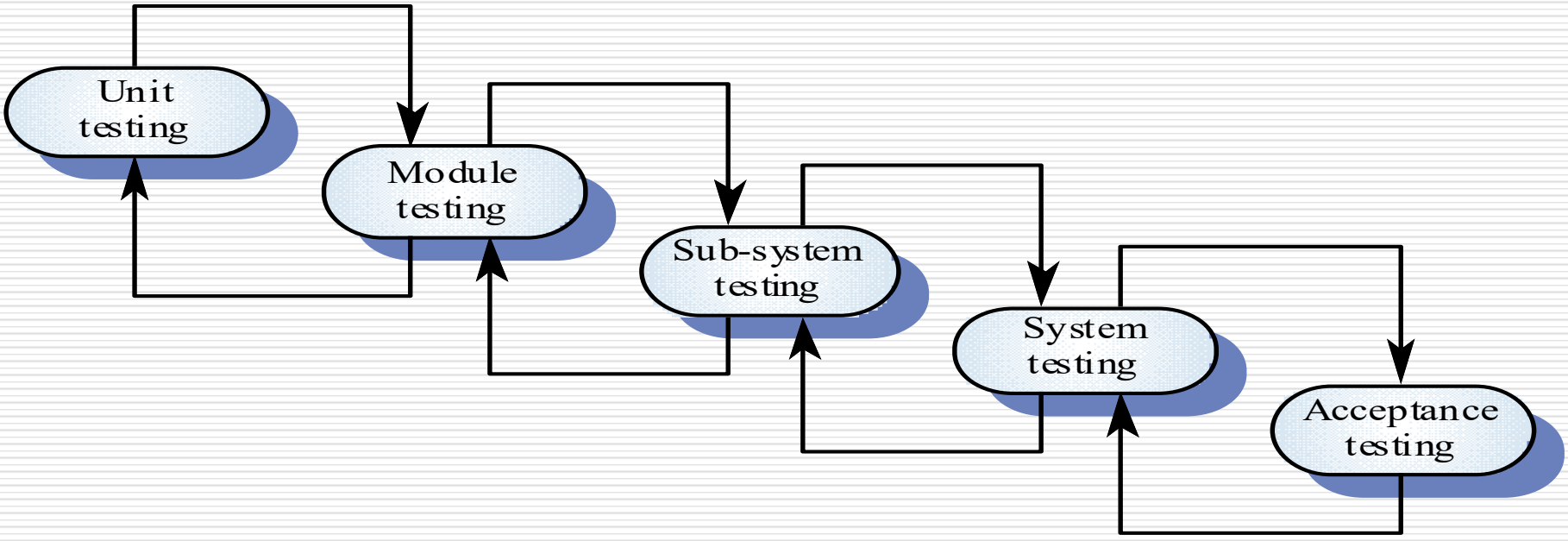
CHAPTER 7

Software Testing

Outline

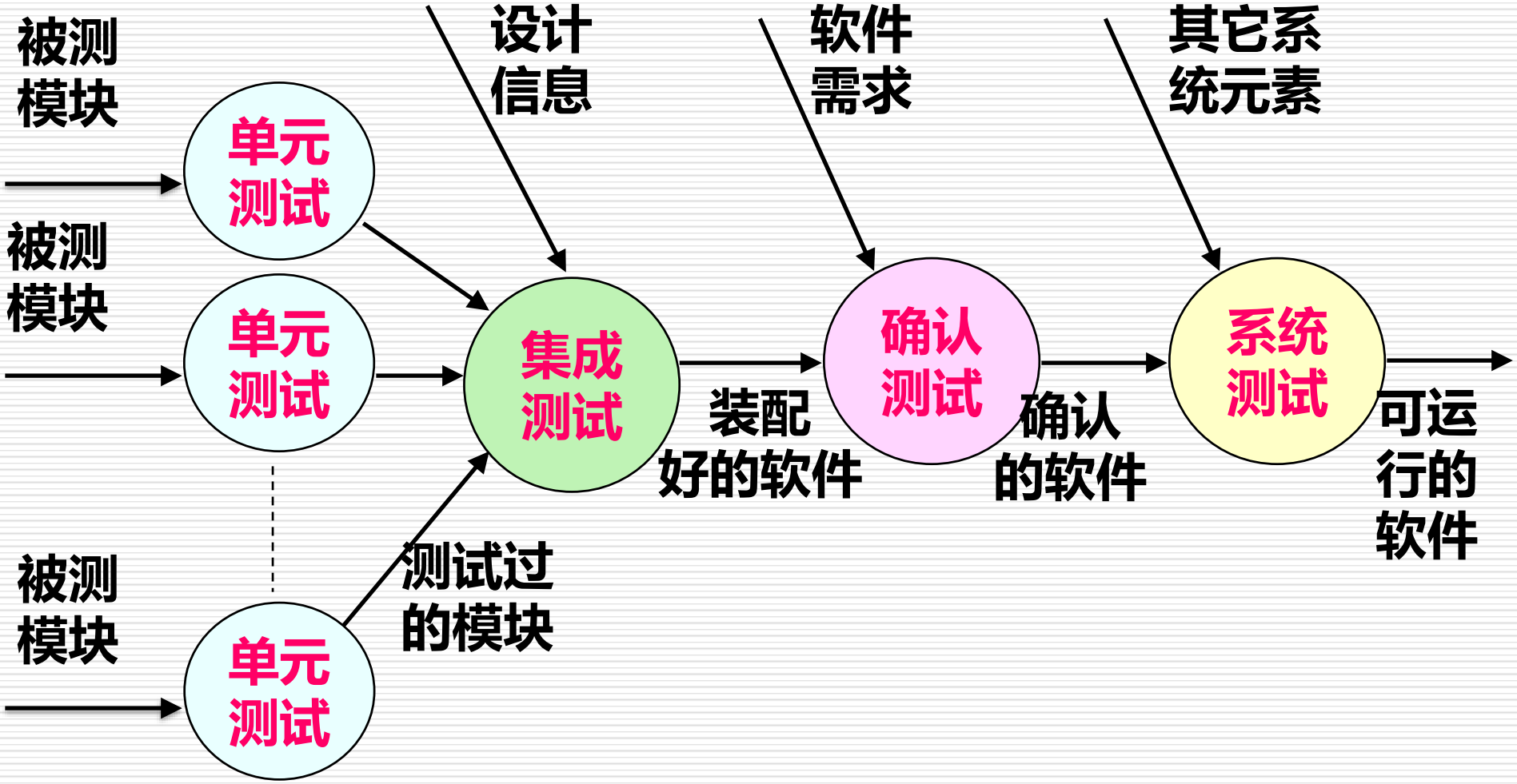
- 软件测试的过程，即软件集成、形成过程
  - ✓ 单元测试
  - ✓ 集成测试
  - ✓ 确认测试
  - ✓ 系统测试
- 软件可靠性和测试终止的条件
- 调试

