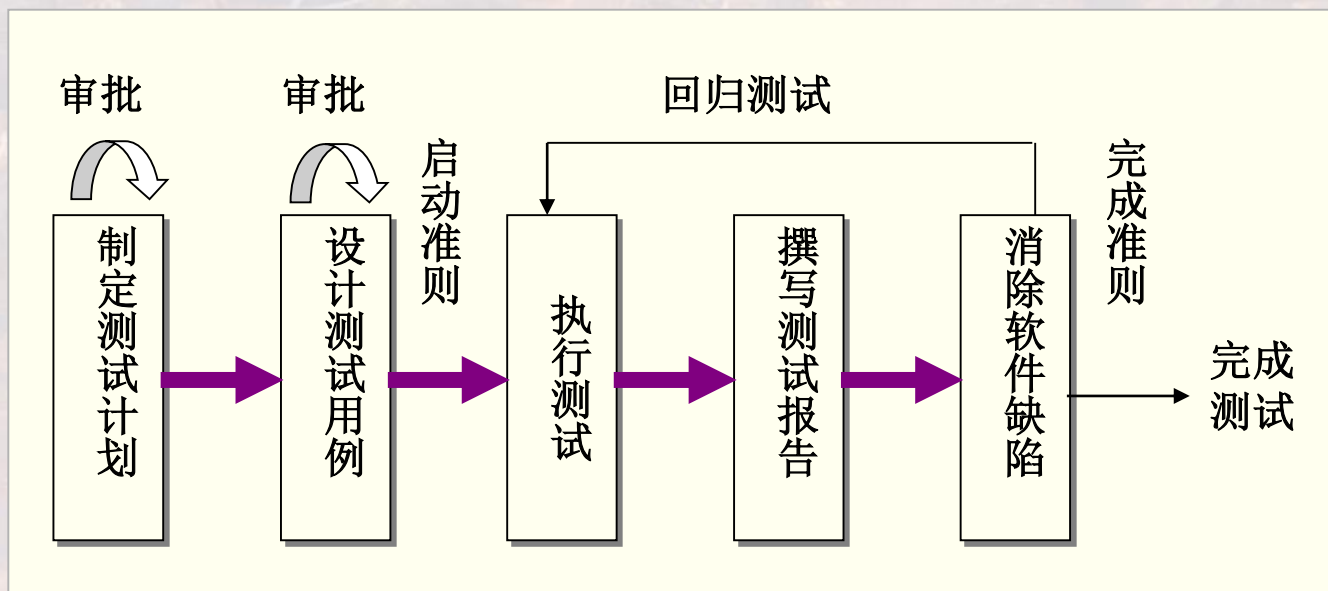
$\beta$ 测试是软件的多个用户在一个或多个用户的实际使用环境下进行的测试。开发者通常不在测试现场，**Beta**测试不能由程序员或测试员完成。



### 回归测试

验证对系统的变更没有影响到以前的功能，**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的。

回归测试**可以发生在软件测试的任何阶段**，包括单元测试、集成测试和系统测试等。往往需要大量频繁的重复性劳动，可通过自动化测试工具来提高测试效率。



# 软件测试技术

### 运行程序?

- **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。

■ **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。

■ **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。

### 分析代码?



软件测试需要通过从无限执行域中适度选择出测试用例的有限集合，动态验证程序是否提供期望行为的工作。

已知 $F_1$ 的计算公式如下，求 $F_n$ 取模17的结果。

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

Main()

```
{ int fn, n;
/* 输入n的语句;
fn = f(n);
fn = fn%17;
/* 输出n,fn的语句;
}
```

int f(int n)

```
{ int sum;
if(n==0 || n==1) return 1;
sum=f(n-1)+f(n-2);
return sum;
}
```

已知内部  
结构相关  
的文件进  
行测试：  
白盒测试

已知 $F_1$ 的计算公式如下，求 $F_n$ 取模17的结果。

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

Main()

```
{ int fn, n;
/* 输入n的语句;
fn = f(n);
fn = fn%17;
/* 输出n,fn的语句;
}
```

int f(int n)

```
{ int sum;
if(n==0 || n==1) return 1;
sum=f(n-1)+f(n-2);
return sum;
}
```

在不知内  
部结构的  
假设下进  
行测试：  
黑盒测试

- ✓ “动态”意味着在特定输入下执行程序；
- ✓ “有限”指测试是在所有可能测试中的一个子集上进行；许多测试技术都涉及到测试集合的选择问题和方法；
- ✓ “期望”指确定观察的输出是可接受的或不可接受的。

### 测试用例

- 测试用例是为特定的测试目的而设计的一组**测试输入、执行条件和预期的结果**组合。
- 测试用例 = {执行条件+测试数据+期望结果}
- 测试用例是执行的最小测试实体。
- 测试用例就是**设计一个场景**，使软件程序在这种场景下，必须能够正常运行并且达到程序所设计的执行结果。
- 测试结果 = {执行条件+测试数据+期望结果+实际结果}

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。

### 测试用例的设计原则：

#### ■ 测试用例的代表性：

- 最有可能抓住错误的，一组相似测试用例中最有效的。
- 能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。

#### ■ 测试结果的可判定性：

- 测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

#### ■ 测试结果的可再现性：

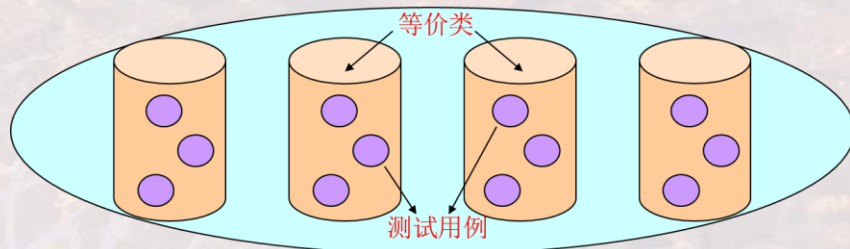
- 对同样的测试用例，系统的执行结果应当是相同的。



### ■ 测试用例设计技术

#### 等价类划分

将程序的输入划分为若干个数据类（等价类），并合理地假定“**在该等价类中的各个输入数据对于揭露程序中的错误都是等效的**”。



—**有效等价类**：合理的、有意义的输入数据组合，可检验程序是否实现了功能；

—**无效等价类**：不合理或无意义的输入数据集，可检验程序应对意外的能力。

—设计测试用例时有效、无效等价类都要考虑，“**完备测试、避免冗余**”

例：计算不超过**1000**的斐波那契数程序，输入**n**可划分为：

- 有效等价类： $n=0$ 、 $n=1$ 、 $1 < n \leq 1000$ 且 $n$ 为整数；进一步可细分为 $n=0$ 、 $n=1$ 、 $1 < n \leq 100$ 、 $100 < n \leq 500$ 、...  $900 < n \leq 1000$ 等。
- 无效等价类： $n < 0$ 、 $n > 1000$ 、 $0 \leq n \leq 1000$ 且为非整数三个无效等价类。
- 针对上述等价类设计测试用例

### ■ 测试用例设计技术



长期的测试经验表明，大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部。

**“错误隐藏在角落里，聚集在边界处”**

例子：

在最容易出现错误的输入或输出范围的边界处取值，更容易发现错误。

边界值分析是等价类测试的补充，主要是考虑等价类的边界条件，在等价类的“边缘”选择元素。

- 对16-bit 的整数，32767和-32768是边界
- 屏幕上光标在最左上、最右下位置
- 报表的第一行和最后一行
- 数组元素的第一个和最后一个
- 循环的第0次、第1次和倒数第2次、最后1次

- 等价类划分和边界值分析多用于黑盒测试用例的设计，在白盒测试用例设计时，要用逻辑覆盖度来衡量测试的完整性，度量测试用例覆盖所有代码行、逻辑表达式、所有执行路径等的程度，主要方法有基本路径测试等。



### 压力测试

- 检查系统在**资源超负荷情况**下的表现，特别是对系统的处理时间、内存和**CPU** 可用性、磁盘空间和网络带宽等有什么影响。压力测试常同性能测试一起进行。

- 如：各类电子商务网站和社交网站，每分钟访问用户数均在万、十万量级，促销或重要节日时，甚至会达到百万量级。如2014年淘宝双11促销活动，参与用户近亿，每分钟支付成功的峰值为79万笔。

如此大规模的访问量下：

- 系统能否正常无故障运行？
- 系统能否正常响应用户请求？  
响应速度是否会下降？
- 软件的性能是否达到了设计要求？
- 各种资源的占用率如何？  
性能瓶颈在何处？





### ■ 以压力测试中的**多用户并发访问测试**为例

#### — 关键点1：测试用例的设计。

- 软件的功能众多，用户的使用方式多样，应选取软件的哪些功能的哪些组合进行压力测试？

- **频繁使用的、重要的、或需要占用大量资源的功能和流程**

用户登录→浏览商品→查看细节→购买操作

用户登录**5%**  
浏览商品**40%**  
查看细节**30%**  
购买操作**20%**  
其他请求**5%**

#### — 关键点2：如何实现多用户并发访问

- **真实用户访问**—让真实用户运行软件。适用于小规模并发访问测试，测试大规模并发访问时，往往无法做到。
- **计算机模拟**—通过测试工具软件。模拟需要数量的用户实例和操作软件的行为，在真实的软件运行环境中模拟用户访问操作软件，以检测软件系统各方面的性能和抗压性。“**仿真的用户，真实的软件**”，适用于测试大规模并发访问。

