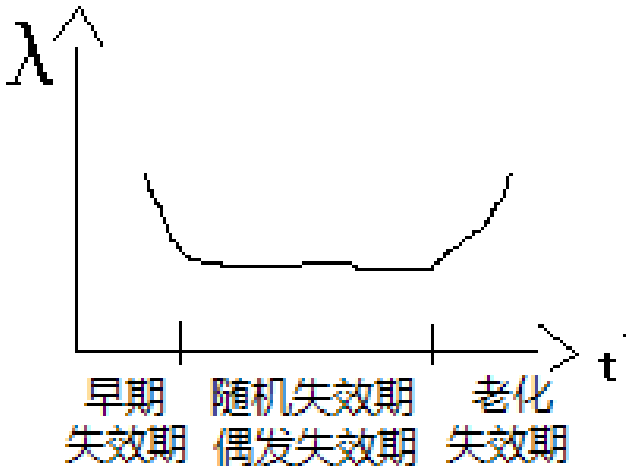
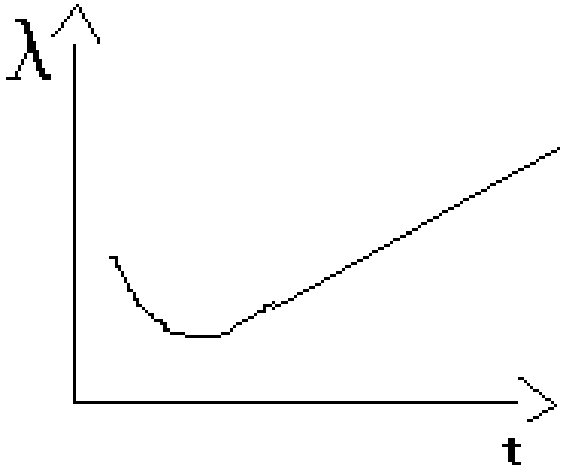
嵌入式软件可靠性设计