Testing process



单元测试　　集成测试　　用户测试

软件测试的过程


被测
模块
被测
模块
被测
模块
单元
测试
单元
测试
单元
测试
设计
信息
软件
需求
其它系
统元素
集成
测试
确认
测试
系统
测试
装配
好的软件
确认
的软件
可运
行的
软件
测试过
的模块

4个步骤简介

- 单元测试：集中对用源代码实现的，每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。
- 集成测试：把已测试过的模块组装起来，主要对与设计相关的软件体系结构的构造进行测试。
- 确认测试：检查已实现的软件是否满足了需求规格说明中确定了的各种需求，以及软件配置是否完全、正确。
- 系统测试：把经过确认的软件纳入实际运行环境中，与其它系统成份组合在一起进行测试。

Unit Testing

单元测试又称模块测试，是软件测试的最小单位。

目的:是发现各模块内部可能存在的各种差错。

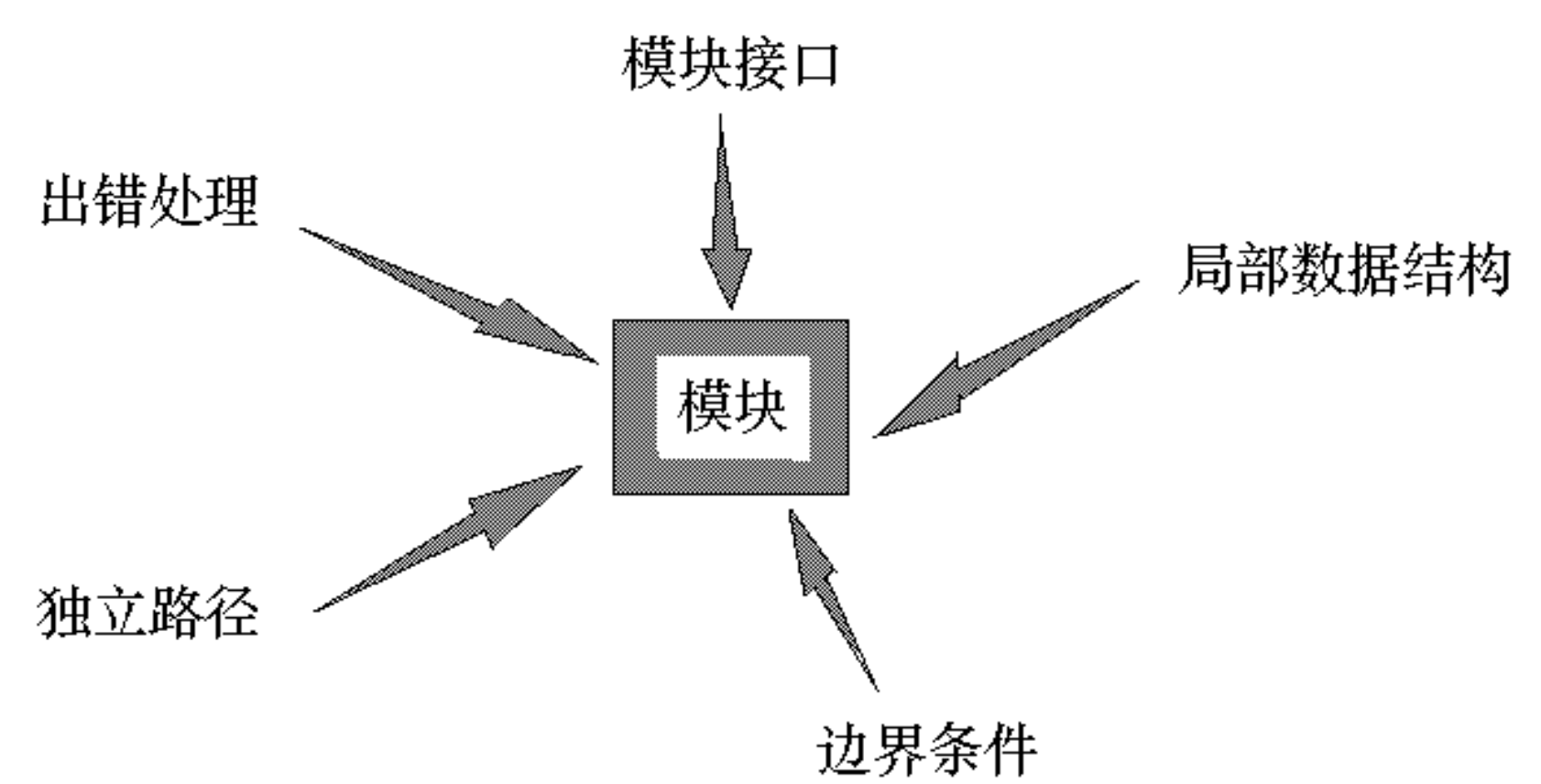
特点：

- **Static test:**
  - ✓ **Hand execution: Reading the source code**
  - ✓ **Walk-Through (informal presentation to others)**
  - ✓ **Code Inspection (formal presentation to others)**
  - ✓ **Automated Tools checking for**
    - **syntactic and semantic errors**
    - **departure from coding standards**

Unit Testing

- Dynamic test：主要采用白盒测试的测试用例，辅之以黑盒测试
- 由编码程序员进行的测试
- 多个模块可以并行地独立进行单元测试

单元测试的内容



模块的I/O条件和模块的逻辑结构

单元测试：局部数据结构

- ✓ 不正确或不一致的数据类型说明
- ✓ 使用尚未赋值或尚未初始化的变量
- ✓ 错误的初始值或错误的缺省值
- ✓ 变量名拼写错或书写错
- ✓ 不一致的数据类型
- ✓ 全局数据对模块的影响

单元测试：路径测试

- ✓ 选择适当的测试用例，对模块中重要的执行路径进行测试。
- ✓ 应当设计测试用例，查找由于错误的计算、不正确的比较，或不正常的控制流而导致的错误。
- ✓ 对基本执行路径和循环进行测试可以发现大量的路径错误。

单元测试：模块接口

在单元测试的开始，应对通过被测模块的数据流进行测试。测试项目包括：

- ✓ 调用本模块的输入参数是否正确；
- ✓ 本模块调用子模块时传给子模块的参数是否正确；
- ✓ 全局量的定义在各模块中是否一致；
- ✓ 单元内外交流数据时，测试要更细致；

单元测试：模块接口

- **单元内外交互时要考虑：**
  - ✓ 文件属性是否正确；
  - ✓ OPEN与CLOSE语句是否正确；
  - ✓ 缓冲区容量与记录长度是否匹配；
  - ✓ 在进行读写操作之前是否打开了文件；
  - ✓ 在结束文件处理时是否关闭了文件；
  - ✓ 正文书写／输入错误，
  - ✓ I／O错误是否检查并做了处理。

