第十二章 软件测试策略

王美红

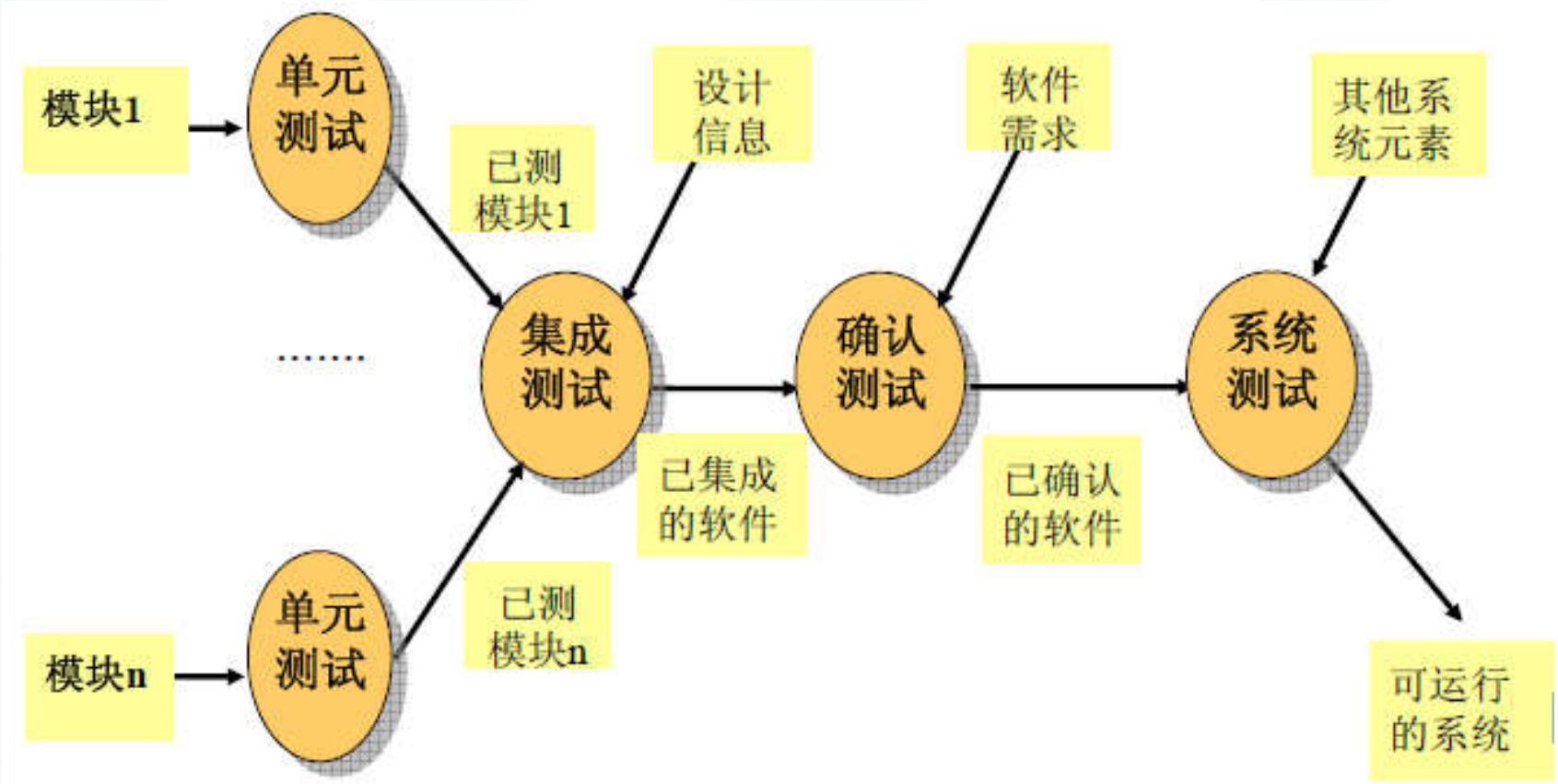
主要内容

- 什么是软件测试
- 软件测试的策略
- 软件调试

软件操作造成损失惊人

- IT失败每年将使全球经济付出惊人的6.2万亿美元的代价--- *IT复杂性危机：威胁与机会*
- 1996年6月发射的阿里亚娜5型火箭升空失败，损失25亿美元
- 美国航空业，1999年损失16亿美元
- 海湾战争中，“爱国者”导弹的雷达跟踪系统被扰乱，发射导弹时产生1/3秒误差，未能击中
- 伊拉克发射的“飞毛腿”导弹，造成多人伤亡
- 2003年8月14日，美国及加拿大部分地区发生了历史上最大的停电事故，原因是电力监测与控制系统出现软件错误

什么是软件测试

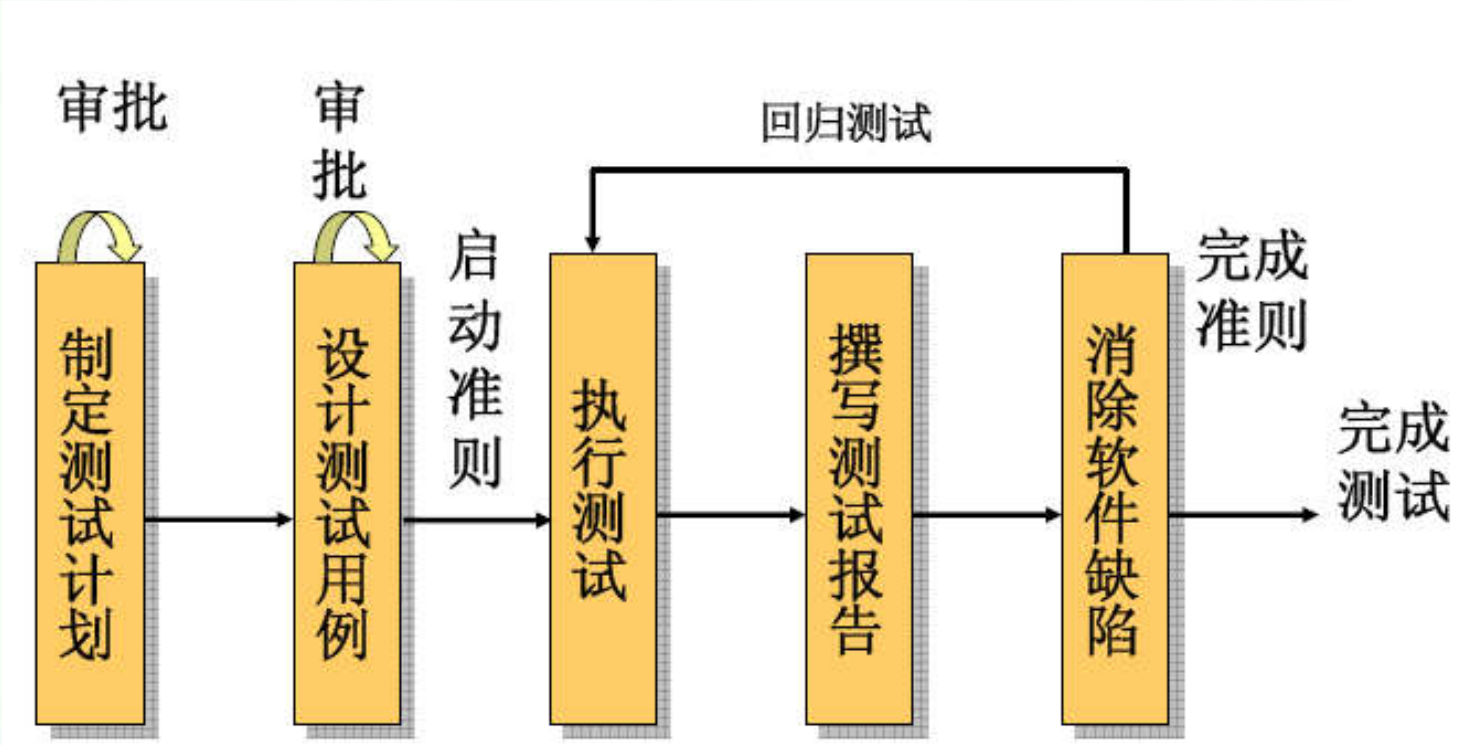


- 对软件需求分析、软件设计和编码的最终审查
- 是为了发现错误而执行程序的过程

12.1 软件测试的策略性方法

- 测试是可以事先计划并可以系统地进行的一系列活动。因此，应该为软件过程定义软件测试模板，即将特定的测试用例设计技术和测试方法放到一系列的测试步骤中去。

软件测试步骤



- 要求：具有足够的灵活性，同时，又必须严格

12.1 软件测试的策略性方法

- 测试策略应具备的特征：
  - 在测试前应经过认真的技术评审，将明显的错误消于无形。
  - 测试开始于构件
  - 不同的测试技术适用于不同的时间点
  - 测试由软件开发人员和独立的测试组执行
  - 测试不同于调试，它包括调试

