软件质量保证与测试

第5章 软件质量度量

内容提要

- 5.1软件度量
  - 5.1.1软件项目度量
  - 5.1.2软件产品度量
  - 5.1.5软件过程度量
- 5.2软件可靠性度量
- 5.3小结

5.1软件度量

- **对于管理层人员来说：**
  - 没有对软件过程的可见度就无法管理；
  - 没有对见到的事物有适当的度量或适当的准则去判断、评估和决策，也无法进行优秀的管理。
- **在软件开发中，软件质量度量的根本目的是为了管理的需要，利用度量来改进软件过程**
- **度量是一种可用于决策的可比较的对象**
  - 已知事物：跟踪和评估
  - 未知事物：预测

软件度量(续)

- **Measure：测量**，对一个产品或过程的某个属性的范围、数量、维数、容量或大小提供了一个定量的指标
- **Measurement: 测度**，确定一个测量的行为
- **Metrics: 度量**，对一个系统、部件或过程具有的某个给定属性的度的一个定量测量
- **Indicator: 指标**，一个度量或度量的组合，对软件过程、软件项目或软件产品提供更深入的了解

软件度量(续)

- **Measure：测量**，对一个产品或过程的某个属性的范围、数量、维数、容量或大小提供了一个定量的指标
- **Measuremer**为
- **Metrics: 度量**具有的某个给定属性的度的一个定量测量
- **Indicator: 指标**，一个度量或度量的组合，对软件过程、软件项目或软件产品提供更深入的了解

获取单个数据点，如测量一个模块评审中发现的错误

软件度量(续)

- **Measure：测量，对一个产品或过程的某个属性的范围、数量、维数、容量或大小提供了一个定量的指标**
- **Measurement: 测度，确定一个测量的行为**
- **Metrics: 度量，对**
  某个给定属性的
- **Indicator: 指**
  件过程、软件项

收集一个或多个数据点的结果，如调研若干次模块评审，以收集每一次评审所发现的错误数

软件度量(续)

- **Measure：测量**，对一个产品或过程的某个属性的范围、数量、维数、容量或大小提供了一个定量的指标
- **Measurement: 测度**，确定一个测量的行为
- **Metrics: 度量**，对一个系统、部件或过程具有的某个给定属性的度的一个定量测量
- **Indicator: 指标**，[illegible]个度量或度量的组合，对软件过程、软件[illegible]

某种程度上与单个测量相关，如每一次评审所发现的平均错误数，或评审中每人-小时所发现的错误数

软件度量(续)

- **Measure：测量**，对一个产品或过程的某个属性的范围、数量、维数、容量或大小提供了一个定量的指标
- **Measurement: 测度**，确定一个测量的行为
- **Metrics: 度量**，对一个系统、部件或过程具有的某个给定属性的度的一个定量测量
- **Indicator: 指标**，一个度量或度量的组合，对软件过程、软件项目或软件产品提供更深入的了解

如：四个软件小组共同完成一个软件项目，每组必须进行技术评审，但可自由选取评审方式

软件度量(续)

- 软件度量是对软件项目、过程及其产品进行数据定义、收集以及分析的持续性定量化过程

- 软件度量的效用：
  - 理解
  - 预测
  - 评估
  - 控制
  - 改善

软件度量(续)

- 软件度量的现状
  - 设计和开发软件产品时，并未制定出度量的目标
  - 由于缺乏清晰的度量目标，开发人员不能使产品各方面的质量特性合格，也不能使用术语向用户说明软件产品具有很高的质量
  - 由于缺乏对软件的度量，看不到清晰的实效，因而对所使用的软件开发技术没有足够的信心

软件度量(续)

- 软件度量的目标
  - ① 需要使用严格的度量术语来指定对软件质量的要求
  - ② 需要度量软件开发过程中不同阶段的费用
  - ③ 需要度量不同小组职员的生产率
  - ④ 需要度量开发的产品的质量
  - ⑤ 需要度量开发过程的属性
  - ⑥ 需要度量不同软件工程方法和工具的效用

软件度量(续)

- 影响软件质量的因素
  - 人
  - 过程
  - 技术
- 软件质量函数：Q={M，P，T}

软件度量(续)

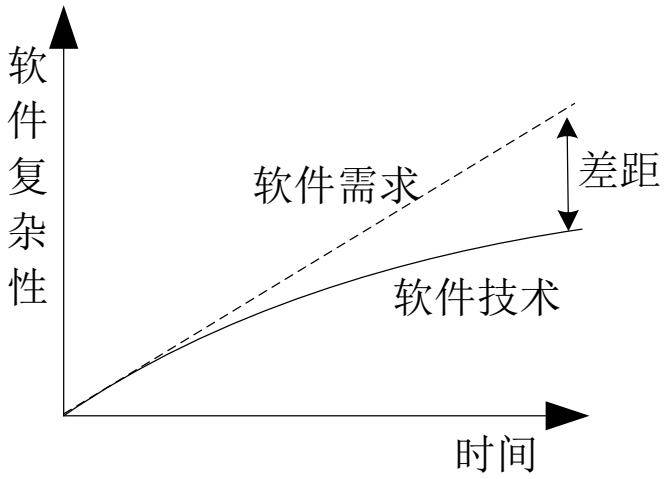
- **软件质量影响因素—人**
  - 积极进取的开发人员
  - 用户、分析员、设计员、程序员、测试员配合得当

软件度量(续)

- **软件质量影响因素—过程**
  - 软件工程过程：软件开发和生产的过程
  - 软件管理过程：对软件开发和生产进行管理的过程
  - 软件支持过程：对软件开发和生产进行支持的过程

软件度量(续)

- 软件质量影响因素—技术



软件度量(续)

- 软件度量贯穿于软件开发的整个生命周期
  - 项目度量
  - 产品度量
  - 过程度量

软件度量(续)

度量维度	侧重点	具体内容
项目度量	理解和控制当前项目的情况和状态	规模、成本、工作量、进度、生产力、风险、顾客满意度
产品度量	理解和控制当前产品的质量	产品的功能性、可靠性、易使用率、效率、可维护性、可移植性
过程度量	理解和控制当前情况和状态，对过程进行改进和预测	能力成熟度、管理、生命周期、生产率、缺陷植入/排除率

5.1.1 软件项目度量

作用：

- 软件项目度量导出的指标由项目管理者和软件项目组使用，以改进项目工作流程和技术活动。
- 从过去项目中收集的度量可用来作为估算当前项目的工作量及时间的基础。
- 项目管理者使用这些数据来监督和控制项目的进展。

软件项目度量(续)

软件项目度量建议每个项目都应该测量：

- **输入**

  完成工作所需的资源的测量

- **输出**

  软件工程过程中产生的交付物或工作产品的测量

- **结果**

  表明交付物的效力的测量

软件项目度量(续)

- 直接测量
  - 成本
  - 工作量
  - 生产的代码行数
- 间接测量
  - 软件的功能
  - 产品的功效

软件项目度量(续)

- 度量
  - 面向规模的度量
  - 面向功能的度量

软件项目度量(续)

项目	LOC(代码行)	工作量	成本(千美元)	文档页数	缺陷	人员
A	12100	24	168	565	29	5
B	27200	62	440	1224	86	5
C	20200	45	514	1050	64	6

- 面向规模的度量：以代码行作为规范化值
  - 每千行代码的缺陷数
  - 每千行代码的花费
  - 每千行代码的文档页数

其它有意义的度量：
- 每人月的错误数
- 每人月的代码行
- 每页文档的花费

软件项目度量(续)

- 面向功能的度量：以软件所提供的功能的测量作为规范化值
- Albrecht提出一种称为功能点的测量

  功能点(FP)是基于软件信息域的特性及软件复杂性的评估而导出的
- 基于功能点的度量：
  - 每个功能点的缺陷数
  - 每个功能点的花费
  - 每个功能点的文档页数
  - 每人月完成的功能点数

5.1.2 软件产品度量

- **CMM对软件质量的定义：**
  - 一个系统、组件或过程符合特定需求的程度；
  - 一个系统、组件或过程符合客户或用户的要求或期望的程度。

“质量越高越好”并非普适的真理

软件质量度量(续)

- **软件质量是许多质量属性的综合体现**
- **软件质量要素：**
  - 技术角度：对软件整体质量影响最大的哪些质量属性
  - 商业角度：客户最关心的、能成为卖点的质量属性

先确定质量要素
后给出质量提高措施

软件质量度量(续)

- 质量需求定义在软件系统所需的输出清单中，通常是多维的：
  - 输出使命
  - 输出所需的准确度
  - 输出信息的完整性
  - 信息的及时性
  - 信息的可用性
  - 软件系统的编码与文档编制标准

软件质量度量(续)

例：俱乐部会员信息系统的质量需求：

- 输出使命：一份明确的清单包括11种报告、4种给会员的标准信函和8种查询，它们都将根据请求显示在显示器上
- 输出所需的准确度：包含一个或多个错误的不准确输出的概率不超过1％
- 输出信息的完整性：一个会员在俱乐部活动的参与情况和付费数据丢失的概率不超过1％
- 信息的及时性：录入有关参加活动的信息不超过2个工作日；录入有关会员付费记录和个人数据不超过1个工作日；查询的反应时间平均少于2秒；报告的反应时间少于4小时
- 软件系统的编码与文档编制标准：要求软件及其文档要符合客户指南