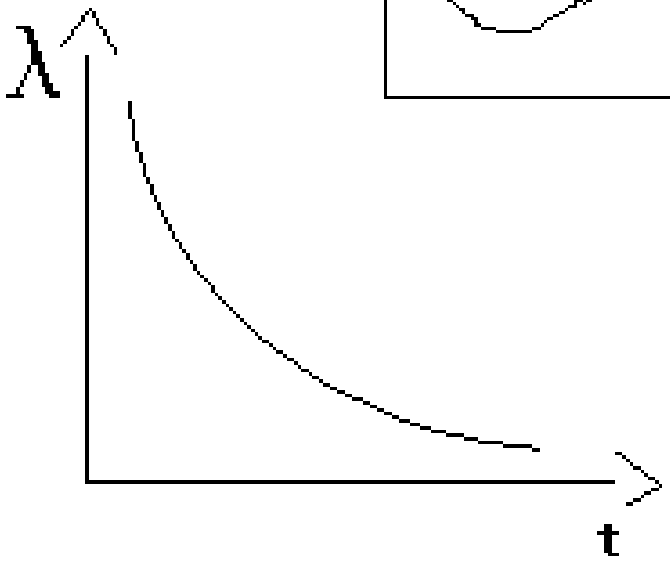
嵌入式软件开发管理方式与电子企业研发组织管理模式的区别

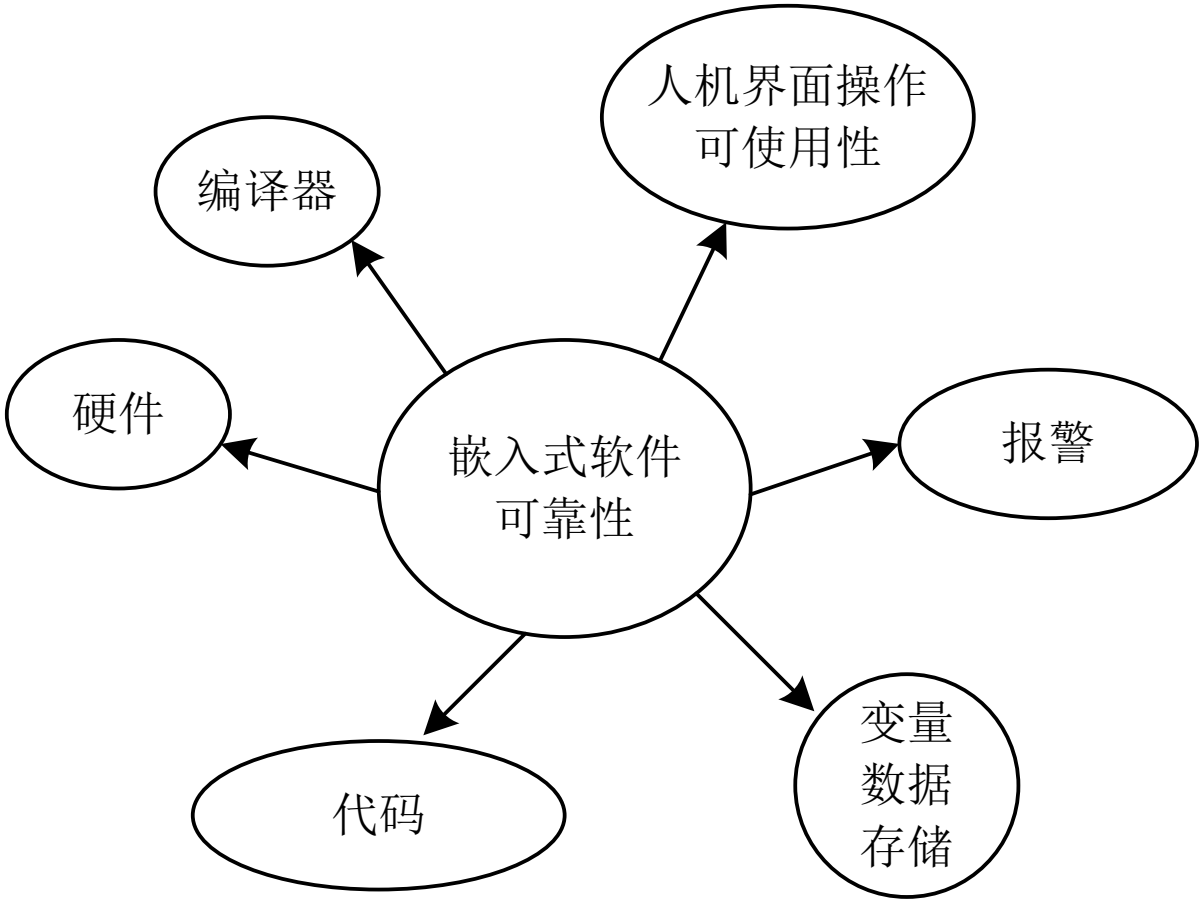
嵌入式软件的特点：通过控制硬件实现功能

电子、机械、软件的失效率分布图对比








人机界面操作
可使用性
编译器
硬件
嵌入式软件
可靠性
报警
代码
变量
数据
存储

编 译 器

```
i=1;
i++;
p=i++;
i++;
```

请问：i=?

p=?

