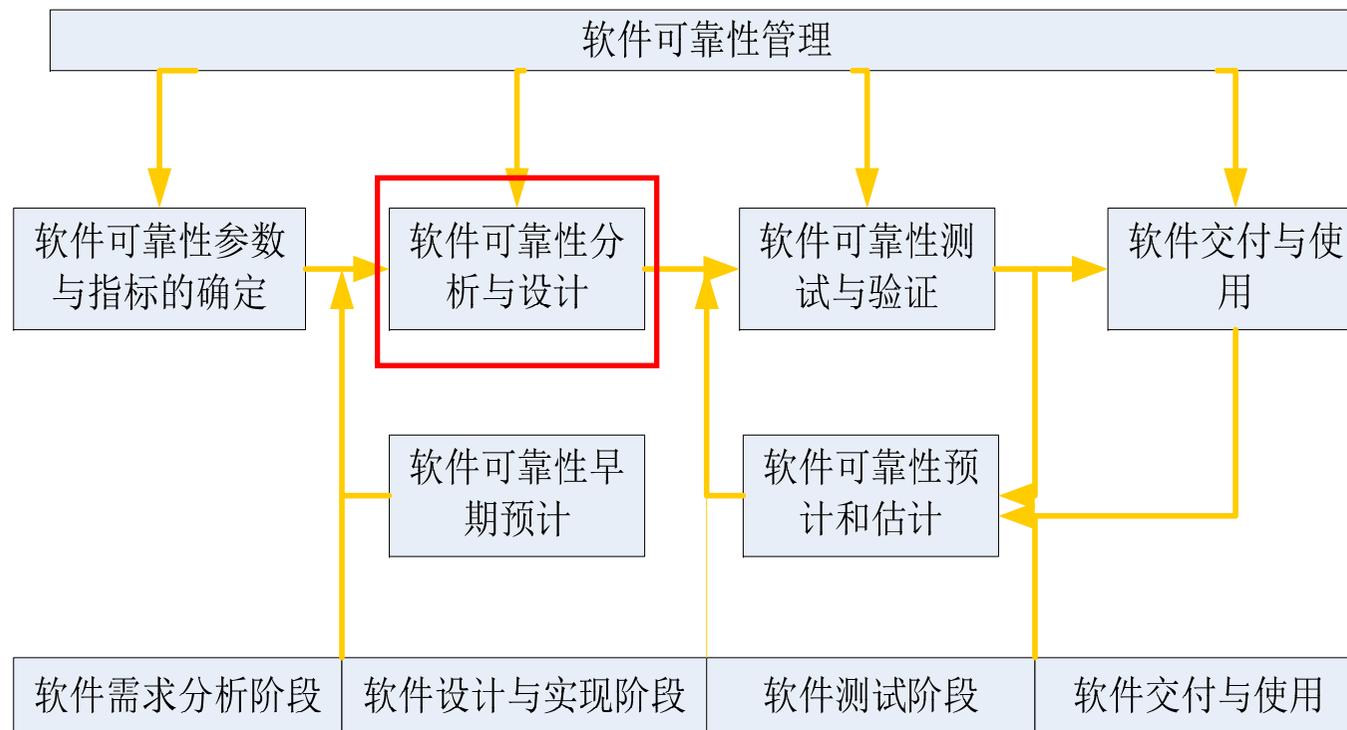
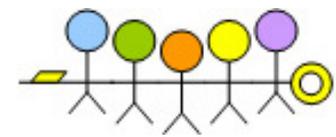


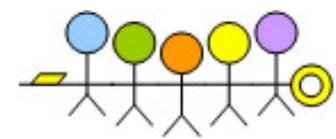
软件可靠性分析与设计





软件可靠性分析与设计的原因

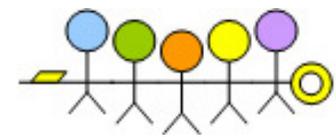
- 软件在使用中发生失效（不可靠）会导致任务的失败，甚至导致灾难性的后果。因此，应在软件设计过程中，对可能发生的失效进行分析，采取必要的措施避免将引起失效的缺陷引入软件，为失效纠正措施的制定提供依据，同时为避免类似问题的发生提供借鉴。
- 这些工作将会大大提高使用中软件的可靠性，减少由于软件失效带来的各种损失。



Myers设计原则

Myers专家提出了在可靠性设计中必须遵循的两个原则：

- 控制程序的复杂程度
 - 使系统中的各个模块具有最大的独立性
 - 使程序具有合理的层次结构
 - 当模块或单元之间的相互作用无法避免时，务必使其联系尽量简单，以防止在模块和单元之间产生未知的边际效应
- 是与用户保持紧密联系

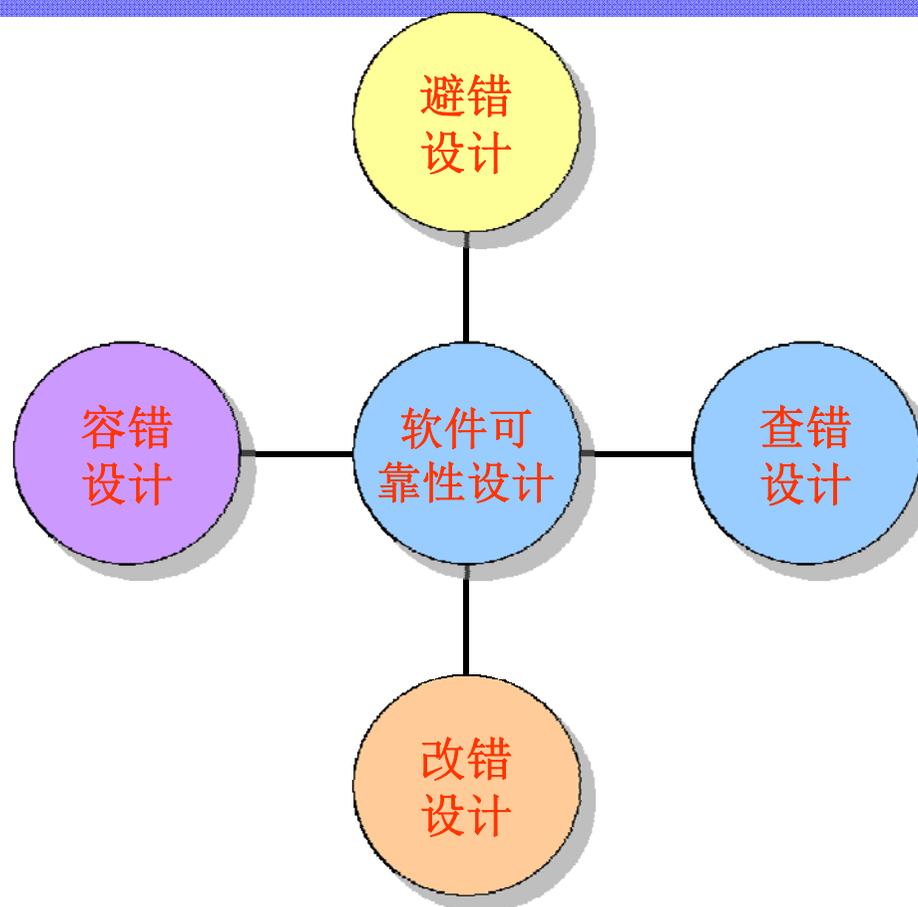


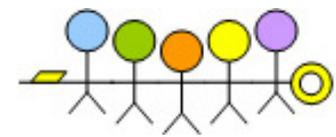
软件可靠性设计

- 软件可靠性设计的实质是在常规的软件设计中，应用各种必须的方法和技术，使程序设计在兼顾用户的各种需求时，全面满足软件的可靠性要求。
- 软件的可靠性设计应和软件的常规设计**紧密地结合，贯穿于常规设计过程的始终。**
- 这里所指的设计是**广义的设计**，它包括了从需求分析开始，直至实现的全过程。



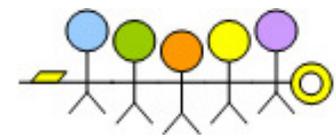
软件可靠性设计的四种类型





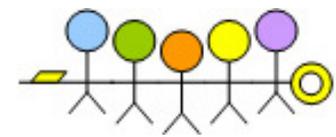
软件避错设计

- 避错设计是使软件产品在设计过程中，不发生错误或少发生错误的一种设计方法。的设计原则是控制和减少程序的复杂性。
- 体现了以预防为主的思想，**软件可靠性设计的首要方法**
- **各个阶段都要进行避错**
- **从开发方法、工具等多处着手**
 - **避免需求错误**
 - 深入研究用户的需求（用户声明的和未声明的）
 - 用户早期介入，如采用原型技术
 - **选择好的开发方法**
 - 结构化方法：包括分析、设计、实现
 - 面向对象的方法：包括分析、设计、实现
 - 基于部件的开发方法（COMPONENT BASED）



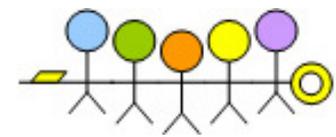
软件避错设计准则

- (1)模块化与模块独立
 - 假设函数 $C(X)$ 定义了问题 X 的复杂性，函数 $E(X)$ 定义了求解问题 X 需要花费的工作量（按时间计），对于问题 $P1$ 和问题 $P2$ ，如果 $C(P1) > C(P2)$ ，则有 $E(P1) > E(P2)$ 。
 - 人类求解问题的实践同时又揭示了另一个有趣的性质： $(P1 + P2) > C(P1) + C(P2)$
 - 由上面三个式子可得： $E(P1 + P2) > E(P1) + E(P2)$
- 这个结论导致所谓的“分治法”----将一个复杂问题分割成若干个可管理的小问题后更易于求解，模块化正是以此为据。
- 模块的独立程序可以由两个定性标准度量，这两个标准分别称为内聚和耦合。耦合衡量不同模块彼此间互相依赖的紧密程度。内聚衡量一个模块内部各个元素彼此结合的紧密程度。



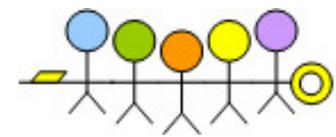
软件避错设计准则

- (2)抽象和逐步求精
 - 抽象是抽出事物的本质特性而暂时不考虑它们的细节
- 举例
 - 抽象 I 该CAD软件系统配有能与绘图员进行可视化通信的图形界面，能用鼠标代替绘图工具画各种直线和曲线；能完成所有几何计算以及所有截面视图和辅助视图的设计。
 - 抽象 II CAD软件任务；
 - 用户界面子任务；
 - 创建二维图形子任务；
 - 管理图形文件子任务；
 - **END CAD**
 - 抽象 III.....
- 软件工程过程的每一步都是对软件解法的抽象层次的一次精化



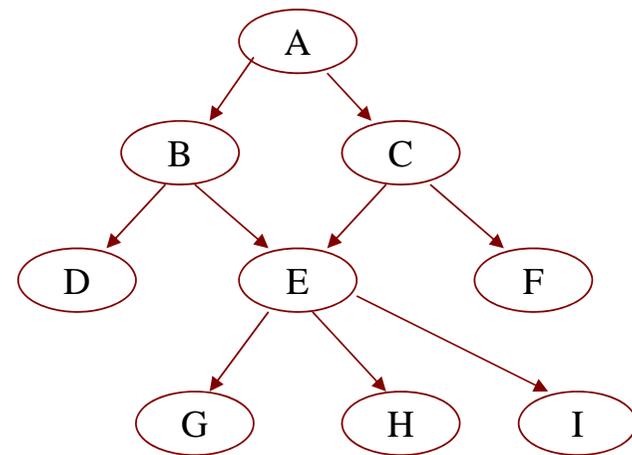
软件避错设计准则

- (3)信息隐蔽和局部化
 - 信息隐藏原理指出：应该这样设计和确定模块，使得一个模块内包含的信息对于不需要这些信息的模块来说，是不能访问的。“**只有需要才能知道**”
 - 如果绝大多数数据和过程对于软件的其他部分而言是隐蔽的，那么在修改期间由于疏忽而引入的错误就很少可能传播到软件的其他部分
 - 局部化是指把一些关系密切的软件元素物理地放得彼此靠近
 - **局部变量**