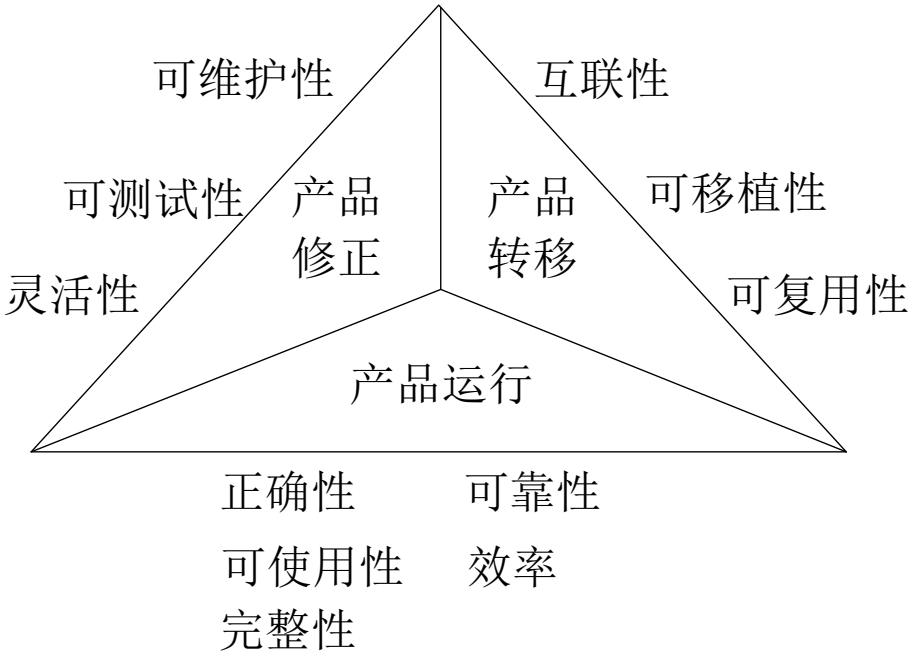
软件质量模型

- McCall模型
- Boehm模型
- FURPS模型
- ISO 9126

McCall模型

McCall质量模型：

- 产品操作运行特性
- 产品修改特性
- 产品的适应特性



McCall模型(续)

产品运行相关的软件质量因素：

① 正确性(correctness)：一个程序满足它的需求规约和实现用户任务目标的程度。

② 可靠性(reliability)：一个程序满足其所需的精确度，完成它的预期功能的程度

③ 效率(efficiency)：一个程序完成其功能所需的计算资源和代码的度量。

④ 完整性(integrity)：为某一目的而保护数据，避免它受到偶然的或有意的破坏、改动或遗失的能力，即对未授权人员访问软件或数据的可控制程度

⑤ 可使用性(usability)：学习、操作、准备输入和解释程序输出所需的工作量。

McCall模型(续)

可靠性：决定允许的最大软件系统失效率，可以是指整个系统或是它的一个或多个功能

例：准备安装在银行的新软件系统，对它的可靠性需求：

- 在银行的办公时间内每月的失效时间不超过10分钟
- 停止时间(所有银行服务的维修和回复所需的时间)超过50分钟的概率要小于0.5％

McCall模型(续)

效率：与所需的硬件资源有关。这些硬件资源是实现软件系统所有功能所需的。

例：一个商店连锁店在为一个软件系统考虑两份标书

- 标书A:占用计算机存储器容量20GB,需要5条28.8KBPS的通信线路
- B标书：占用计算机存储器容量10GB,需要2条28.8KBPS的通信线路

McCall模型(续)

完整性： 与软件系统的保密性有关，即防止非授权人员的访问

例：某地方市局的工程部使用一个GIS(地理信息系统)。这个工程部允许市民通过Internet访问GIS文件，可以观看和拷贝，但不得在所访问的地图中插入更改，如果要进入正在制作的图或由确定为受限访问的地图时，访问将被拒绝。

McCall模型(续)

可使用性：与培训新员工或操作软件系统所需的人力资源的范围有关

例：由一个家庭用品服务公司启动的一个新服务系统，其可使用性需求如下：

- 一位员工应当一天至少能够处理60个服务电话
- 训练一个新雇员不超过16个小时，训练完毕后，受训者立即能够每天处理45个服务电话

McCall模型(续)

产品修改软件质量因素：

① 可维护性(maintainability)：定位和修复程序中一个错误所需的工作量。

② 灵活性(flexibility)：修改一个运行的程序所需的工作量。

③ 可测试性(testability)：测试一个程序以确保它完成所期望的功能所需的工作量。

McCall模型(续)

可维护性： 维护人员识别软件失效的原因、失效的改正、验证改正成功所需的工作

例：代表性的可维护性需求：

- 软件模块的大小不超过50条语句
- 编程遵守公司的编码标准和指南

McCall模型(续)

- 软件维护所占的工作量比任何其它软件工程活动都大
- 可维护性无法直接测量
- 一个简单的间接测量方式：平均修改时间(mean-time-to-change, MTTC)：即分析改变的需求，设计合适的修改方案，实现修改，测试，并将修改后的结果发布给用户所花的时间

McCall模型(续)

产品移植软件质量因素：

① 可移植性(portability)：把一个程序从一个硬件和或软件系统环境移植到另一个环境所需的工作量。

② 可复用(reusability)：一个程序可以在另外一个应用程序中复用的程度

③ 互连性(interoperability)：连接一个系统和另一个系统所需的工作量。

McCall模型(续)

可移植性： 关注的是软件系统对由不同硬件、不同操作系统等组成的其它环境的适应

例：设计和编制一个在Windows 8环境下运行的软件，需要它能够以低费用转移到Linux环境。

McCall模型(续)

可复用性：与原先为一个项目设计的软件模块在当前正在开发的新项目中的使用有关

例：为游泳池的运营和控制开发一个软件系统，该游泳池为宾馆客人和游泳池俱乐部成员服务，要求某些模块应当设计和编制得能够在游泳池的未来软件系统中重用，计划这个未来软件系统在下一年开发，这些模块包括：

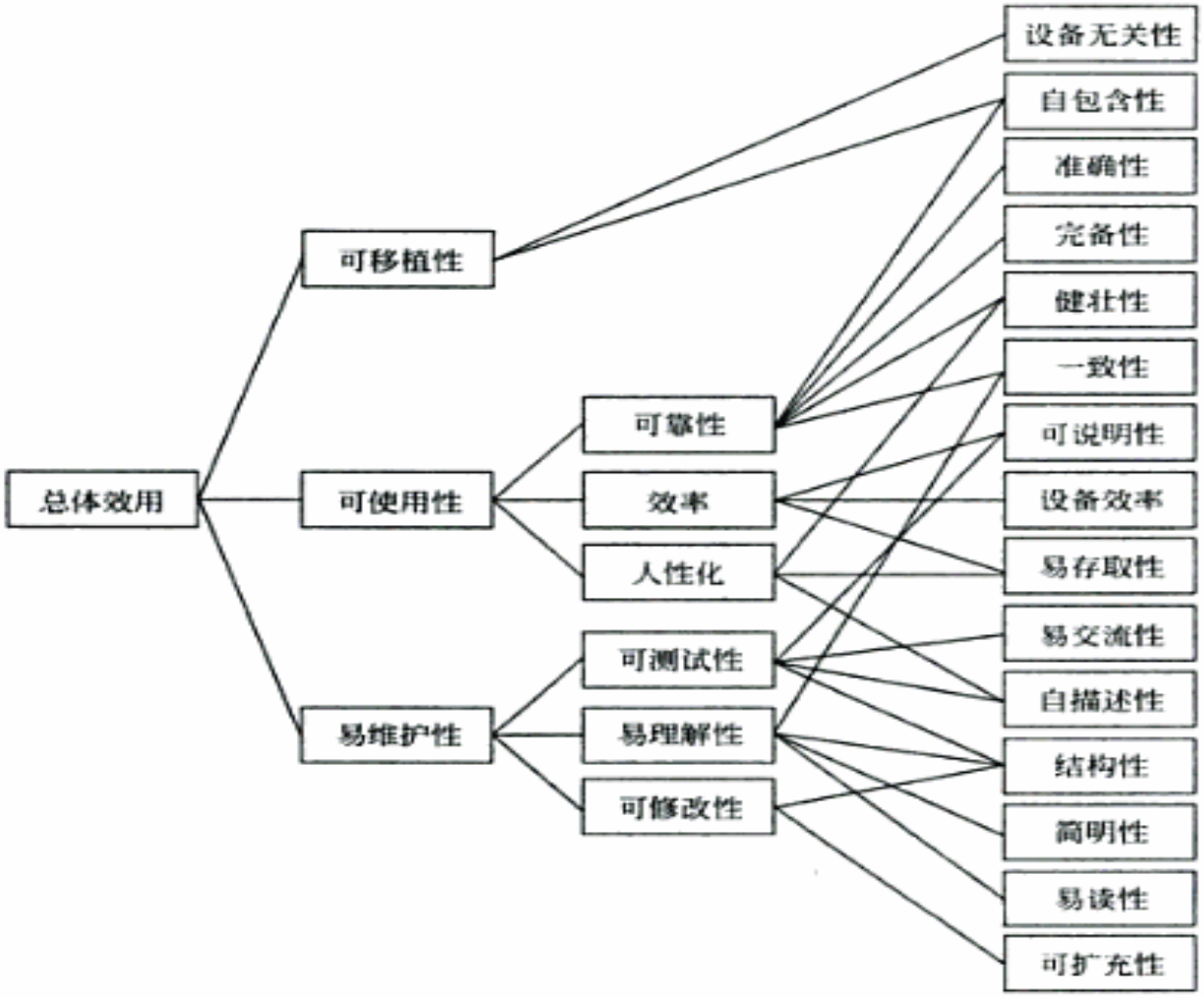
- 会员卡的入口确认检查
- 会员资格延期信件的处理

McCall模型(续)