- 单一语句，单一功能

1. 转生产过程的编译器版本控制
2. 多人开发软件集成时的版本控制
3. 强制定义默认值

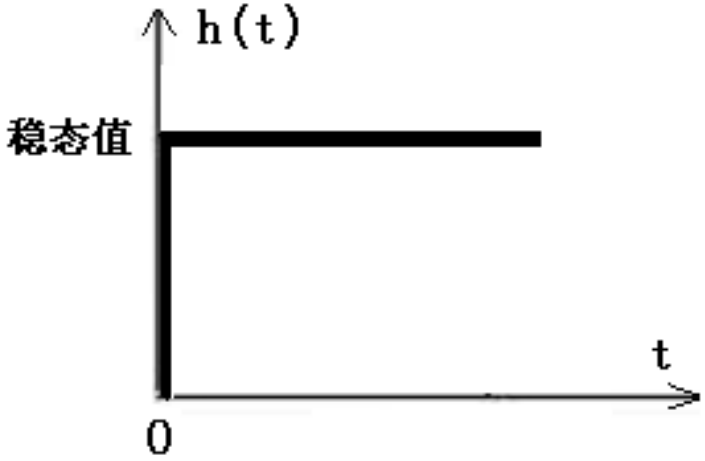
代　码

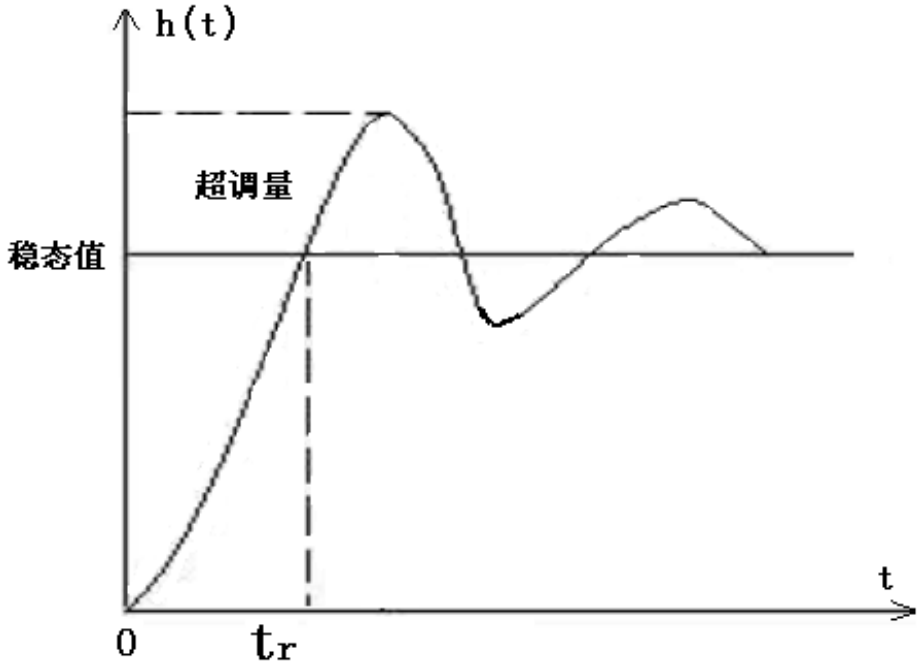
1. 隐性问题显性化
2. 单一功能，单一函数（代码复用）
3. 安全性内核
4. 形参只做数值传递不参与运算
5. 中间数据的有效性判断
6. 数据数值范围有效性判断
7. 工作时间不超过1/2时启动休眠
8. 代码中不重复性的出现同一数值，用宏定义
   避免笔误和数值更改可能的遗漏
9. 全局变量的操作采用单独的函数
10. 设计更改时接口向前兼容
11. 代码冗余编程方式
12. 防跑飞的软件陷阱

硬　　件

1. 硬件和机械的执行特性对软件的要求
2. 通过软件避免上电时序触发的闩锁效应
3. 中断与查询方式哪个运算速度更快？
4. 光耦的传输速率导致的信号失真问题
5. 继电器、传感器、负载等故障时的系统反应须安全，即SFC状态时，系统输出须安全
6. 供电系统的串并联结构导致信号波动
7. 精度分配

执行机构的启动特性





人机界面操作 / 可使用性

1. 安全关键信息的保存必须有提示
2. 安全关键功能的执行必须经过再次确认
3. 字体按照显示信息的重要性分级
4. 不超过2个主色调，其他均为点缀
5. 不超过3次按键要能进入程序的功能最底层
6. 显示界面不翻页

变量 / 数据 / 存储

1. 变量命名避免类型强制转化和作用域超标
2. 初始参数的多次刷新、控制参数多次检查
3. 异地容灾备份存储
4. 同一存储区域的不同数值存储方式
5. Flash的块存储方式
6. 强数据类型

报　　警

报警频率

报警优先级

报警提示的要求

软件测试

1. 无限接近于100%覆盖
2. 圈复杂度

测试-基本概念

测试的原则

- 不应测试自己开发的程序
- 设计测试用例时，不仅有确定的输入数据，还有确定的输出数据
- 测试用例不仅有合理的，也要有非合理的
- 除了检查程序是否做完了它应该做的事，还要检查它是否做了不应该做的事。
- 保留全部测试用例，作为软件的组成部分
- 程序中存在错误的概率与在该段程序中已发现的错误数成正比

基本概念

1. 什么是测试
   - 测试也称调试，它包括模块测试（单调）、子系统测试（分调）、系统测试（联调）
   - 测试是假定程序中存在错误，因而想通过测试来发现尽可能多的错误。
2. 测试的目标
   - 测试是为了发现程序中的错误而执行程序的过程
   - 好的测试方案是尽可能发现迄今为止尚未发现的错误的测试方案
   - 成功的测试是发现了至今为止尚未发现的错误。