### ■ 以压力测试中的多用户并发访问测试为例

#### — 关键点3：运行场景的设计

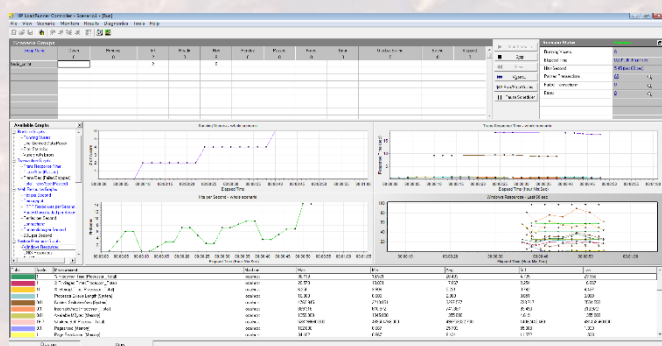
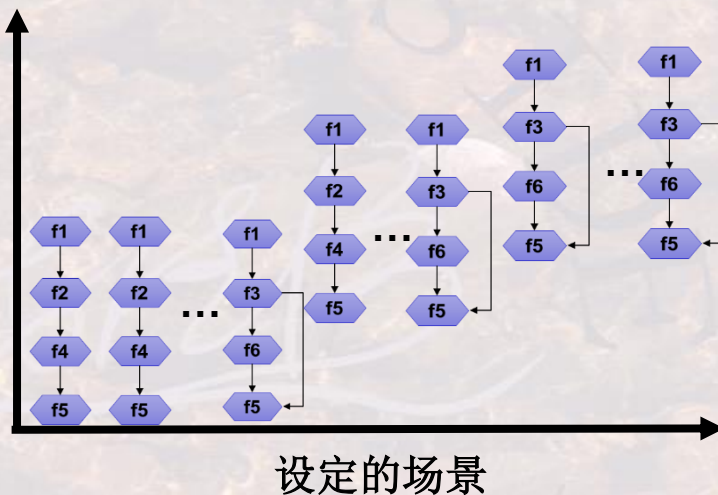
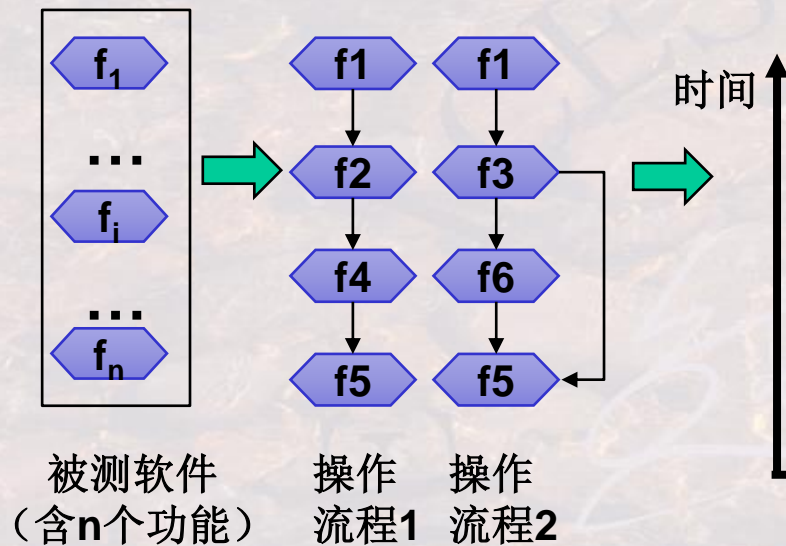
- 为保证测试时的使用场景尽可能接近真实情况，需要设定若干参数
- 如：设定选定的各种流程和操作的执行顺序、用户各操作之间的时间间隔、模拟的用户数量变化方式(恒定或随时间递增递减)、各种浏览器的选择及比例、各种带宽的网络及比例等。

#### — 关键点4：效能数据的采集和分析

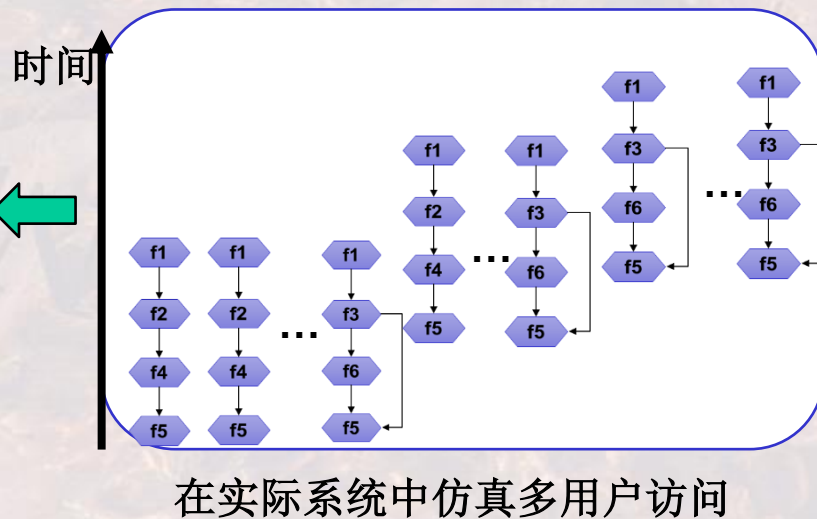
- 服务器上是否出现错误，系统能否承受压力
- 收集网络服务器中的各种性能数据：响应时间变化、内存占用、CPU占用、网络带宽变化等，为软件性能优化提供参考和支持

1.选取代表性的操作流程，  
使用“录制”技术记录用户的操作过程

2.设定多用户使用软件的场景  
(多用户如何使用软件)



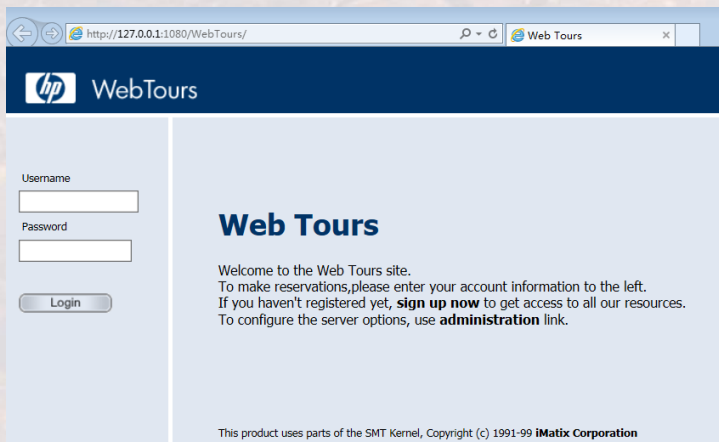
4.采集数据，汇总统计，性能分析



3.在真实环境下  
根据场景运行测试，  
访问软件



- 订票网站例子（以HP Loadrunner 压力测试软件为例）
- 第1步：确定要测试的核心业务，登陆→查询航班→选择航班→支付



WebTours

Username  
Password

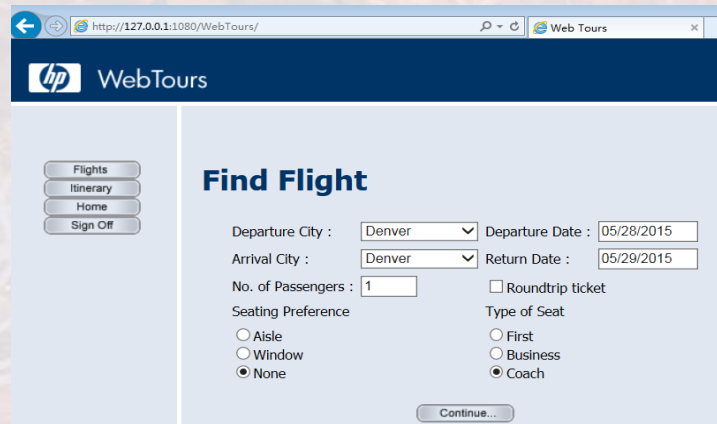
Login

**Web Tours**

Welcome to the Web Tours site.  
To make reservations, please enter your account information to the left.  
If you haven't registered yet, **sign up now** to get access to all our resources.  
To configure the server options, use **administration** link.

This product uses parts of the SMT Kernel, Copyright (c) 1991-99 iMatix Corporation

登陆



WebTours


Flights  
Itinerary  
Home  
Sign Off

**Find Flight**

Departure City : Denver Departure Date : 05/28/2015  
Arrival City : Denver Return Date : 05/29/2015  
No. of Passengers : 1 ☐ Roundtrip ticket  
Seating Preference  
☐ Aisle ☐ First  
☐ Window ☐ Business  
☒ None ☒ Coach

Continue...

查询航班



WebTours

Flights  
Itinerary  
Home  
Sign Off

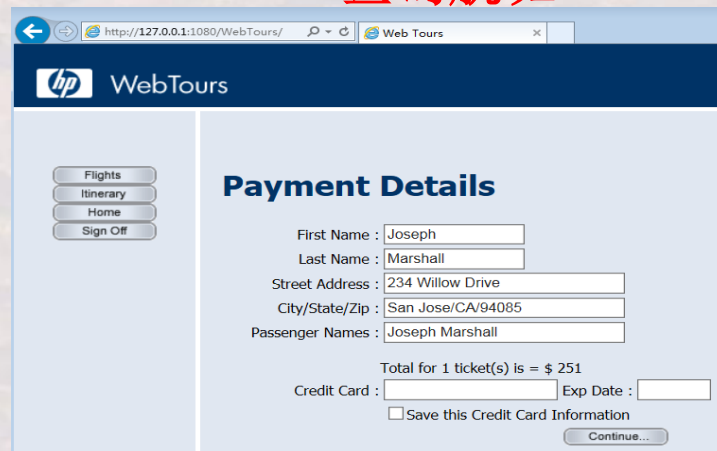
**Find Flight**

Flight departing from Denver to Los Angeles on 05/28/2015

Flight	Departure time	Cost
<input checked="" type="radio"/> Blue Sky Air 030	8am	\$ 251
<input type="radio"/> Blue Sky Air 031	1pm	\$ 224
<input type="radio"/> Blue Sky Air 032	5pm	\$ 238
<input type="radio"/> Blue Sky Air 033	11pm	\$ 206

Continue...