启发规则

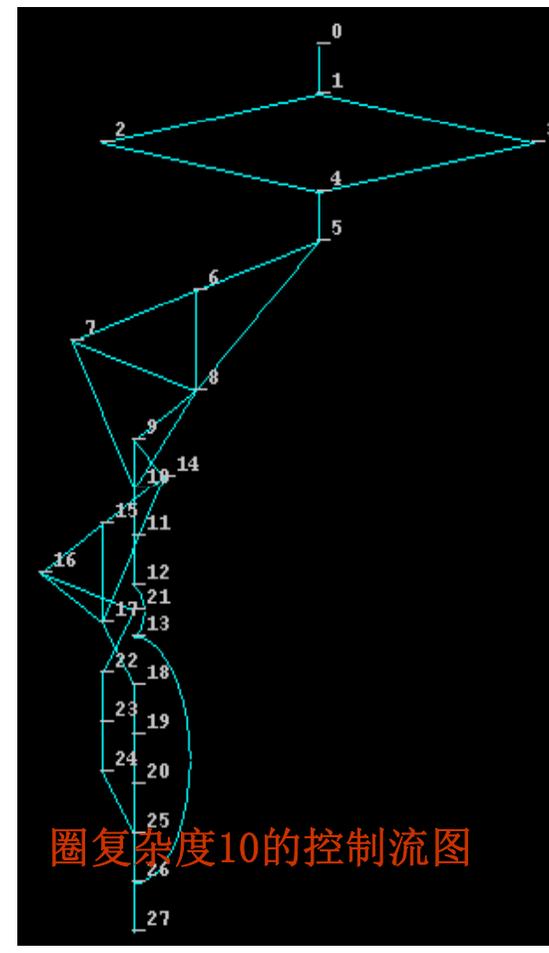
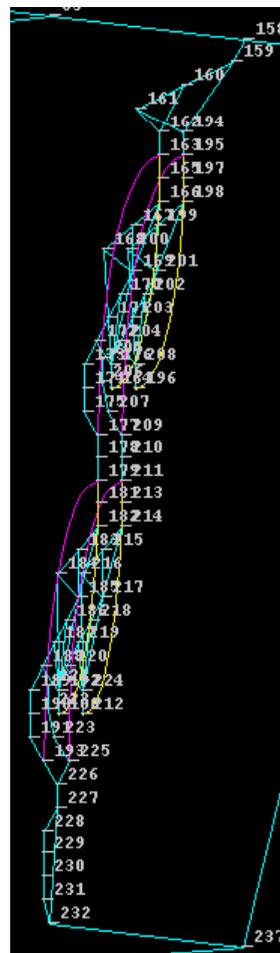
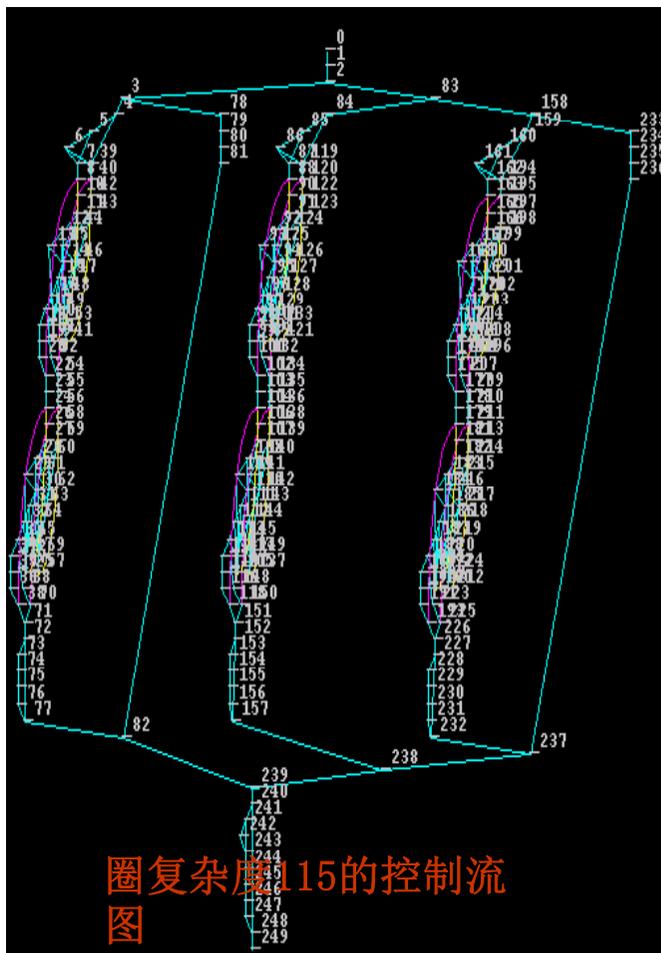
- 改进软件结构提高模块独立性
- 模块规模应该适中
- 深度、宽度、扇出和扇入都应适当
 - 深度表示软件结构中控制的层数，它往往能粗略地标志一个系统的大小和复杂程度。
 - 宽度是软件结构内同一层次上的模块总数的最大值。
 - 扇入是指有多少个上级模块直接调用它，扇入越大则共享该模块的上级模块数目越多，这是有好处的。
 - 扇出是一个模块直接调用的模块数目，扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块。



其中E函数扇入数为2，扇出数为3。



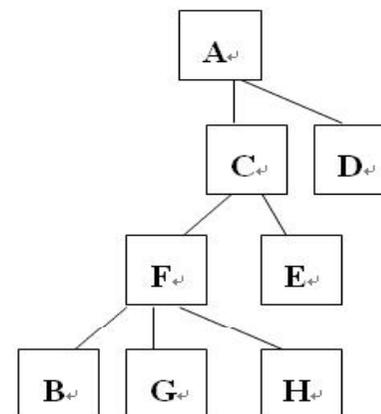
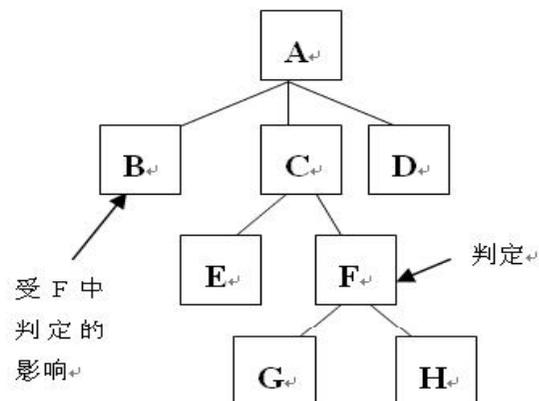
启发规则





启发规则

- 模块的作用域应该在控制域之内



- 力争降低模块接口的复杂程度
 - $QUAD-ROOT(TBL, X) \Rightarrow QUAD-ROOT(A, B, C, ROOT1, ROOT2)$
- 设计单入口单出口的模块
- 模块功能应该可以预测



软件避错设计

- 慎重使用容易引入缺陷的结构和技术
 - 浮点数
 - 指针
 - 动态内存分配
 - 并行
 - 递归
 - 中断
 - 继承
 - 别名
 - 默认输入的处理

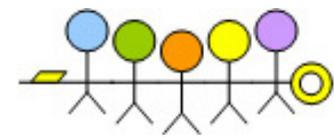


GJB/Z 102-97软件可靠性和安全性设计准则

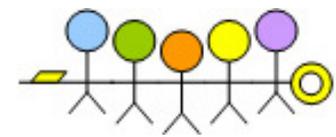
——5.12 防错程序设计

5.12 防错程序设计

- 参数化
- 公用数据和公共变量
- 标志
- 文件
- 非授权存取的限制
- 无意指令跳转的处理
- 程序检测点的设置
- 寻址模式的选用
- 数据区隔离
- 安全关键信息的要求
- 信息存储要求
- 算法选择要求

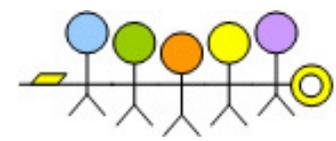


软件查错设计



软件查错设计

- 软件查错设计是指在设计中赋予程序某些特殊的功能，使程序在运行中自动查找存在错误的一种设计方法。
- **被动式错误检测**
 - 在程序的若干部位设置检测点，等待错误征兆的出现
- **主动式错误检测**
 - 对程序状态主动进行检查



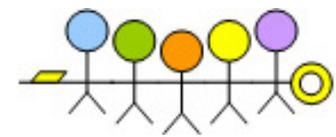
被动式错误检测

- 检测原则

- 相互怀疑原则：在设计任何一个单元、模块时，假设其它单元、模块存在着错误；
- 立即检测原则：当错误征兆出现后，要尽快查明，以限制错误的损害并降低排错的难度。

- 负效应

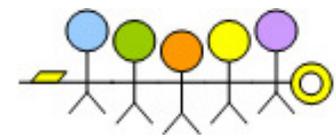
- 所设置的“接收判据”不可能与预期的正确结果完全吻合，导致错判或漏判；
- 软件增加了冗余可能降低可靠性



被动式错误检测的实施方法

- 看门狗定时器
 - 当出现潜在不安全的系统状态或有可能转移到这种状态时，将系统转移到规定的安全状态。
- 循环等待次数控制
- 配合硬件进行处理的设计
 - 如：电源失效、电磁干扰、系统不稳定、接口故障、干扰信号，以及错误操作等。
- 按照已知的数据极限检查数据；
- 按照变量间恒定关系检验；
- 检查所有多值数据的有效性；
- 对冗余的输入数据进行一致性检验；
-





看门狗的设计

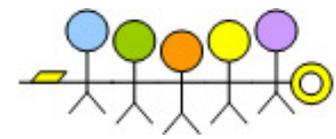
看门狗技术是控制运行时间的一种有效方法。看门狗实际上是一种计时装置，当计时启动后看门狗在累计时间，当累计时间到了规定值时触发到时中断（即狗叫），看门狗在不需要时可以关闭。看门狗的设计要首先明确其目的性。如：

(1)要防某段程序可能的死循环，则在此段程序前启动狗，在此段程序后关闭狗，在狗叫中断中进行超时异常处理。

(2)要防外来的信息长时间不来，则在开始等外来信息时启动狗，在接收到外来信息时关闭狗，在狗叫中断中进行超时异常处理。

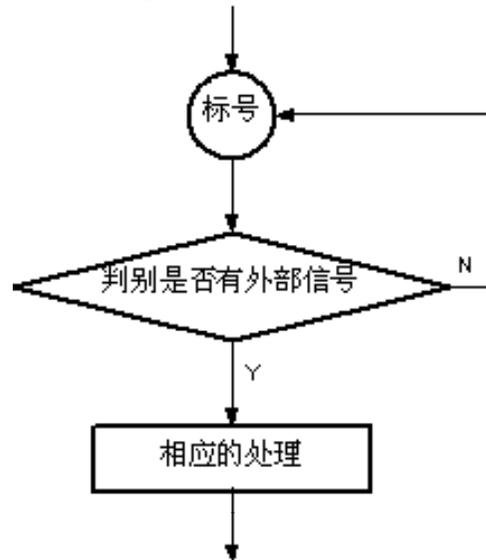
(3)要防计算超时，则在开始计算时启动狗，在计算完毕后关闭狗，在狗叫中断中进行超时异常处理。

显然，不可能要求一个狗可以看管好所有的超时情况。

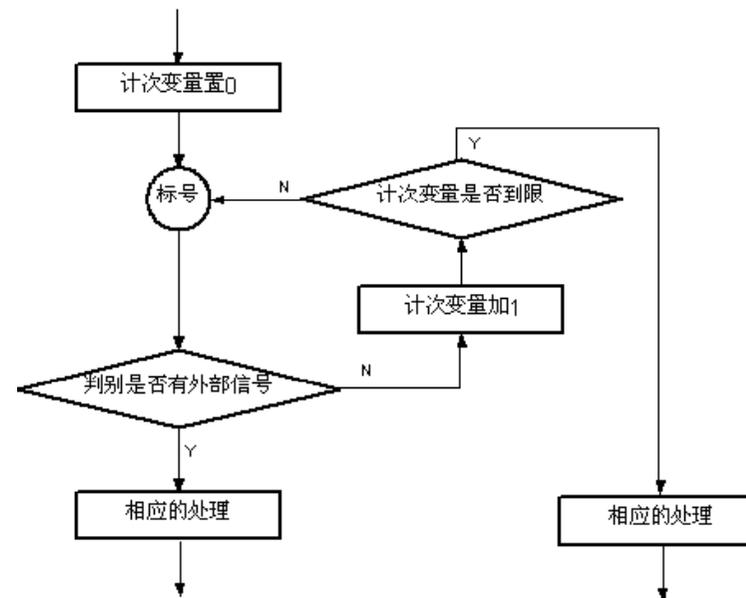


避免潜在的死循环

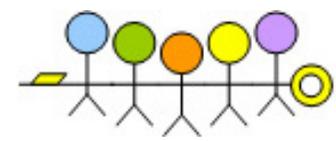
在等待外部信号的程序段中，不允许无限制地等待。正确的做法应是，或采用循环等待次数控制，或使用定时器，使得规定时间内（无论成功或失败）必须保证退出等待外部信号的程序段。