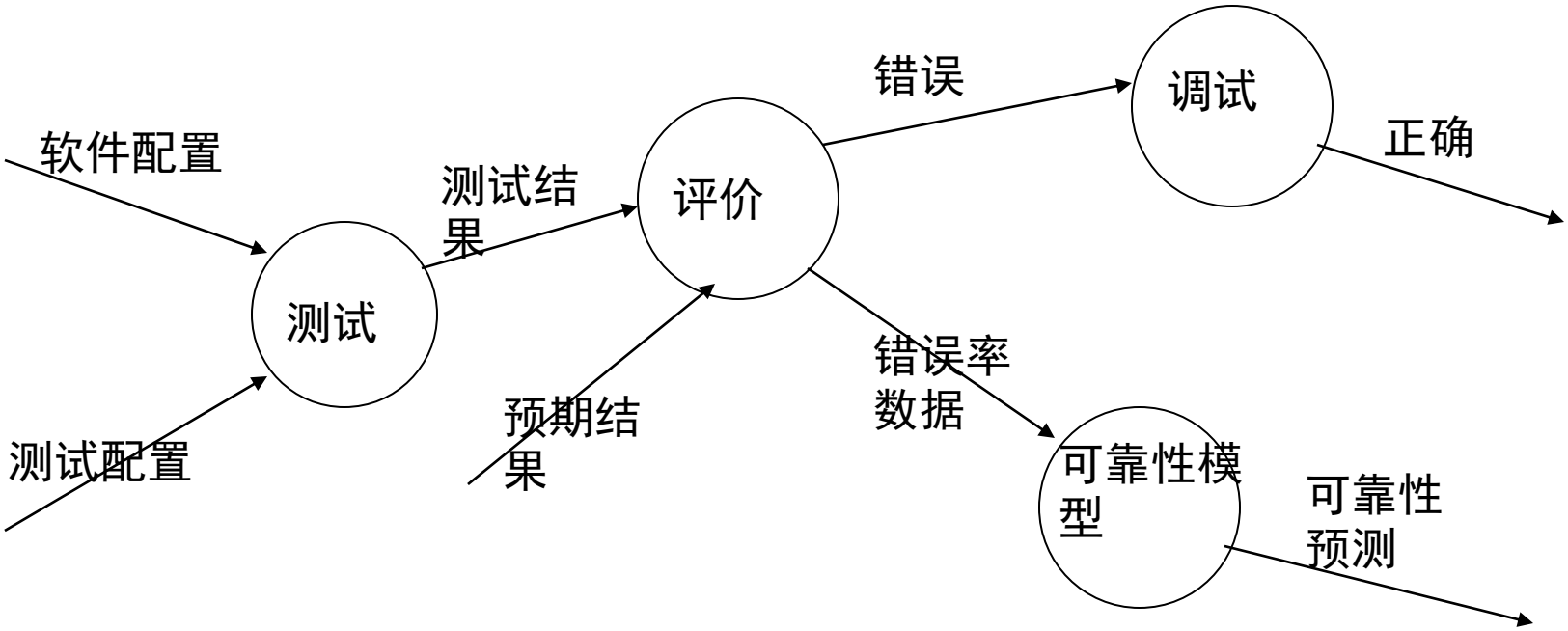
测试-基本概念

测试的步骤

- 模块测试
- 子系统测试
- 系统测试
- 验收测试
- 平行测试

测试-基本概念

测试阶段的信息流



测试-模块测试

模块测试-单元测试

1. 模块测试内容
   - 模块接口
   - 局部数据结构
   - 重要的执行通路
   - 出错处理通路
   - 影响上述各方面特性的边界条件

测试-模块测试

(1)模块接口测试要点

A. 参数数目和由调用模块送来的变元的数目是否相等?

B. 参数的属性和变元的属性是否匹配?

C. 参数和变元的单位系统是否匹配?

D. 传送给被调用模块的变元的数目是否等于那个模块的参数的数目?

E. 传送给被调用模块的变元属性和参数的属性是否一致?

F. 传送给被调用模块的变元的单位系统和该模块参数的单位系统是否一致?

G. 传送给内部函数的变元属性、数目和次序是否正确?

H. 是否修改了只做输入用的变元。

I. 全程变量的定义和用法在各个模块中是否一致?

测试-模块测试

(2)输入/输出的测试要点

A. 文件属性是否正确?

B. 打开文件语句是否正确?

C. 格式说明书与输入/输出语句是否一致?

D. 缓冲区大小与记录长度是否匹配?

E. 使用文件之前先打开文件了吗?

F. 文件结束条件处理了吗?

G. 输入/输出错误检查并处理了吗?

H. 输出信息中由文字书写错误吗?

测试-模块测试

(3)局部数据结构的测试要点

A. 错误的或不相容的说明

B. 使用尚未赋值或尚未初始化的变量

C. 错误的初始值或不正确的缺省值

D. 错误的变量名字（拼写错或截短了）

E. 数据类型不相容

F. 上溢、下溢或地址异常

测试-模块测试

(4)计算中的常见错误

A.计算次序不对或误解了运算符的优先次序

B.混合运算（运算对象的类型彼此不相容）

C.变量初始值不正确

D.精度不够

E.表达式的符号表示错误

测试-模块测试

(5)测试方案中的错误