互连性：关心建立同其它软件系统或其它设备固件的接口之间的互操作，能够规定那些需要接口的软件或固件的名字，还能够规定已经被接受为特定行业或应用领域标准的数据结构。

例：医学实验室设备的固件产生标准数据结构的结果，要求实验室信息系统的输入符合该数据结构

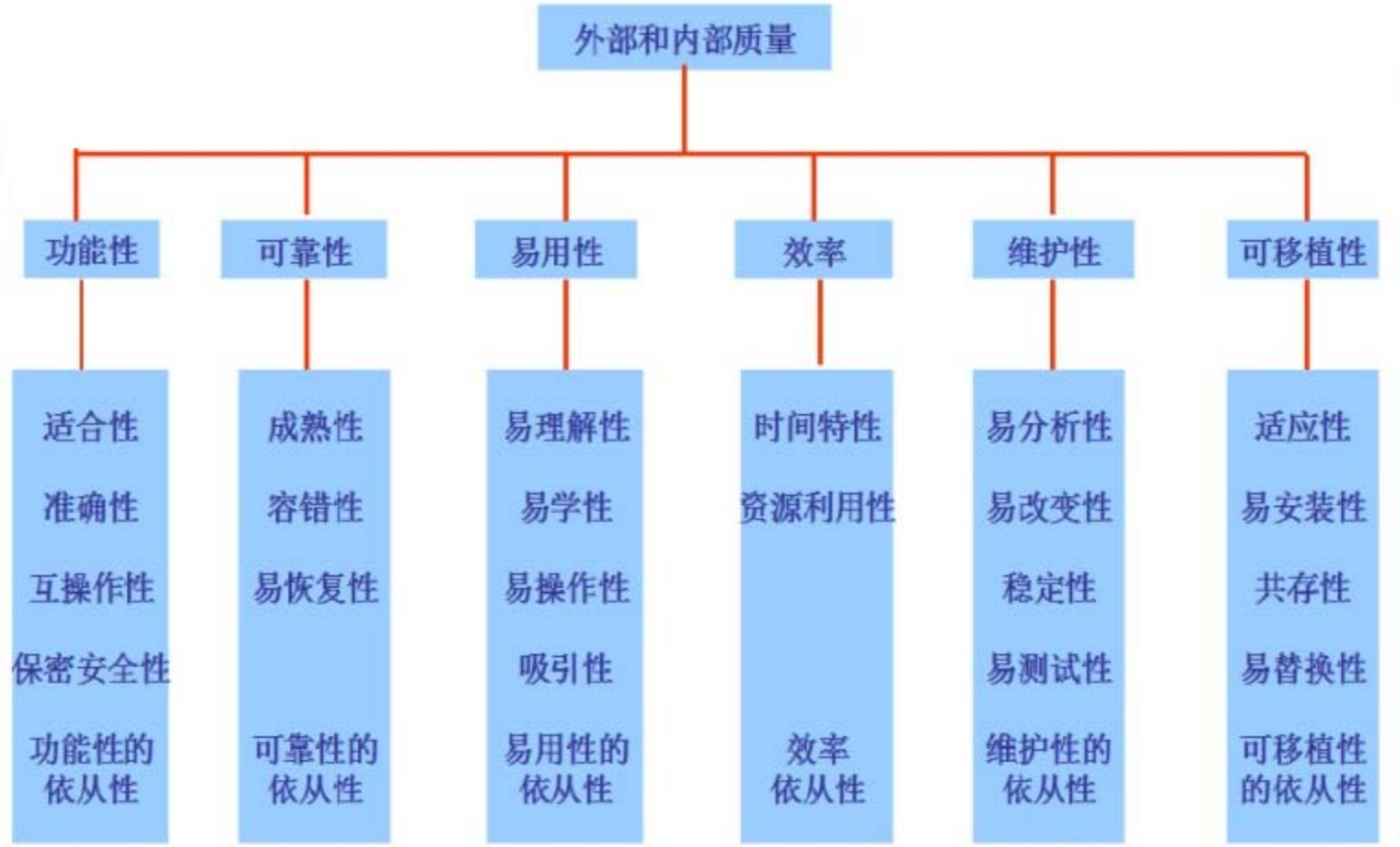
Boehm模型


总体效用
可移植性
可使用性
易维护性
可靠性
效率
人性化
可测试性
易理解性
可修改性
设备无关性
自包含性
准确性
完备性
健壮性
一致性
可说明性
设备效率
易存取性
易交流性
自描述性
结构性
简明性
易读性
可扩充性

FURPS模型

- 功能性(Functionality)：功能集、能力、通用性、安全性
- 可用性(Usability)：人性化因素、美学、一致性、文档
- 可靠性(Reliability)：故障频率及严重程度、可恢复性、可预测性
- 性能(Performance)：响应时间、吞吐量、资源利用率、效率
- 可支持性(Supportability)：适应性、可维护性、兼容性、可测试性、可配置性

ISO 9126


外部和内部质量
功能性
可靠性
易用性
效率
维护性
可移植性
适合性
准确性
互操作性
保密安全性
功能性的依从性
成熟性
容错性
易恢复性
可靠性的依从性
易理解性
易学性
易操作性
吸引性
易用性的依从性
时间特性
资源利用性
效率依从性
易分析性
易改变性
稳定性
易测试性
维护性的依从性
适应性
易安装性
共存性
易替换性
可移植性的依从性

ISO 9126(续)

1、功能性

① 合适性
- 所提供的功能是用户所需要的
- 用户所需要的功能系统已提供

② 准确性：是否满足用户对该功能的精确度要求

③ 互操作性：与一个或多个周边系统进行信息交互的能力

④ 安全性
- 防止未得到授权的人或系统能正常访问相关的信息或数据
- 保证得到授权的人或系统能正常访问相关的信息或数据

⑤ 功能性的依从性：遵循相关的标准(国际标准、国家标准、行业标准、企业内部规范等)约定或法规以及类似规定的能力

ISO 9126(续)

2、可靠性

① 成熟性：软件系统防止内部错误扩散而导致失效的能力

② 容错性：软件系统防止外部接口错误扩散而导致系统失效的能力

③ 易恢复性

- 系统原有能力的恢复程度
- 系统原有能力的恢复速度

ISO 9126(续)

3、可用性

① 易理解性：用户在使用软件系统的过程中，展示给用户的信息是否准确、清晰、易懂

② 易学性：软件提供相关的辅助手段，帮助用户学习使用它的能力

③ 易操作性

- 常用功能路径不要太深，最好能提供快捷键，且这些快捷键具有普适性
- 最好提供一键返回桌面的功能
- 操作尽量简单

④ 吸引性：软件具备某些独特的、能让用户眼前一亮的属性，包括GUI，多媒体应用

ISO 9126(续)

4、效率

① **时间效率**：软件系统在各业务场景下完成用户指定的业务请求所需的响应时间

② **资源效率**：软件系统在完成用户指定的业务请求所消耗的系统资源，如CPU占有率、内存占有率、通信带宽占有率等

ISO 9126(续)

5、可维护性

① 易分析性：软件系统提供辅助手段帮助开发人员分析识别缺陷产生的原因，找出待修复部分的能力

② 易改变性：软件缺陷修复的实施容易程度

③ 稳定性：软件系统在长时间连续工作环境下正常工作的能力

④ 易测试性：软件被测试的难易程度

ISO 9126(续)

6、可移植性

① 适应性：软件系统无需做任何变动就能适应不同运行环境的能力

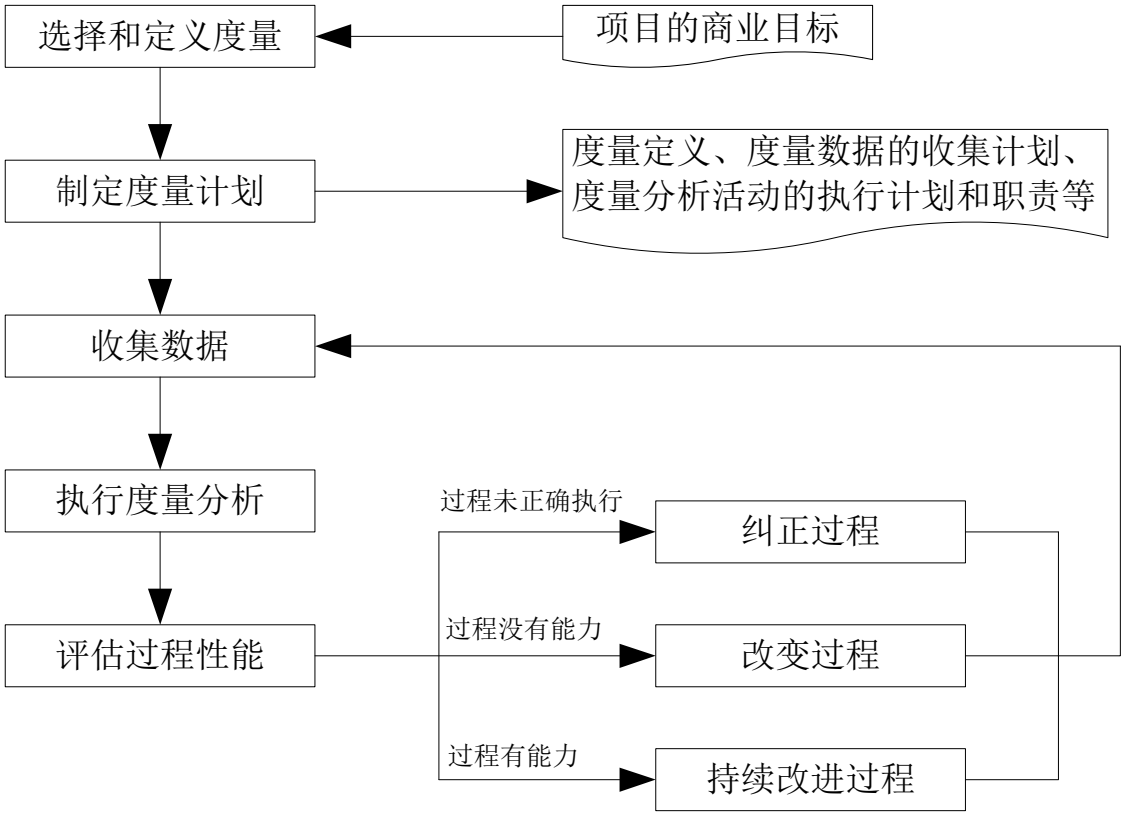
② 易安装性：平台变化后成功安装软件的难易程度

③ 共存性：软件系统在公共环境下与其共享资源的其它系统共存的能力

④ 易替换性：软件系统的升级能力，包括在线升级、打补丁升级等

5.1.3 软件过程度量

- **软件过程度量是对软件过程进行度量的定义、方法、活动和结果的集合。**



软件过程度量(续)