选择航班



WebTours

Flights  
Itinerary  
Home  
Sign Off

**Payment Details**

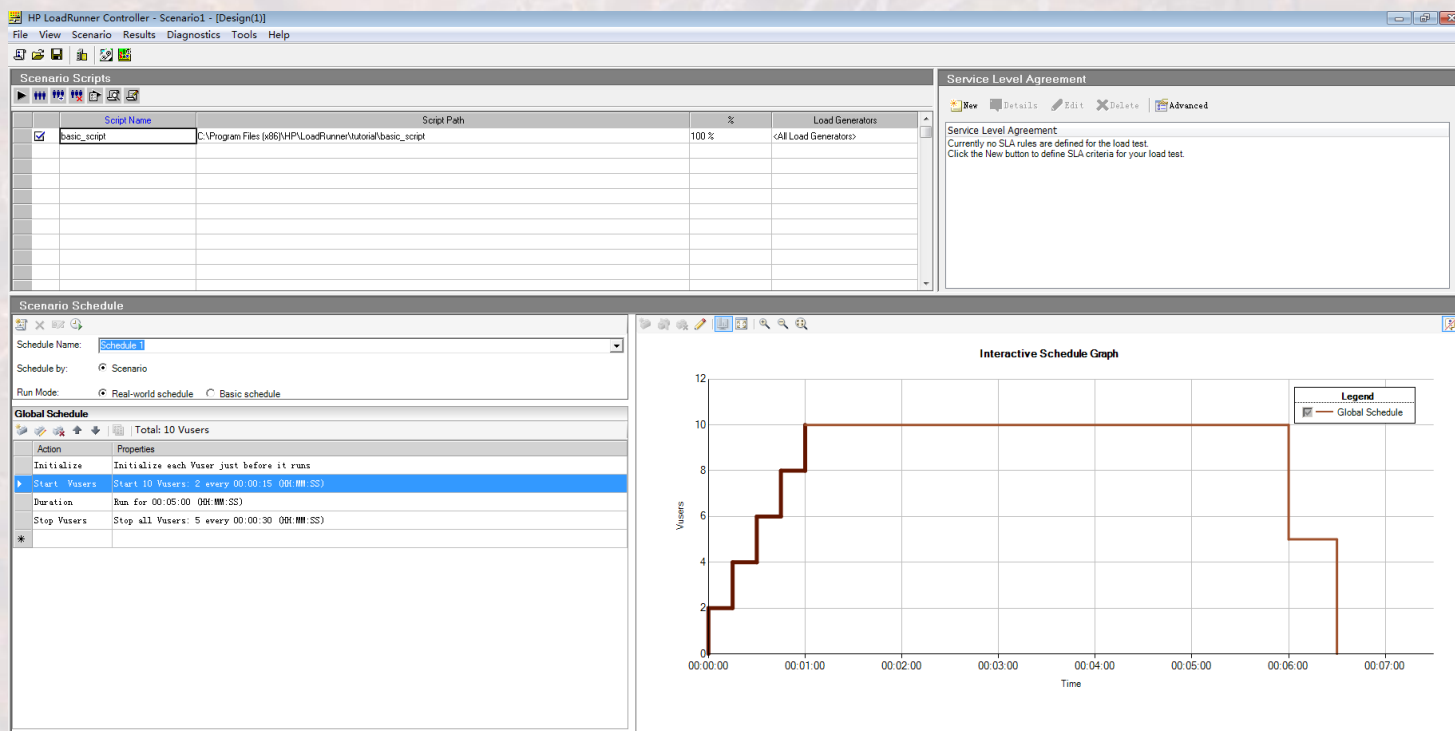
First Name : Joseph  
Last Name : Marshall  
Street Address : 234 Willow Drive  
City/State/Zip : San Jose/CA/94085  
Passenger Names : Joseph Marshall

Total for 1 ticket(s) is = \$ 251  
Credit Card : Exp Date :  
☐ Save this Credit Card Information

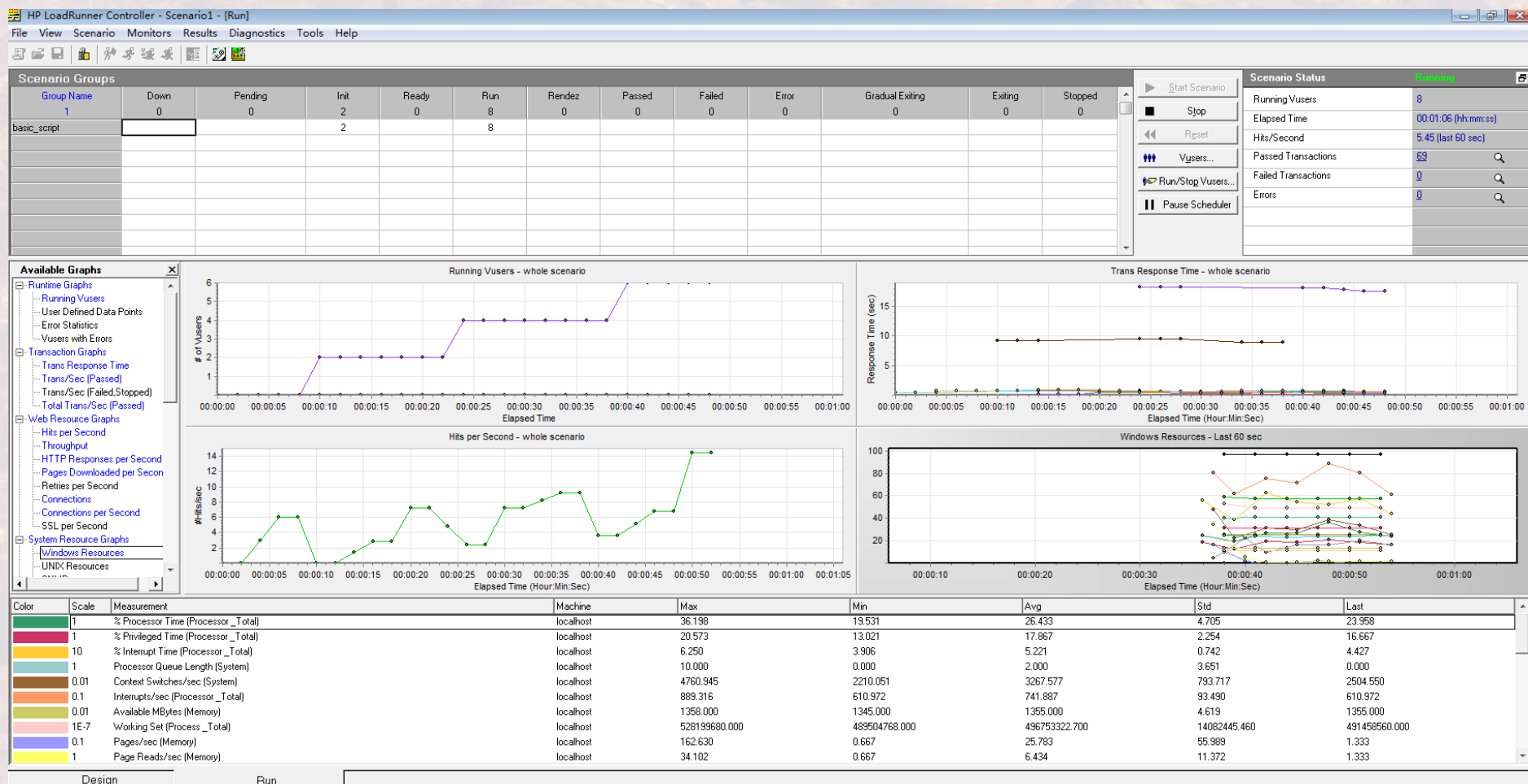
Continue...

支付

- 第2步：“录制”用户对典型流程的操作
  - 按照标准的方式操作软件，压力测试工具会录制下来所有的操作（鼠标移动和点击、键盘的输入等）。录制后的操作，将做为模板来设计各种仿真用户的操作场景，并能在测试时还原执行。
- 第3步：设置用户数和场景（停顿时间、用户数目递增方式等）

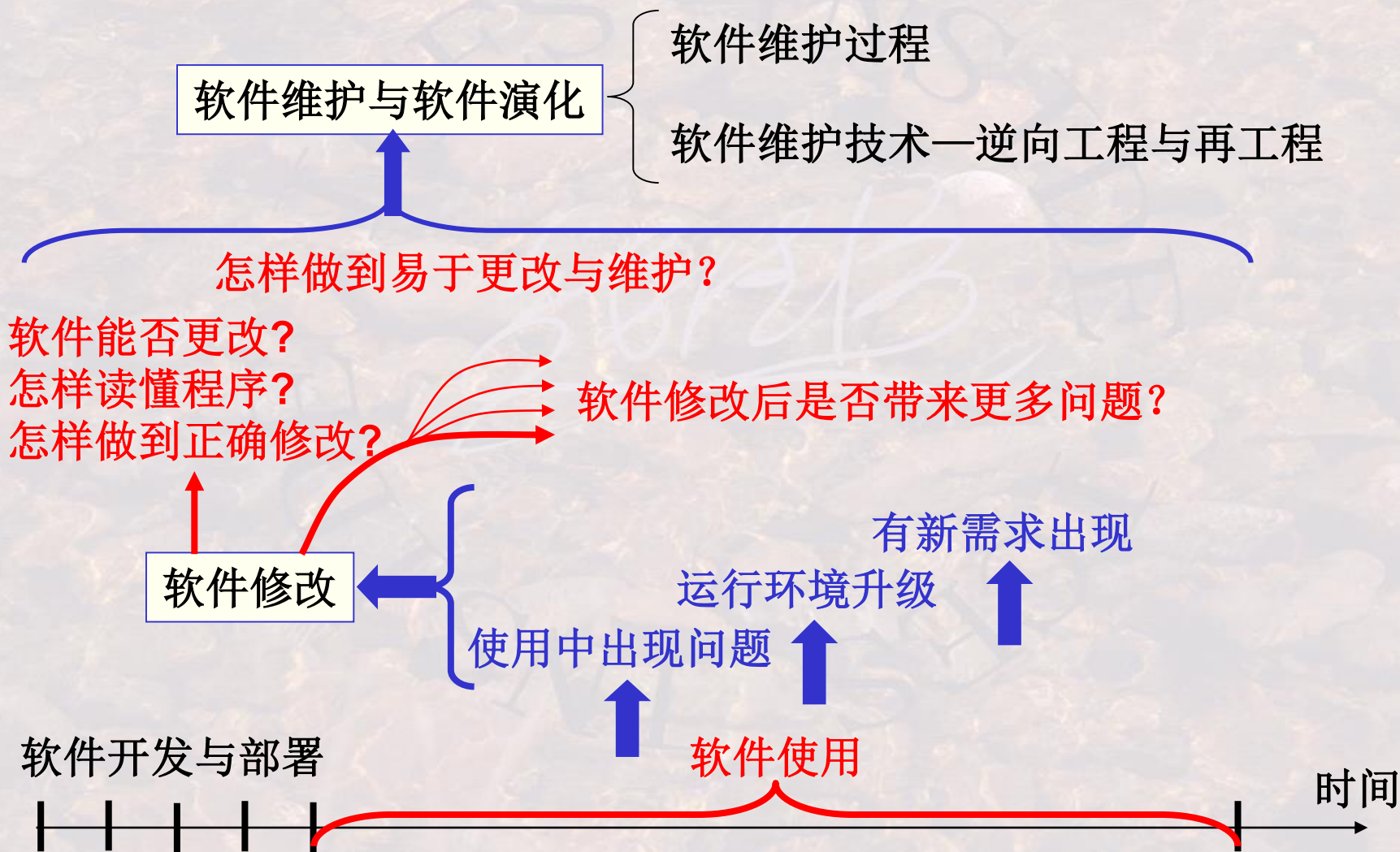


- 第4步：运行压力测试，收集数据
- 第5步：结果分析





# 软件维护与软件演化概念



## (2)什么是软件维护?什么是软件演化?

### 软件维护

- 是指软件系统交付使用以后, 为了改正错误或满足新的需要而修改软件的过程

### 国标GB/T 11457-95给出如下定义

在一软件产品交付使用后对其进行修改, 以纠正故障;