单元测试：边界条件

- 注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。
- 如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。

单元测试：错误处理

- ✓ 出错的描述是否难以理解
- ✓ 出错的描述是否能够对错误定位
- ✓ 显示的错误与实际的错误是否相符
- ✓ 对错误条件的处理正确与否
- ✓ 在对错误进行处理之前，错误条件是否已经引起系统的干预等

单元测试环境

C语言程序中的一个函数如何测试？ Java中？

```
main( )
{
    ……
    root(a, b, c, x1, x2);
    ……
}
```

```
main( )
{
    /*
    ……
    */
    a=1; b=2; c=1;
    root(a, b, c, x1, x2);
    printf(x1, x2)
    /*
    ……
    */
}
```

单元测试环境

模块并不一定是一个独立的程序。

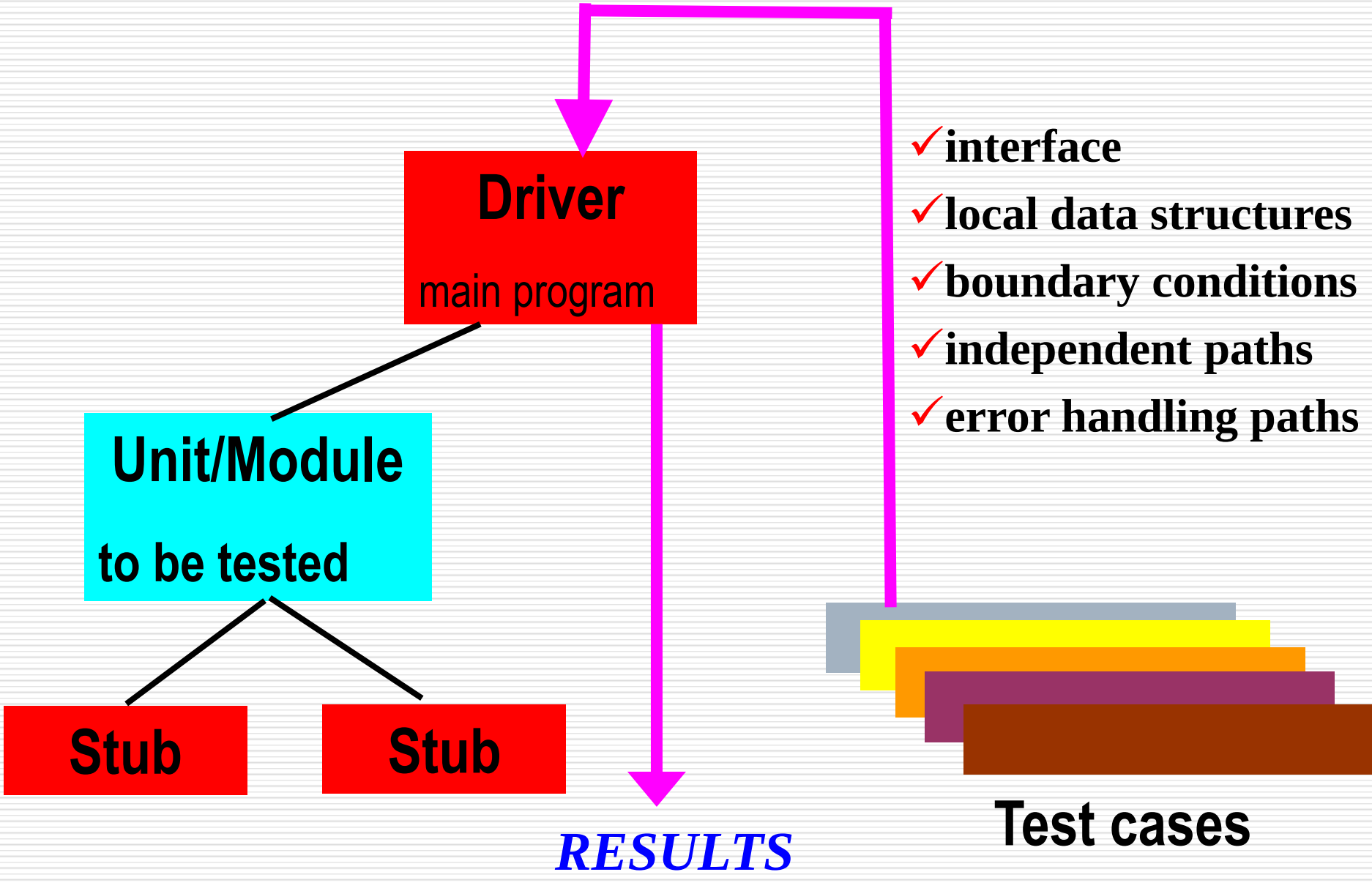
辅助模块，模拟与被测模块相联系的其它模块。

驱动模块 (driver)， 桩模块 (stub) —— 存根模块

驱动模块 —— 相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后再输出实测结果。

桩模块 (stub) —— 存根模块。用以代替被测模块调用的子模块。

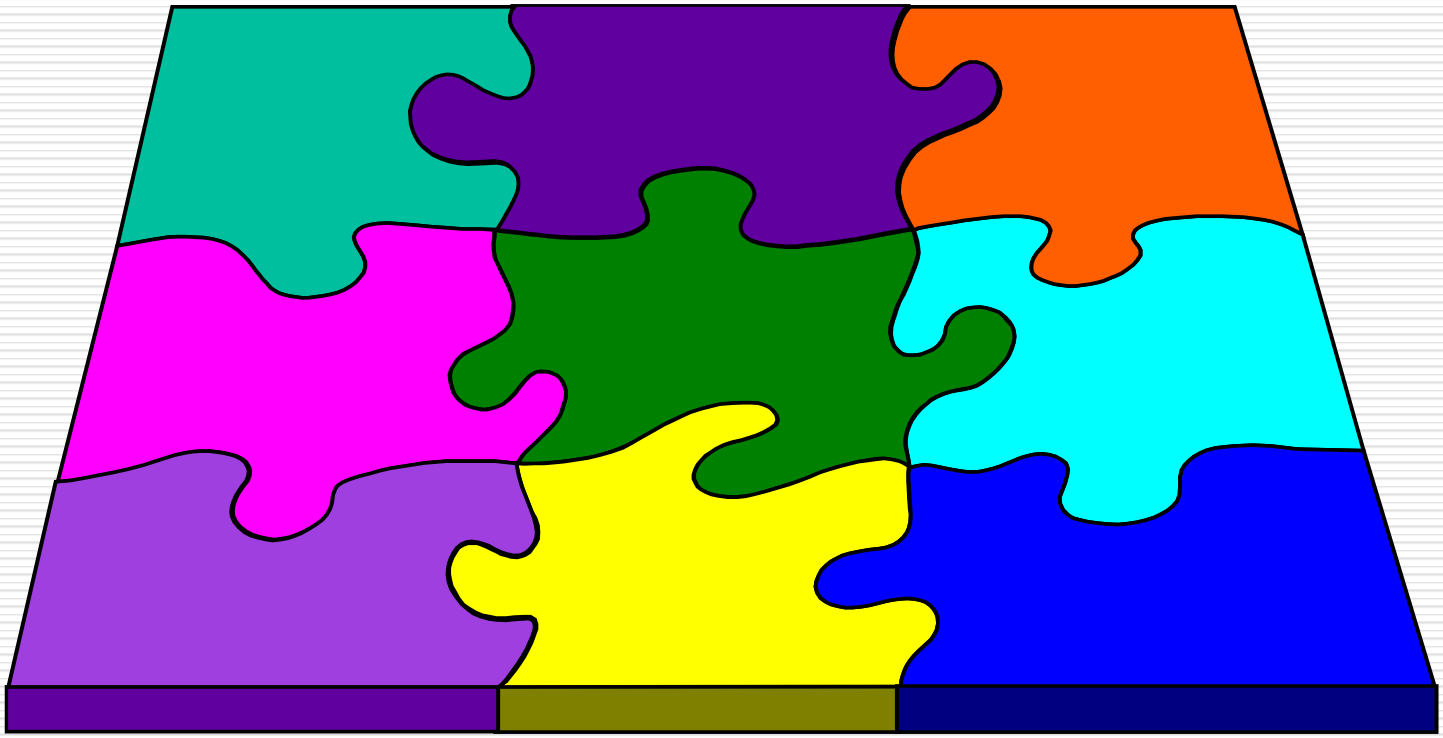
(Unit) Testing environment



集成/组装测试

Why integration testing?

If all units work individually, why doubt that they work together?