A.比较数据类型不同的量

B.逻辑运算符不正确或优先次序的错误

C.当由于精度问题两个量不会相等时，程序中却期待着相等条件的出现

D.“差1”错（即，多循环一次或少循环一次）

E.错误的或不存在的循环终止条件

F.当遇到发散的迭代时不能终止循环

G.错误地修改循环变量

测试-模块测试

(6)评价出错处理时的常见错误

A. 对错误的描述是难于理解的

B. 记下的错误与实际遇到的错误不同

C. 在对错误进行处理之前，错误条件已经引起系统干预。

D. 对错误的处理不正确

E. 描述错误的信息不足以帮助确定造成错误的位置。

测试-模块测试

2. 测试过程

(1) 代码审查

- 人工测试程序可以由编写者本人非正式地进行，也可以由审查小组正式进行。
- 审查小组最好由四人组成：
  - 组长：有能力的程序员、没有直接参与这项工程。
  - 程序的设计者
  - 程序的编写者
  - 程序的测试者

测试-模块测试

(2) 测试软件

- 为每个模块开发测试驱动软件，它好比一个“主程序，它接收测试数据，把这些数据传送给被测试的模块，并且打印出有关的结果。

测试-集成测试

三、集成测试

- 集成测试主要有两种方法：非渐增式测试方法、渐增式测试方法。
- 非渐增式测试方法
  - 先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序。
- 渐增式测试方法
  - 把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。

测试-集成测试

- 两种测试方法的比较：
  1. 非渐增式测试方法需要编写的软件较多，工作量较大；渐增式测试方法开销小。
  2. 渐增式测试方法发现模块间接口错误早；而非渐增式测试方法晚。
  3. 非渐增式测试方法发现错误，较难诊断；而使用渐增式测试方法，如果发生错误则往往和最近加进来的那个模块有关。
  4. 渐增式测试方法测试更彻底
  5. 渐增式测试方法需要较多的机器时间
  6. 使用非渐增式测试方法，可以并行测试。