- **软件过程度量的目标**

  对软件过程的行为进行管理

- **软件过程度量的对象**

  组织标准软件过程和项目定义软件过程

- **软件过程度量的方法**

  采集方法、数据分析方法

- **软件过程度量的结果**
  - 模型：如花费模型、质量模型
  - 关系：如人员投入与质量的关系，进度与质量的关系
  - 过程基线

软件过程度量(续)

- **需求阶段**
  - 需求一致性度量

    Q1＝所有评审者都有相同解释的需求数目/需求说明书中的需求个数
  - 需求确认程度度量

    Q2＝已确认为正确的需求个数/需求说明书中的需求个数
  - 需求稳定性度量

    Q3＝(所有确定的需求数-累计的需求变化请求书)/所有确定的需求数

软件过程度量(续)

- **测试阶段**
  - 测试用例的深度
    - 每KLOC的测试用例数
    - 每个功能点的测试用例数
  - 测试用例的有效性
    - 每100个测试用例发现的缺陷数
  - 测试执行的效率
    - 每个人日所执行的测试用例数
    - 每个人日所发现的缺陷数
  - 测试的覆盖率
    - 基于需求的测试覆盖
    - 基于代码的测试覆盖

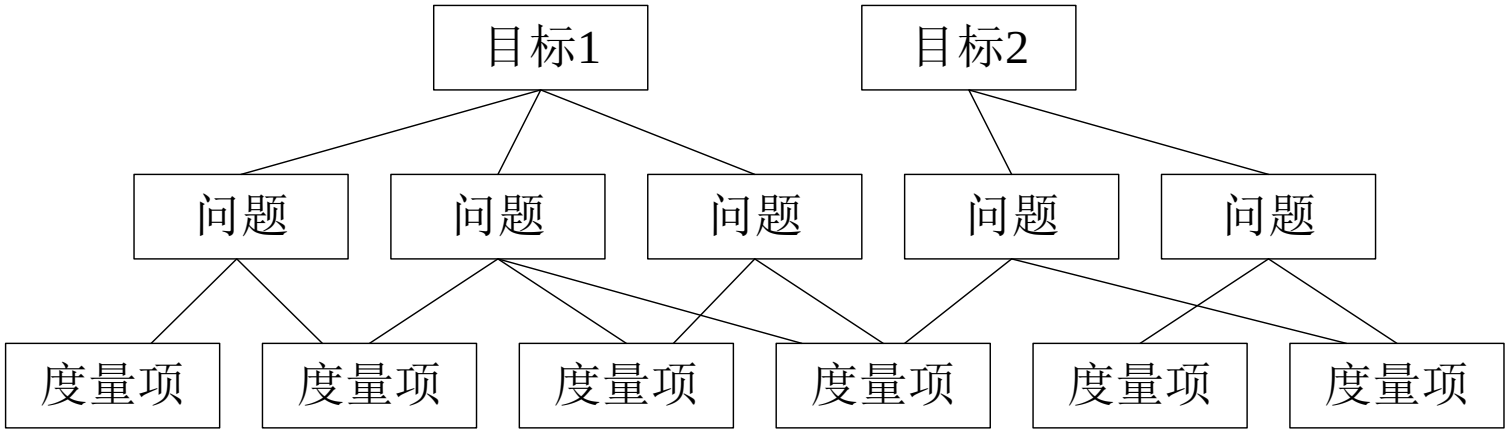
缺陷排除效率

- 缺陷排除效率DRE（Defect Removal Efficiency）是对质量保证及控制活动的过滤能力的一个测量
- 项目级：$DRE = E / (E + D)$
  - E=软件交付给最终用户之前所发现的错误数
  - D=软件交付之后所发现的缺陷数
- 过程级： $DRE_i = E_i / (E_i + E_{i+1})$
  - Ei=在软件工程活动i中所发现的错误数
  - Ei+1 =在软件工程活动i+1中所发现的缺陷数

软件过程度量(续)

- **基于目标的软件过程度量方法GQM**

  Goal-Question-Metric



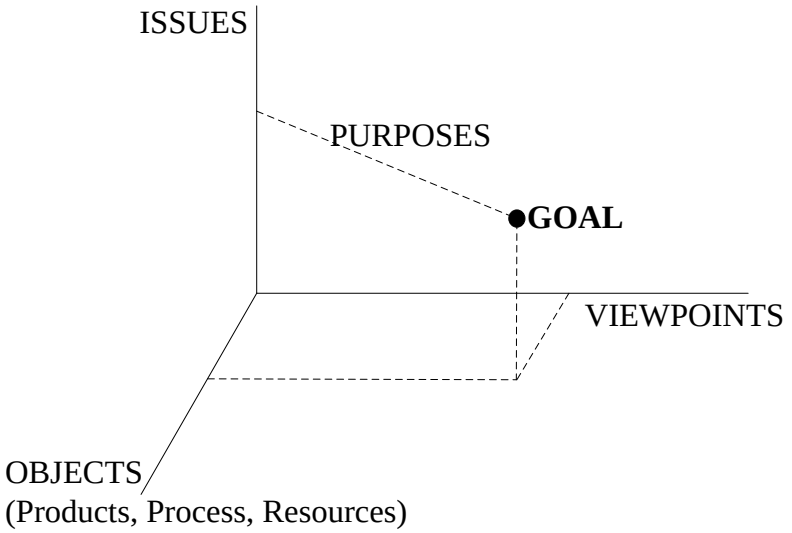
GQM方法

1. 确定度量的目标
2. 提出能够满足目标的问题
3. 确定回答问题所需要的度量

GQM方法(续)

1. 确定目标

- ISSUES(侧重点)
  度量对象的质量重点
- VIEWPOINT(立场)
  信息使用者
- OBJECT(对象)
  要度量对象
- PURPOSES(目的)
  一般是理解、控制和改进要度量的对象



GQM方法(续)

1. 确定目标

① 分析组织的经营战略和方针

② 分析组织的过程和产品定义

③ 分析软件组织的结构

GQM方法(续)

2. 提出问题

- **对于特定目标陈述中的对象，应该抓住哪些可以量化的特征？**
  - 什么是当前同行评审的效率？
  - 实际同行评审过程是按照文档化的流程执行的吗？
  - 同行评审发现缺陷的数量与评审对象规模、评审小组人数有关系吗？
- **结合模型中的侧重点，这些特征应该怎么来描述？**
  - 同行评审的效率与其基线的偏差是多少？
  - 同行评审的效率正在提高吗？
- **结合模型中的侧重点，应该如何评价度量对象的这些特征？**
  - 每人时发现的缺陷数量明显提高了吗？
  - 项目经理能够明显觉察到评审效率的提高吗？

GQM方法(续)

3. 确定度量项

- **现有数据的有效性**

  最大限度地利用现有数据

- **度量对象的稳定性**
  - 对于稳定的度量对象，多应用客观度量
  - 对于不稳定的度量对象，可结合主观判断
- **GQM建模的渐进性**

软件过程度量(续)

常见问题

- 度量的太多、太频繁
- 度量的太少、太迟
- 度量了不正确的事物或属性
- 度量的定义不精确
- 收集了数据却没有利用
- 错误的解释度量数据
- 自动化工具欠缺

5.2软件可靠性度量

- **软件可靠性比硬件可靠性更难保证**
  - 美国F-18飞空系统，首飞前试验2万小时

    故障总数　　硬件故障数　　软件故障数

    580　　271(48.4%)　　509(51.6%)
  - 我国某军舰计算机系统，运行850小时

    故障总数　　硬件故障　　软件故障

    120多次　　约50%　　约70%

软件可靠性度量(续)

- **软件可靠性的发展**
  - 1950年~1958年：没有提及
  - 1959年~1967年：不受重视
  - 1968年~1978年：开创时期
  - 1978年至今：日趋成熟

    1988年提出软件可靠性工程：软件可靠性设计、分析、测试与管理的系统方法

软件可靠性度量(续)

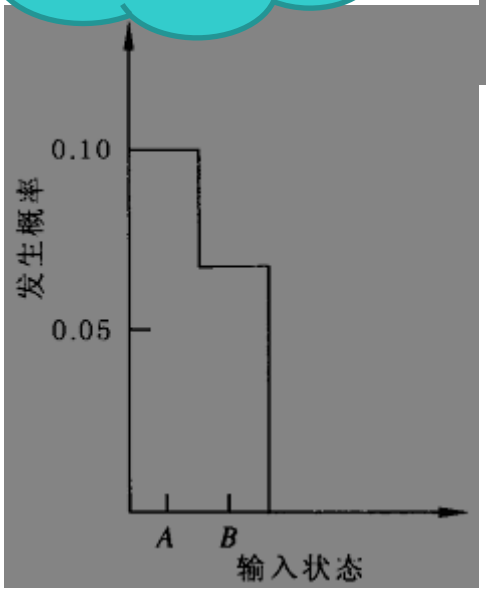
- **IEEE对“软件可靠性”的定义：**
  - [在规定的条件下，在规定的时间内]，软件不引起系统失效的概率〈该概率是系统输入和系统使用的函数，也是软件中存在的错误的函数；系统输入将确定是否会遇到已存在的错误（如果错误存在的话）〉
  - [在规定的时间周期内，在所述条件下]软件执行所要求的功能的能力。