集成测试考虑的问题

- ✓ 在把各个模块连接起来的时侯，穿越模块接口的数据是否会丢失
- ✓ 一个模块的功能是否会对另一个模块的功能产生不利的影响
- ✓ 各个子功能组合起来，能否达到预期要求的父功能
- ✓ 全局数据结构是否有问题
- ✓ 单个模块的误差累积起来，是否会放大，从而达到不能接受的程度
- ✓ 在单元测试的同时可进行组装测试，发现并排除在模块连接中可能出现的问题，最终构成要求的软件系统

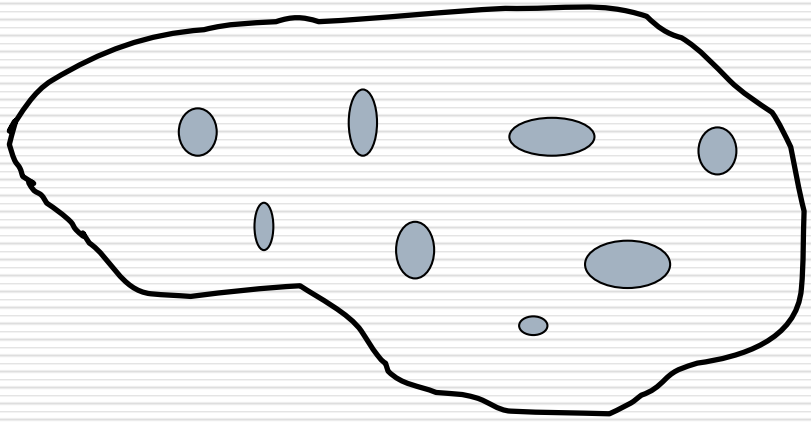
集成组装的方式

- 模块组装集成有两种方式：
  - ✓ 一次性组装方式
  - ✓ 增殖式组装方式

- 增殖式组装方式：
  - ✓自顶向下的增殖方式
  - ✓自底向上的增殖方式
  - ✓混合增殖式测试

一次性组装方式 (big bang)

- 它是一种非增殖式组装方式，也叫做整体拼装。
- 使用这种方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。

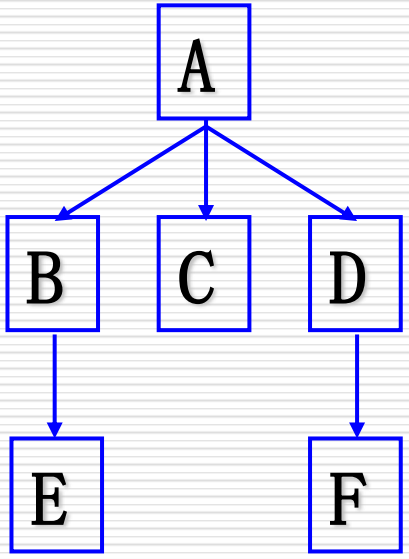


增量式组装方式

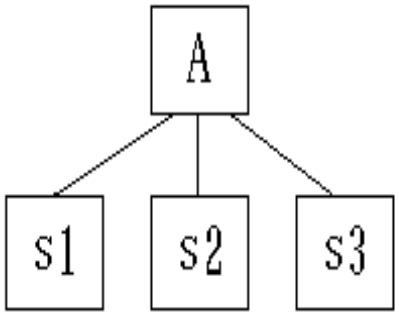
- 这种组装方式又称渐增式组装
- 首先对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统
- 在组装的过程中，一边连接一边测试，以发现连接过程中产生的问题
- 通过增殖方式逐步组装成为要求的软件系统。

自顶向下的增量方式

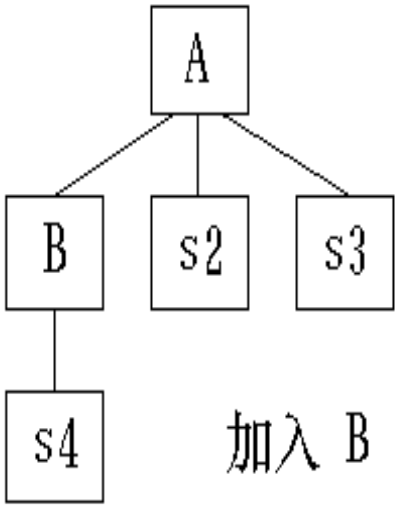
- 这种组装方式将模块按系统程序结构，沿控制层次自顶向下进行组装。
- 自顶向下的增殖方式在测试过程中较早地验证了主要的控制和判断点。
- 选用按深度方向组装的方式，可以早期实现和验证一个完整的软件功能。