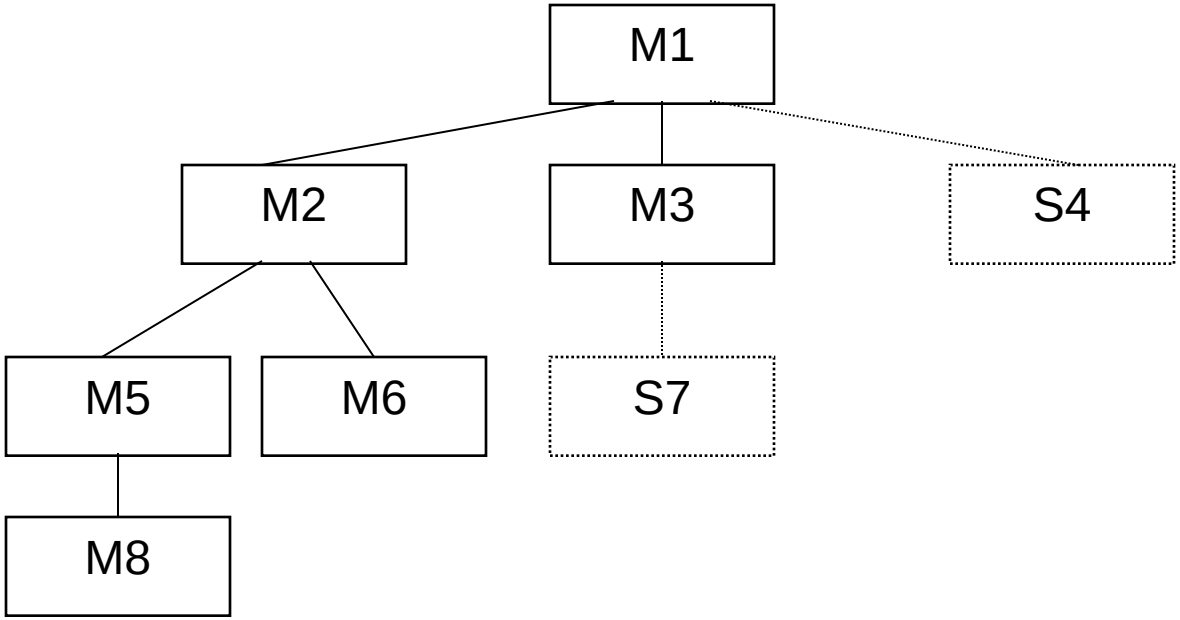
测试-集成测试

- 在实际测试中，应该将两种方法有机集合起来。
- 当使用渐增式测试方法时，具体有自顶向下和自底向上两种方法。

测试-集成测试

1. 自顶向下法
   - 从主控模块（“主程序”）开始，沿着软件的控制层次向下移动，从而逐渐把各个模块结合起来。
   - 在组装过程中，可以使用深度优先的策略，或宽度优先的策略。

测试-集成测试



深度优先：M1->M2->M5->M8->M6->M3->S7->S4

宽度优先：M1->M2->M3->S4->M5->M6->S7->M8

测试-集成测试

- 步骤：
  (1) 对主控模块进行测试，测试时用存根程序代替所有直接附属于主控模块的模块。
  (2) 根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块代替一个存根程序（新结合进来的模块往往又需要新的存根程序）
  (3) 在结合下一个模块的同时进行测试
  (4) 为了保证加入模块没有引进新的错误，可能需要进行回归测试（即，全部或部分地重复以前做过的测试）。
  - 从第2步开始不断地重复进行上述过程，直至完成。