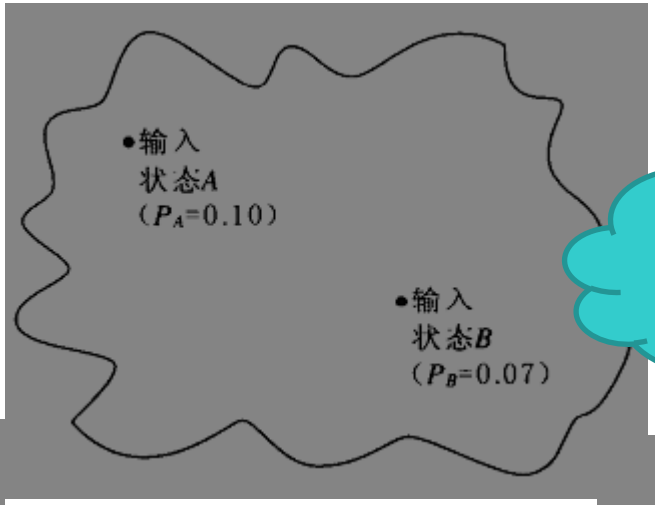
软件可靠性(续)

1. **“规定的时间”**
   - 日历时间
   - 时钟时间
   - CPU时间
2. **“规定的条件”——环境条件**
   - 与程序存储、运行有关的计算机及其操作系统
   - 软件的输入分布
     - **输入空间：输入向量的集合**
     - **输出空间：输出向量的集合**
     - **运行剖面：输入元素选用的概率分布**

运行剖面图

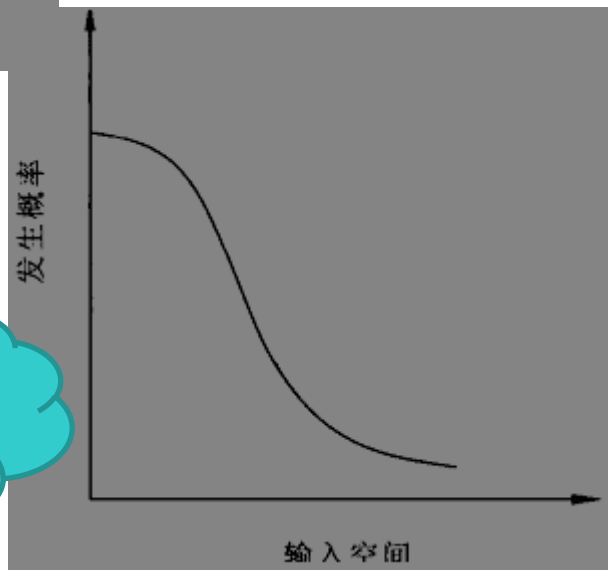
离散型运行剖面图





输入空间示意图

连续型运行剖面图



软件可靠性度量(续)

- **软件可靠性是硬件可靠性的引申和扩展**
  - 软件失效的随机性
  - 软件的运行建立在硬件可靠性基础之上

软件可靠性度量(续)

- **软件和硬件在可靠性特征上的差异:**
  - 硬件有老化损耗现象，软件有陈旧落后的问题
  - 硬件可靠性的决定因素是时间，软件可靠性的决定因素是与输入数据有关的软件差错
  - 硬件的纠错维护可通过修复或更换失效的系统重新恢复功能，软件只有通过重设计。
  - 对硬件可采用预防性维护技术预防故障，采用断开失效部件的办法诊断故障，而软件则不能采用这些技术。
  - 为提高硬件可靠性可采用冗余技术，而同一软件的冗余不能提高可靠性。
  - 硬件可靠性检验方法已建立，并已标准化且有一整套完整的理论，而软件可靠性验证方法仍未建立，更没有完整的理论体系。

软件可靠性度量(续)

- **异常。偏离期望的状态(或期望值)的任何情形都可称为异常。**
- **缺陷。不符合使用要求或与技术规格说明不一致的任何状态常称为缺陷。**
- **差错。从一般意义上说，差错有下面几个方面不同的含义：**
  - 计算的、观测的或测量的值与真实的、规定的或理论上正确的值或条件之间的差别。
  - 一个不正确的步骤、过程或数据定义。
  - 一个不正确的结果。
  - 一次产生不正确的结果的人的活动。
- **故障。在一个计算机程序中出现的不正确的步骤、过程或数据定义常称为故障。**
- **失效。一个程序运行的外部结果与软件产品的要求出现不一致时称为失效。软件失效证明了软件中存在着故障。**

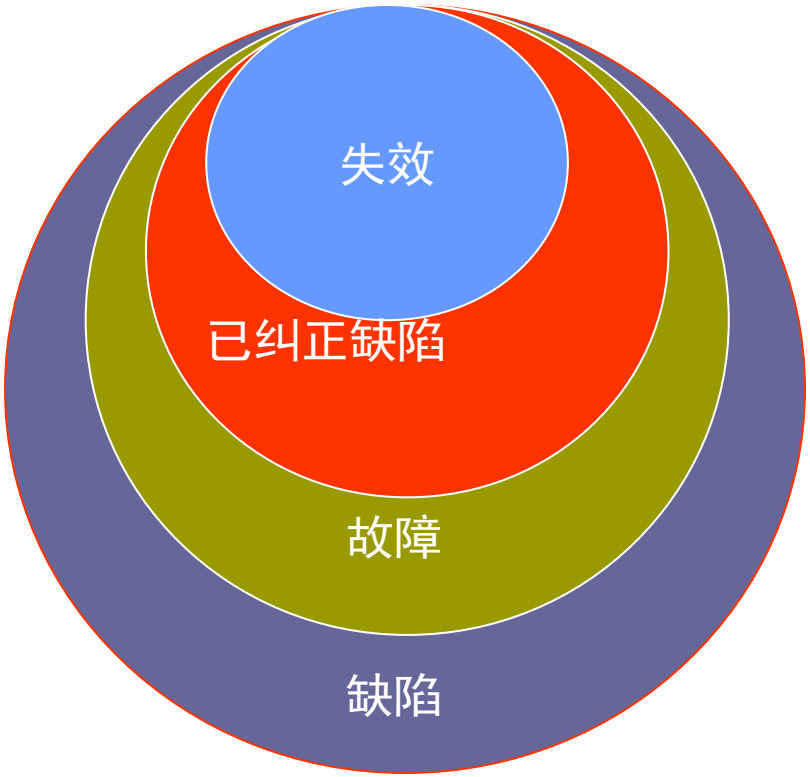
软件可靠性度量(续)

- **差错(Error,)：设计和构造时产生的缺陷**
  - 总数是不可预知的
  - 缺陷分为已知和未知
  - 已知的缺陷分为已纠正和未纠正
- **故障(Fault)：运行结果不正确**
  - 故障是差错的表现形式
  - 故障总是由缺陷引起
  - 缺陷不一定导致故障
- **失效(Failure)：系统不能完成所需要的功能**
  - 失效总是由故障引起
  - 故障不一定导致失效

软件可靠性度量(续)

失　效	故　障
面向用户	面向开发者
软件运行偏离用户需求	程序执行输出错误结果
可根据对用户应用的严重性等级分类	可根据定位和排除故障的难度分类
如，5次失效/1000 CPU小时	如，6个故障/1KLOC

软件可靠性度量(续)



软件可靠性度量(续)

影响软件可靠性的因素

① **软件差错是软件开发各阶段潜入的人为错误：**

- 需求分析定义错误
- 设计错误
- 编码错误
- 测试错误
- 文档错误

引入错误的方式：

- 程序代码特性
- 开发过程特性

② **产品设计的容错性**

软件可靠性度量(续)

可靠性常用度量

① **可靠度 R(t)**

② **不可靠度/累积故障分布函数　F(t)**

③ **故障/失效率 λ(t)**

④ **平均失效前时间MTTF(Mean Time To Failure)**

⑤ **平均故障间隔时间MTBF(Mean Time Between Failures)**

⑥ **平均故障修复时间MTRF(Mean Time To Repair Fault)**

软件可靠性度量(续)

① 可靠度R(t)：在给定条件下，在时间[0,t]内，软件无故障运行的概率

令T表示软件运行开始到发生故障所经历的时间，

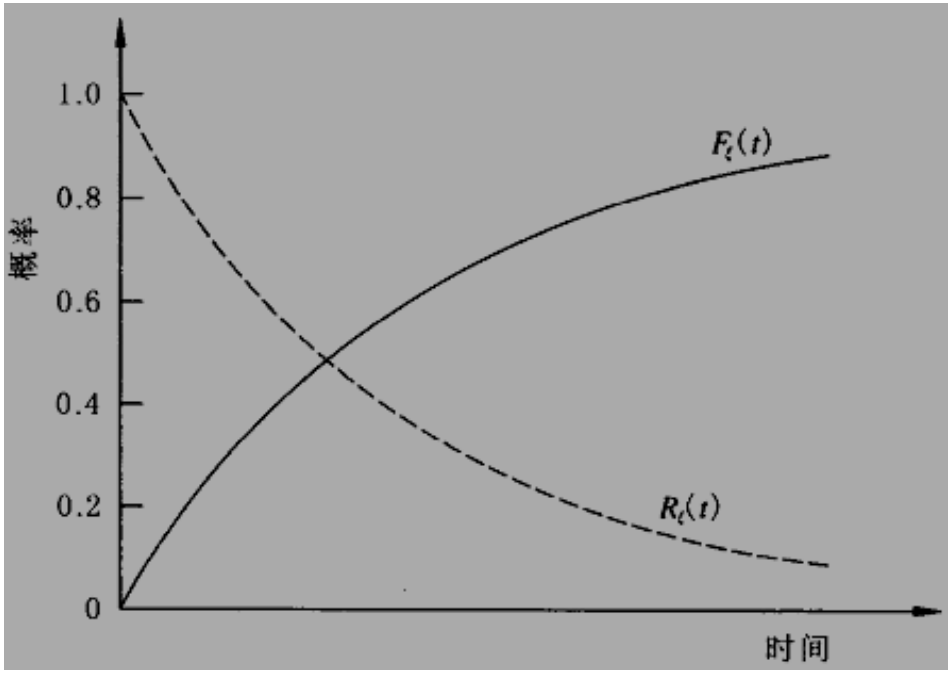
则 $R(t) = P(T>t)$

② 不可靠度/累积故障分布函数　F(t)