验证、确认与测试的关系

- 验证是确保软件正确执行具有某一特定功能的一系列活动。
- 确认指的是确保开发的软件可追溯到客户需求的另外一系列活动。
- 另一种说法：
  - 验证：“我们在正确地构建产品吗？”
  - 确认：“我们在构建正确的产品吗？ ”
- 验证和确认包括---正式技术评审、质量和配置审核、性能监控、仿真等等。
- 测试是验证和确认活动的重要组成部分。

测试&质量

- 无论是大规模系统还是小规模系统，程序测试的根本动机都是使用经济且能有效应用的方法认可软件质量

软件测试的组织

- 软件测试是破坏性的（心理）
- 软件测试的组织—应由什么人来组织测试（看似正确的想法）：
  - 软件开发人员不应该参加测试
  - 让刻意挑毛病的陌生人来测试
  - 测试人员只在测试开始时才参与工作

实际上：

- 测试贯穿于整个软件的开发过程中，开发人员必须参与测试。
- 对即将交付软件的集成测试，开发人员往往也要参加。
- 集成测试中，有独立测试组（ITG）的介入
- ITG的目的是为了避免开发人员进行测试所引发的固有问题，以找错误为主要工作内容，避免利益冲突。
- ITG应从一开始就要介入项目，只是不像开发人员过多地注意细节。

12.1.3面向对象软件体系结构的测试策略

- 从小型测试走向大型测试
- 小型测试的对象是类
- 对相关类通信协作的集成测试
- 最后，作为一个整体来测试系统

12.1.4 测试完成的标准

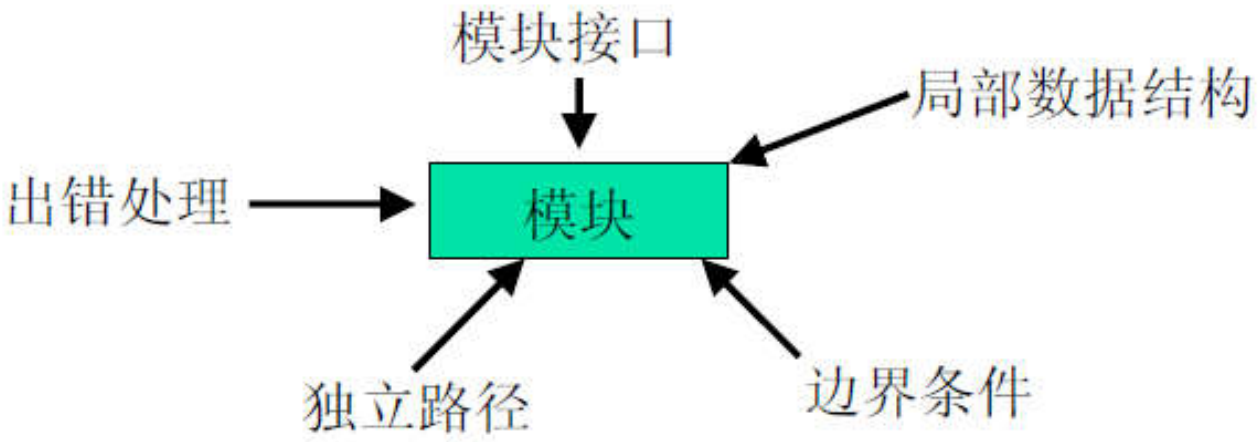
- 测试只能是在某个阶段告以段落，由于软件使用的软环境可能要永恒地变化，所有，它时刻、永远地面临考验，没有尽头，即测试是永远也完不成的。
- 统计质量保证方法提出了统计使用技术，在测试过程中搜集度量数据并利用现有的统计模型

12.2 策略问题

- 在开发软件时，应以量化方式设计测试指标
- 显示地陈述测试目标
- 为软件的不同类用户量身定做测试计划
- 软件设计时应设计测试接口
- 正式技术评审能完成测试所需的前期工作
- 对测试策略和用例进行正式技术评审
- 为测试过程建立一种持续改进方法

12.3 传统软件测试策略

- 单元测试
  - 内容：根据详细设计说明书、程序清单，采用白盒、黑盒测试的测试用例，对模块进行检查，主要包括以下五部分



（1）模块接口测试

- 被测模块的入口参数
- 被测模块调用子模块时的入口参数
- 输出给标准函数的参数
- 全局量的定义在各模块中是否一致
- 当模块对外部设备进行操作时，要测试：
  - 文件属性、**I/O格式等**

(2) 局部数据结构测试

- 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

(3) 路径测试

设计测试用例对重要路径进行测试

- 常见的计算错误：
  - 运算的优先次序
  - 不同类型的运算对象进行计算
  - 运算精度不够

- 常见的比较和控制流错误：
  - 不同数据类型相比较
  - 浮点数比较
  - 循环次数不正确
  - 不可能的循环中止条件

(4) 错误处理测试

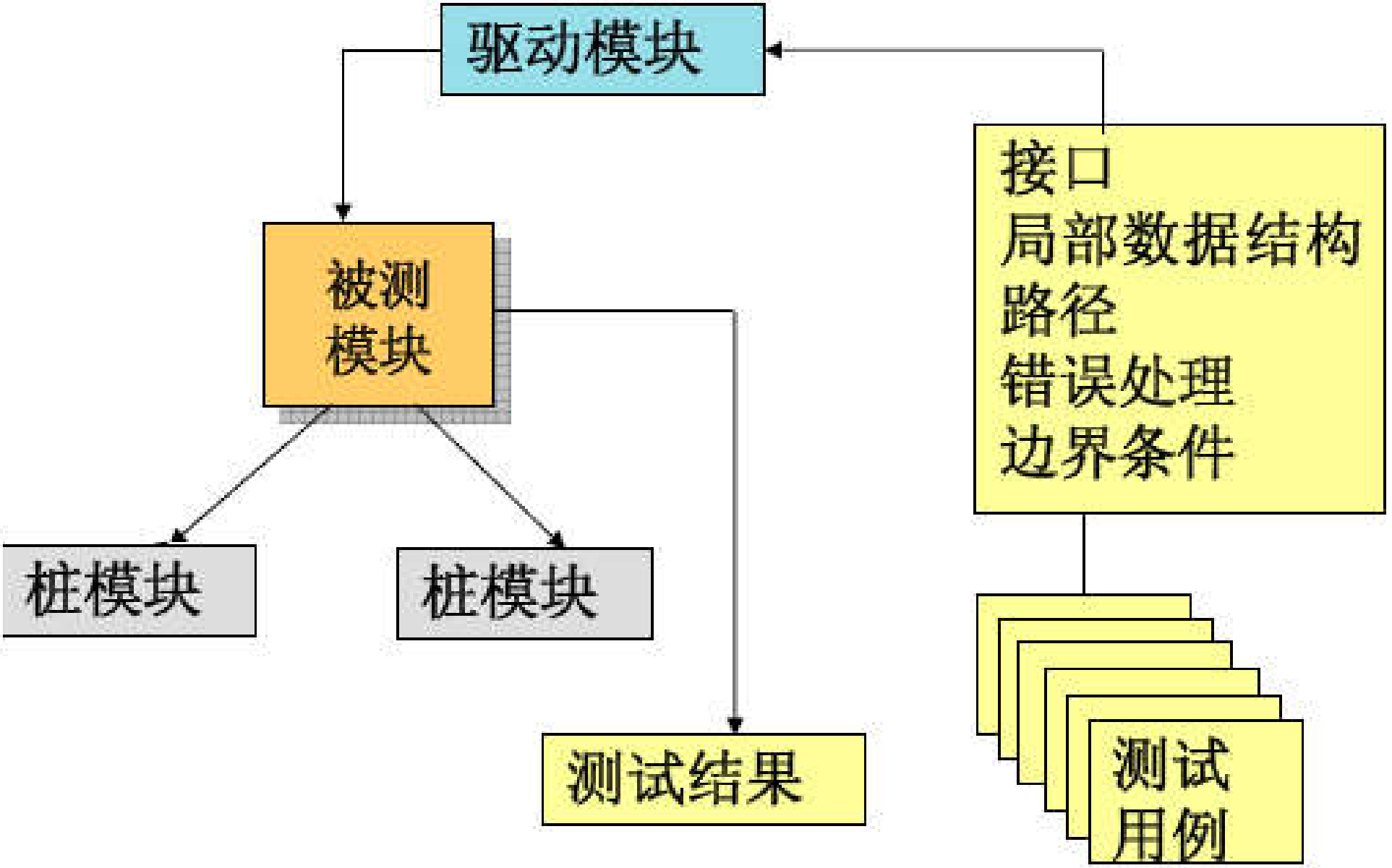
- 错误处理测试：好的模块设计要能预见出错的条件，设置出错处理
  - 常见的有问题的出错处理
    - 出错描述难以理解
    - 出错不易定位
    - 出错显示与实际错误不符
    - 在对错误处理之前已引起系统的干预