测试-集成测试

2. 自底向上法

- 自底向上测试从“原子”模块（即在软件结构最低层的模块）开始组装和测试，具体策略是：

(1) 把低层模块组合成实现某个特定的软件子功能的族。

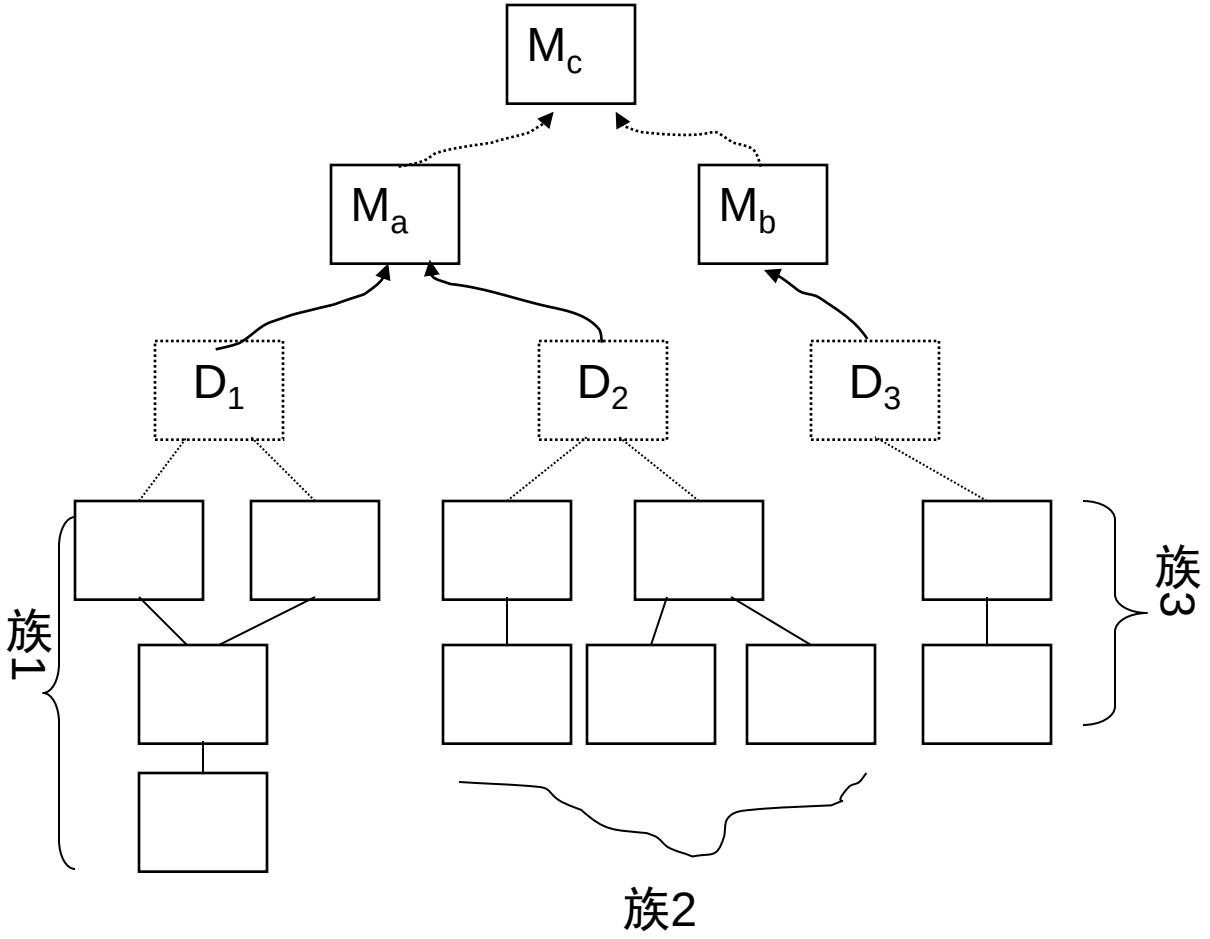
(2) 写一个驱动程序（用于测试的控制程序），协调测试数据的输入和输出。

(3) 对由模块组成的子功能族进行测试。

(4) 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族。

- 循环(2)－(4)步

测试-集成测试



测试-集成测试

3. 两种策略的比较

- “自顶向下”法的主要优点：不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。
- “自顶向下”法的主要缺点：需要存根程序，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力。
- “自底向上”法的优缺点与“自顶向下”法刚好相反。

测试-集成测试

4. 混合策略

- 在具体测试中，采用混合策略：

(1) 改进的“自顶向下”法：基本使用“自顶向下”法，但在测试早期，使用“自底向上”法测试少数的关键模块。

(2) 混合法：对软件结构中较上层，使用的是“自顶向下”法；对软件结构中较下层，使用的是“自底向上”法，两者相结合。

测试-验收测试

四、验收测试

1. 验收测试目的

- 验收测试的目的是向未来的用户表明系统能够象预定要求那样工作。

2. 验收测试的范围

- 验收测试的范围与系统测试的范围类似，但有一些差别。
- 验收测试必须有用户参与，甚至以用户为主。
- 验收测试一般使用黑盒测试法。

测试-验收测试

3. 验收测试的结果

(1) 功能和性能与用户要求一致，软件是可以接收的。

(2) 功能或性能与用户的要求有差距。

- 在这个阶段发现问题往往与“需求分析”阶段的差错有关，涉及的面比较广，难以解决。

4. 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，文档与程序一致。
- 除合同规定的内容和要求外，一般严格遵循用户指南以及其它操作程序。

测试-测试方案

五、设计测试方案

- 设计测试方案是测试阶段的关键技术问题。
- 测试方案包括要测试的功能，应该输入的测试数据和预期的结果。
- 设计技术主要有两种方法：黑盒法与白盒法。一般用黑盒法设计基本的测试方案，再用白盒法补充一些方案。

测试-测试方案

1. 黑盒法
    - 测试人员将程序看成是一个“黑盒”，即不关心程序内部是什么，只要检查程序是否符合它的“功能说明”。黑盒法可分为下列几种方法（关键在于确定测试数据）：

测试-测试方案

(1)等价分类法

- 等价分类法是将输入数据的可能值分成若干“等价类”，每一类以一个代表性的测试数据进行测试，这个数据就等价于这一类中的其它数据。
- 该法的关键在于如何将输入数据分类。
- 例如：输入的数据范围是1~999，我们可以划分三类：x<1，1<=x<999，x>=999

测试-测试方案

(2)边缘值分析法

- 用边缘特殊值测试。
- 经验表明：程序往往在边缘情况时犯错误，故测试边缘情况比较有效。
- 例如：输入数据的值的范围是：-1.0至1.0，则可选-1.0，1.0，-1.001，1.001等数据作为测试数据。