$F(t) = P(T \leq t)$

$R(t) + F(t) = 1$

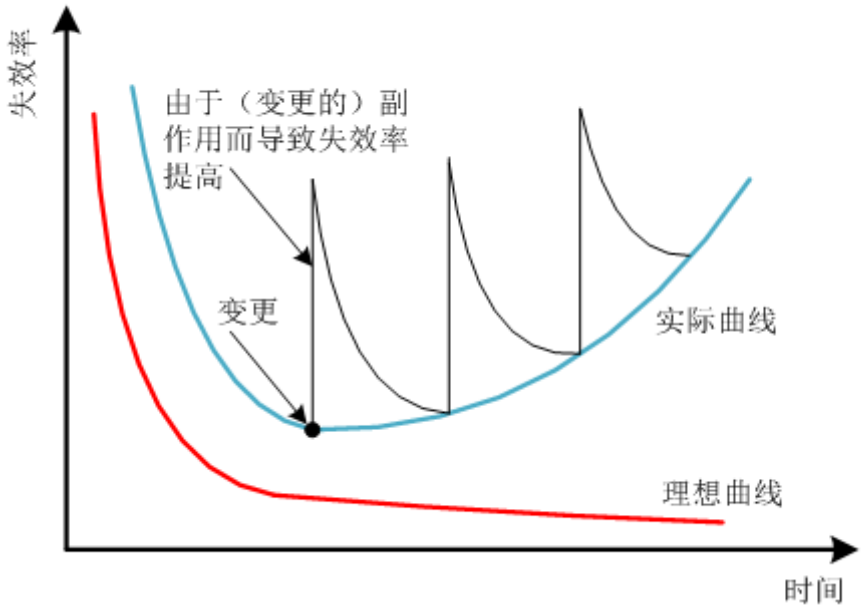
软件可靠性度量(续)



软件可靠性度量(续)

③ 故障/失效率 λ(t)：软件在t时刻没有发生故障/失效的条件下，在单位时间内发生故障/失效的概率

$$\lambda(t)=\lim_{\Delta t\to 0}\frac{P(t+\Delta t\geq T\geq t\,|\,T>t)}{\Delta t}$$



软件可靠性度量(续)

④ **平均失效前时间MTTF：当前时间到下一次失效时间的均值**

$$MTTF = \int_0^{\infty} R(t)dt$$

$$MTTF = \sum_{k=1}^{N} \frac{t_k}{N}$$

软件可靠性度量(续)

⑤ **平均失效间隔时间MTBF：**

产品在使用过程中发生了N次故障，每次故障修复后又重新投入使用，测得其每次工作持续时间为t1, t2, ….，则平均失效间隔时间时间为：

$$MTBF = \sum_{k=1}^{N} \frac{t_k}{N}$$

例：某产品工作了1万小时，共发生5次故障，该产品的MTBF是？

软件可靠性度量(续)

⑥ **平均失效修复时间MTRF：系统出现故障后到恢复正常工作时的平均时间**

$$MTRF = \sum_{k=1}^{N} \frac{t_k}{N}$$

例：某产品从工作到停止共1.2万小时，其中实际工作时间1万小时共发生5次故障，该产品的MTRF是？

软件可靠性度量(续)

- 软件可靠性模型：（Software Reliability Model）：为预计或估算软件的可靠性所建立的可靠性框图和数学模型。
- 建立可靠性模型的目标：定量预计、分配、估算和评价复杂系统的可靠性。

软件可靠性度量(续)

- 软件可靠性建模：根据软件过去的故障行为建立软件可靠性数学模型的过程
  - 通过度量获得历史数据
  - 对故障数据进行分析，拟合成概率分布函数
  - 对拟合函数进行参数分析
  - 确定所期望的可靠性度量值并预测可能的故障行为

软件可靠性度量(续)

- 可靠性模型的现状
  - 数量多，表达形式不同，且新模型不断发表
  - 适用于某些软件故障数据集合的模型，不适用于其他故障数据集合
  - 同一模型适用于软件项目开发的某个阶段，不适用于其他阶段
  - 某些模型针对特定的软件或过程而开发

模型分类

模型统一

软件可靠性度量(续)

- 软件可靠性模型
  - ① 给定时间间隔内的失效数模型
  - ② 两相邻失效间的时间间隔模型

软件可靠性度量(续)

- **Musa和Okumoto根据软件可靠性模型的五种特征进行分类：**
  - 时间域(Time Domain)：按时钟时间、执行时间（或ＣＰＵ时间）分类；
  - 类别(Category)：根据软件在无限的时间内运行时可能经历的故障数是有限的还是无限的进行分类；
  - 型式(Type)：根据软件在运行时间t时的失效数分布来分类。其中主要考虑泊松分布和二项式分布。
  - 种类(Class)：根据软件发生故障的故障密度对时间的函数形式分类（仅对有限故障类）。
  - 族(Family)：根据软件的故障密度对它的期望故障数的函数形式分类（仅对无限故障类）。

软件可靠性度量(续)

- **模型统一**

  为软件可靠性模型所涉及的范围、结构等建立一个统一的框架，从中反映出现有模型在理论和实际应用间的差别，并能在对象和特性分析的基础上，为用户提供合适的模型

软件可靠性度量(续)

- **Musa模型**
  - 基本模型
  - 对数模型
- **Goel-Okumoto模型**

软件可靠性度量(续)

Musa基本模型

- **假设**
  - 软件中存在的差错对故障率有同等的贡献
  - 每纠正一个差错，故障率均匀减少
  - 软件中固有的故障总数是一个有限的数，但不一定固定
- **由以下参数确定**

$$\lambda_p = \lambda_0 e^{-\frac{\lambda_0 \tau}{ETV}}$$

- **贡献**

  确定要达到某个可靠性目标还必须要发现或检测出的差错数以及仍需要的运行时间

软件可靠性度量(续)

Musa基本模型

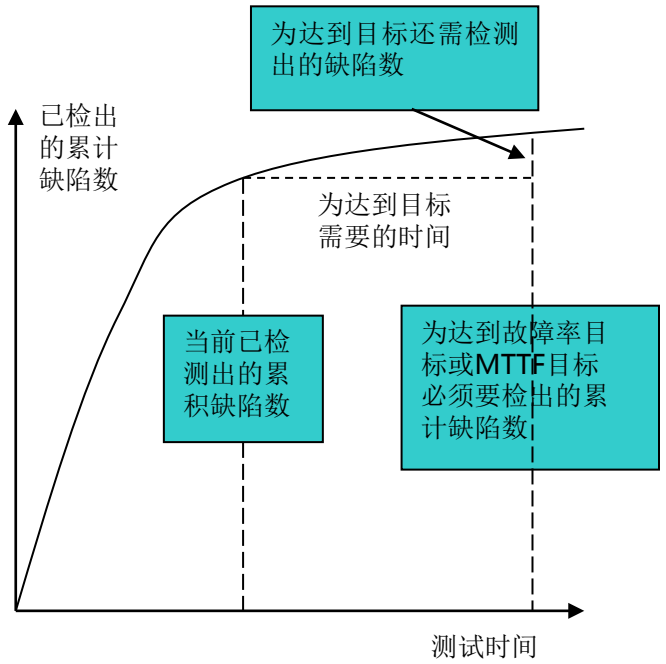
设要达到的可靠性目标 λ_f

需发现的故障数 N_f 为:

$$N_f = \frac{1}{\ln(\lambda_p / \lambda_f)}$$

为达到这一目标所需时间 T_f 为:

$$T_f = \frac{1}{\lambda_p - \lambda_f}$$



软件可靠性度量(续)

Musa对数模型

- 假设
  - 软件中存在的差错对故障率的贡献是不同的
  - 差错发现得越早，故障率随时间减少得越大
  - 软件中固有的故障总数是无限
- 由以下参数确定

$$\lambda_p = \frac{\lambda_0}{\lambda_0 \theta \tau + 1}$$

- 为达到目标仍需要的执行时间为

$$T_f = \frac{\lambda_p \lambda_f}{\lambda_p - \lambda_f}$$

软件可靠性度量(续)

Goel-Okumoto模型

- 假设
  - 差错对时间的分布是非时齐的
  - 当差错发现时不一定会立即取消，还可能引起另外的差错
- 由以下参数确定

$$\lambda(t) = abe^{-bt}$$

- 可估计：

$$ED(t) = a - ae^{-bt}$$

软件可靠性度量(续)

模型	假设	适用阶段	难易度
Musa基本模型	①有限固有缺陷数 ②常数故障率 ③指数分布	集成测试后	E
Mssa对数模型	①无限固有缺陷数 ②对数分布 ③故障率随时间变化	单元测试到系统测试	E
Goel-Okumoto	①非时齐缺陷分布 ②缺陷可能因修复而产生 ③指数、Weibull分布	集成测试后	M

软件可靠性度量(续)