(5) 边界测试

- 检查各种边界情况
  - 循环的第**n**次
  - 取最大和最小值时
  - 比较中刚好等于、大于、小于时

12.3 传统软件测试策略（续）

- 单元测试规程----所测试单元不是独立的程序，它可能和其它部分交互信息，在测试时，应模拟这些数据或信息（驱动软件和桩软件）


驱动模块
被测
模块
桩模块
桩模块
测试结果
接口
局部数据结构
路径
错误处理
边界条件
测试
用例

12.3 传统软件测试策略（续）

- 集成测试
  - 又称组装测试，联合测试
  - 将所有模块按设计要求组装成系统
  - 一步到位与增量集成

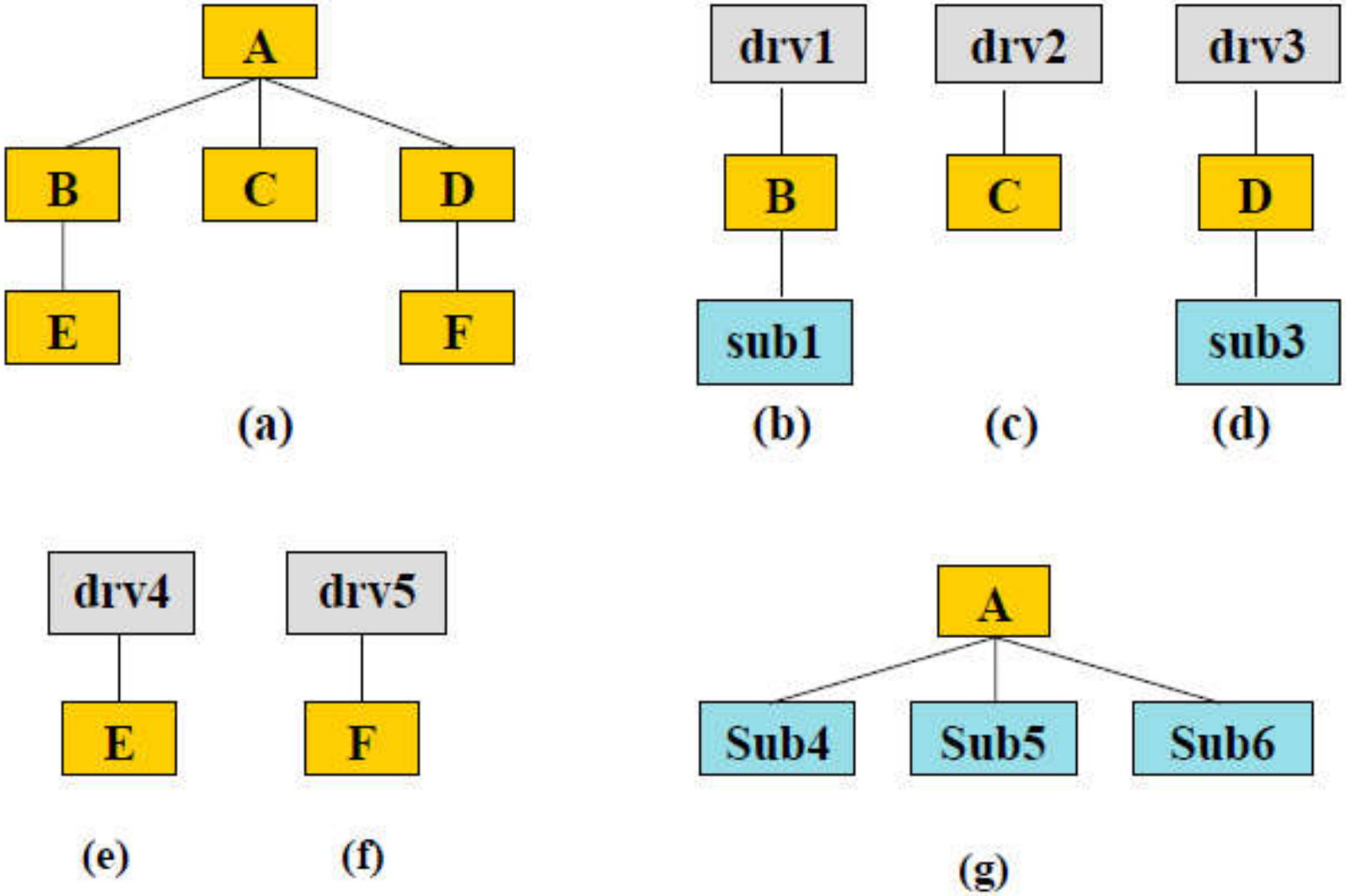
12.3 传统软件测试策略（续）

1. 一次到位：

– 又称为非增式组装，一次性组装或大爆炸集成，这种集成将所有单元在一起编译并进行一次性测试。

- 缺点：
  – 一次成功的可能性不大
  – 当发现缺陷时，没有多少线索能够用来帮助确定缺陷位置。

一次性组装 实例


A
B
C
D
E
F
(a)
drv1
drv2
drv3
B
C
D
sub1
sub3
(b)
(c)
(d)
drv4
drv5
E
F
(e)
(f)
A
Sub4
Sub5
Sub6
(g)