不允许的设计方法



建议采用的设计方法



注意通过双口RAM进行握手

通过双口RAM进行信息交换是设计师经常采用的一种设计方案。的确双口RAM提供了信息交换双方的方便读写，但仅靠双口RAM要做到读写的时序要求就要格外小心。

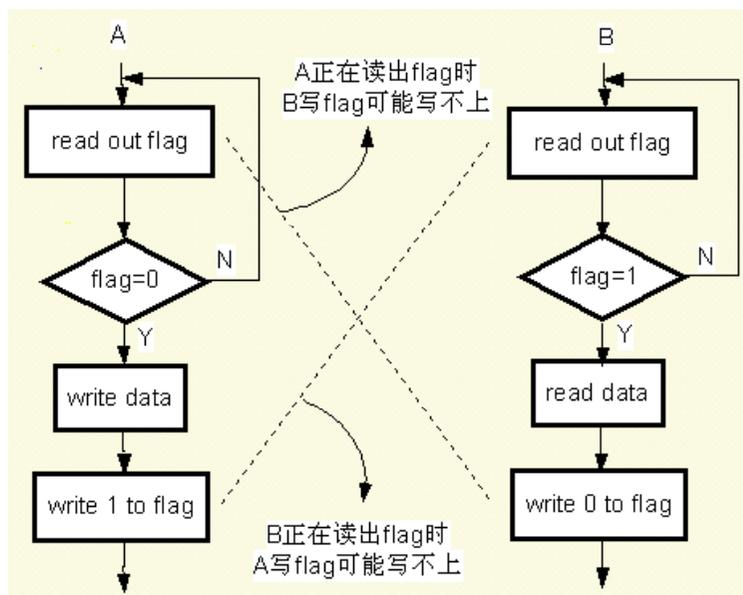
如此的设计是要避免的：通过双口RAM交换信息，在双口RAM中设置了握手信号单元。读方检查到握手信号为01H，表明对方已准备好数据，再读数据，读完后将握手信号置为00H；写方检查到握手信号为00H，表明对方已取走数据，再写数据，写完数据后再将握手信号置为01H，表明自己已准备好数据。

这种设计不一定可靠，可能会出现写方要写握手信号时，读方正读握手信号，则写方要写的值写不进去。可靠的设计应用硬件连线保证握手，而不要靠双口RAM中的握手信号。如果一定要靠双口RAM进行握手，则写握手信号单元数据时一定要写完后接着再读出，经验证确实写成功后再进行下面的操作，否则需继续写。

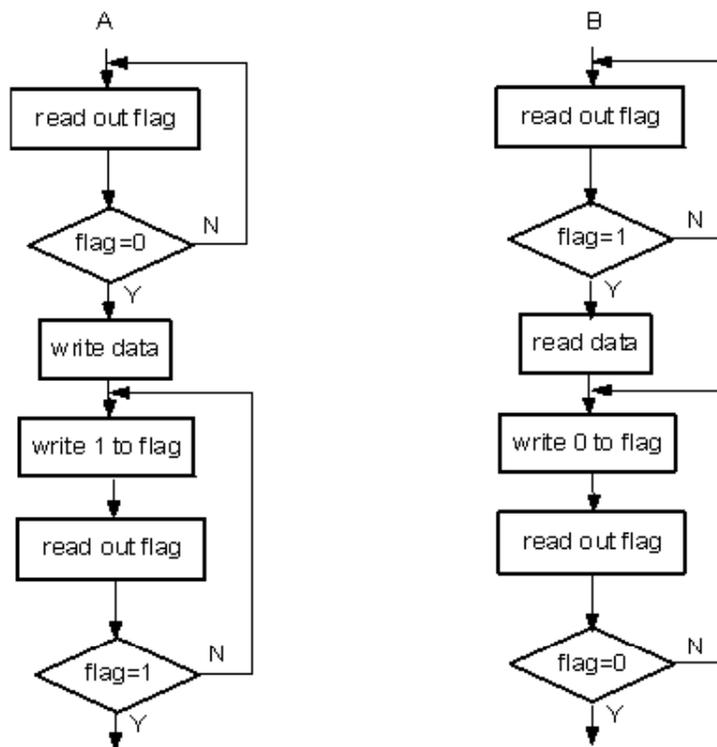
当然这必须与避免潜在的死循环的设计准则联合使用。



注意通过双口RAM进行握手



握手标志置不上的可能



可靠的设计方法



数据采集的多路冗余设计

关键数据的采集可采用多路冗余设计，即可以从多个通讯口对同一数据进行采集，通过表决进行有效数据的裁决。通常多采用奇数路的冗余设计，如3路、5路等。

(1) 开关量的裁决可采用多数票的裁决，如3取2、5取3等。

(2) 模拟量的裁决可采用中间数平均值的裁决，如3路数的中间值、5路数去掉最大最小值后的平均值等。



主动式错误检测

● 软件BIT

- 对ROM中的代码进行和检验 (sum check) ;
- 检测RAM, 以保证正确的操作用数码;
- 检测内存的正确性, 以确保正确操作;
- 对关键及重要的函数功能及逻辑功能进行典型较核。
-

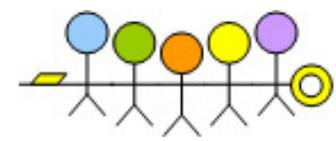
据统计, 将硬件看门狗定时器及软件BIT相结合, 一般可检测出98%的故障。

- 主动式错误检测可作为周期性任务来安排, 也可作为一个低优先级的任务来执行。

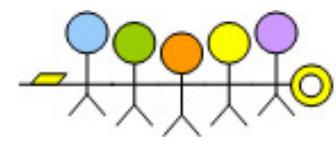


故障恢复措施

- 完全恢复（依靠冗余备份）
- 降级恢复（只提供重要的功能）
- 立即停止程序运行（安全停机）
- 记载错误
 - 将发生错误时的状态记录在一个外部文件上，然后让系统恢复运行，再由维护人员对记录进行深入的分析研究。

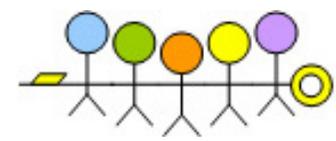


软件改错设计



软件改错设计

- **改错设计**是指在设计中，赋予程序自我改正错误、减少错误危害程度的一种设计方法。
- 改正错误的前提是已经准确地找出软件错误的起因和部位（故障检测与故障定位合称**故障诊断**），程序又有能力修改、剔除有错误的语句。
- 现阶段仅限于减少软件错误造成的有害影响，或将有害影响限制在一个较小的范围。常采用**故障隔离技术**。
- 现阶段没有人的参与几乎不可能



故障隔离

- **权限最小化原则**是实现故障隔离的主要思想。
- 为了限制故障的蔓延，要求对过程和数据加以严格的定义和限制。例如，针对操作系统的故障隔离方法：
 - 不允许一个用户的应用程序引用或修改其它用户的应用程序或数据；
 - 绝对不允许一个应用程序引用或修改操作系统的编码或操作系统内部的数据；
 - 保护应用程序及其数据，使其不致由于操作系统的错误而引起程序和数据的偶然变更；
 - 应用程序绝对不能终止系统工作、不能诱发操作系统去改变其它的应用程序或数据；
 -

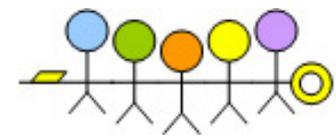


软件容错设计



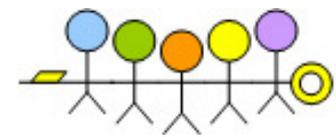
软件容错设计

- 软件容错设计是指在设计中赋予程序某种特殊的功能，使程序在错误已被触发的情况下，系统仍然具有正常运行能力的一种设计方法。
 - 时间容错
 - 结构容错
 - 信息容错



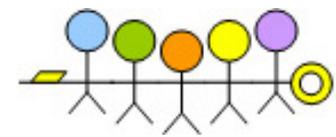
时间容错

- 所谓“时间容错”就是不惜以牺牲时间为代价来换取软件系统高可靠性的一种手段，它包括指令的重复执行和程序(一个模块或一个子程序)重复执行，两种常被采用而行之有效的办法
 - 指令重复执行是当应用软件系统检查出正在执行的指令出错误后，让当前指令重复执行 n 次($n \geq 3$)，若故障是瞬时的干扰，在指令重复执行时间内，故障有可能不再复现，这时程序就可以继续往前执行下去。这时指令执行时间比正常时大 n 倍
 - 程序(一个模块或一个子程序)重复执行是当应用软件系统检查出正在执行的程序出错误后，中断当前正在运行的软件，反复调用 n 次($n \geq 3$)运行出错的程序(一个模块或一个子程序)。在反复调用过程中若故障不再复现，这时程序继续从中断的地址往F执行。这时所需的运行时间比重复执行 n 条指令的时间要大得多
- 对于复执不成功的情况，通常的处理办法是发出中断，转入错误处理程序，或对程序进行复算，或重新组合系统，或放弃程序处理。



时间容错

```
For (i=0,i<3,i++)  
{  
    ...  
    x[i]=ReadData(...); //读入数据  
    ...  
}  
2From3(x); //三取二方法函数
```



结构容错

- 结构容错的基本思想来源于硬件可靠性中的冗余技术。

静态冗余 → *N*版本程序设计法 (NVP)

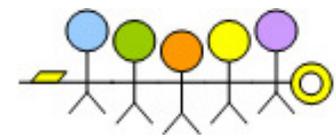
动态冗余 → 恢复块法 (RB)

Q: 使用多个完全相同的软件能够实现容错吗

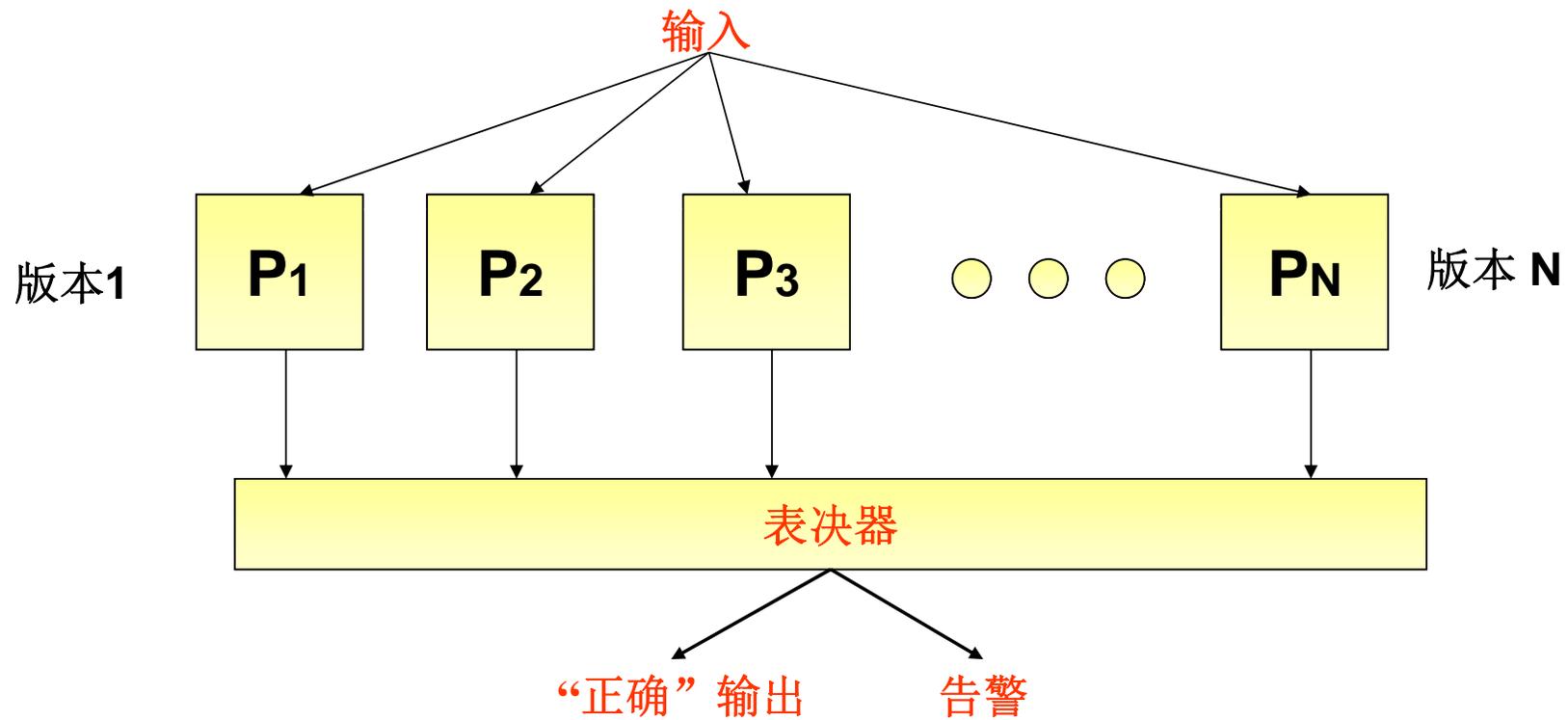


软件N版本程序设计

- 美加州大学Avizienis和L.Chen提出
- 基本思想
 - 对相同的要求进行多个不同版本的设计
 - 对相同的初始条件和相同的输入的操作结果实行表决（多数决定、一致决定）
- 针对的错误来源：设计人员对需求说明理解不正确、不完全导致的缺陷
- 要求需求说明完全、精确