在一软件产品交付使用后对其进行修改, 以纠正故障、改进其性能和其它属性, 或使产品适应改变了的环境

### 软件演化

- 软件演化是指软件在交付以后, 对软件进行的一系列活动的总称。包括**软件维护**和**软件再工程**。
- 软件维护**阶段覆盖了从软件交付使用到软件被淘汰为止的整个时期。
- 软件再工程**的主要目的是为遗留系统转化为可演化系统提供一条现实可行的途径, 是在软件生命周期终止后开始的一个新的阶段。

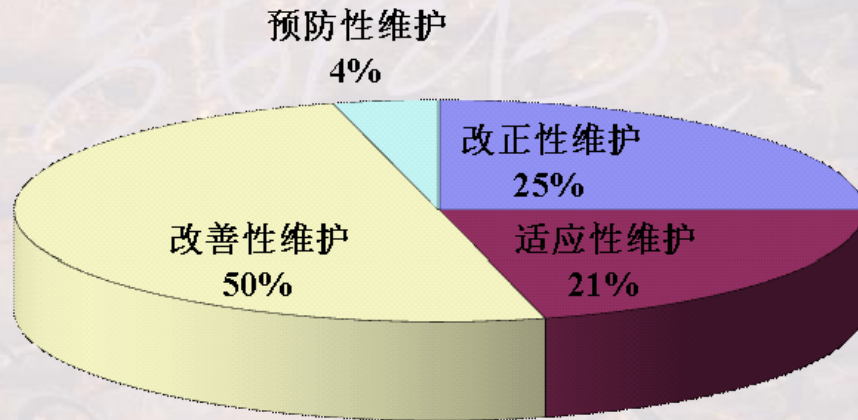


### 软件维护

- **改正性维护**：为改正软件系统中潜藏的错误而进行的活动。
- **适应性维护**：为适应软硬件运行环境的变化而修改软件的活动。
- **改善性维护**：根据用户在软件使用过程中提出的建设性意见而进行的维护活动。
- **预防性维护**：为了进一步改善软件系统的可维护性和可靠性，并为以后的改进奠定基础。

## (3)软件维护的分类？

- 实践表明，在几种维护活动中，**改善性维护所占的比重最大**，即大部分维护工作是改变和加强软件，而不是纠错。
  - 改善性维护不一定是救火式的紧急维修，而可以有计划、有预谋的一种再开发活动。
  - 来自用户要求扩充、加强软件功能、性能的维护活动约占整个维护工作的50%。



- **据统计：软件维护活动所花费的工作占整个生存期工作量的70%以上。**这是由于在漫长的软件运行过程中需要不断对软件进行修改，以改正新发现的错误、适应新的环境和用户新的要求，这些修改需要花费很多精力和时间，而且有时会引入新的错误。

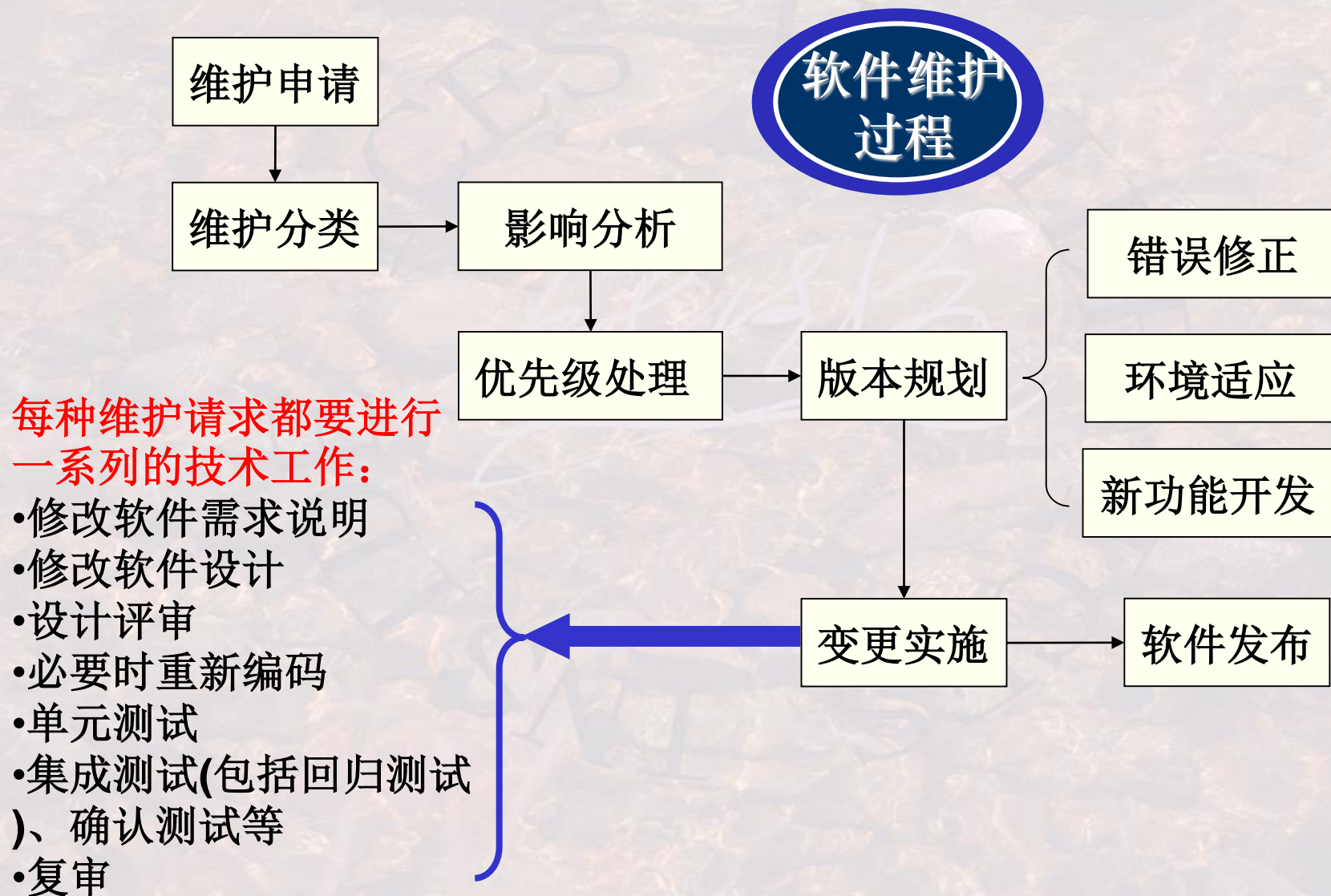
- 软件维护中出现的大部分问题都可归咎于**软件规划和开发方法的缺陷**：
  - 软件开发时采用急功近利还是放眼未来的态度，对软件维护影响极大，软件开发若不严格遵循软件开发标准，维护就会遇到许多困难。
- 例如：
  1. 读懂原开发人员写的程序通常相当困难
  2. 软件人员的流动性，使得软件维护时，很难与原开发人员沟通
  3. 没有文档或文档严重不足
  4. 软件设计时，欠考虑软件的可修改性
  5. 频繁的软件升级，要追踪软件的演化变得很困难，使软件难以修改

**因此：在设计和开发时，要有长远的规划，严格遵循开发标准和过程控制，提升软件的可维护性**

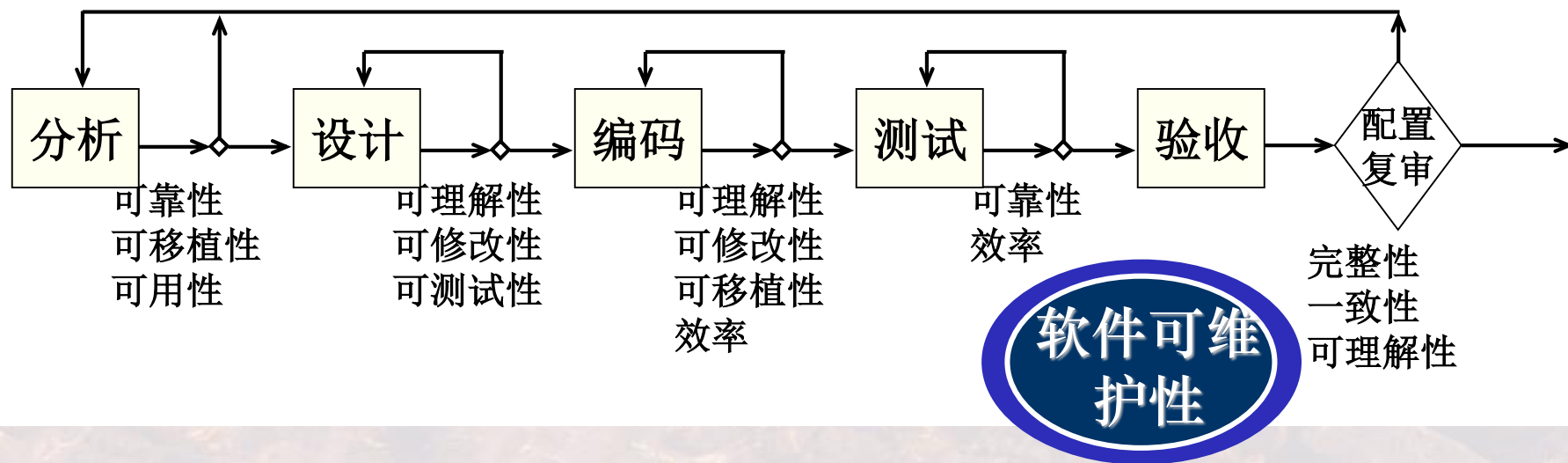
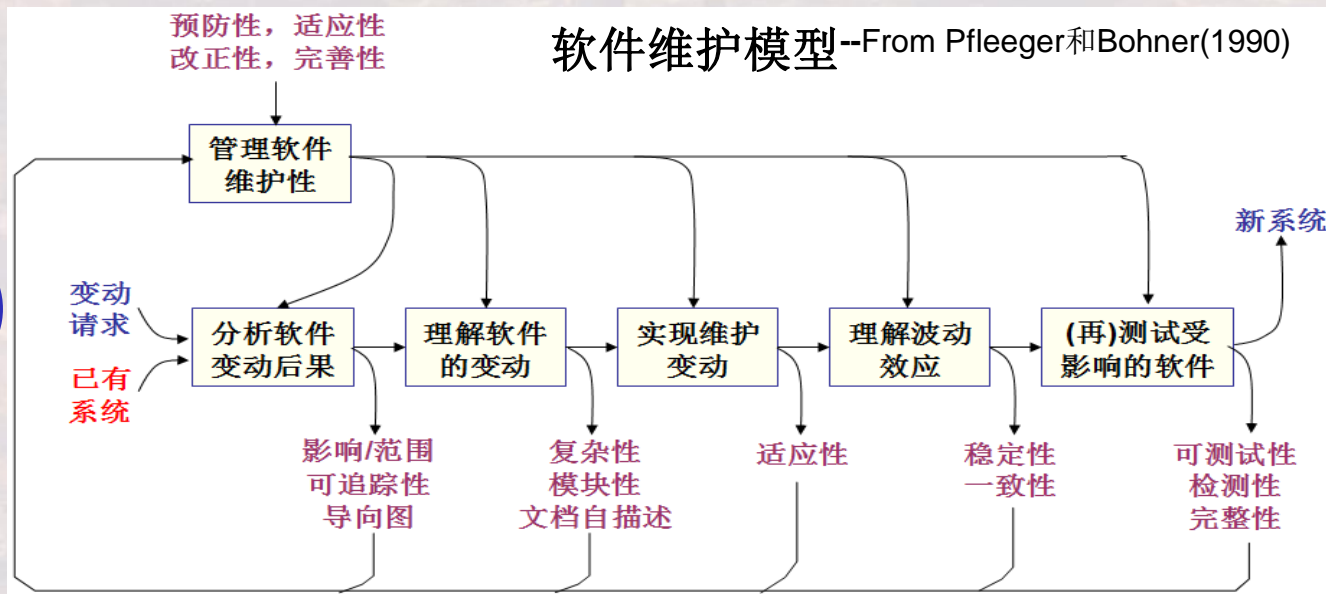