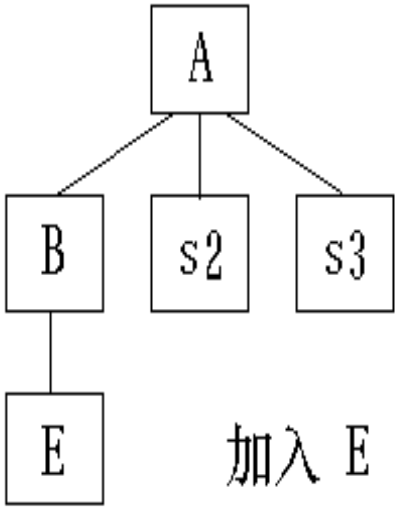
举例



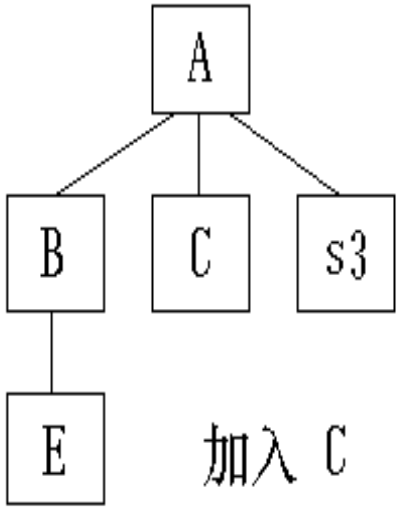
测试 A



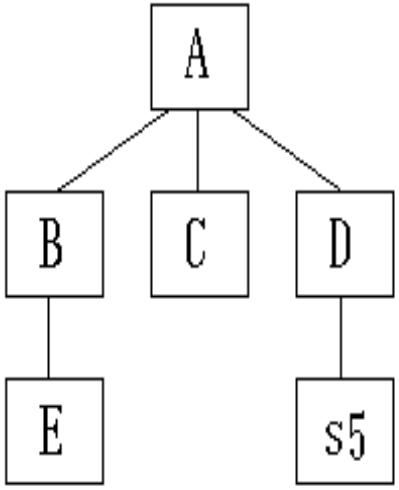
加入 B



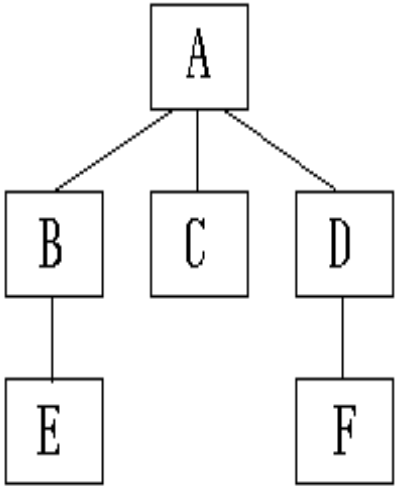
加入 E



加入 C



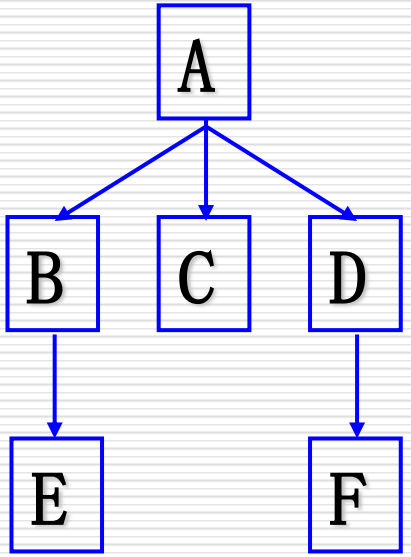
加入 D



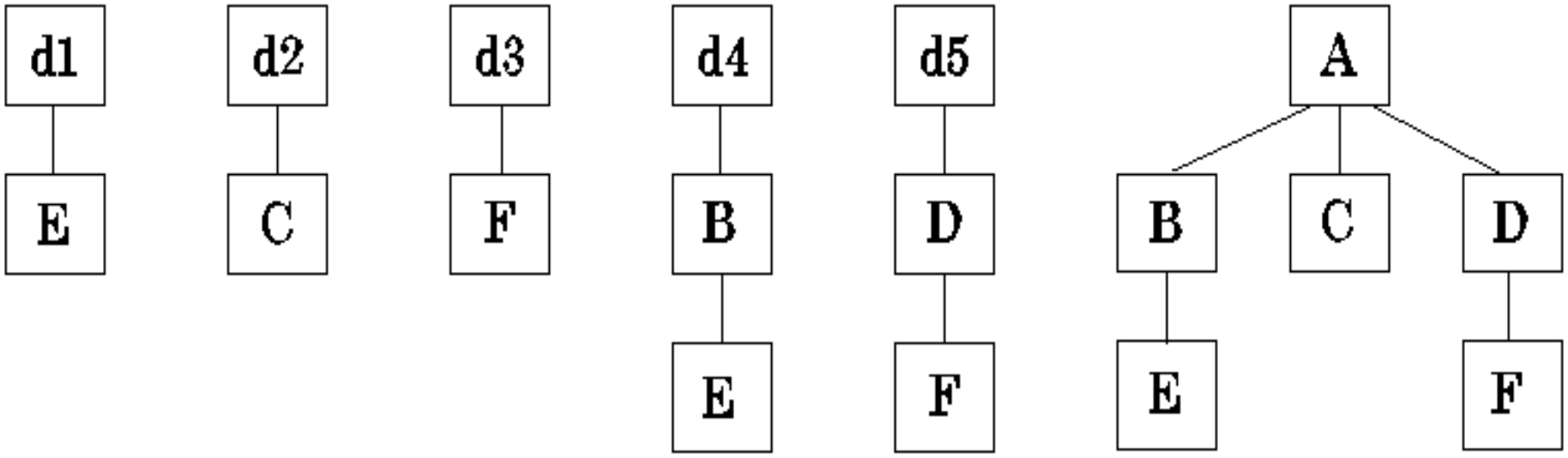
加入 F

自底向上的增量方式

- 这种组装的方式是从程序模块结构的最底层的模块开始组装和测试。
- 因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块（包括子模块的所有下属模块）已经组装并测试完成，所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以直接运行子模块得到。



举例



- 自顶向下增殖的方式和自底向上增殖的方式各有优缺点。
- 一般来讲，一种方式的优点是另一种方式的缺点。

比较两种组装方式

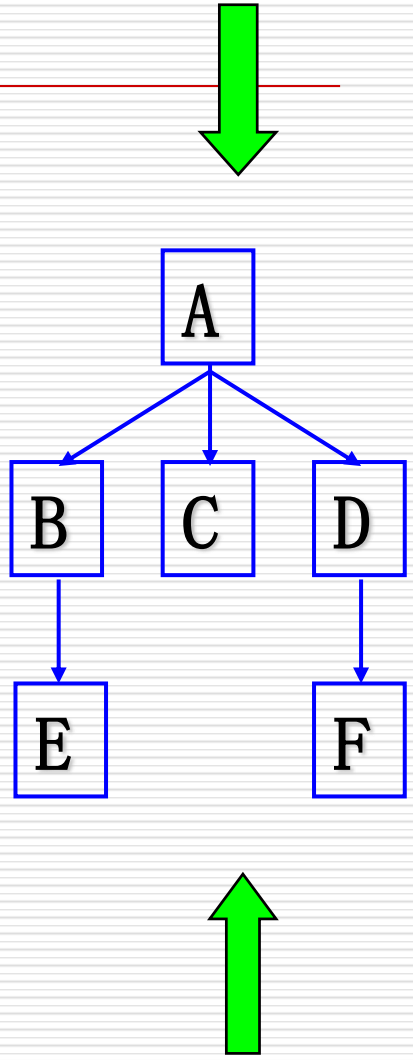
- ✓驱动程序
- ✓存根程序
- ✓系统整体功能
- ✓低层关键模块
- ✓充分展开测试人力方面
- ✓并行测试