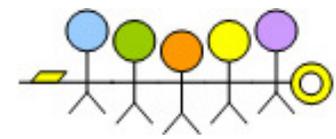
NVP的基本结构



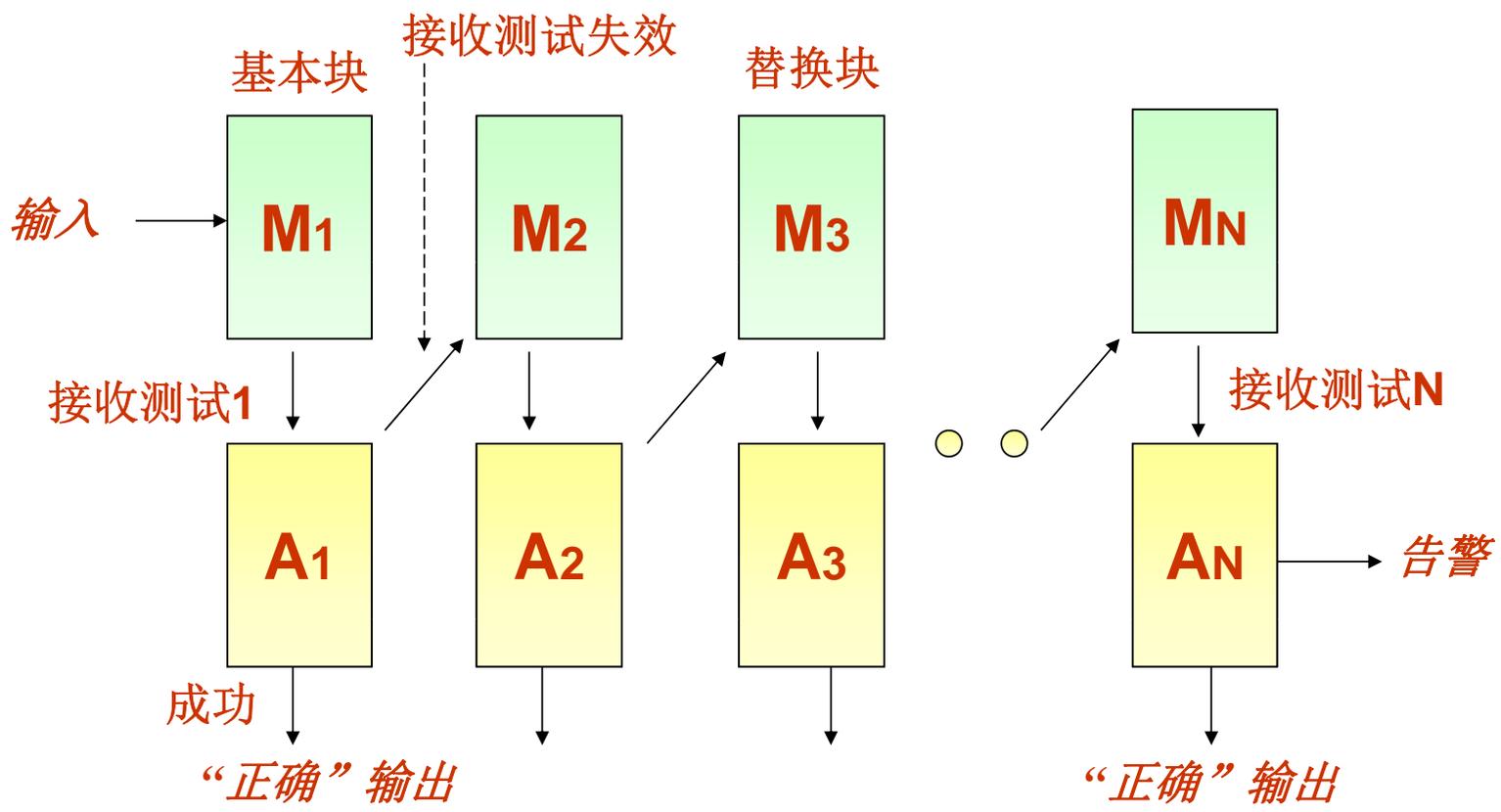


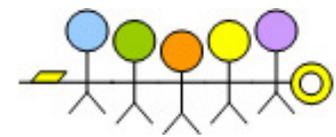
冗余软件的设计原则

- **相异性**是指对实现同一功能的多版本产品，在制作时为防止出现共因故障而有意识地使之产生某种差异的性质；
- 冗余版本之间相互屏蔽故障不能完全依靠背靠背编程（**随机相异**）；
- 应有意识地实行相异化编程（**强制相异**）：
 - 相互独立的不同人员
 - 不同的算法
 - 不同的编程语言
 - 不同的编译程序
 - 不同的设计工具、测试方法
 -



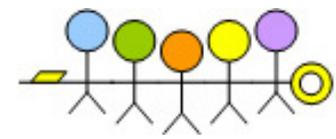
恢复块法





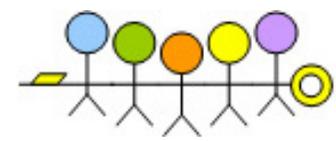
恢复块法

- 恢复块法适合只有一台计算机的情况；
- 允许只对较为复杂、容易出故障的程序段进行冗余；
- 基本块与替换块的设计应尽可能相异；
- 接收测试的作用非常重要，应能检测程序执行结果与预期结果的偏离，或检测和防止能触发安全事故的输出：
 - 需求检查法
 - 帐目检查法
 - 合理性检查法
 - 计算机运行时检查法



信息容错（检错及纠错）

- 一般非关键信息可以使用**奇偶校验码**。
- 关键、重要信息与其它信息之间应保持一定的**Hamming 距离**，不会因一位或两位差错而引起系统故障。
 - 例如不能用01表示一级点火，10表示二级点火，11表示三级点火。
- 如安全关键信息有差错，应能检测出来，并返回到规定的状态（例如安全状态）。
 - 如循环码
- 安全关键信息的决策判断不得依赖于全“1”或全“0”的输入（尤其是从外部传感器传来的信息）。



软件可靠性分析



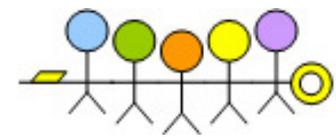
软件可靠性分析的作用

- 发现设计缺陷
- 指导软件设计
- 指导软件测试



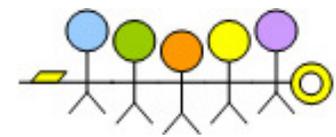
软件可靠性分析方法

- 软件失效模式和影响分析（**SFMEA**）
- 软件故障树分析（**SFTA**）
- 软件**Petri**网分析法
- 软件潜藏分析法(**SSCA**)



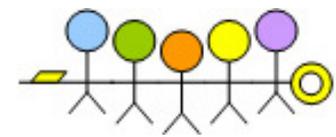
SFMEA概述

- 软件失效模式和影响分析（**SFMEA**）是在软件开发阶段的早期，通过识别软件失效模式，分析造成的后果，研究分析各种失效模式产生的原因，寻找消除和减少其有害后果的方法，以尽早发现潜在的问题，并采取相应的措施，从而提高软件的可靠性和安全性。



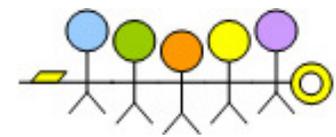
SFMEA的优缺点

- 优点：
 - 突出需要更改的部分
 - 分析底层故障如何在整个系统中逐级向上传播
 - 分析应在何处采取容错、故障检测和运行监控等设计手段
 - 识别将导致系统崩溃的单点故障
 - 可以按照对设备和人员影响的严重程度对软件失效严重性进行分级
- 局限性：
 - 不能识别人为错误造成的潜在失效
 - 作为一个工具尚未广泛的应用于对软件错误的评估
 - 用于分析路径十分复杂或相互交联的软件时，**SFMEA**是一项繁琐、劳动强度高的工作



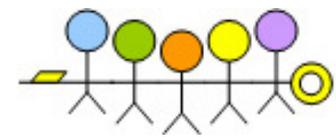
SFMEA的基本概念

- **软件失效**就是泛指程序在运行中丧失了全部或部分功能、出现偏离预期的正常状态的事件。软件失效是由软件的缺陷引起的。
- **软件失效模式**指软件失效的表现形式，即软件失效发生的方式，有时也被描述为对设备运行产生的影响。在IEEE软件异常分类标准中给出了软件失效模式的分类，在进行SFMEA分析时可作为参考。
- **软件失效影响**是指软件失效模式对软件系统的运行、功能或状态等造成的后果。



IEEE软件异常分类

软件异常类型	具体软件异常
操作系统失败	——
程序挂起	——
程序失败	程序不能启动 程序运行不能终止 程序不能退出
输出问题	错误的格式 不正确的结果、数据 不完全或遗漏 拼写问题、语法问题 美化问题
未达到性能要求	不能满足用户对运行时间的要求 不能满足用户对数据处理量的要求 多用户系统不能满足用户数的要求
整个产品失效	——
系统错误信息	——
其它	程序运行改变了系统配置参数 程序运行改变了其它程序的数据 其它



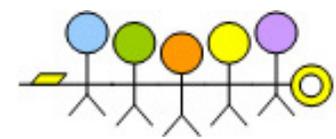
IEEE软件异常分类

- 软件错误(error): 在软件生存期内的不希望或不可接受的人为错误。
 - 其结果是导致软件缺陷的产生。
 - 软件错误是一种人为过程, 相对于软件本身, 是一种外部行为。
- 软件缺陷(defect): 是存在于软件(文档、数据、程序)之中的那些不希望或不可接受的偏差。
- 其结果是软件运行于某一特定条件时出现软件故障, 这时称软件缺陷被激活。
- 有时又称“software bug”。



IEEE软件异常分类

- 软件故障(fault): 是指软件运行过程中出现的一种不希望或不可接受的内部状态。
 - 此时若无适当措施(容错)加以及时处理,便产生软件失效。
 - 是一种动态行为。
- 软件失效(failure): 是指软件运行时产生的一种不希望或不可接受的外部行为结果。
- Is a **dynamic property** of software
- Users may choose several severity levels of failures:
- ◆ Catastrophic
 - ◆ Major
 - ◆ minor



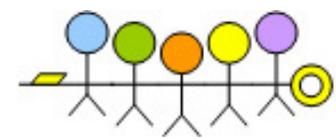
SFMEA分析的一般步骤

- **SFMEA**的分析对象可以是开发早期阶段的高层次的子系统、部件，也可以是详细设计阶段的单元、模块。
- 对于不同的分析对象，其软件失效模式是不同的，采用的**SFMEA**分析方法也不同，前者采用系统级分析方法（**system FMEA**），后者为详细级分析方法（**detailed FMEA**）。



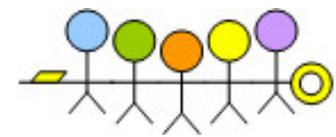
系统FMEA分析的一般步骤

- 1) 系统定义
- 2) 软件失效模式分析
- 3) 软件失效原因分析
- 4) 软件失效影响分析
- 5) 改进措施分析



1) 系统定义

- 在系统定义中应说明系统的主要功能和次要功能、用途、系统的约束条件和失效判据等。系统定义还应包括系统工作的各种模式的说明、系统的环境条件，以及软、硬件配置。
- 进行结构分解，就是将软件按照一定的分解原则划分为若干逻辑部分，以确定系统的分析级别，以及最小分析单元。
 - 结构分解的原则可基于软件系统的功能、结构特征或工作模式等，应根据软件的具体情况确定