# 软件维护与软件演化概念

## (5)软件维护过程是怎样的？



## (6)由“软件维护”到“软件可维护性”？



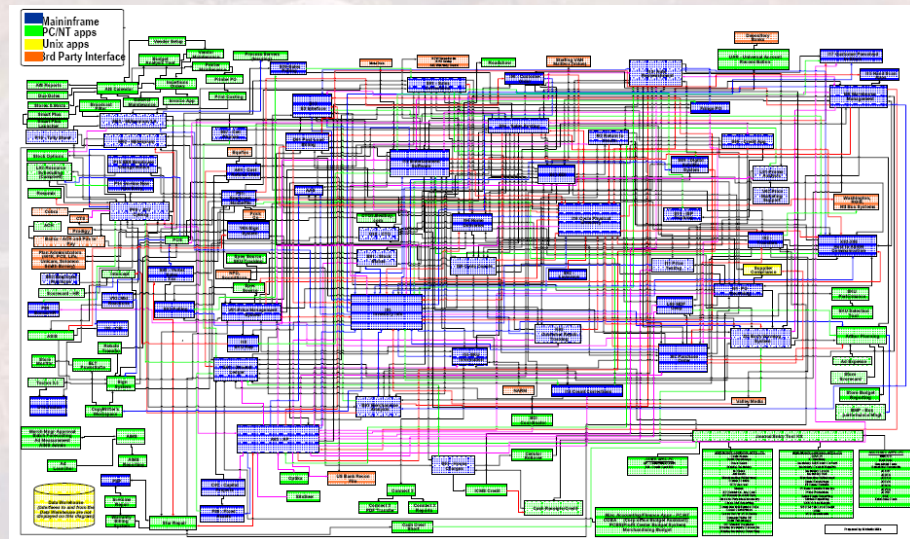
## (7)由“软件维护”到“软件再工程”？

### 遗留系统(Legacy System)

- “已经运行了很长时间的、对用户来说很重要的、但是目前已无法完全满足要求，难以重新实现和处理的软件系统”。

#### 特点

- 遗留系统往往缺乏完整的描述，文档不完整，修改记录简略
- 业务过程和遗留系统的操作方式紧密地“交织”在一起
- 重要的业务规则隐藏在软件内部
- 经过多年维护，系统结构可能已经破坏，理解设计难度大
- 开发新软件本身存在较大的风险



一种可行的解决方案：  
**软件再工程**



## (7)由“软件维护”到“软件再工程”？

### ■ 软件再工程

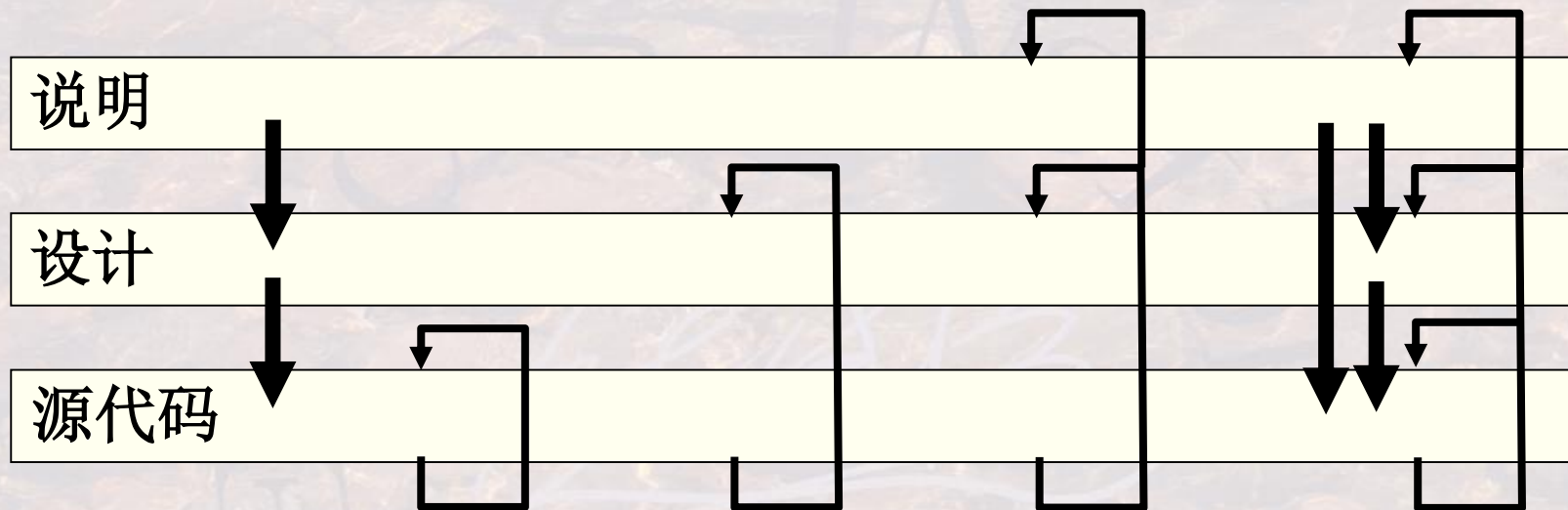
- 采用先进的软件工程方法对整个软件或软件中的一部分重新进行设计、编写和测试，以提高软件的可维护性和可靠性，保证系统的正常运行。
- “把今天的软件工程方法学用于昨天的系统以满足明天的需要”

### ■ 目的：

- 努力使系统更加易于维护，系统需要被再构造和再文档化

### ■ 优势：

- **减少风险**：重新开发一个在用的系统具有很高的风险，可能会有开发问题、人员问题和规格说明问题
- **降低成本**：再工程的成本比重新开发软件的成本要小得多



### 正向工程

- 正向按处理过程推进

### 结构重组

- 从代码开始
- 内部表示
- 迭代简化结构，消除死代码

### 文档重构

- 从代码开始
- 对结构、复杂性、数据等进行静态分析，得出信息
- 不基于软件方法

### 逆向工程

- 从代码开始
- 基于已经广泛接受的软件方法产生设计和说明
- 管理表示

### 软件再工程

- 从代码开始
- 对代码进行逆向工程
- 正向工程：完成并修改表示，重新产生代码

软件再工程的分类(Bohner1990)