混合增量式测试

✓首先对输入／输出模块和关键算法模块进行测试；

✓ 再自底向上组装成为功能相当完整且相对独立的子系统；

✓ 然后由主模块开始自顶向下进行增殖测试。

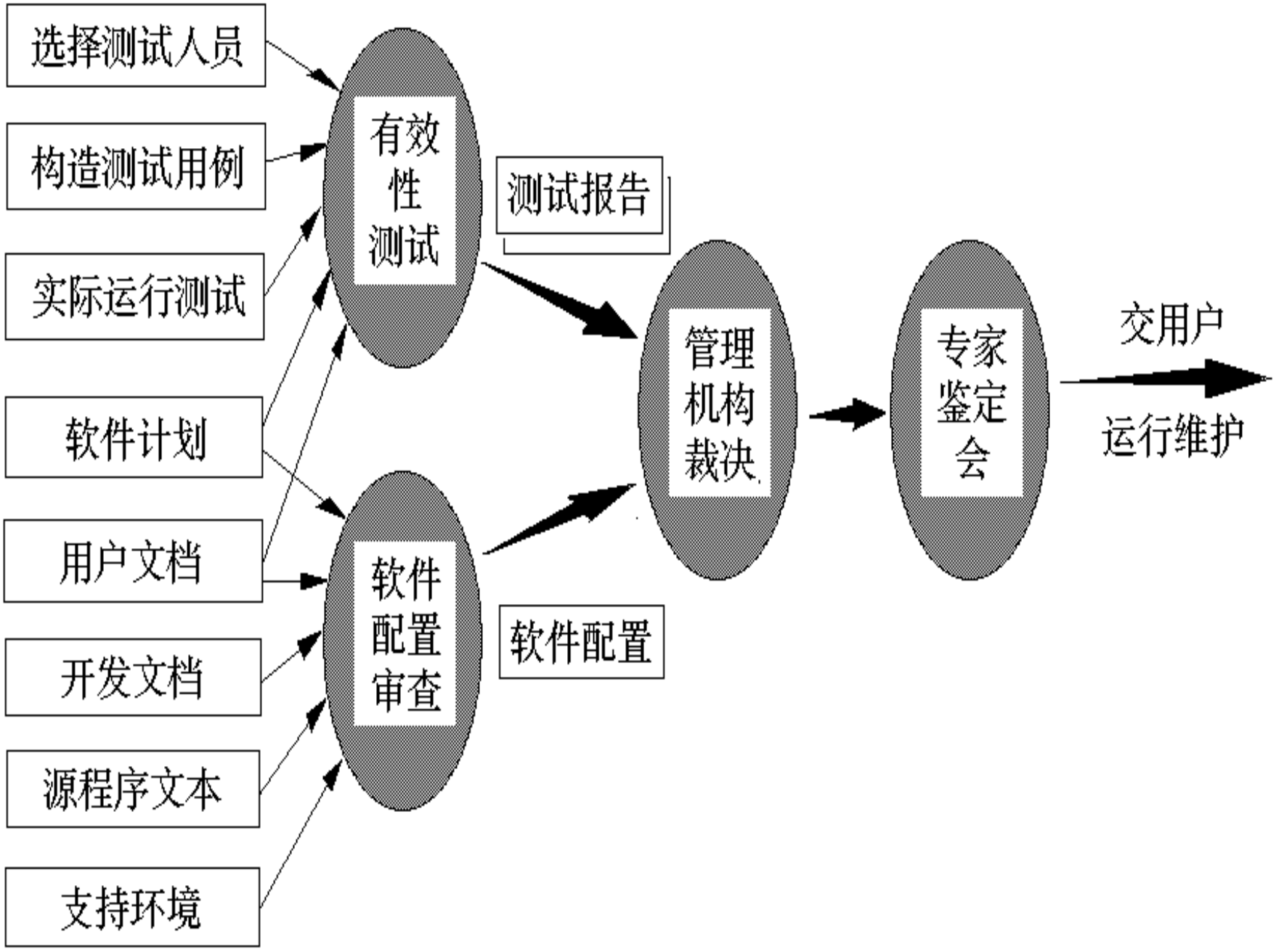


确认测试（Validation Testing）

- ✓ 确认测试又称有效性测试。任务是验证软件的功能和性能及其它特性是否与用户的要求一致。
- ✓ 对软件的功能和性能要求在软件需求规格说明书中已经明确规定。包含的信息就是软件确认测试的基础。

主要的工作：

- ✓ 有效性测试
- ✓ 软件配置复查
- ✓ α 测试和 β 测试
- ✓ 验收测试


选择测试人员
构造测试用例
实际运行测试
软件计划
用户文档
开发文档
源程序文本
支持环境
有效性测试
软件配置审查
测试报告
软件配置
管理机构裁决
专家鉴定会
交用户
运行维护

软件有效性测试（黑盒测试）

- ✓ 有效性测试是在模拟的环境（可能就是开发的环境）下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。
- ✓ 首先制定测试计划，规定要做测试的种类。还需要制定一组测试步骤，描述具体的测试用例。
- ✓ 通过实施预定的测试计划和测试步骤，确定
  - 软件的功能和性能特性是否与需求相符；
  - 所有的文档都是正确且便于使用；
  - 同时，对其它软件需求，例如可移植性、兼容性、出错自动恢复、可维护性等，也都要进行测试

软件配置复查

- 软件配置复查的目的是保证
  - ✓ 软件配置的所有成分都齐全；
  - ✓ 各方面的质量都符合要求；
  - ✓ 具有维护阶段所必需的细节；
  - ✓ 而且已经编排好分类的目录。
- 应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性。

α 测试

- α 测试是由一个开发者在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。
- α 测试的目的是评价软件产品的FLURPS（即功能、局域化、可使用性、可靠性、性能和支持）。尤其注重产品的界面和特色。
- α 测试可以从软件产品编码结束之时开始，或在模块（子系统）测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。