- 可靠性模型的评价标准
  - 基于合理的假设
  - 预测的有效性
  - 模型实现的可操作性

软件可靠性度量(续)

定量的评价准则

- **模型拟合性**

  模型估计出的失效数据与实际失效数据的吻合程度

- **模型预计有效性**

  从预测的角度来反映模型评估的有效性

- **模型偏差**

  通过u-结构图判断预测与实际之间的误差

- **模型偏差趋势**

  通过y-结构图探测u-结构图中难以发现的预测与实际数据之间的偏差

- **模型噪声**

  模型本身给模型预测引入噪声的程度

软件可靠性度量(续)

软件可靠性评测：运用统计技术对软件可靠性测试和系统运行期间采集的软件失效数据进行处理并评估软件可靠性的过程。

- **针对软件开发过程**

① 更快速地找出对可靠性影响最大的错误

② 估计软件当前的可靠性

③ 预计软件要达到相应的可靠性水平所需要的时间和测试量

- **针对最终软件产品**

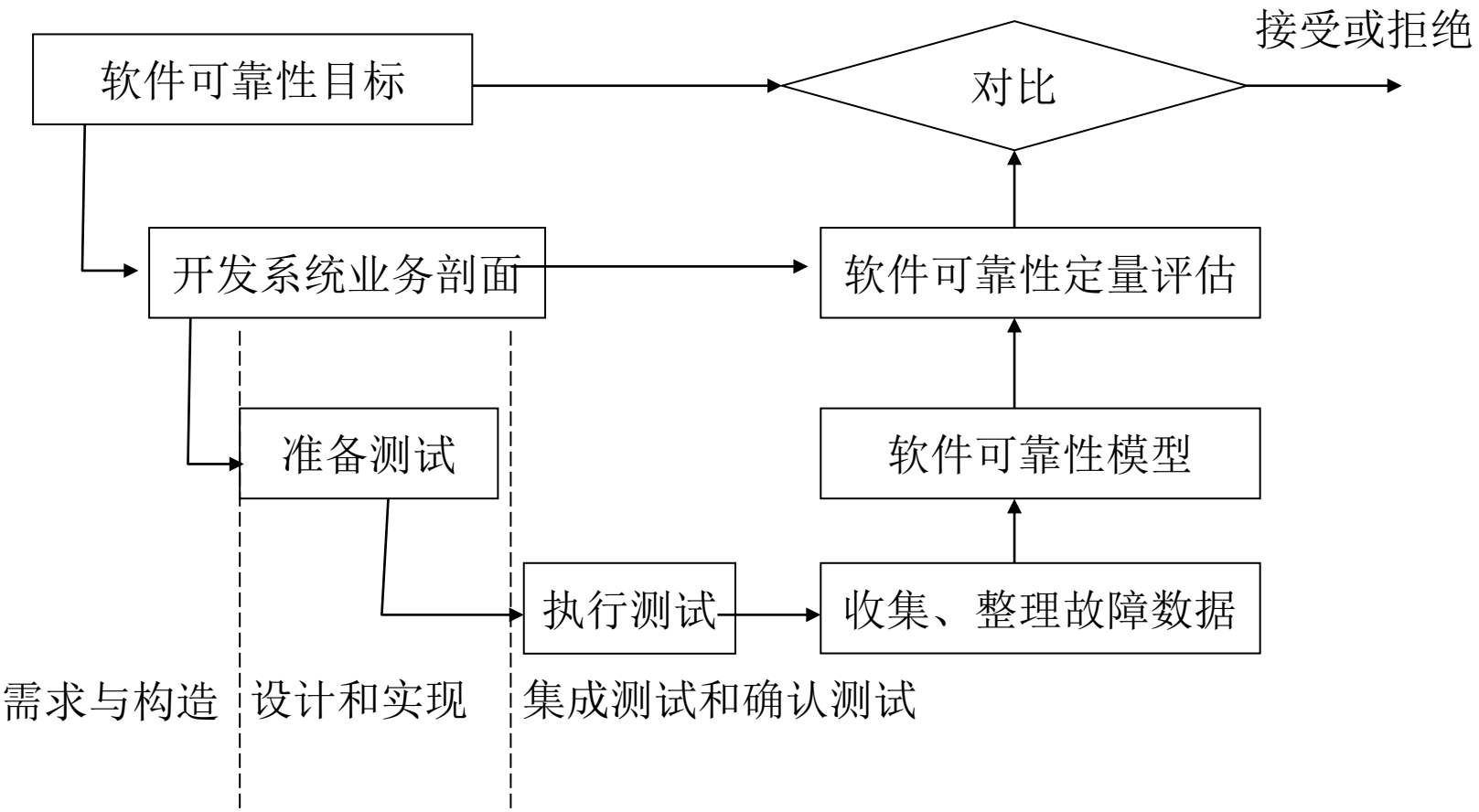
① 确认软件的执行与需求的一致性

② 确定最终软件产品所达到的可靠性水平

软件可靠性度量(续)

- **软件可靠性评测也是对软件测试过程的一种完善**
  - 一般测试是面向错误的测试
  - 可靠性测试需要在软件的预期使用环境下进行

软件可靠性度量(续)


软件可靠性目标
对比
接受或拒绝
开发系统业务剖面
软件可靠性定量评估
准备测试
软件可靠性模型
执行测试
收集、整理故障数据
需求与构造
设计和实现
集成测试和确认测试

软件可靠性度量(续)

软件可靠性评测的步骤

1. 可靠性目标的确定
2. 运行剖面的开发
3. 测试的计划与执行
4. 测试结果的分析

软件可靠性度量(续)

1. 可靠性目标的确定

- 可靠性目标：客户对软件可靠性满意程度的期望
  可靠度、故障强度、MTTF等
- 建立定量的可靠性目标需要权衡

软件可靠性度量(续)

2. 运行剖面的开发

- **目的：使测试真实地反映软件的使用情况**
- **过程**
  1. 在运行剖面中为每个元素赋予发生概率值和关键因子
  2. 分配测试资源、挑选和生产测试用例

软件可靠性度量(续)

3. 测试的计划与执行

- **针对软件过程和最终产品的可靠性测试，在测试计划的制定上有所不同**
  - 可靠性增长测试：“测试-排错-新版本”，测试执行多次
  - 可靠性验证测试：测试执行一次
- **测试执行阶段：收集可靠性故障数据**

软件可靠性度量(续)

4. 测试结果的分析与反馈

- **从不同的角度理解故障数据**
  - 可靠性增长测试：根据测试结果估计故障强度
  - 可靠性验证测试：根据测试结果确定软件系统是被接受还是被拒绝

软件可靠性度量(续)

- **可靠性测试的目的**
  - 正确地完成规定的功能
  - 满足性能要求
  - 不完成没有规定的功能
  - 提供运行中的故障数据

软件可靠性度量(续)

- **测试准备和执行的注意点**
  - 执行不同的业务剖面
  - 质量管理部门要参与到测试中
    - 规定测试要求
    - 评审和批准测试计划和程序
    - 评审和批准需求的认证方法和技术
    - 提供环境及操作人员
    - 要求独立的测试组织

软件可靠性度量(续)

提高软件可靠性的方法和技术

1. 建立以可靠性为核心的质量标准
2. 选择开发方法
3. 软件重用
4. 使用开发管理工具
5. 加强测试
6. 可靠性设计

建立以可靠性为核心的质量标准

- **软件质量**
  - 产品质量和过程质量
  - 静态质量和动态质量
- **建立时间：软件项目规划和需求分析阶段**
- **度量所定的质量标准应达到的目的：**
  ① 明确划分开发过程，通过质量检验的反馈作用确保差错及早排除并保证一定的质量
  ② 在各开发过程中实施进度管理，产生阶段度量评价报告，对不合要求的产品及早采取对策

选择开发方法

- 建议采用面向对象的方法，借鉴Parnas和瑞理模式的思想，在开发过程中结合使用其他方法，吸取其他方法的优点

选择开发方法(续)

- **Parnas方法：在概要设计时预先估计未来可能发生变化，提出了信息隐藏的原则以提高软件的可靠性和可维护性**
  - 划分模块时，将可能发生变化的因素隐藏在某个模块内部，使其他模块与此无关
  - 接近硬件的模块要对硬件行为进行检查
  - 输入模块对输入数据进行合法性检查
  - 加强模块间检查

选择开发方法(续)

- **瑞理开发方法**
  - 面向对象
  - 螺旋式上升
  - 管理与控制
  - 高度自动化

软件重用

- **软件重用包括三个方面内容的重用：**
  - 开发过程重用，指开发规范、各种开发方法、工具和标准等。
  - 软件构件重用，指文档、程序和数据等。
  - 知识重用，如相关领域专业知识的重用。

使用开发管理工具

- **开发一个大的软件系统，离不开开发管理工具**
- **如PVCS软件开发管理工具：**
  - 规范开发过程；
  - 自动创造完整的文档；
  - 管理软件多重版本；
  - 管理和追踪开发过程中危及软件质量和影响开发周期的缺陷和变化；
  - 管理人员。

加强测试

- **为了最大限度地除去软件中的差错，要对软件进行尽可能完备的测试**
- **测试过程：**
  - 测试前：确定测试标准、规范
  - 测试过程：建立完整的测试文档
  - 测试后：查错，确保任何错误以及对错误的动作及时归档

加强测试(续)

- **测试规范：**
  - 测试设计规范：测试方法，设计及其有关测试所包括的特性，测试用例，测试规程，判定准则。
  - 测试用例规范：输入的具体值及预期输出结果，各种限制。
  - 测试规程规范：测试步骤。

加强测试(续)

- **测试的方法：**
  - 走查（Walk-through）
  - 机器测试
  - 程序证明或交替程序表示。
  - 模拟测试
  - 设计审查
  - 可靠性测试