# 软件配置技术



### 软件开发的复杂

产品不可见--**抽象**

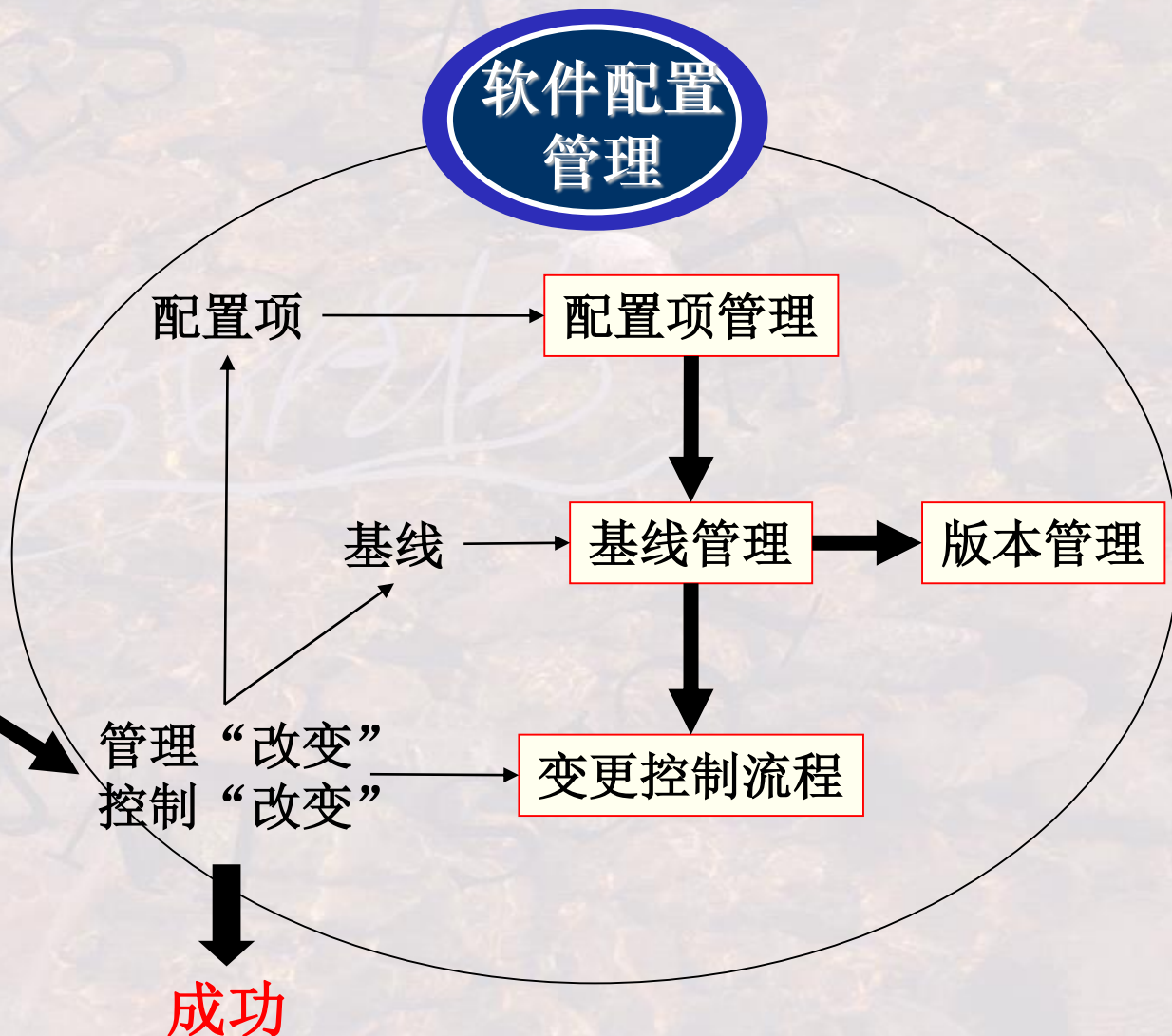
产品类别/开发工具--**多样**

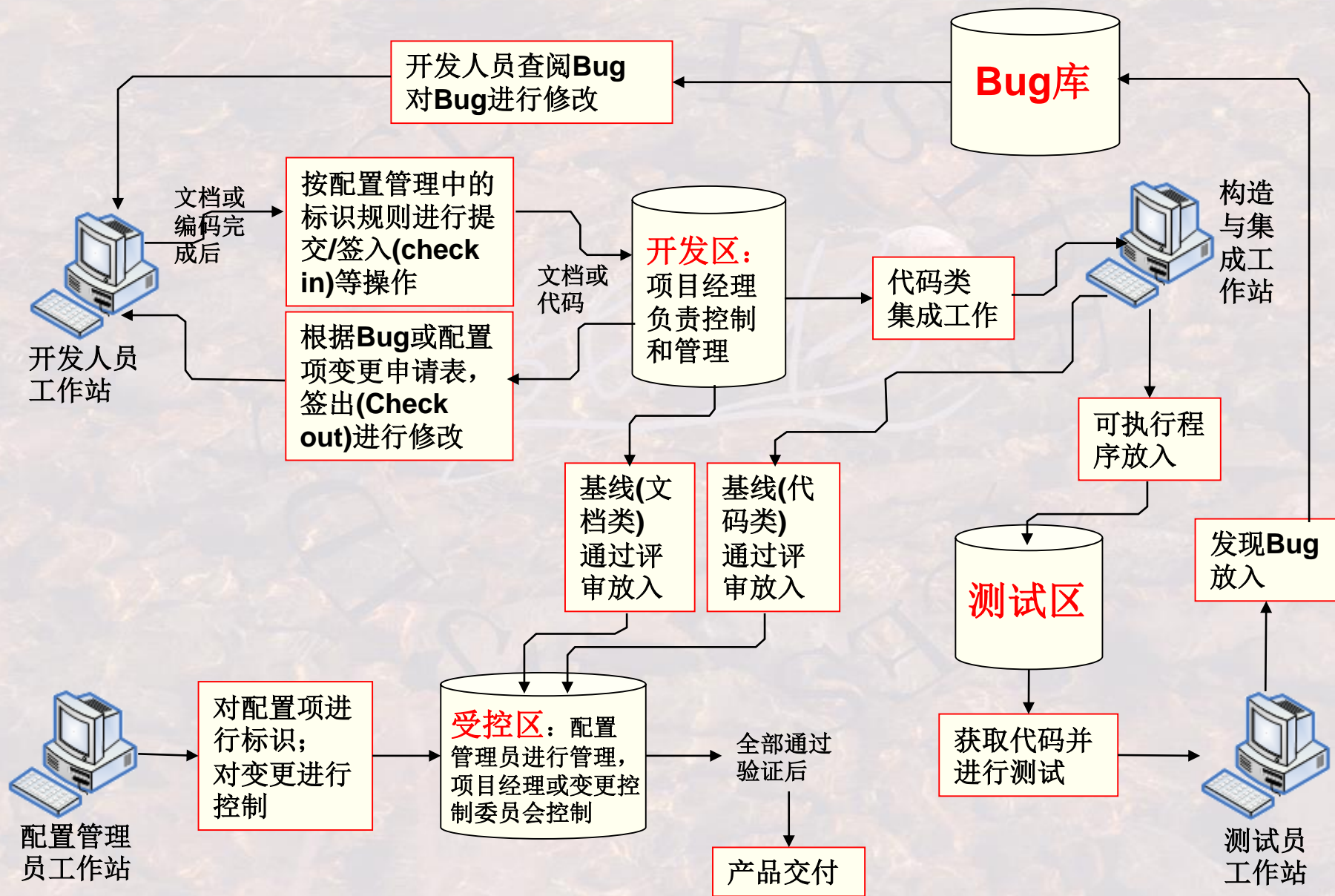
团队开发/异地开发--**协作**

不断地“改变”



失败





### 软件配置

■ 软件配置是由在软件工程过程中产生的**所有信息项**构成的，它可以看作该软件的具体形态（软件配置项）在某一时刻的瞬间影像

### 软件配置管理的目标

- 标识变更
- 控制变更
- 确保变更的正确实现
- 向开发组织内各角色**报告变更**
- 总结：**当变更发生时，能够提高适应变更的容易程度，并且能够减少所花费的工作量。**

### 软件配置的基本元素

- 配置项
- 基线
- 配置管理数据库



### 配置项

■配置项是软件全生命周期内受管理和控制的基本单位，大到整个系统，小到某个硬件设备或软件模块。

#### 主要包括：

■代码类配置项，比如：源代码、可执行代码以及相关的数据文件。

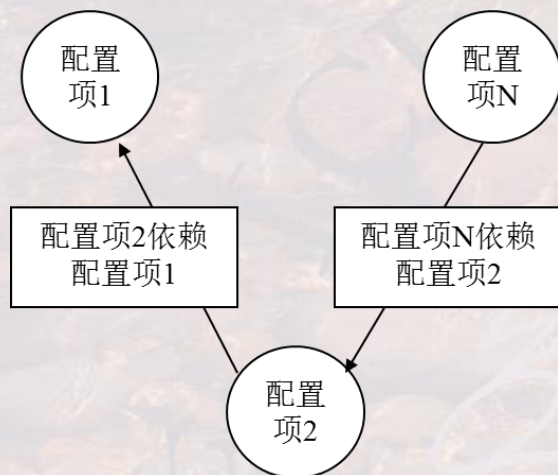
■开发环境、测试环境和运行环境--作为单独的配置项。

■文档类配置项，比如：需求类文档、计划类文档、设计类文档、测试用例/方案文档、测试报告、用户手册等。

### 配置项识别的参考标准：

- ✓可能被两个或两个以上组使用的工作产品；
- ✓无论是因为错误还是因为需求变更而导致变更的工作产品；
- ✓工作产品相关依赖，其中一个变更会导致另外一个变更；
- ✓对项目非常重要的工作产品（环境类文档应当属于这一类）。

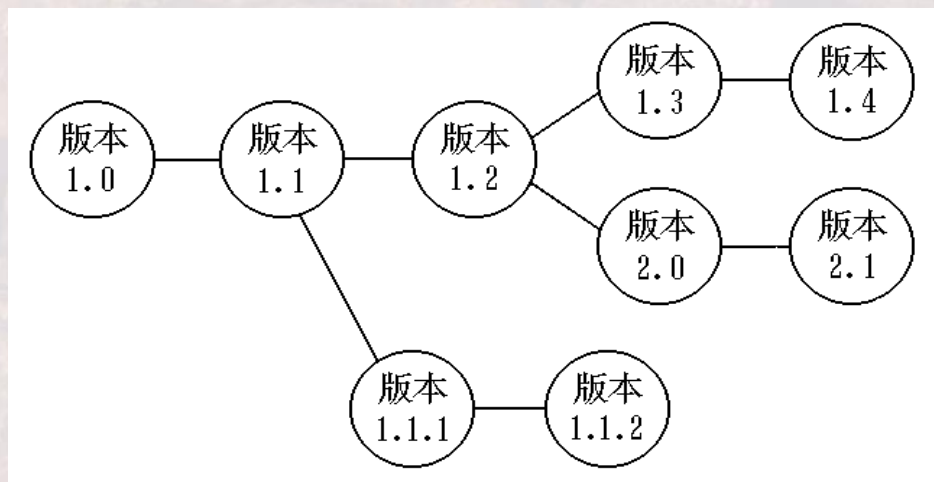
### 配置项间的依赖关系管理



依赖关系	配置项1	配置项2	...	配置项N
配置项1		X		
配置项2				X
...				
配置项N				

### 配置项的版本演化管理

- 在配置项成为基线以前可能要做多次变更，在成为基线之后也可能需要频繁的变更。对于每一配置项都要维护版本演变信息，以记录对象的变更历史。

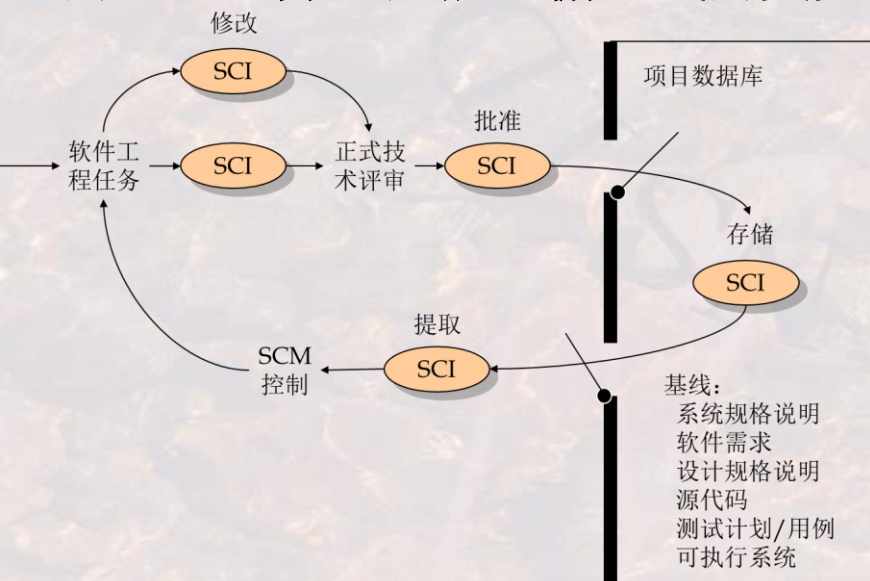


### 基线

■基线，由一个或若干个通过(正式)评审并得到确认的配置项组成，是项目进入下一个生命周期阶段的出发点(或基准)。

■基线，是软件文档或源码(或其它产出物)的一个稳定版本，它是进一步开发的基础，只有经过授权后才能变更。建立一个初始基线后，以后每次对其进行的变更都将记录为一个差值，直到建成下一个基线。