1) 系统定义

- 2) 软件失效模式分析
 - 针对每个最小分析单元，确定其潜在的失效模式，例如软件无响应或者返回错误的值等。
- 3) 软件失效原因分析
 - 针对每个软件失效模式分析其所有可能的原因。软件的失效原因往往是软件开发过程中形成的各类缺陷所引起的。



软件失效原因

- 逻辑遗漏或执行错误
- 算法的编码错误
- 软硬件接口故障
- 数据操作错误
- 数据错误或丢失





具体失效原因——逻辑遗漏或执行错误

- 遗忘细节或步骤
- 逻辑重复
- 忽略极限条件
- 需求的错误表述
- 未进行条件测试
- 检查错误变量
- 循环错误

返回



具体失效原因——算法编码错误

- 等式不完整或不正确
- 丢失运算结果
- 操作数错误
- 操作错误
- 括号使用错误
- 精度损失
- 舍入和舍去错误
- 混合类型
- 标记习惯不正确

返回



具体失效原因——软硬件接口错误

- I/O时序错误
- 时序错误导致数据丢失
- 子函数调用不当
- 子函数调用位置错误
- 调用不存在的子函数
- 子函数不一致

返回



具体失效原因——数据操作错误

- 数据初始化错误
- 数据存取错误
- 标志或索引设置不当
- 数据打包解包错误
- 变量参考错误数据
- 数据越界
- 变量缩放比率或单位不正确
- 变量维度不正确
- 变量类型错误
- 变量下标错误
- 数据范围不对

返回



具体失效原因——数据错误或丢失

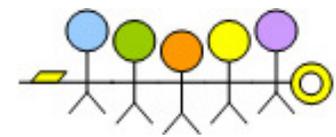
- 传感器数据错误或丢失
- 操作数据错误或丢失
- 嵌入到表中的数据错误或丢失
- 外部数据错误或丢失
- 输出数据错误或丢失
- 输入数据错误或丢失

返回



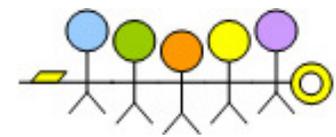
4) 软件失效影响分析

- 针对每个失效模式分析其造成的影响，一般可分为局部影响、上一层次影响和最终影响三级，并分析失效所造成影响的严重性。



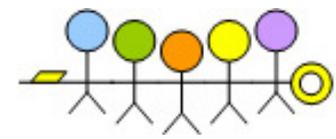
失效影响严重性分类

严重性类别	严重性定义
致命/非常严重 9-10	引起人员死亡，系统报废，或对周围环境造成灾难性破坏
严重 7-8	引起人员严重伤害，系统严重损坏，任务失败，或环境严重破坏
一般 4-6	引起人员轻度伤害，系统轻度损坏，导致任务延误或降级，环境受影响
轻微 1-3	不导致人员伤害，不影响任务完成，不影响环境，但使用的方便性或舒适性降低



5) 改进措施分析

- 根据每个失效模式的原因、影响及严重性等级，综合提出有针对性的改进措施。



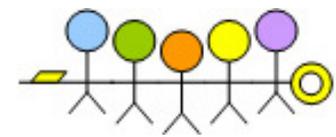
SFMEA工作表格

编号	单元	功能	失效模式	可能的失效原因	失效影响			严重性	改进措施
					单元	子系统	系统		
1.1	输出	输出数据提交用户显示	数值高于正常范围	逻辑问题 计算问题 数据操作问题	N/A	无	任务降级	一般	...
1.2			数值低于正常范围	逻辑问题 计算问题 数据操作问题	N/A	无	任务降级	一般	...
1.3			输出数据没有显示	逻辑问题 接口或时序问题	N/A	无	任务中止	严重	...
1.x			x	x	x	x	x	x	x



SFMEA实例 (1/5)

- 某舰载防御武器型号的转塔控制与伺服软件的并行通讯软件采取主从双机**8255**通讯（固定字节长度）的方式。其中控制计算机为主计算机（**A**），伺服计算机为从计算机（**B**）。主从双机之间的通讯协议如下：
 - 先由**A**向**B**发送**N**字节，待**B**接收完**N**字节后，**B**再向**A**发送**M**字节；



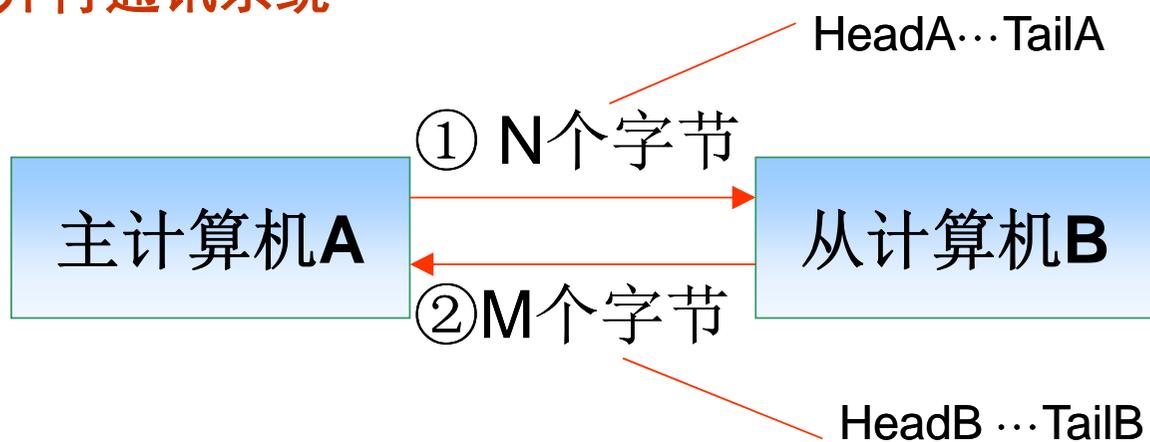
SFMEA实例 (2/5)

- A发送的第一字节为HeadA，HeadA为控制机命令标识，A发送的第N字节为TailA，TailA为控制机校验标识。B发送的第一字节为HeadB，HeadB为从属机接收状况回应标识，B发送的第M字节为TailB，TailB为从属机校验标识；
- A发送过程的容忍时间为TSA，A接收过程的容忍时间为TTA，B发送过程的容忍时间为TSB，B接收过程的容忍时间为TTB；
- A进入通讯子程序后，先发中断，B响应中断后进入通讯子程序准备接收A的数据；
- 采用8255，实现能判别是否对方已提供字节数据，能判别对方是否已取走字节数据。



SFMEA实例 (3/5)

某并行通讯系统



发送容忍时间TSA
接收容忍时间TTA

发送容忍时间TSB
接收容忍时间TTB



SFMEA实例 (4/5)

A的失效模式

通讯时间异常

- ✓无法按时进入通讯中断
- ✓规定的时间不能完成发送, 即 $TS > TSA$
- ✓规定的时间不能完成接收, 即 $TT > TTA$

通讯数据异常

- ✓HeadA错
- ✓TailA错
- ✓中间字节错



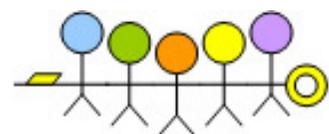
SFMEA实例 (5/5)

编号	失效模式	失效影响	严重性	措施
1	A由于其它中断或其它原因，无法按时进入通讯中断。	导致通讯无法进行。	非常严重	由A的通讯外程序采取措施保证按时进入通讯子程序。
2	A的通讯程序在规定的未能完成发送/接收。	通讯周期将乱，通讯混乱。	非常严重	在A中设置定时器，或通过计数控制，以保证规定时间内返回。
3	A发送头字节HeadA由于外干扰出现错误。	B执行错误的命令。	严重	B接收A的头字节后，判别头字节的合法性。
4
5	A发中断后，由于外干扰，B没能响应中断。	B无法进入通讯程序，因此无法接收A发送的数据。	严重	在A中设置定时器，当在允许的时间后仍无B的响应信息，放弃此帧通讯。



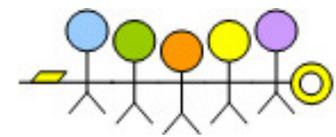
详细级SFMEA步骤

- 分析每个软件模块的输入输出变量和算法的失效模式，追踪每个失效模式对软件系统的影响直至输出，最后将最终的软件状态与预先定义的软件危险状态对比判断是否存在危险的软件失效。
 - 建立变量映射关系
 - 软件模块使用哪些变量、变量类型、输入变量来源
 - 建立软件线索
 - 变量经过一系列处理到系统输出变量之间的映射关系
 - 确定软件失效模式
 - 变量及算法的失效模式
 - 失效影响分析
 - 由模块追溯到系统影响



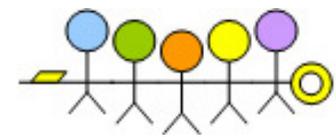
表格 6 软件 FMEA 表格

失效模式		关键变量			
		输出变量 1	输出变量 2	输出变量 3
输入变量 1	失效模式 1				
	失效模式 2				
				
输入变量 2	失效模式 1				
	失效模式 2				
				
输入变量 3	失效模式 1				
	失效模式 2				
				
.....					



软件故障树分析 (SFTA)

Software Fault Tree Analysis



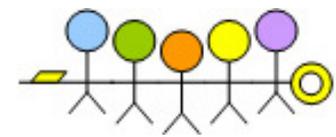
软件故障树分析 (SFTA)

- 软件故障树分析 (SFTA) 是一种自顶向下的软件可靠性分析方法，即从软件系统不希望发生的事件 (顶事件)，特别是对人员和设备的安全及可靠性产生重大影响的事件开始，向下逐步追查导致顶事件发生的原因，直至基本事件 (底事件)，从而确定软件故障原因的各种可能组合方式和 (或) 发生概率。



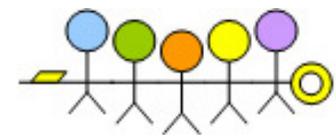
SFTA步骤

- 软件故障树的建立
- 软件故障树的定性分析
- 软件故障树的定量分析

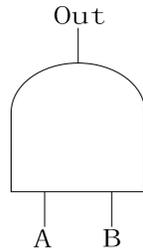


软件故障树的建立

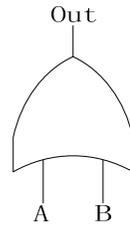
- 建立软件故障树通常采用演绎法：
 - 首先选择要分析的顶事件（即不希望发生的故障事件）作为故障树的“根”；
 - 然后分析导致顶事件发生的直接原因（包括所有事件或条件），并用适当的逻辑门与顶事件相连，作为故障树的“节”；
 - 按照这个方法逐步深入，一直追溯到导致顶事件发生的全部原因（底层的基本事件）为止。这些底层的基本事件称为底事件，构成故障树的“叶”。



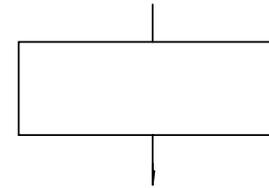
故障树符号(部分)