β 测试

- β 测试是由软件的多个用户，在实际使用环境下进行的测试。这些用户返回有关错误信息给开发者。
- 测试时，开发者通常不在测试现场。因而，β 测试是在开发者无法控制的环境下进行的软件现场应用。
- 在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告。
- β 测试主要衡量产品的FLURPS。着重于产品的支持性，包括文档、客户培训和支持产品生产能力。
- 只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。它处在整个测试的最后阶段。同时，产品的所有手册文本也应该在此阶段完全定稿。

系统测试（System Testing）

- ✓ 系统测试，是将通过确认测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其它系统元素结合在一起，在实际运行环境下，对计算机系统进行一系列的组装测试和确认测试。
- ✓ 系统测试的目的在于通过与系统的需求定义作比较， 发现软件与系统的定义不符合或与之矛盾的地方。

功能测试 vs. 非功能测试

功能测试是在规定的一段时间内运行软件系统的所有功能，以验证这个软件系统有无严重错误。

- 如果一个软件各个功能的运行结果完全符合需求，是否就没问题了？
  - ✓ **ATM**存取款机系统
  - ✓ 课程注册管理系统
  - ✓ 月球车控制软件

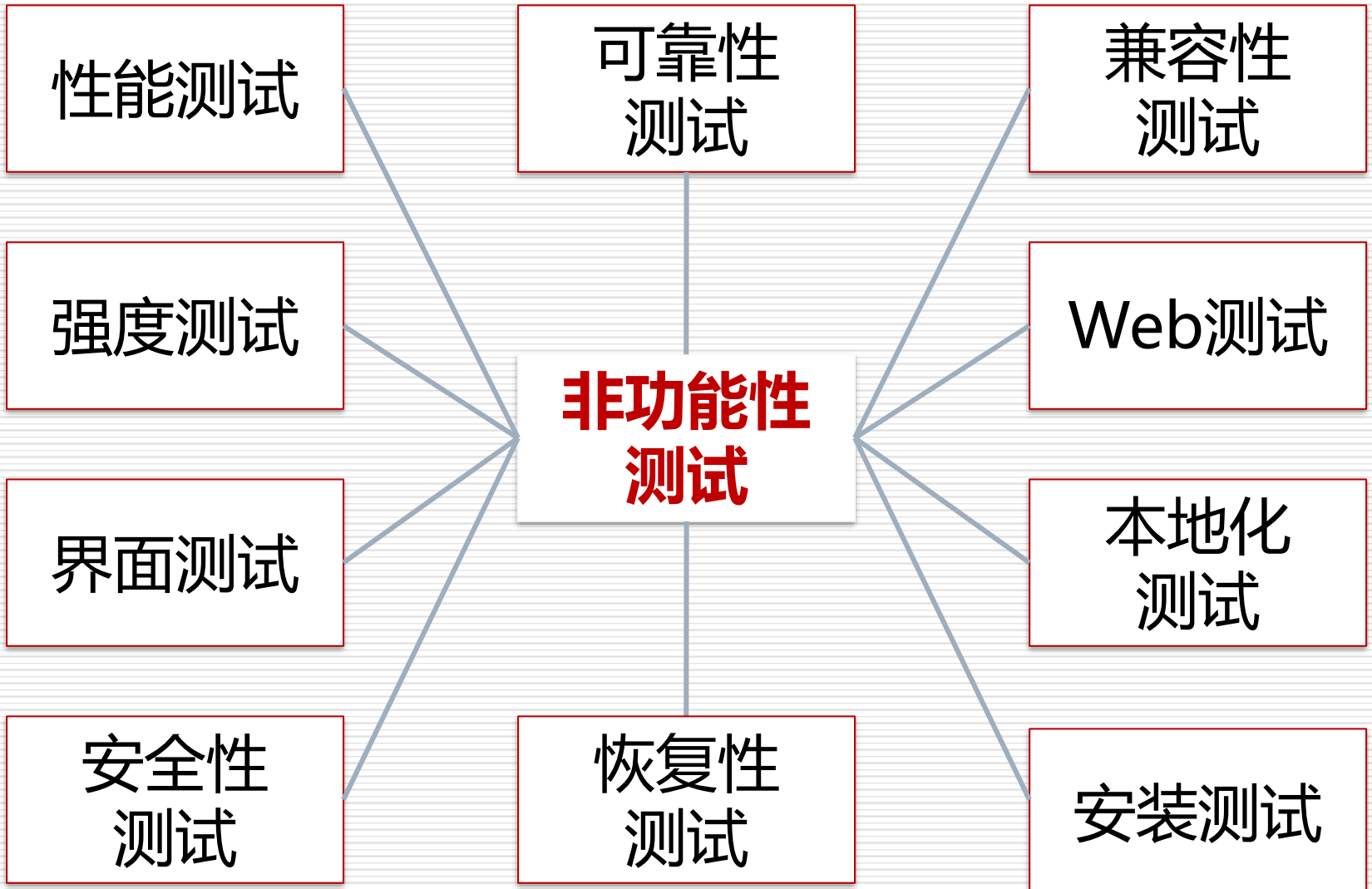
软件非功能特性能否满足要求非常重要！

需要对哪些非功能特性进行测试？

其它测试

- 关心和关注的一组功能和性能类型
  - ✓可靠性测试
  - ✓强度测试
  - ✓性能测试
  - ✓恢复测试
  - ✓配置测试
  - ✓安全性测试
  - ✓可使用性测试
  - ✓可支持性测试
  - ✓安装测试
  - ✓兼容测试
  - ✓过程测试
  - ✓容量测试
  - ✓文档测试

非功能性测试


性能测试
可靠性测试
兼容性测试
强度测试
非功能性测试
Web测试
界面测试
本地化测试
安全性测试
恢复性测试
安装测试

其它测试

- ✓性能测试
- ✓可靠性测试
- ✓强度测试
- ✓恢复测试
- ✓配置测试
- ✓安全性测试
- ✓可使用性测试
- ✓可支持性测试
- ✓安装测试
- ✓兼容测试
- ✓过程测试
- ✓容量测试
- ✓文档测试
- ✓……

性能测试

- 检查软件系统是否满足在需求规格说明中规定的性能或效率

  例如：响应时间、吞吐量，并发操作数、存储规模等

示例：课程注册管理系统
(1) 能支持同时有1000个用户登录使用；
(2) 用户操作平均响应时间少于1.5秒

通常需要自动化测试工具协助完成测试工作

示例：
(1) 月球车控制软件的主循环需要每50毫秒完成一次对各种负载状态和传感器数据进行更新；
(2) 月球车控制软件需要把运行时信息记录在某特定的内存区域。

强度测试

- 强度测试：不断施加更大的压力和使用强度，来获得系统能提供的最大服务能力。

与性能测试有何区别?