■“已经通过正式复审和批准的某规约或产品，它因此可以作为进一步开发的基础，并且只能遵循正式的变更控制过程得到改变” —from IEEE





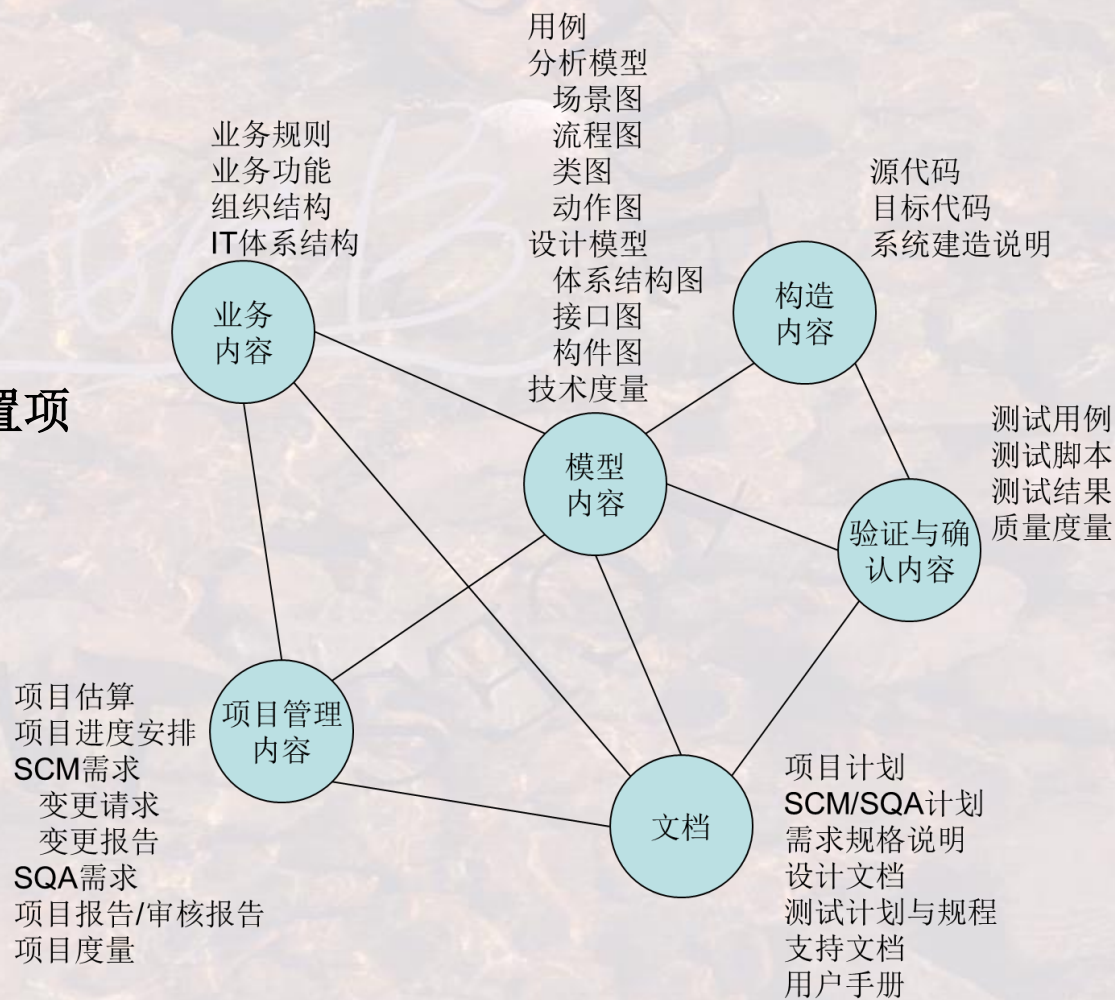
# 软件配置管理过程与技术

## (6)什么是配置管理数据库？

### 配置管理数据库

■ 配置管理数据库用于保存于软件相关的所有配置项的信息以及配置项之间关系的数据库。

- 每个配置项及其版本号
- 变更可能会影响到的配置项
- 配置项的变更路线及轨迹
- 与配置项有关的变更内容
- 计划升级、替换或弃用的配置项
- 不同配置项之间的关系



### 软件配置管理(Software Configuration Management, SCM)

“协调软件开发使得混乱减到最小的技术叫做软件配置管理，它是一种标识、组织和控制修改的技术，目的是使错误达到最小并最有效地提高生产效率。” --- Wayne Babich 《SCM Coordination for Team Productivity》

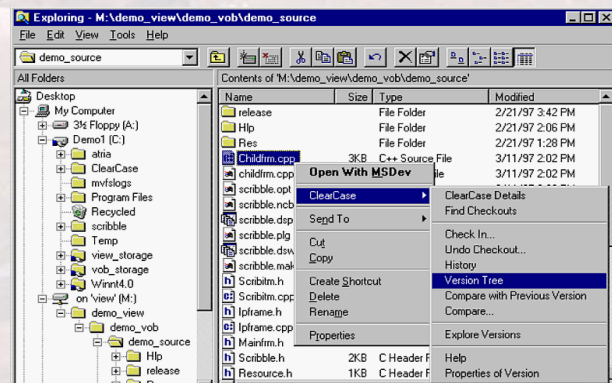
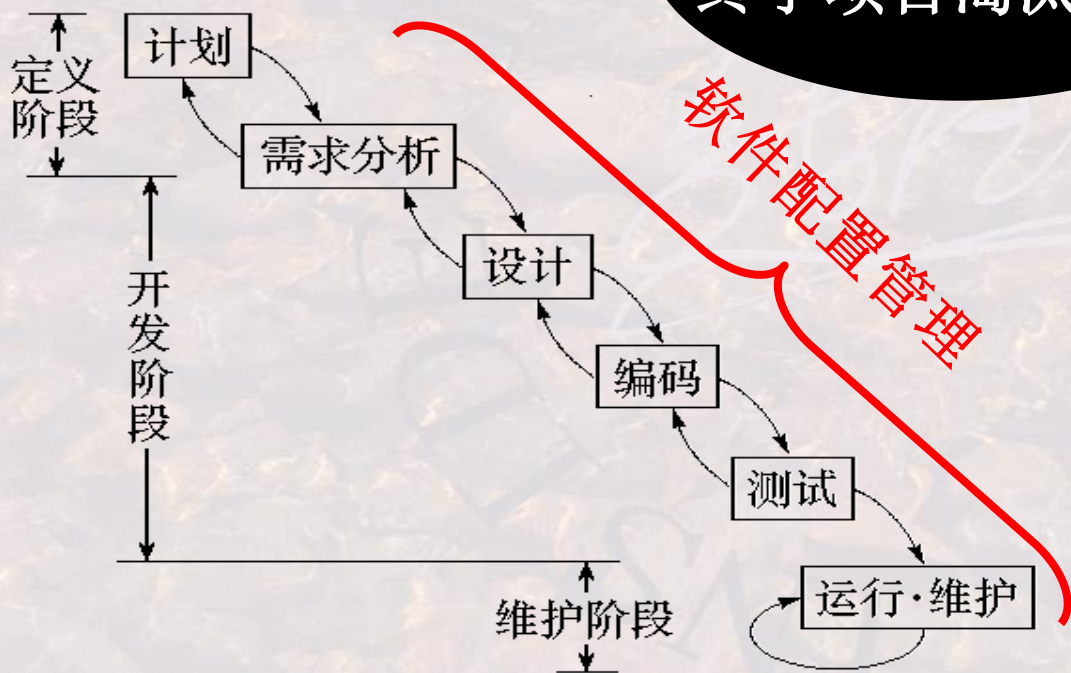
**软件配置管理：**是在贯穿整个软件生命周期中建立和维护项目产品的完整性, 包含版本控制、工作空间管理、并行开发控制、过程管理、权限管理、变更管理、配置审计等内容。借助于这些工作，使基线变更和工作产品发布得到监督和控制。

# 软件配置管理过程与技术

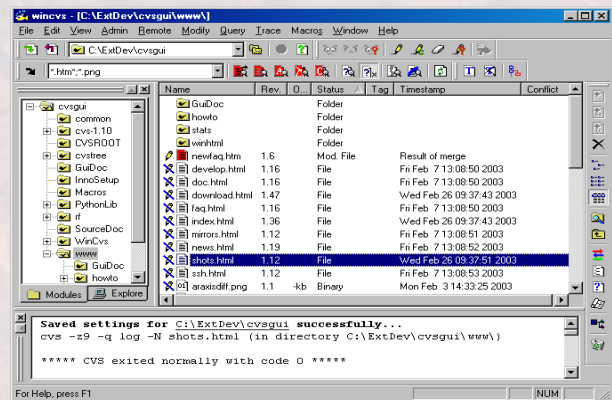
## (7) 深入理解软件配置管理?