与门: 只有所有输入事件都发生时才能导致输出事件发生。



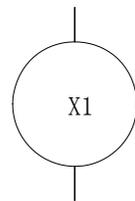
或门: 输入事件中至少有一个发生时, 就能导致输出事件发生。



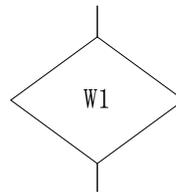
顶事件或中间事件: 表示需进一步分析的事件。

顶事件 是不希望发生的系统不可靠或不安全事件。

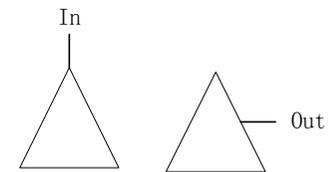
中间事件 是故障树中除底事件和顶事件之外的所有事件。



基本事件: 即底事件, 对于故障树中的基本事件不必作更深入的分析

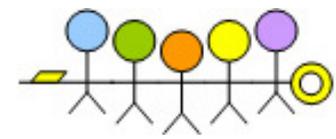


未展开事件: 由于发生概率较小, 或由于信息不足, 不能作更深层的分析的事件。

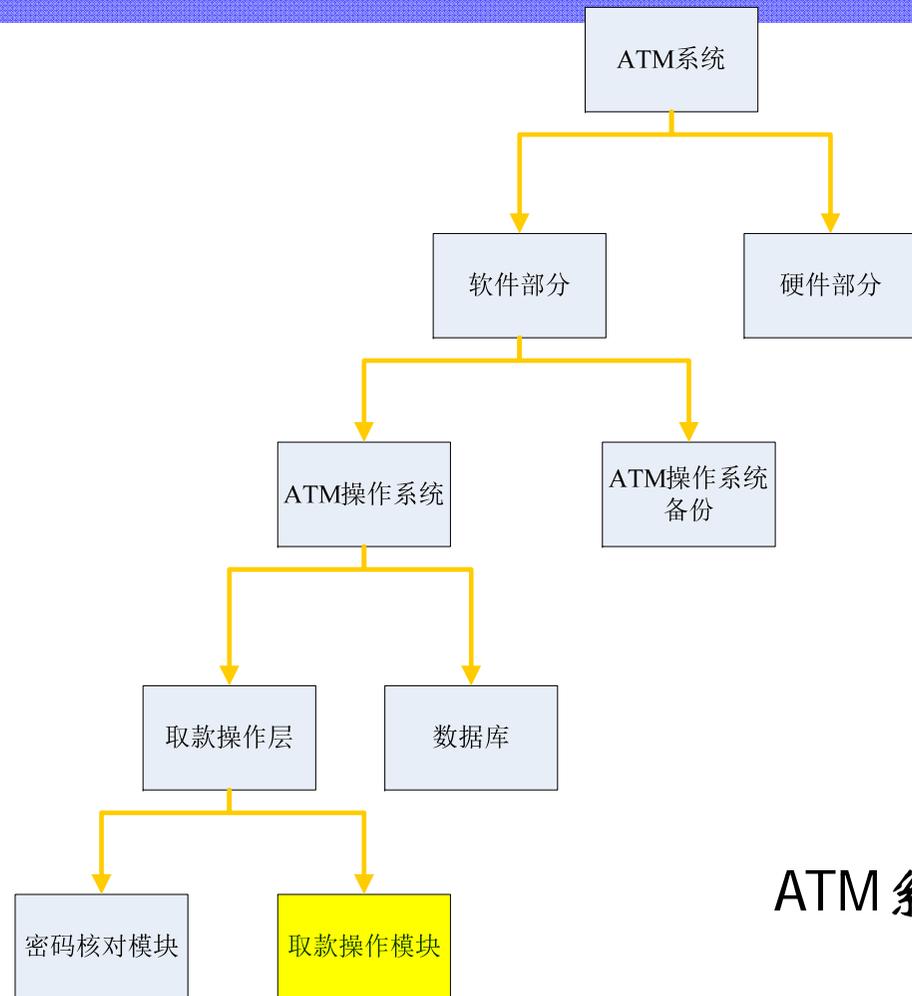


入三角形: 表示树的部分分支在另外地方绘制, 用以简化故障树;

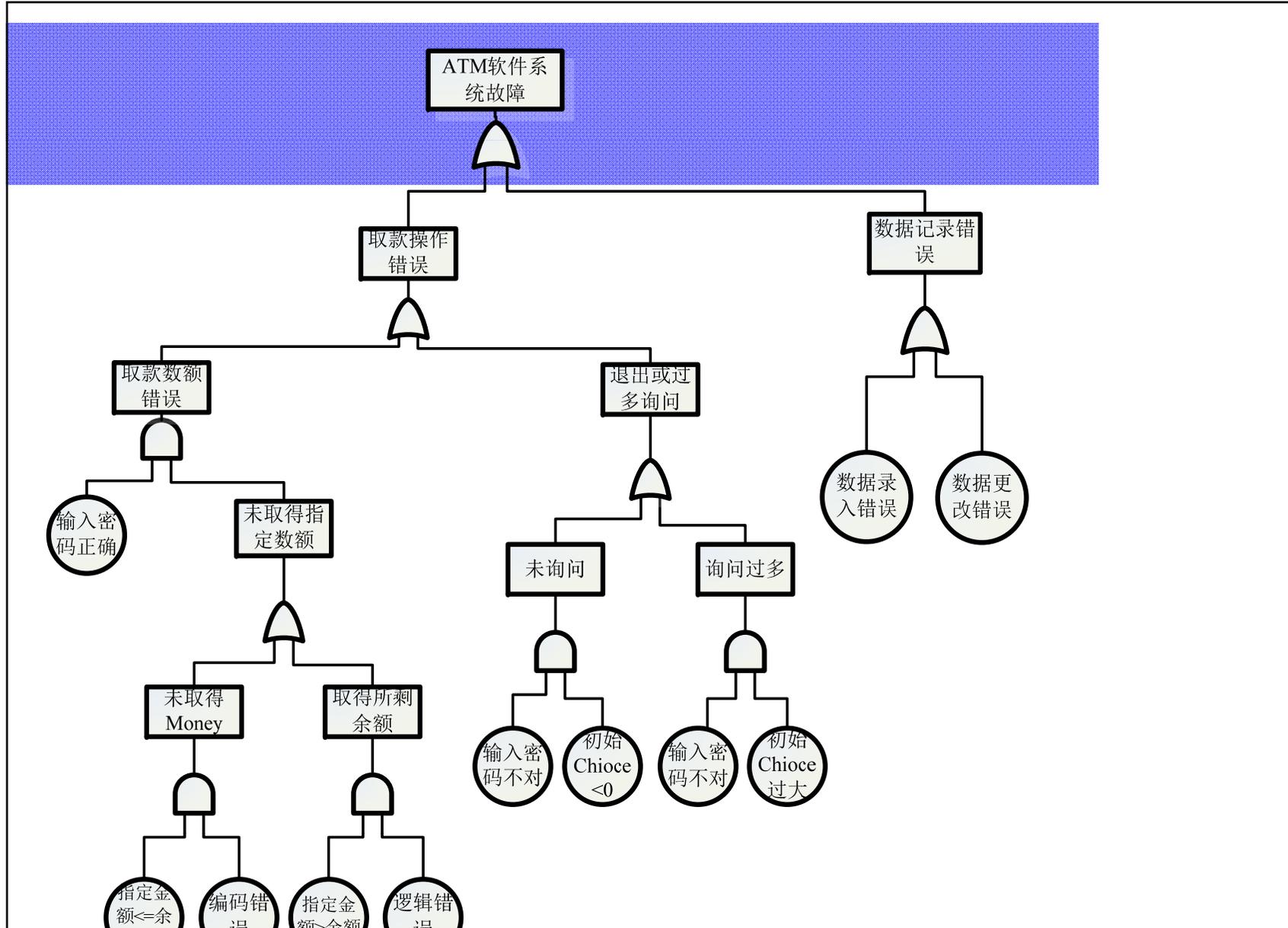
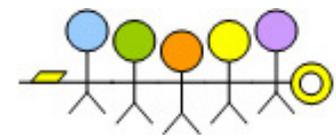
出三角形: 表示该树是在另外部分绘制的一颗故障树的子树, 用以简化故障树。