可靠性设计

- 避错设计
- 查错设计
- 改错设计
- 容错设计

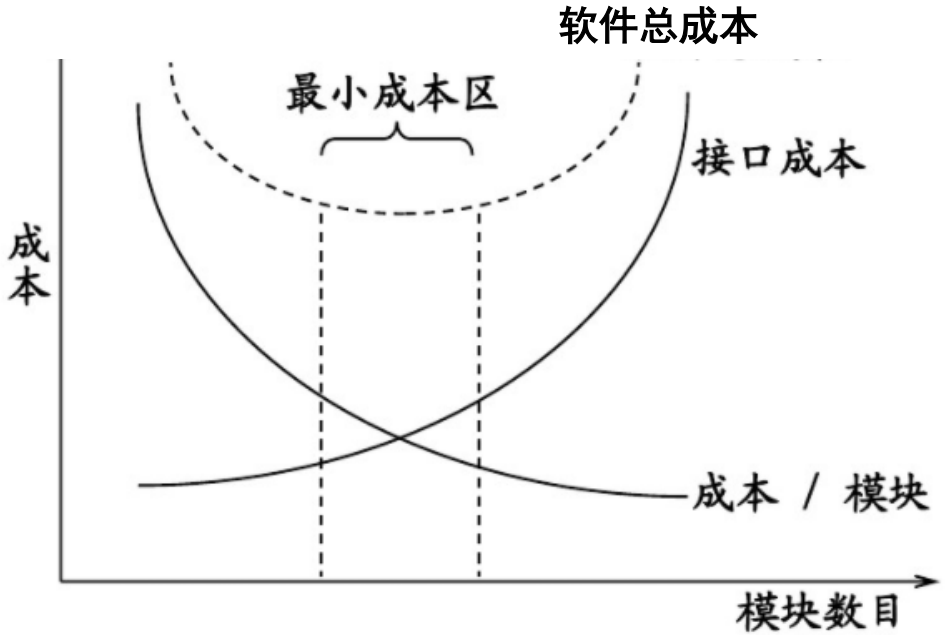
可靠性设计(续)

- 避错设计

1. 模块化

把程序划分为若干个模块，每个模块完成一个子功能，把这些模块集成起来组成一个整体，可以完成指定的功能

- 模块的规模
- 模块的数量

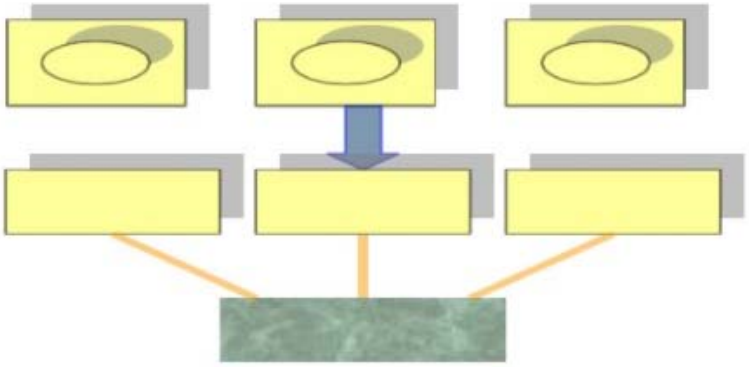


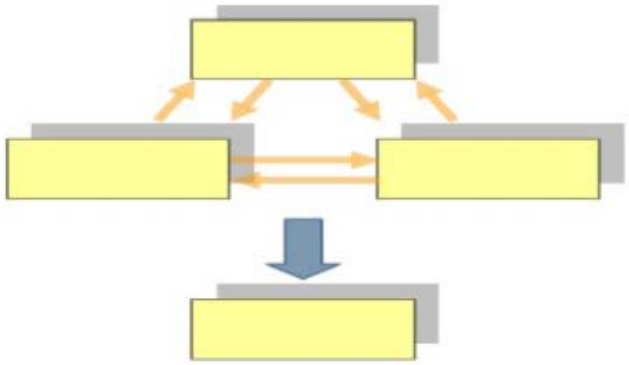
可靠性设计(续)

- 避错设计

2. 模块独立性
   - 耦合性：衡量不同模块间互相依赖的程度
   - 内聚性：衡量一个模块内部各元素间结合的紧密程度

   低耦合、高内聚





可靠性设计(续)

- 避错设计

3. 信息隐蔽

   设计模块，使其内部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

4. 慎重使用容易引入缺陷的结构和技术
   - 指针
   - 动态内存分配
   - 并行
   - 递归
   - 继承
   - 别名

可靠性设计(续)

- 查错设计

1. 被动式检测

   发生错误以后检测

2. 主动式检测

   周期性地搜查整个程序或数据

可靠性设计(续)

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

可靠性设计(续)

- 改错设计

  在设计中，赋予程序自我改正错误、减少错误危害程度的能力的一种设计方法

- 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
- 故障隔离技术：主要思想是权限最小化原则

  要求对过程和数据加以严格的定义和限制，以限制故障的蔓延

可靠性设计(续)

- 容错设计

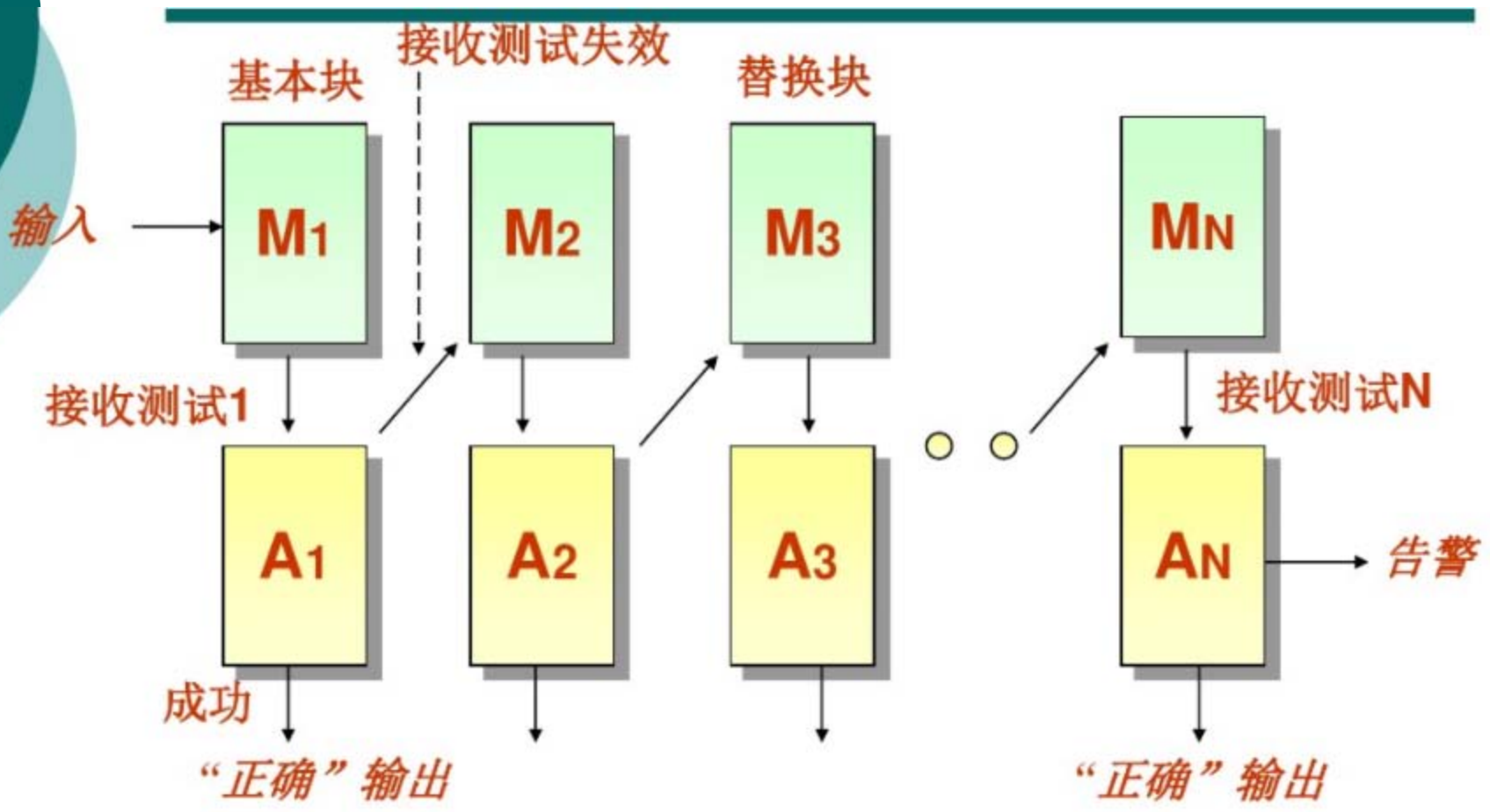
1. N-版本程序设计：静态冗余
2. 恢复块技术：动态冗余

可靠性设计(续)

- 容错设计

1. N-版本程序设计：
   - 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
   - 对于一个给定的功能，由N个不同的设计组独立编制出N个不同的程序，然后同时在N个机器上运行并比较运行的结果
     - 如果结果一致，认为结果正确
     - 如果结果不尽相同，按多数表决或其它预先制定的策略，判定结果的正确性

可靠性设计(续)



恢复块技术

5.3小结

- **对过程、项目及产品的特定属性的测量被用于计算软件度量。分析这些度量可产生指导管理及技术行为的指标**
  - 改善软件过程
  - 辅助软件项目的计划、跟踪及控制
  - 评估软件产品的质量
- **软件可靠性是软件质量特性中重要的固有特性和关键因素。**

  软件可靠性不但与软件中存在的缺陷有关，而且与系统输入和系统使用有关