软件配置管理贯穿整个软件生命周期与软件工程过程

始于软件项目之初  
终于项目淘汰之时



常用的软件配置管理工具：  
CVS、SVN、Git等





# 软件过程

## (1)为什么要研究软件过程？

- 软件过程是近十年来人们关注的焦点。
- 软件过程是为开发高质量软件所需要完成的任务的框架。
- 软件工程是有创造力、有知识的人在定义好的、成熟的软件过程框架中进行的创造性工作。



### 为什么要有过程？

- ✓过程能保证各活动之间是有组织的和一致的，从而，不同的人使用同一过程能获得在某一层次上一致的产品；
- ✓一致性并不排斥灵活性，一致性是在某一层次上实现的；
- ✓过程可被检查、理解、控制和改进；
- ✓过程也是传授经验的一种途径。

## (2)什么是过程?什么是软件过程?

**过程**：产生某种预定输出的一系列可预测的步骤，包含一组活动(**activities**)、约束(**constraints**)、资源要素(**resources**)。

**软件过程**：开发和维护软件及其相关产品所涉及的一系列活动。**过程**是活动的集合；**活动**是任务的集合；**任务**是把**输入****转换**为**输出**的操作。是一个为建造高质量软件所需完成的任务的框架，它规定了完成各项任务的工作步骤。

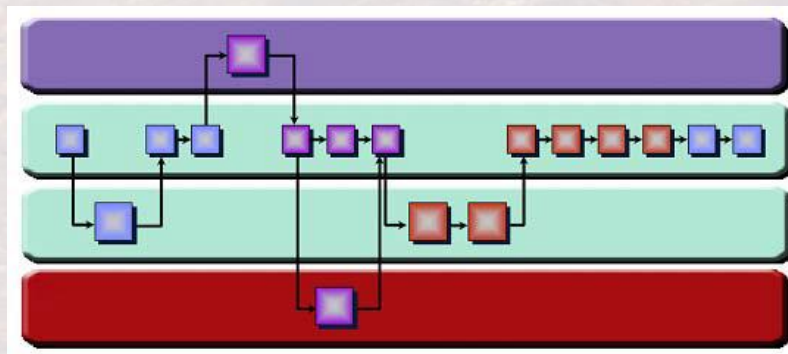
**过程的性质**：时间性、并发性、嵌套性和度量性等。



## (2)什么是过程?什么是软件过程?

### ■ 软件过程定义以下内容

- 所执行的活动
- 活动的细节和步骤
- 人员与分工



### ■ 软件过程通过以下方式组织和管理软件生命周期

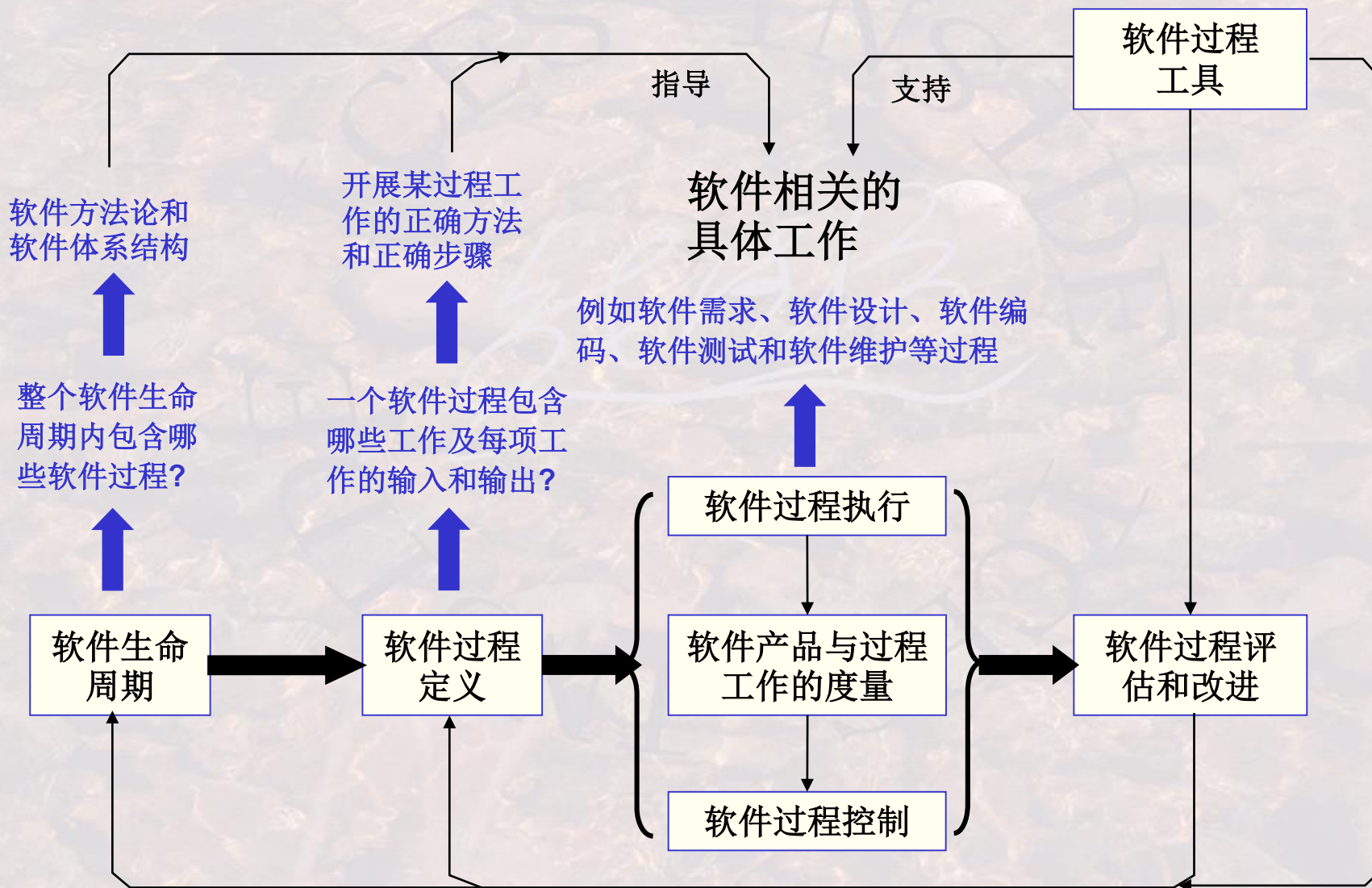
- 定义软件生产过程中的活动
- 定义这些活动的顺序及其关系

### ■ 软件过程的目的

- 标准化(可模仿)、可预见性(降低风险)、提高开发效率、得到高质量产品
- 提升制定时间和预算计划的能力

软件工程具有“产品(质量)与过程二相性”的特点，必须把二者结合起来去考虑，而不能忽略其中任何一方。

### 软件过程管理的一种模型



## (4)什么是软件能力成熟度？

软件过程成熟度，是指一个特定的软件过程被显式定义、管理、度量、控制和能行的程度，它可用于指示企业加强其软件过程能力的潜力。

当一个企业达到了一定的软件过程成熟级别后，它将通过制定策略、建立标准和确立机构结构使它的软件过程制度化。而制度化又促使企业通过建立基础设施和公司文化来支持相关的方法、实践和过程。从而使之可以持续并维持一个良性循环。

### 不成熟企业的标志

- ✓缺乏确定的软件过程和相应的管理和控制；
- ✓即使给出了软件过程，也不严格的遵循和强制执行；
- ✓管理是完全被动的，管理者采用的策略是救火式的，即出了事才去解决，解决的时候也难以纵观全局，往往只顾眼前；



## (5)什么是软件能力成熟度?

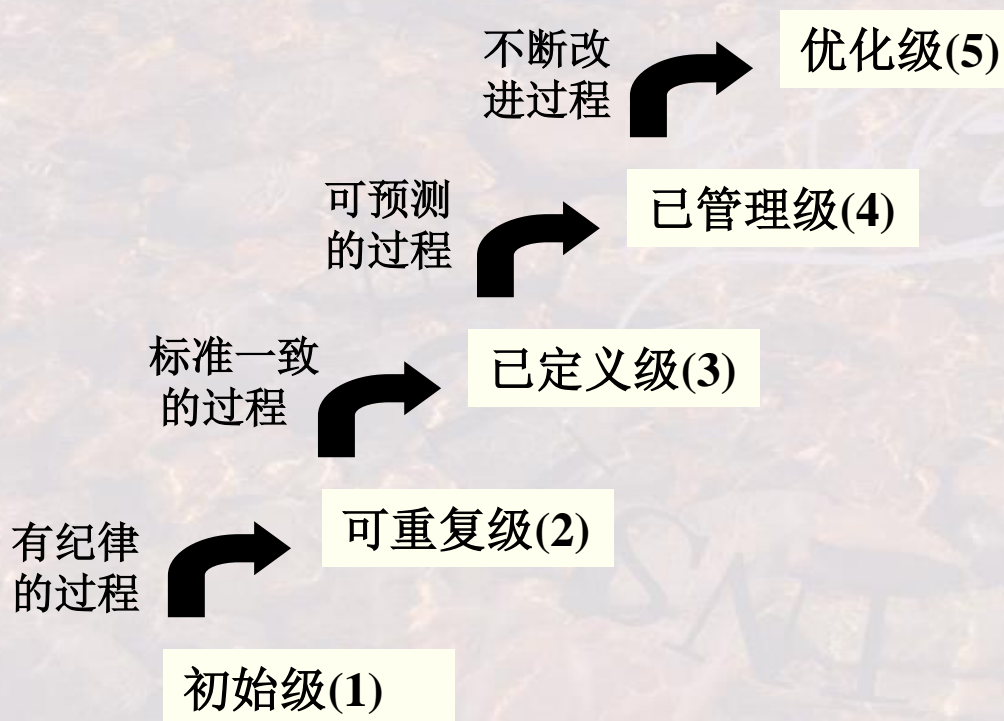
### 软件能力成熟度模型(CMM, Capability Maturity Model)

●**CMM**: 是软件行业标准模型, 用来定义和评价软件企业开发过程的成熟度, 提供如何做才能够提高软件质量的指导。

●**CMM的基本原理**: **CMM**将软件组织的过程能力成熟度分为5个级别, 每一个级别定义一组过程能力目标, 并描述要达到这些目标应该采取的各种实践活动。

●**CMM的主要作用**: 提供了一个软件过程改进的框架。根据**CMM**模型, 软件开发者(机构或组织)可以去定义、实施、度量、控制和改进自己的软件过程, 能够大幅度的提高按计划、高效率、低成本的提交有质量保证的软件产品的能力。

### CMM的5个分级标准



### CMM的分级结构和特征

**初始级：**软件过程无秩序，有时甚至是混乱的。

**可重复级：**已建立了基本的项目管理过程，可用于对成本、进度和功能特性进行跟踪。

**已定义级：**用于管理的、工程的软件过程均已实现文档化、标准化，并形成了整个软件组织的标准软件过程。

**已管理级：**软件过程和产品质量有详细的度量标准，软件过程和产品质量得到了定量的认证和控制。

**优化级：**通过对来自过程、新概念和新技术等方面各种有用信息的定量分析，能够不断地、持续性地对过程进行改进。

## 第9讲 软件工程技术-软件测试与维护