测试-测试方案

(3) 因果图法

- 等价类法与边缘值分析法的缺点是没有检查各种输入条件的组合。
- 因果图法则着重分析输入条件的各种组合，每种组合条件就是“因”，它必然有一个输出的结果，这就是“果”。

测试-测试方案

(4) 错误推测法

- 通过经验或直觉推测程序中可能存在的各种错误，从而有针对性设计测试用例。

测试-测试方案

2. 白盒法

- 白盒法需要了解程序的功能与结构，测试用例必须根据程序内部的逻辑来设计。如果想用白盒法发现程序中的所有错误，则至少必须使程序中每种可能的路径都执行一次。
- “彻底地测试”是不可能的，故策略是：在一定的研制时间、研制经费的限制下，通过执行有限的测试用例，尽可能多地发现一些错误。
- 白盒法又称为逻辑覆盖法，目前常用的覆盖法有：

测试-测试方案

(1) 语句覆盖

- 即每个语句至少能执行一次

测试-测试方案

(2)判定覆盖

- 判定覆盖又叫分支覆盖，含义是：每个判定的分支至少执行一次。

测试-测试方案

(3) 条件覆盖

- 即一个判断语句中往往包含了若干条件。通过给出测试用例，使判断中的每个条件都获得各种可能的结果。

测试-测试方案

(4)判断/条件覆盖

- 选取足够多的测试数据，使判断中每个条件都取得各种可能值，并使每个判断表达式也取到各种可能的结果。

测试-测试方案

⑸ 条件组合覆盖

- 使得每个判断中条件的各种可能组合都至少出现一次。

测试-测试方案

3. 实用测试策略

(1) 在任何情况下都使用边界值分析的方法。

(2) 必要时用等价划分法补充测试方案。

(3) 必要时再用错误推测法补充测试方案。

(4) 对照程序逻辑，检查已经设计出的测试方案。可以根据对程序可靠性的要求采用不同的逻辑覆盖标准，如果现有测试方案的逻辑覆盖程度没达到要求的覆盖标准，则应再补充一些测试方案。

- 注意：即使采用综合策略设计方案，仍不能保证测试将发现一切程序错误。

测试-调试

六、调试

1. 基本任务
   - 诊断和改正程序中的错误
2. 调试技术

(1) 输出存储器内容

- 通常以八进制或十六进制的形式输出存储器的内容。

(2) 打印语句

- 打印关键输出变量的值

(3) 自动工具

- 利用程序设计语言的调试功能或者使用专门的软件工具分析程序的动态行为。

测试-调试

3. 调试策略

(1) 试探法

- 调试人员分析错误征兆，猜想故障的大致位置，然后使用前述的一两种调试技术，获取程序中被怀疑的地方附近的信息。
- 该策略缓慢而低效。

测试-调试

(2) 回溯法

- 确定最先发现“症状”的地方，然后人工沿程序的控制流往回追踪源程序代码，直到找出错误根源或确定故障范围为止。
- 回溯法的另一种形式是正向跟踪，也就是使用输出语句检查一系列中间结果，以确定最先出现错误的地方。
- 回溯法适用于小程序。

测试-调试

(3) 对分查找法

- 如果已经知道每个变量在程序内若干个关键点的正确值，则可以用赋值语句或输入语句在程序中间点附近“注入”这些变量的正确值，然后检查程序的输出。如果输出结果是正确的，则故障在程序的前半部分；反之，故障在程序的后半部分。对于程序中有故障的那部分再重复使用这个方法，直到把故障范围缩小到容易诊断的程度为止。

测试-调试

(4)归纳法

- 所谓归纳法就是从个别推断一般的方法。从线索（错误征兆）出发，通过分析这些线索之间的关系而找出故障，具体步骤如下：

A. 收集有关的数据

B. 组织数据

C. 导出假设

D. 证明假设

测试-调试

(5)演绎法

- 演绎法从一般原理或前提出发，经过删除和精化的过程推导出结论。
- 用演绎法调试开始时先列出所有看来可能成立的原因或假设，然后一个一个地排除列举出的原因，最后证明剩下的原因确实是错误的根源。具体步骤如下：

A. 设想可能的原因

B. 用已有的数据排除不正确的假设

C. 精化余下的假设

D. 证明余下的假设

相互拥抱着才好飞翔。
部门间都是只有一只翅膀的天使，

祝大家：产品越来越可靠，设计更上一层楼！