故障树分析示例



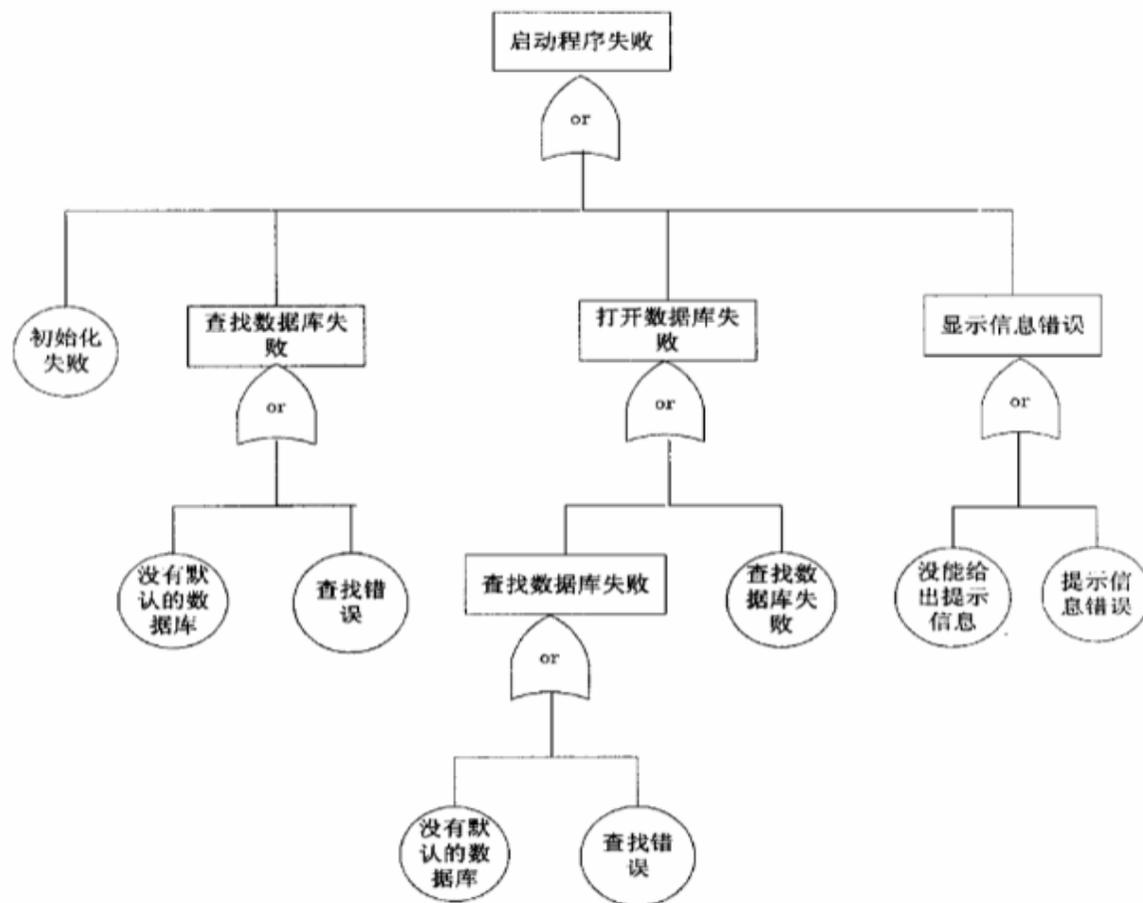
ATM系统结构

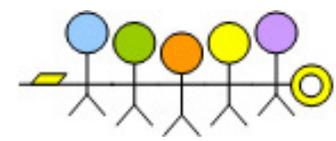




故障树分析示例

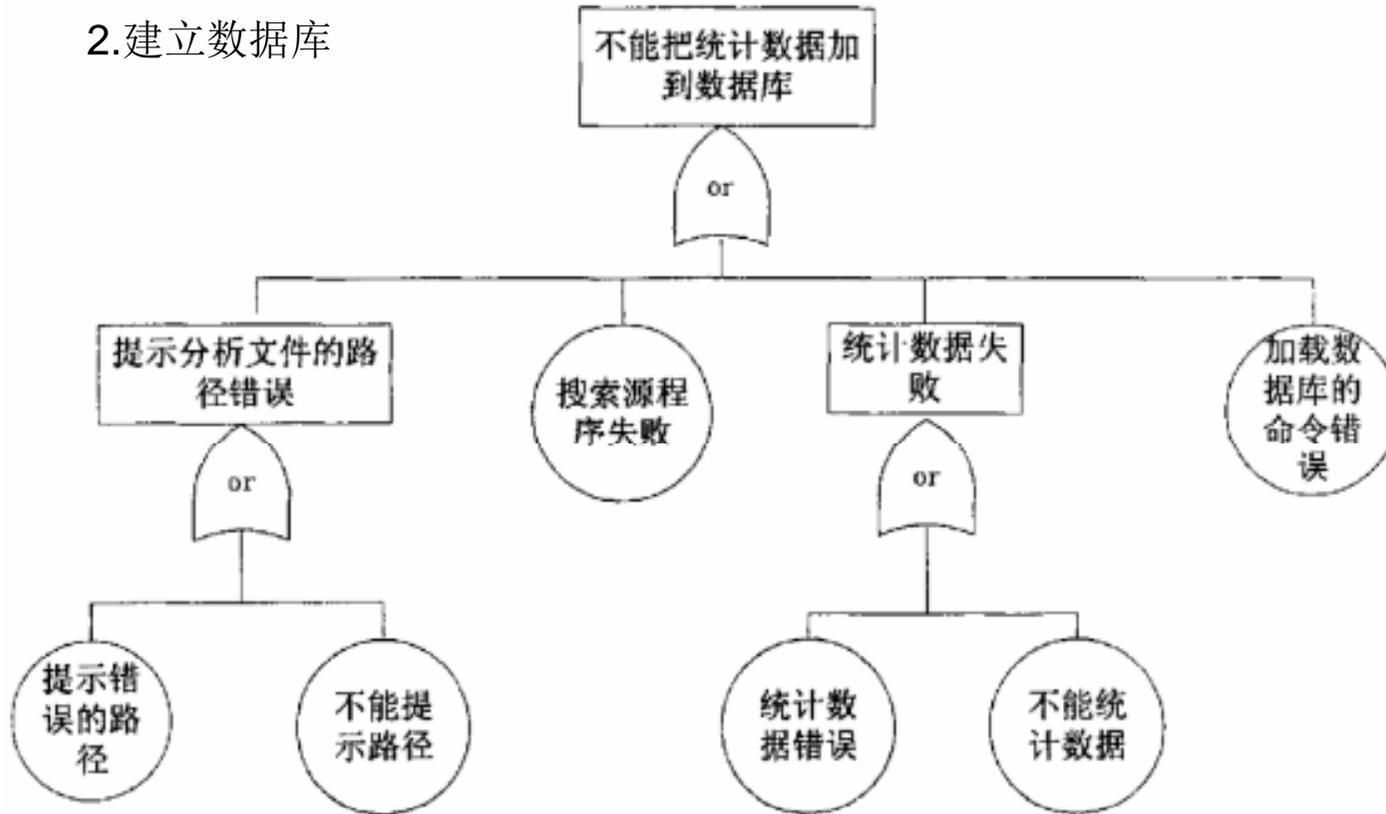
1. 启动运行





故障树分析示例

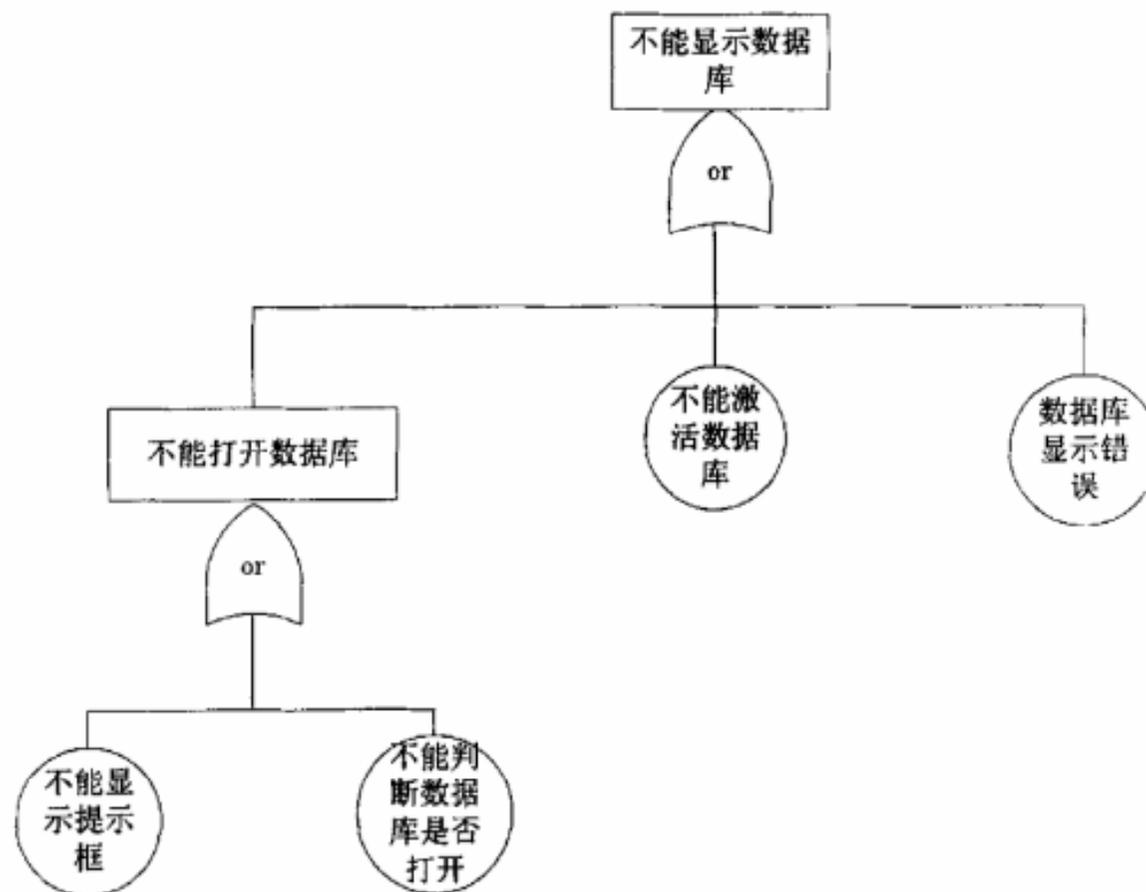
2. 建立数据库





故障树分析示例

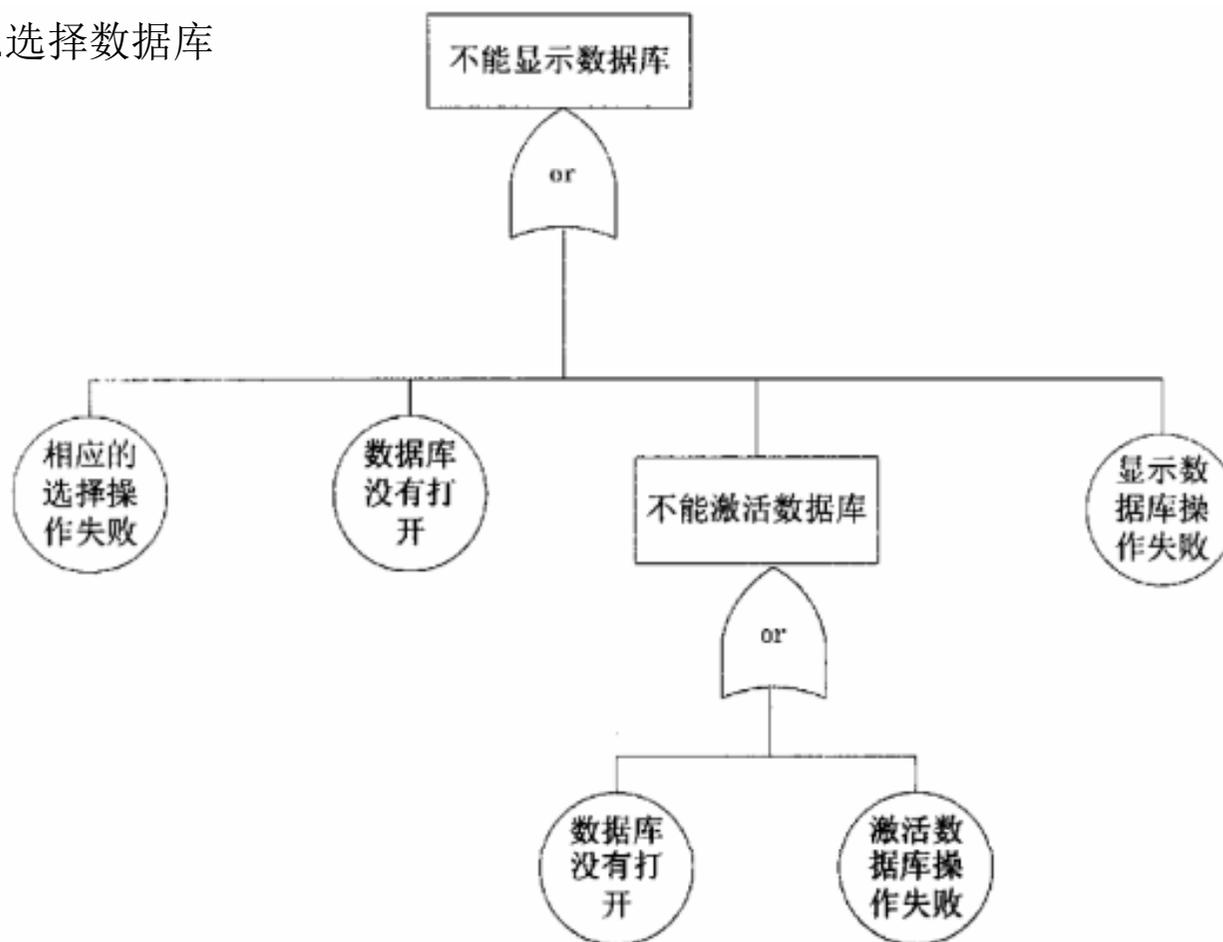
3. 打开数据库





故障树分析示例

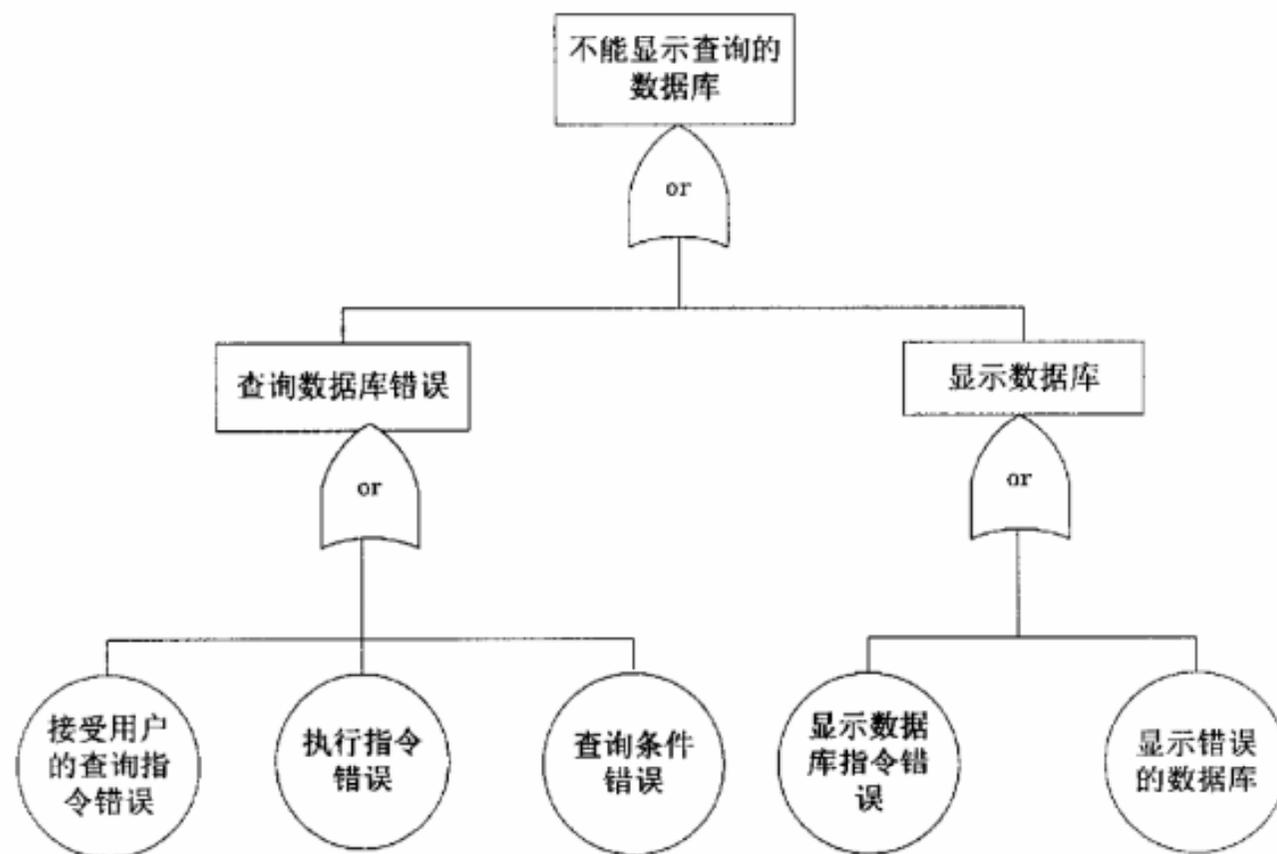
3.选择数据库





故障树分析示例

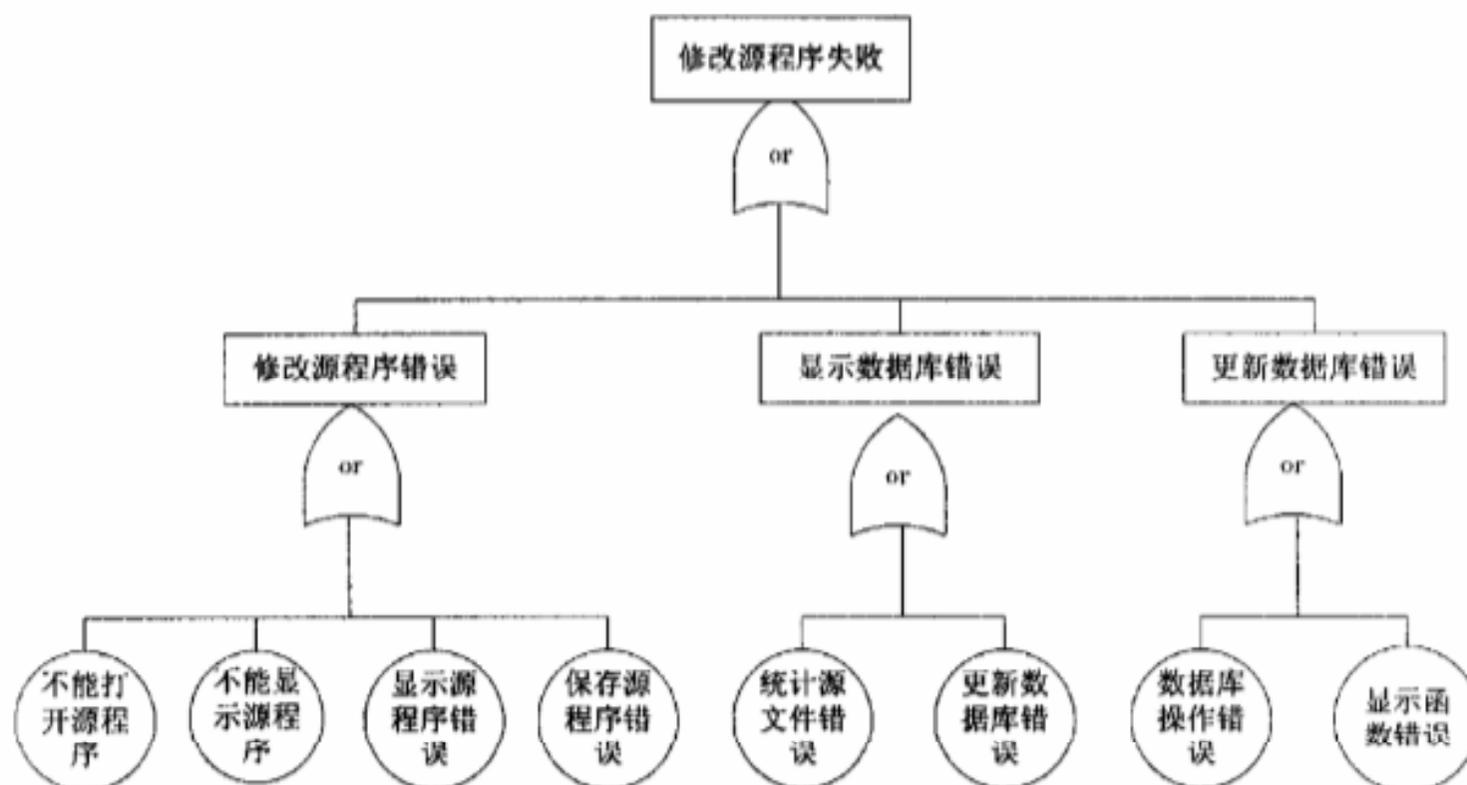
5. 查询数据库

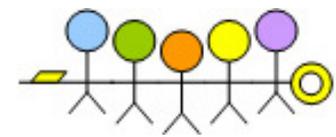




故障树分析示例

修改源程序





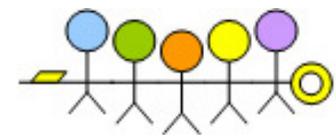
软件故障树定性分析

- 目的
 - 软件故障树分析最重要的意义在于，根据分析的结果找出关键性的事故发生的原因，指导软件可靠性、安全性设计以及软件测试，从而提高软件的可靠性和安全性。
- 方法
 - 识别所有最小割集，并对最小割集进行定性比较，对最小割集及底事件的重要性进行排序。



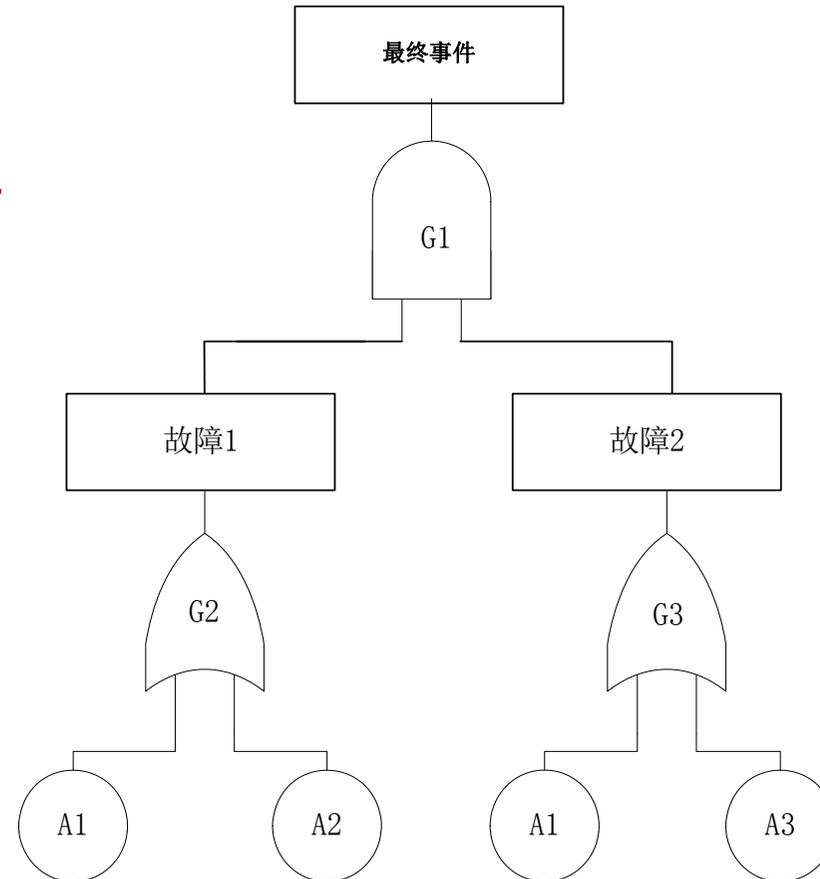
软件故障树定性分析

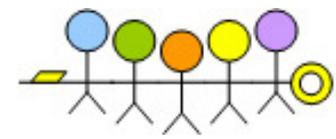
- 割集：能引起顶事件发生的基本事件集合。
- 最小割集：不包含任何冗余因素的割集。如果去掉最小割集中的任何事件或条件，它就不再成为割集。
 - 阶数越小的最小割集越重要；
 - 低阶最小割集所包含的底事件比高阶最小割集中的底事件重要；
 - 在不同最小割集中重复出现次数越多的底事件越重要。



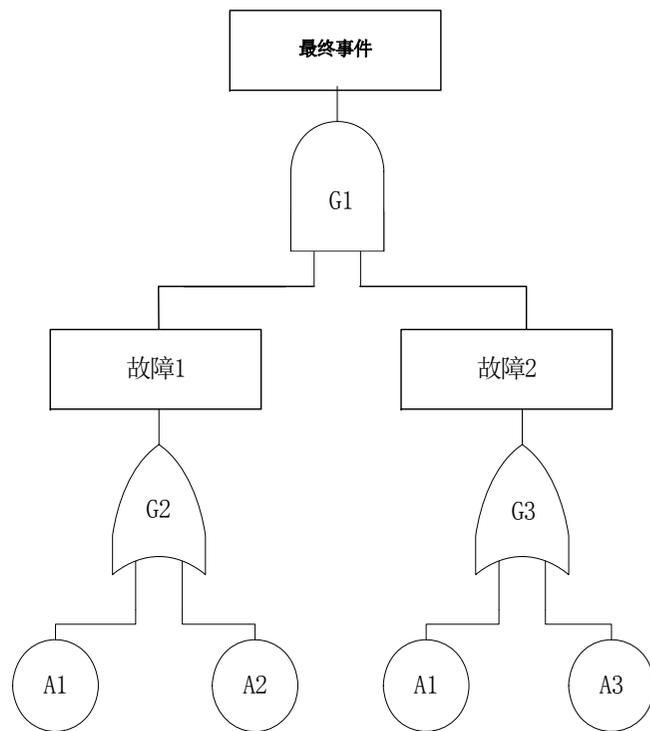
故障树分析示例

最小割集: {A1} {A2,A3}



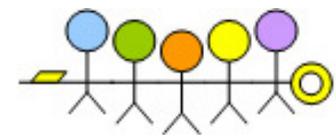