12.3 传统软件测试策略（续）

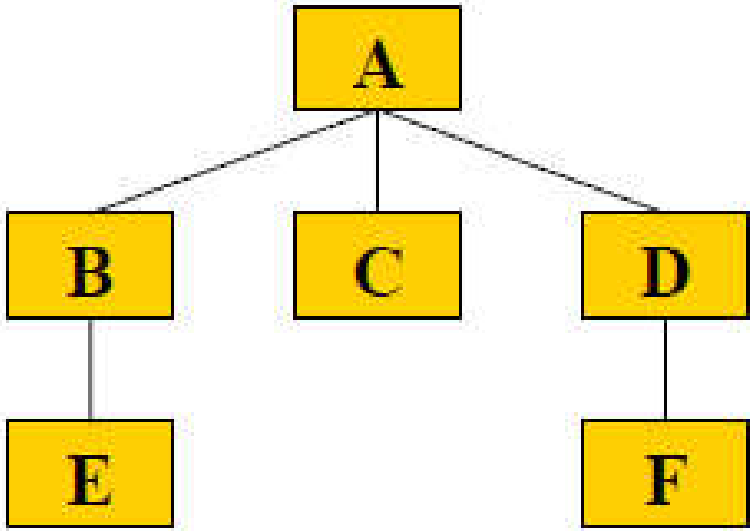
2. 增式组装：

– 逐步组装成较大的系统，边组装边测试

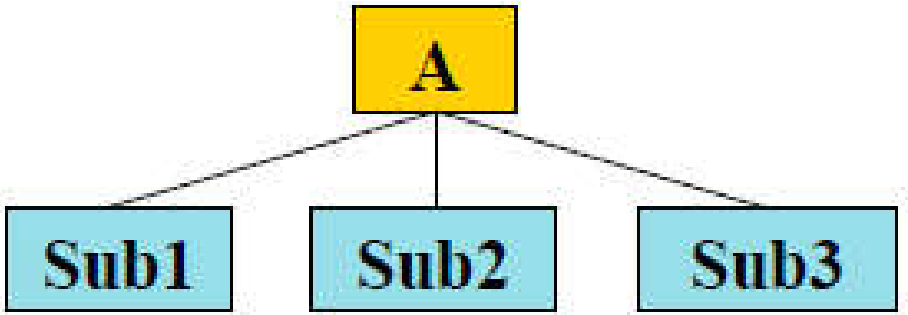
- 自顶向下的增殖方式
- 自底向上的增殖方式
- 混合增殖方式

自顶向下的增殖方式

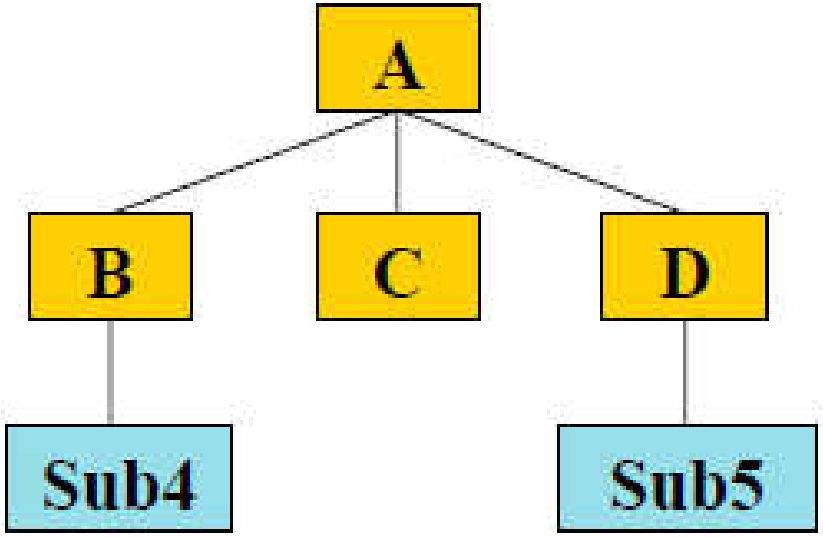
- 以主模块为被测模块及驱动模块，所有直属于主模块的下属模块用桩模块代替，对主模块进行测试
- 采用深度优先或广度优先的分层策略，用实际模块代替相应桩模块，再用桩模块代替它们的直接下属模块，组装成新的子系统
- 进行回归测试（重新执行以前做过的测试），排除组装过程中引入新的错误的可能
- 完成所有组装



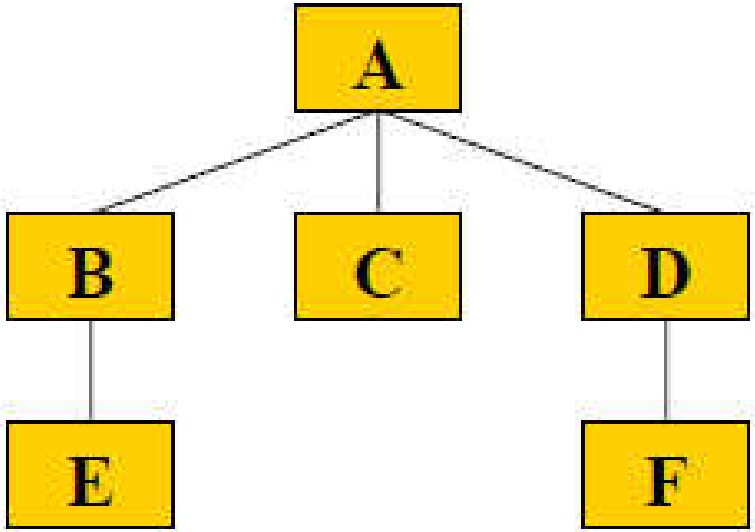
(a)



(b)



(c)



(d)

编写测试驱动模块以及桩模块的实例被测代码：

被测代码：

```
int Max()
    {
    int a = fun1(); //需要打桩
    int b = fun2(); //需要打桩

    if (a>b)
        return a;
    else
       return b;
    }
```

驱动模块：

```
int main()
    {
        //初始化
        int ret = Max();
        //判断结果是否符合预
    期
    }
```

桩模块：

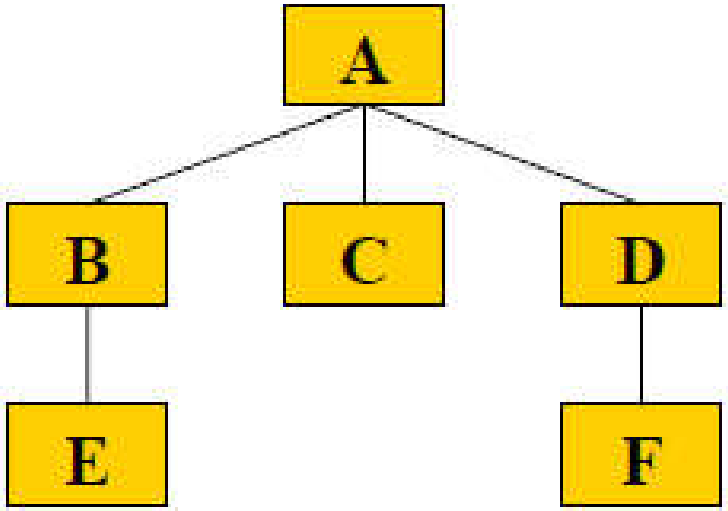
```
int fun1(){return 0;};
int fun2(){return 0;};
```

自顶向下的增殖方式

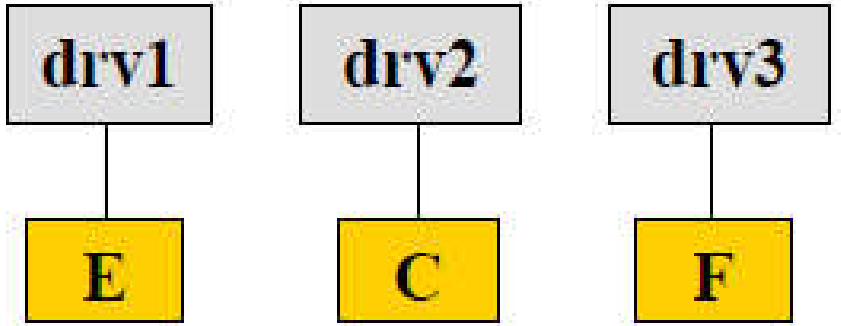
- 桩模块的几种情况：
  - 显示跟踪信息
  - 显示传递的信息
  - 从一个表（或外部文件）返回一个值
  - 进行一项表查询以根据输入参数返回输出参数