- 性能测试：软件系统在正常使用时是否能够达到一定的性能指标

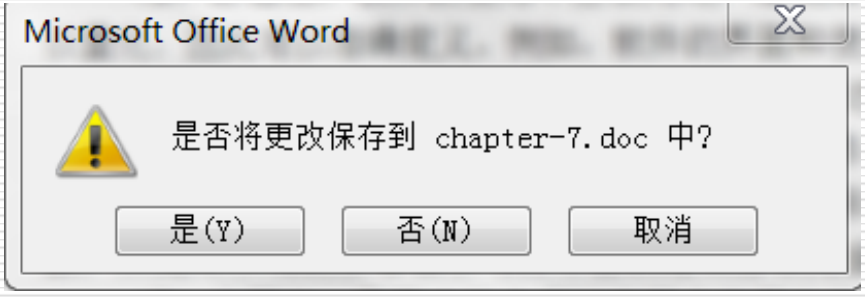
示例：课程注册管理系统
(1)最多能支持多少用户同时登录使用？

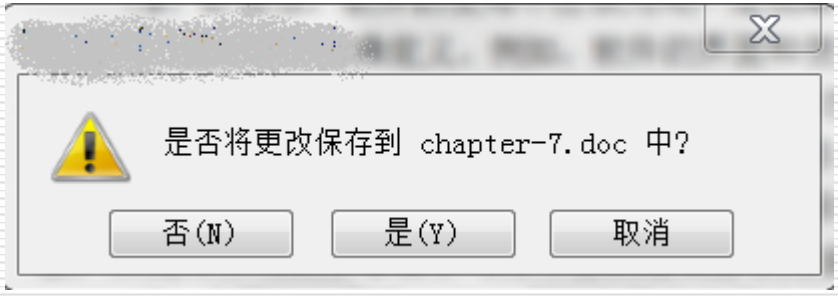
示例：月球车控制软件
(1)假定要求5米外发现障碍时能够及时停车，那么最近多少米时，能够不撞到障碍？

界面测试

- 软件的交互界面是否舒适、直观、符合使用习惯、提示及时且易于理解……

界面友好
界面美观
适应各种终端





可靠性测试

如果系统需求说明书中有对可靠性的要求，则需进行可靠性测试。

① 平均失效间隔时间 MTBF (Mean Time Between Failures) 是否超过规定时限?

② 因故障而停机的时间 MTTR (Mean Time To Repairs) 在一年中应不超过多少时间。

例如：

- ✓ 火箭控制软件在下次发射不会发生故障的概率为**99.9999%**；
- ✓ 软件的平均无故障时间为**34**小时；等等

安全性测试

- 要检验在系统中已经存在的系统安全性、保密性措施是否发挥作用，有无漏洞，能否抵御黑客的攻击…

力图破坏系统的保护机构，以进入系统。方法有：

✓正面攻击或从侧面、背面攻击系统中易受损坏的那些部分；

✓以系统输入为突破口，利用输入的容错性进行正面攻击；

申请和占用过多的资源压垮系统，以破坏安全措施，从而进入系统；

✓故意使系统出错，利用系统恢复的过程，窃取用户口令及其它有用的信息；

✓ 通过浏览残留在计算机各种资源中的垃圾（无用信息），以获取如口令，安全码，译码关键字等信息；

✓浏览全局数据，期望从中找到进入系统的关键字；

✓浏览那些逻辑上不存在，但物理上还存在的各种记录和资料等。

恢复测试

- 证实在克服硬件故障(包括掉电、硬件或网络出错等)后，系统能否正常地继续进行工作，并不对系统造成任何损害。

可采用各种人工干预的手段，模拟硬件故障，故意造成软件出错。并由此检查：

- ✓ 错误探测功能——系统能否发现硬件失效与故障；
- ✓ 能否切换或启动备用的硬件；
- ✓ 在故障发生时能否保护正在运行的作业和系统状态；
- ✓ 在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业，等等。
- ✓ 掉电测试：其目的是测试软件系统在发生电源中断时能否保护当时的状态且不毁坏数据，然后在电源恢复时从保留的断点处重新进行操作。

配置测试

这类测试是要检查计算机系统内各个设备或各种资源之间的相互联结和功能分配中的错误。

它主要包括以下几种：

- ✓ 配置命令测试：验证全部配置命令的可操作性（有效性）；特别对最大配置和最小配置要进行测试。软件配置和硬件配置都要测试。
- ✓ 循环配置测试：证明对每个设备物理与逻辑的，逻辑与功能的每次循环置换配置都能正常工作。
- ✓ 修复测试：检查每种配置状态及哪个设备是坏的。并用自动的或手工的方式进行配置状态间的转换。

可使用性测试

- 主要从使用的合理性和方便性等角度对软件系统进行检查，发现人为因素或使用上的问题。

要保证在足够详细的程度下，用户界面便于使用；对输入量可容错、响应时间和响应方式合理可行、输出信息有意义、正确并前后一致；出错信息能够引导用户去解决问题；软件文档全面、正规、确切。

可支持性测试

- 这类测试是要验证系统的支持策略对于公司与用户方面是否切实可行。

它所采用的方法是：

- ✓ 试运行支持过程(如对有错部分打补丁的过程，热线界面等)；
- ✓ 对其结果进行质量分析；
- ✓ 评审诊断工具；
- ✓ 维护过程、内部维护文档；
- ✓ 修复一个错误所需平均最少时间。

安装测试

- 安装测试的目的不是找软件错误，而是找安装错误。

- 在安装软件系统时，会有多种选择。
  - ✓ 要分配和装入文件与程序库
  - ✓ 布置适用的硬件配置
  - ✓ 进行程序的联结。

而安装测试就是要找出在这些安装过程中出现的错误。

- 安装测试是在系统安装之后进行测试。它要检验：
  - ✓ 用户选择的一套任选方案是否相容；
  - ✓ 系统的每一部分是否都齐全；
  - ✓ 所有文件是否都已产生并确有所需要的内容；
  - ✓ 硬件的配置是否合理，等等。

过程测试

- 在一些大型的系统中，部分工作由软件自动完成，其它工作则需由各种人员，包括操作员，数据库管理员，终端用户等，按一定规程同计算机配合，靠人工来完成。

- 指定由人工完成的过程也需经过仔细的检查，这就是所谓的过程测试。

兼容性测试

- 这类测试主要想验证软件产品在不同版本之间的兼容性。
- 有两类基本的兼容性测试：
  - ✓ 向下兼容
  - ✓ 交错兼容

容量测试

- 容量测试是要检验系统的能力最高能达到什么程度。例如，
  - ✓ 对于编译程序，让它处理特别长的源程序；
  - ✓ 对于操作系统，让它的作业队列“满员”；
  - ✓ 对于信息检索系统，让它使用频率达到最大。

在使系统的全部资源达到“满负荷”的情形下，测试系统的承受能力。

文档测试

这种测试是检查用户文档(如用户手册)的清晰性和精确。

用户文档中所使用的例子必须在测试中一一试过，确保叙述正确无误。