下行法

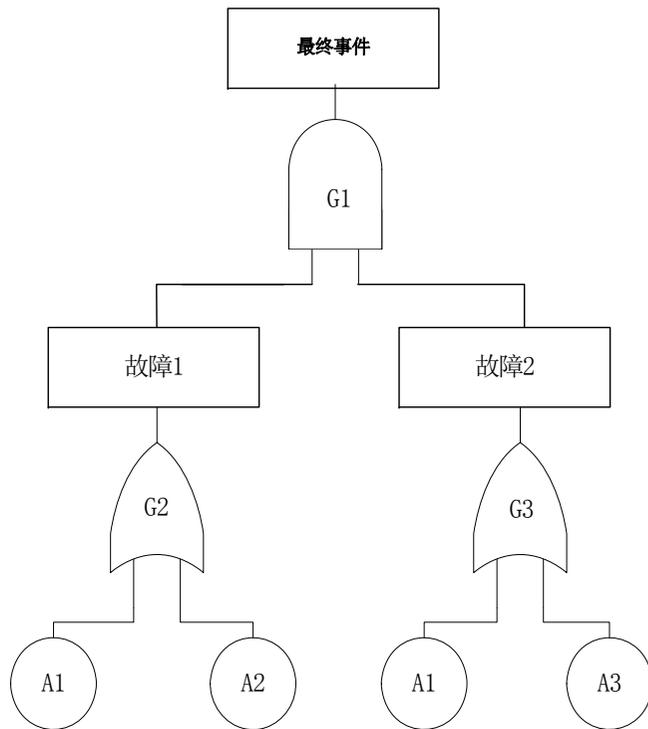


步骤	1	2	3	4
	$F_1 F_2$	$A_1 F_2$	$A_1 A_1$	A_1
		$A_2 F_2$	$A_1 A_3$	$A_1 A_3$
			$A_2 A_1$	$A_1 A_2$
			$A_2 A_3$	$A_2 A_3$

最小割集: {A1} {A2,A3}



上行法



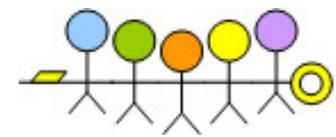
$$F_1 = A_1 + A_2$$

$$F_2 = A_1 + A_3$$

$$G1 = F1F2 = (A_1 + A_2)(A_1 + A_3)$$

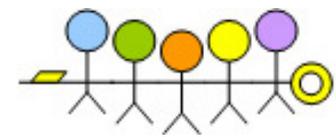
$$= A_1 + A_1A_3 + A_1A_2 + A_2A_3$$

最小割集: {A1} {A2,A3}



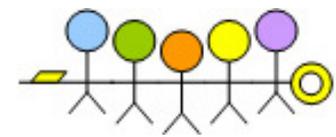
软件故障树定量分析

- 目的
 - 软件故障树定量分析的目的是计算或估计软件故障顶事件发生的概率。
- 方法
 - 如果每一个底事件的发生概率是已知的，可根据软件故障树的逻辑关系，计算顶事件的发生概率。
- 软件故障树分析的重点在于定性分析，而非定量分析。



SFMEA与SFTA的区别与联系

		SFMEA	SFTA
区别	分析方法	由系统的最低分析层次，分析潜在的失效模式及原因，自底向上分析对系统产生的影响，即 由因到果 。	是一种自顶向下的分析方法，从系统不希望发生的事件（顶事件）开始，向下逐步追查导致顶事件发生的原因，直到基本事件（底事件），即 由果到因 。
	分析级别	SFMEA分析的最低级别可以是实现一定功能的 模块 。	最低分析级别可以是程序的 语句 。
	分析对象	SFMEA用于分析系统和子系统所有可能的失效模式和原因。	SFTA只分析顶层的故障事件。
联系	将SFMEA和SFTA相综合，是一种有效的软件可靠性分析方法。		



小结

- 软件可靠性设计和分析是软件可靠性工程的重要内容；
- 软件可靠性设计包括避错设计、查错设计、改错设计和容错设计；
- 软件可靠性分析包括软件失效模式和影响分析、软件故障树分析、软件Petri网分析和软件潜藏分析。