自底向上的增殖方式

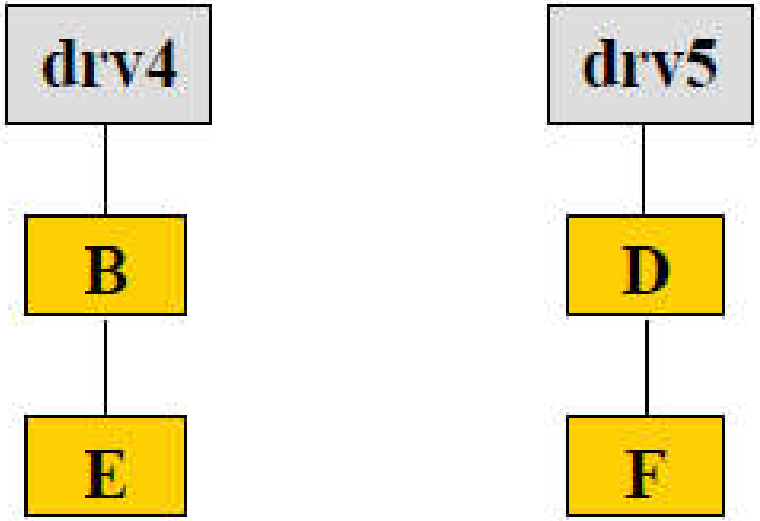
- 由驱动模块控制最底层模块进行测试
- 用实际模块代替驱动模块，组装成子系统
- 为子系统配备驱动模块，进行测试
- 完成所有组装



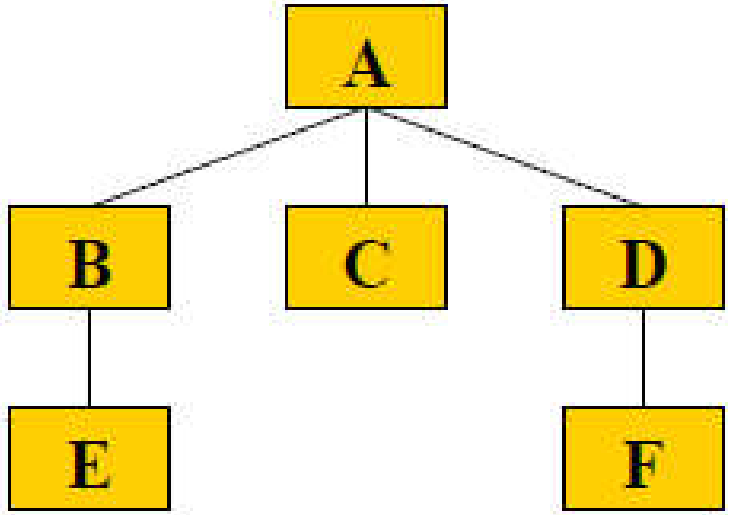
(a)



(b) (c) (d)



(e) (f)



(g)

自底向上的增殖方式

- 驱动模块的几种情况：
  - 调用从属模块
  - 从表（或外部文件）中传送参数
  - 显示参数
  - 兼有上面两个功能

比较

自顶向下方式

- 缺点：需要建立桩模块；重要而复杂的算法模块一般在底层
- 优点：较早的发现主要控制方面的问题

自底向上方式

- 优点：不需要建立桩模块，建立驱动模块一般比建立桩模块容易；容易出问题的部分在早期解决；易于并行测试，提高效率
- 缺点：最后才接触到主要的控制

混合增殖方式

- 衍变的自顶向下的增殖测试：
  - 强化对输入输出模块和新算法的测试，并自底向上组装成功能完整且相对独立的子系统，然后再由主模块自顶向下进行增殖测试
- 自底向上—自顶向下的增殖测试：
  - 对含读操作的子系统自底向上进行组装测试，对含写操作的子系统做自顶向下的组装测试

回归测试

- 对所修改的模块及其子模块进行自顶向下测试
- 测试完成后将其视为子系统，再自底向上测试

冒烟测试

- 冒烟测试的对象是每一个新编译的需要正式测试的软件版本，目的是确认软件基本功能正常，可以进行后续的正式测试工作。冒烟测试的执行者是版本编译人员。
- 可以理解为耗时短，仅用一袋烟功夫足够了

冒烟测试

1. 将已经转换为代码的软件构件集成为“Build”
2. 设计一系列测试以暴漏影响build正确地完成其功能的错误
3. 每天该“build”与其他“build”及整个软件产品集成起来进行冒烟测试

测试策略的选择

- 策略选择：
  - 依赖于软件的特征，有时也与项目进度安排有关。
  - 组合方法：
    - 自顶向下测试程序高层；自底向上测试从属层；应着重测试关键模块

测试策略的选择

- 关键模块：
  - 满足某些软件需求
  - 处于较高层次
  - 较复杂、易发生错误
  - 有明确定义的性能要求
- 关键模块应尽早测试

测试集成文档

- 测试计划和步骤文档化，是今后软件配置的一部分。软件集成的计划和测试描述等都应该是测试文档一部分。
- 测试相关文档：
  - 测试计划；测试需求分析（可不写，依项目而定）；测试方案；测试用例；测试执行计划（可不写）；测试报告。
  - 例：来自微软网站上的一份测试计划

12.4 面向对象软件的测试策略详解

- 单元测试
  - 测试对象是类，类包含属性、操作等，有些类之间有类似的属性与操作，此时可以考虑同时测试这些指标

12.4 面向对象软件的测试策略详解

- 集成测试

1. 基于线程的测试

  - 集成响应系统的一个输入或事件所需的一组类，每个线程被集成并分别测试，应用回归测试以保证没有产生副作用。

面向对象—集成测试

2.基于使用的测试

①通过测试那些几乎不使用服务器类的类(称为独立类)来开始系统的构造

②在独立类测试完成后，下一层的使用独立类的类，称为依赖类，被测试

③这个依赖类层次的测试序列一直持续到构造完整个系统。

面向对象—集成测试

- 驱动程序和桩程序：
  - 驱动程序可用于：
    - 测试低层中的操作和整组类的测试
    - 代替用户界面以便于在界面实现之前进行系统功能的测试
  - 桩程序可用于：
    - 在需要类间的协作但其中一个或多个协作类未完全实现的情况

12.5 确认测试

- 在传统软件和面向对象软件间没有明显差别。
- 始于集成测试的结束
- 验证软件的功能和性能及其它特性是否与用户要求一致

确认测试的步骤

- 进行有效性测试（黑盒测试）
  - 制定测试计划、测试步骤，设计测试用例
  - 确定所有的功能、性能均满足要求
- 软件配置复查
- α测试和β测试
- 验收测试：由用户用实际数据进行测试。考虑软件的功能、性能、可移植性、兼容性、可维护性

α测试和β测试

- α测试：
  - 由一个用户在受控环境下进行的测试。
  - 最终用户在开发者的场所进行。
  - 目的是评价软件产品的**FLURPS（功能、局域化、可使用性、可靠性、性能、支持）**，产品的界面和特色

α测试和β测试

- β测试：
  - 由多个用户在实际使用环境下的测试。
  - 用户定期向开发者报告软件运行的问题。
  - 主要衡量产品的**FLURPS**

确认测试完成的标志

- 功能和性能与用户的要求一致，用户接受
- 应交付的文档：
  - 确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告

12.6 系统测试

- 系统测试是基于实际应用环境对计算机系统的一种多方位的测试，每一种测试都具有不同的目的，但所有的测试都是为了检验各个系统成分能否正确集成到一起并且是否能完成预定的功能。

12.6 系统测试（续）

- 测试方法：
  - 恢复测试： 测试计算机系统在一定时间内在错误情况下恢复并能继续运行的能力
  - 安全性测试：测试计算机系统内的保护机制能否保护系统不受到非法侵入
  - 压力测试：测试软件对非正常情况的处理能力
  - 性能测试：测试软件在集成系统中的运行性能

验收测试的内容

- 明确验收项目，规定验收测试通过的标准。
- •确定测试方法。
- 决定验收测试的组织机构和可利用的资源。
- 选定测试结果分析方法。
- 指定验收测试计划并进行评审。
- 设计验收测试所用测试用例。
- 审查验收测试准备工作。
- 执行验收测试。
- 分析测试结果。
- 阐明验收测试结论，决定通过验收或拒绝。

测试种类（自学）

- 软件测试是由一系列不同的测试组成。主要目的是对以计算机为基础的系统进行充分的测试。
- 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。

测试种类

- 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。

测试种类

- 强度测试
  - 强度测试是要检查在系统运行环境不正常乃至发生故障的情况下，系统可以运行到何种程度的测试。例如：
    - 把输入数据速率提高一个数量级，确定输入功能将如何响应。
    - 设计需要占用最大存储量或其它资源的测试用例进行测试。

测试种类

- 设计出在虚拟存储管理机制中引起“颠簸”的测试用例进行测试。
- 设计出会对磁盘常驻内存的数据过度访问的测试用例进行测试。

– 强度测试的一个变种就是敏感性测试。在程序有效数据界限内一个小范围内的一组数据可能引起极端的或不平稳的错误处理出现，或者导致极度的性能下降的情况发生。此测试用以发现可能引起这种不稳定性或不正常处理的某些数据组合。

测试种类

- 性能测试
  - 性能测试是要检查系统是否满足在需求说明书中规定的性能。特别是对于实时系统或嵌入式系统。
  - 性能测试常常需要与强度测试结合起来进行，并常常要求同时进行硬件和软件检测。
  - 通常，对软件性能的检测表现在以下几个方面：响应时间、吞吐量、辅助存储区，例如缓冲区，工作区的大小等、处理精度，等等。

测试种类

- 恢复测试
  - 恢复测试是要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
  - 为此，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；

测试种类

- 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。

测试种类

- 启动/停止测试
  - 这类测试的目的是验证在机器启动及关机阶段，软件系统正确处理的能力。
  - 这类测试包括
    - 反复启动软件系统（例如，操作系统自举、网络的启动、应用程序的调用等）
    - 在尽可能多的情况下关机。

测试种类

- 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。

测试种类

- 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。

测试种类

- 安全性测试
  - 安全性测试是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。
  - 力图破坏系统的保护机构以进入系统的主要方法有以下几种：
    - 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；

测试种类

- 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。

测试种类

- 可使用性测试
  - 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。

测试种类

- 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。

测试种类

- 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。

测试种类

- 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。

测试种类

- 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。

测试种类

- 互连测试
  - 互连测试是要验证两个或多个不同的系统之间的互连性。
- 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

测试种类

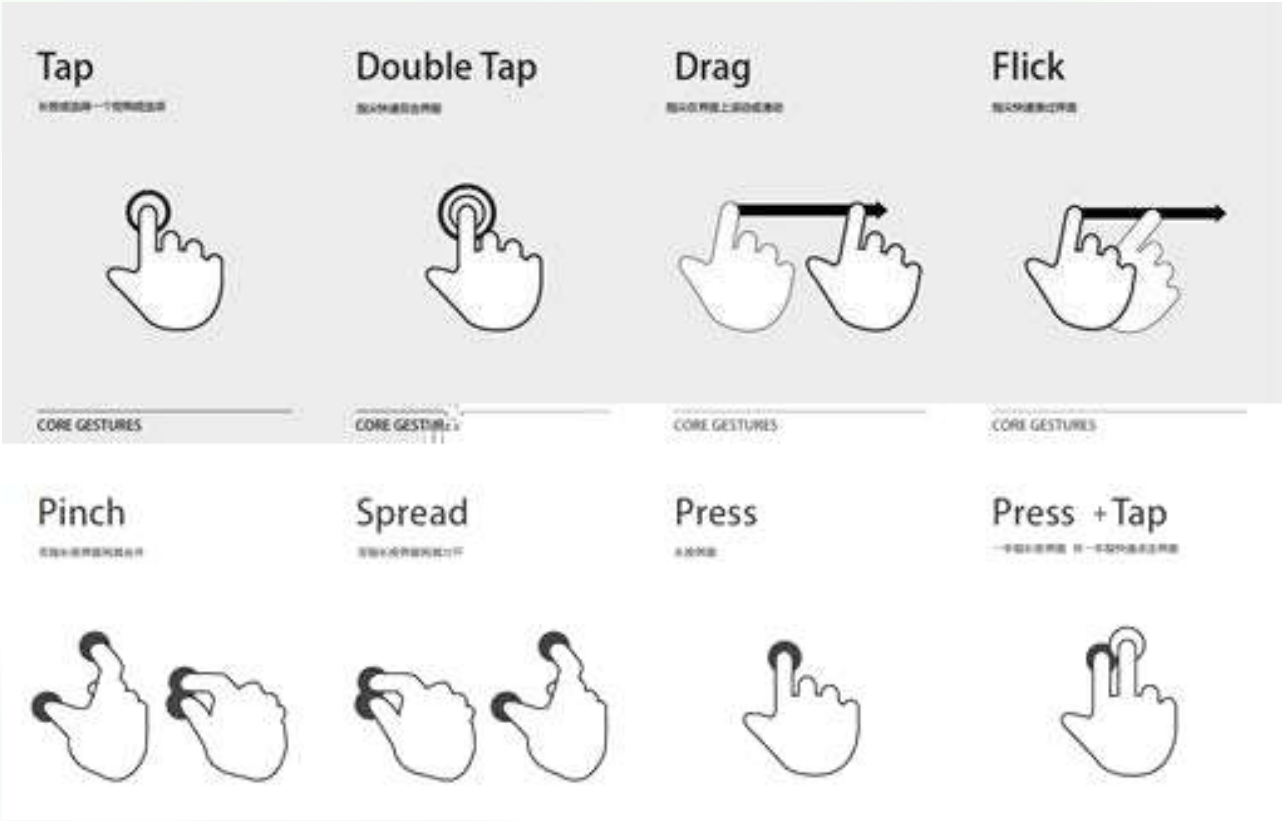
- 容量测试
  - 容量测试是要检验系统的能力最高能达到什么程度。例如，
    - 对于编译程序，让它处理特别长的源程序
    - 对于操作系统，让它的作业队列“满员”；
    - 对于信息检索系统，让它使用频率达到最大。

  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。

测试种类

- 文档测试
  - 这种测试是检查用户文档(如用户手册)的清晰性和精确性。
  - 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。

移动APP手势测试


Tap
Double Tap
Drag
Flick
CORE GESTURES
CORE GESTURES
CORE GESTURES
CORE GESTURES
Pinch
Spread
Press
Press +Tap

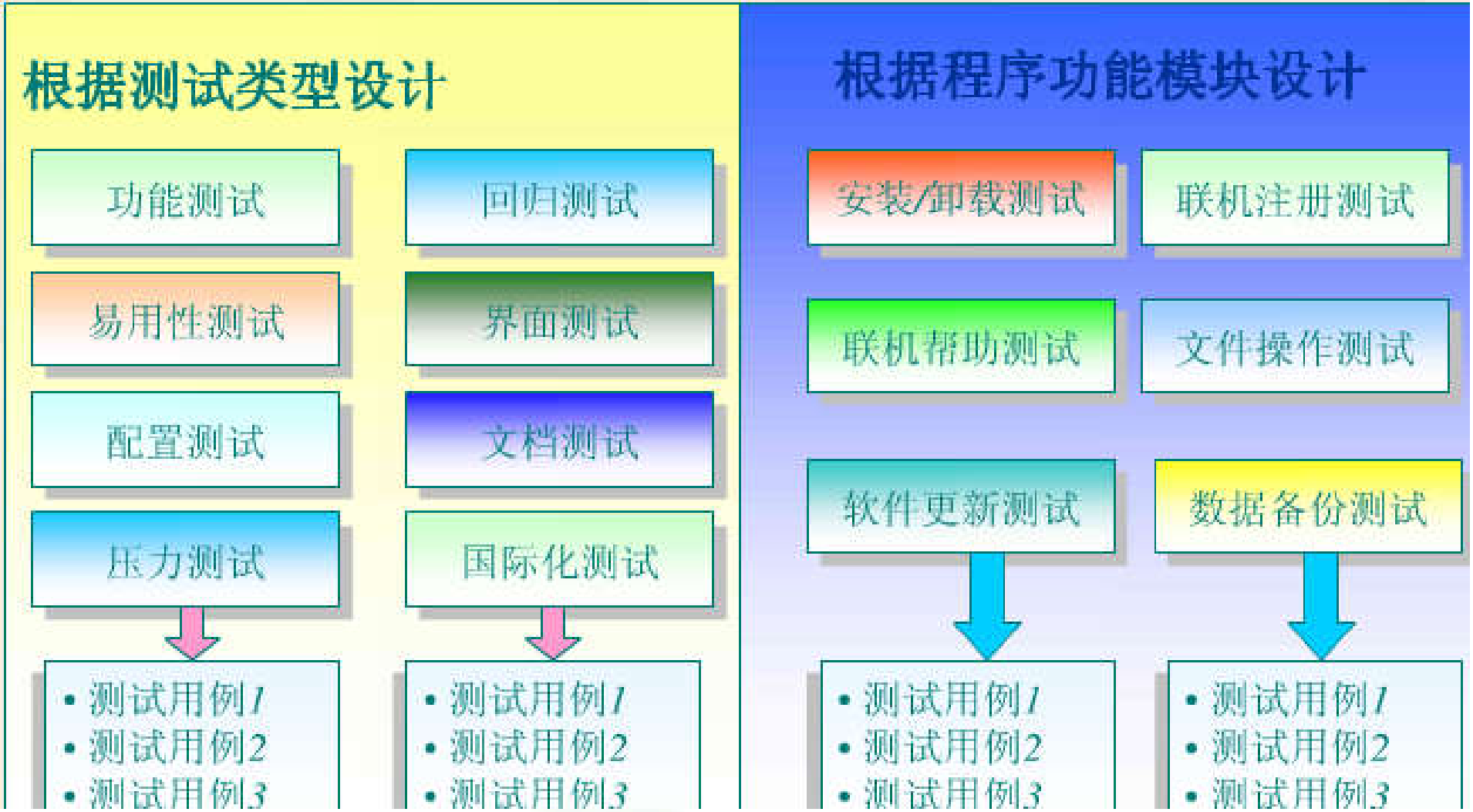
导航测试

- 对导航机制（链接及所有类型的锚、重定向、书签、框架和框架集、站点地图以及内部搜索工具的准确性）进行测试，以确保每个机制都能执行其预期功能。

国际化测试

- 国际化是一个创建软件产品的过程，它使得在多个国家、操着各种语言来使用产品成为可能，而不需做任何工程的改变。
- 除了语言，还需要考虑不同货币、不同文化、税收和标准。

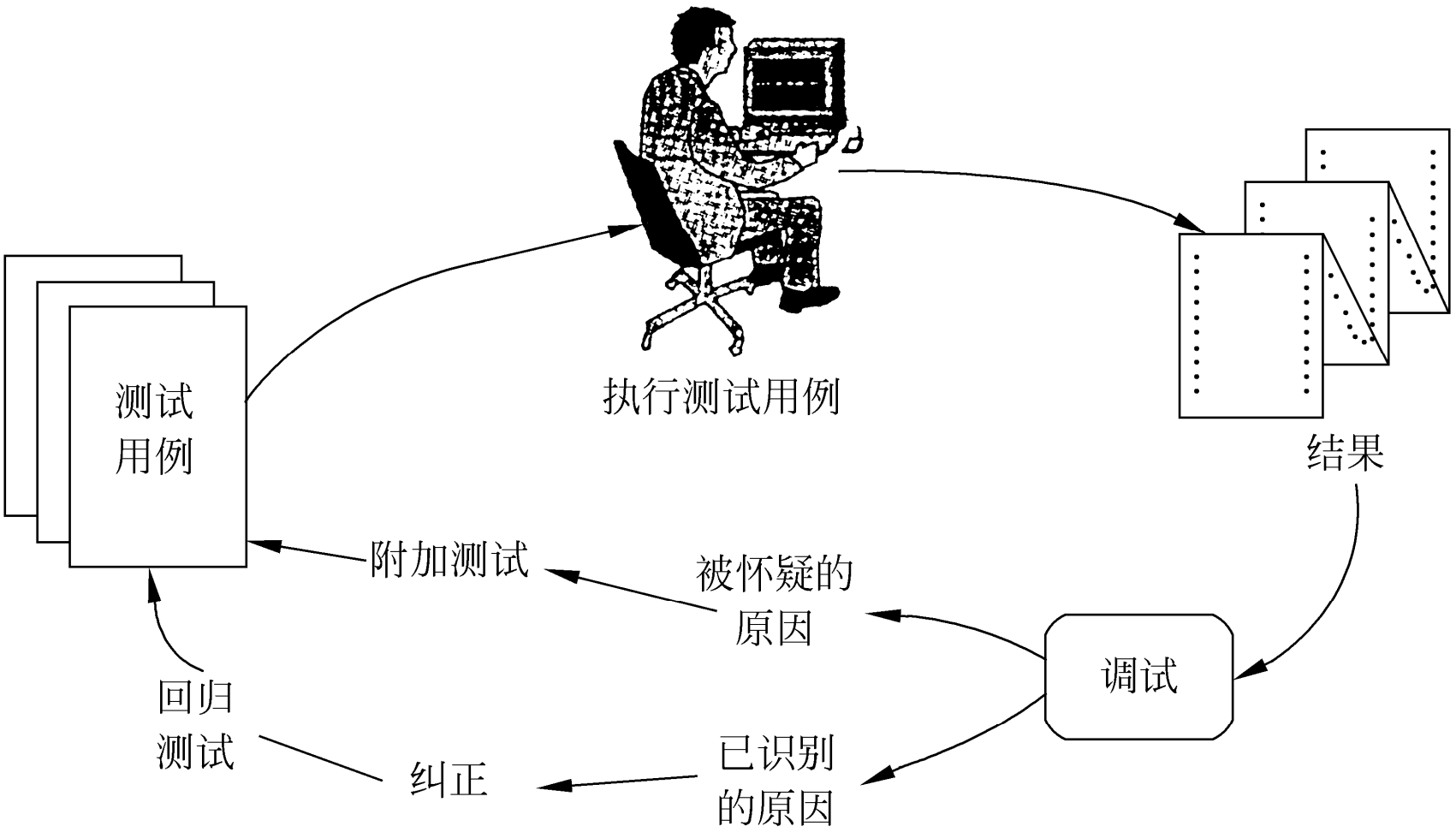
测试类型与测试用例设计



12.7 调试技巧

- 软件调试是在进行了成功的测试之后才开始的工作。它与软件测试不同，调试的任务是进一步诊断和改正程序中潜在的错误。
- 调试活动由两部分组成：
  - 确定程序中可疑错误的确切性质和位置.
  - 对程序(设计,编码)进行修改，排除这个错误。

软件调试过程



测试与调试的比较

测试 (test)	调试 (debug)
发现错误	找出错误位置，排除
有计划	被动的
以已知条件开始， 使用预先定义的程序， 有预知的结果	以不可知内部条件开始， 结果一般不可预见
由独立的测试组，在 不了解软件设计的条 件下完成	由程序作者进行

找出错误是困难的

- 现象与原因的位置可能相距甚远
- 改正其它错误时，现象可能暂时消失，但并没有排除
- 现象由一些非错误原因引起的（如计算不精确）
- 错误是由于时序问题引起，与处理过程无关
- 错误征兆时有时无
- 错误是由于任务分布在若干台不同处理机上运行而造成的

调试的步骤

- 从错误的外部表现入手，确定出错位置
- 找出错误的内在原因
- 修改代码，排除错误
- 重复测试，以确认：是否排除了该错误，是否引进了新的错误
- 如果修改无效，则恢复原样，重复以上过程，直到改正了错误

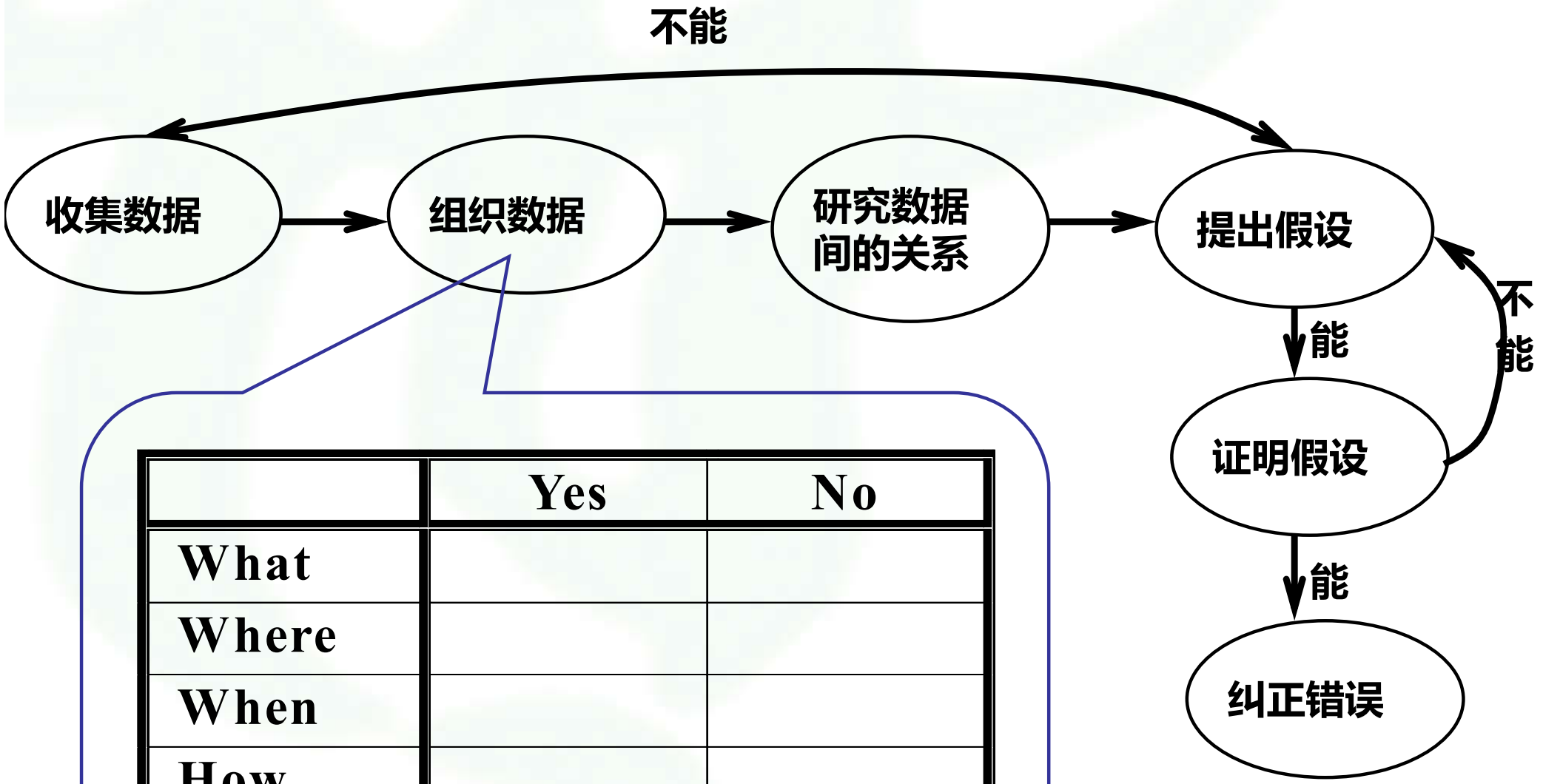
主要的调试方法

- 强行排错
  - 将内存全部打印来排错
  - 在程序特定部位设置打印语句：
    - 在关键变量改变部位、重要分支部位、子程序调用部位设置打印语句，跟踪程序的执行，监视变量的变化
  - 自动调试工具：利用某些程序语言的调试功能，分析程序的动态过程

主要的调试方法（续）

- 回溯法
  - 从出现问题的地方，回溯跟踪源程序，直到找到错误根源
- 归纳法：从特殊推断一般
  - 收集数据
  - 组织数据
  - 提出假设
  - 证明假设

例 归纳法：从错误症状中找出规律，推断根源



	Yes	No
What		
Where		
When		
How		

3W1H Table

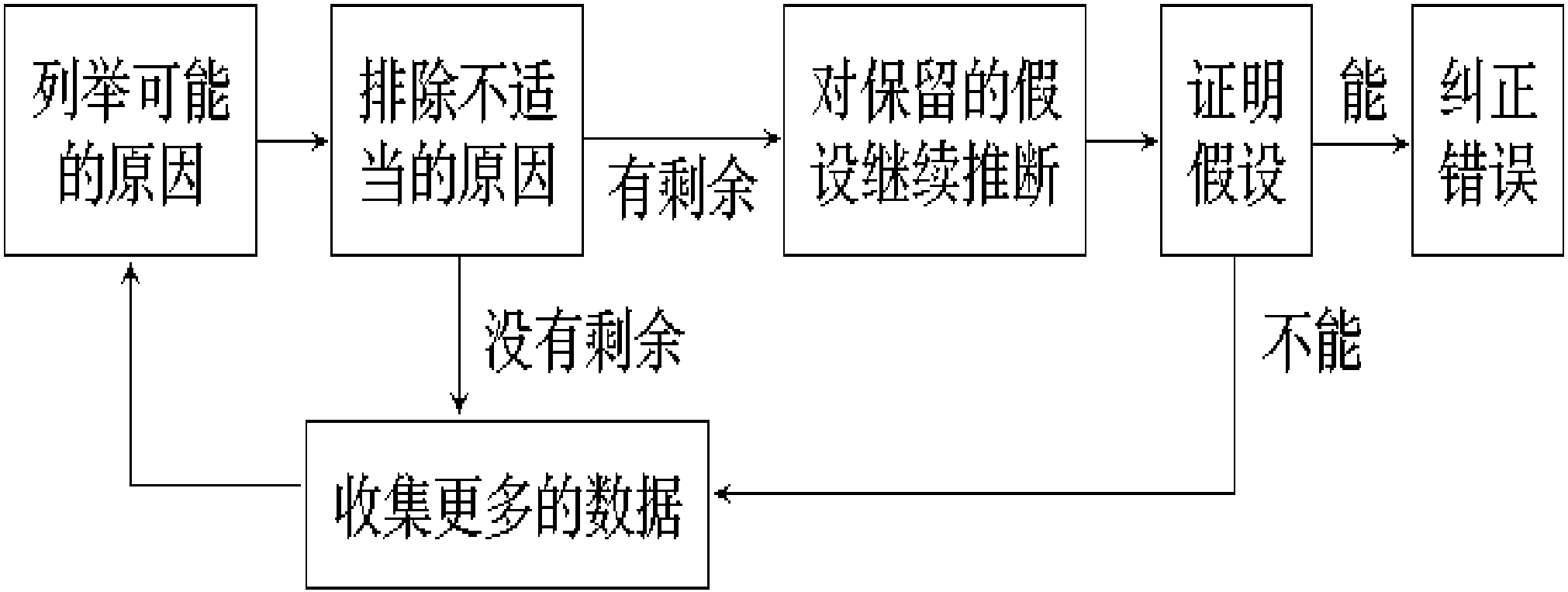
发现错误：对51个学生评分 ⇒ 中间值为26（期望值80）
对1个学生评分 ⇒ 中间值为1

	Yes	No
What	第2号报告中打印的中间值有误	平均值和标准偏差的计算
Where	在第 2 号报告中	在其它报告中，学生分数计算似乎正确
When	在对 51 个和 1 个学生评分的测试时发现	
How	51 个学生 → 中间值 = 26 1 个学生 → 中间值 = 1	对 2 个和 200 个学生计算时并无错误

观察分析：取奇数时出错?
打印的是中间学生的编号而非分数?
加测试来验证上述推测。

主要的调试方法（续）

- 演绎法：从一般原理或前提出发，用排除法来得到结论



错误改正

- 一旦找到错误，就必须纠正
  1. 这些错误的原因在程序的另一部分也产生过吗？ --有助于发现其他错误
  2. 进行修改可能引发的“下一个错误”是什么？
  3. 为避免这个错误，我们首先应当做些什么？