

# 软件质量与评测技术

## Software Quality & Evaluation Technology

计算机学院 单纯  
[sherryshan@bit.edu.cn](mailto:sherryshan@bit.edu.cn)  
2025年11月

# 软件测试实践

## Software Testing Practice

计算机学院 单纯

[sherryshan@bit.edu.cn](mailto:sherryshan@bit.edu.cn)

2025年12月1日

## 2.5 测试计划内容与编制

- 2.5.1 测试计划内容
- 2.5.2 测试项目的计划过程
- 2.5.3 制定有效的测试计划

## 2.5.1 测试计划内容

- 软件测试计划是指导测试过程的纲领性文件，描述测试活动的范围、方法、策略、资源、任务安排和进度等，并确定测试项、哪些功能特性将被测试、哪些功能特性将无需测试，识别测试过程中的风险
- 内容主要集中在测试目标和需求说明、测试工作量估算、测试策略、测试资源配置、进度表、测试风险等

# GBT 9386-2008 (1)

## ■ 测试计划应有如下结构

- 测试计划标识符
- 引言
- 测试项
- 要测试的特征
- 不要测试的特征
- 方法
- 测试项通过准则

# GBT 9386-2008 (2)

## ■ 测试计划应有如下结构（续）

- 暂停准则和恢复要求
- 测试交付项
- 测试任务
- 环境要求
- 职责
- 人员配备和培训要求

# GBT 9386-2008 (3)

## ■ 测试计划应有如下结构（续）

- 进度
- 风险和应急
- 批准

## 2.5.2 测试项目的计划过程（1）

一、 概述 .....	1
1.1 文档目的 .....	1
1.2 项目背景 .....	1
1.3 定义和缩写 .....	2
1.4 参考文档 .....	2
二、 测试目标 .....	2
三、 测试范围 .....	2
四、 测试策略与方法 .....	3
五、 准入准出条件 .....	3
5.1 准入条件 .....	3
5.2 准出条件 .....	4
六、 测试资源 .....	4
6.1 测试工具 .....	4
6.2 培训需求 .....	4
七、 测试任务与进度安排 .....	4
八、 风险管理 .....	5
九、 测试管理方式 .....	6
十、 测试交付件 .....	6

计划不是文档，是一个过程

## 2.5.2 测试项目的计划过程（2）

- 确认测试目标、范围和需求
- 识别测试风险，制订相应的测试策略
- 对测试任务和工作量进行估算
- 确定所需的时间和资源
- 进度安排和资源分派，包括团队角色、责任和培训
- 测试阶段划分，包括阶段性任务和成果
- 计划评审与批准
- 跟踪和控制机制

## 2.5.2 测试项目的计划过程 (3)



## 2.5.3 制定有效的测试计划（1）

- 确定测试项目的任务，清楚测试范围和测试目标
- 尽量识别出各种测试风险，制定出相应的对策
- 让所有合适的相关人员参与测试项目的计划制定
- 客观、准确、留有余地
- 制定测试项目的输入、输出和质量标准

## 2.5.3 制定有效的测试计划（2）

- 识别出可变因素，建立变化处理和控制的流程规则
- 不要忽视技术上的问题
- 要对测试的公正性、遵照的标准做一个说明
- 测试计划应简洁、易读并有所侧重

# 测试输入/输出标准（1）

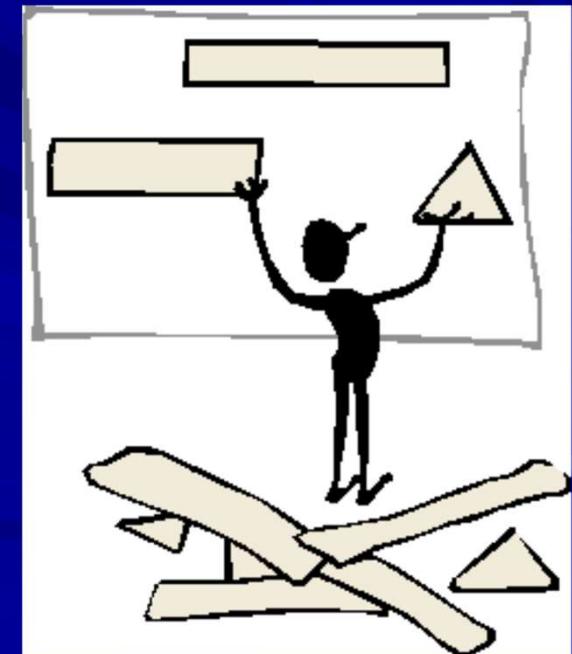
## ■ 测试的输入标准

- 整体项目计划框架
- 需求规格说明书
- 技术知识或业务知识
- 标准环境
- 设计文档
- 足够的资源
- 人员组织结构

# 测试输入/输出标准（2）

## ■ 测试的输出标准

- 测试执行标准
- Bug描述和处理标准
- 文档标准和模板
- 测试分析、质量评估标准等



# 进入/退出标准

- 进入准则：对软件测试切入点的确立，即满足什么条件，才启动测试
- 退出标准：满足某个阶段结束/里程碑达到的事先定义的要求
- 软件测试每个阶段需要控制进入/退出标准以保障测试的质量

# 示例：系统测试结束标准

- 对于非严格系统，可以采用“基于测试用例”的准则
  - 功能性测试用例通过率达到100%
  - 非功能性测试用例通过率达到95%
- 对于严格系统，应当补充“基于缺陷密度”的规则
  - 相邻n个CPU小时内“测试期缺陷密度”全部低于某个值m。具体值根据项目的类型来确定

## 2.6 小结

- 从产品需求到测试需求，逐层分析：业务、功能、场景、用例
- 清楚测试上下文（目标、条件、环境等）
- 测试覆盖率是起点也是终点
- 风险识别、分析与防范
- 有效制定测试计划
- 计划评审也不可忽视



# 项目测试实践（1）

## ■ 测试计划（过程控制，1分）

- 小组讨论，重新明确测试目标，包括功能+非功能性要求
- 分析哪些要测试、哪些可以不测，即确定测试的范围
- 对测试范围进行分解，列出测试项（包括功能测试项和专项测试项）
- 用WBS或其他方法来估算每一个测试项的工作量，并进行汇总

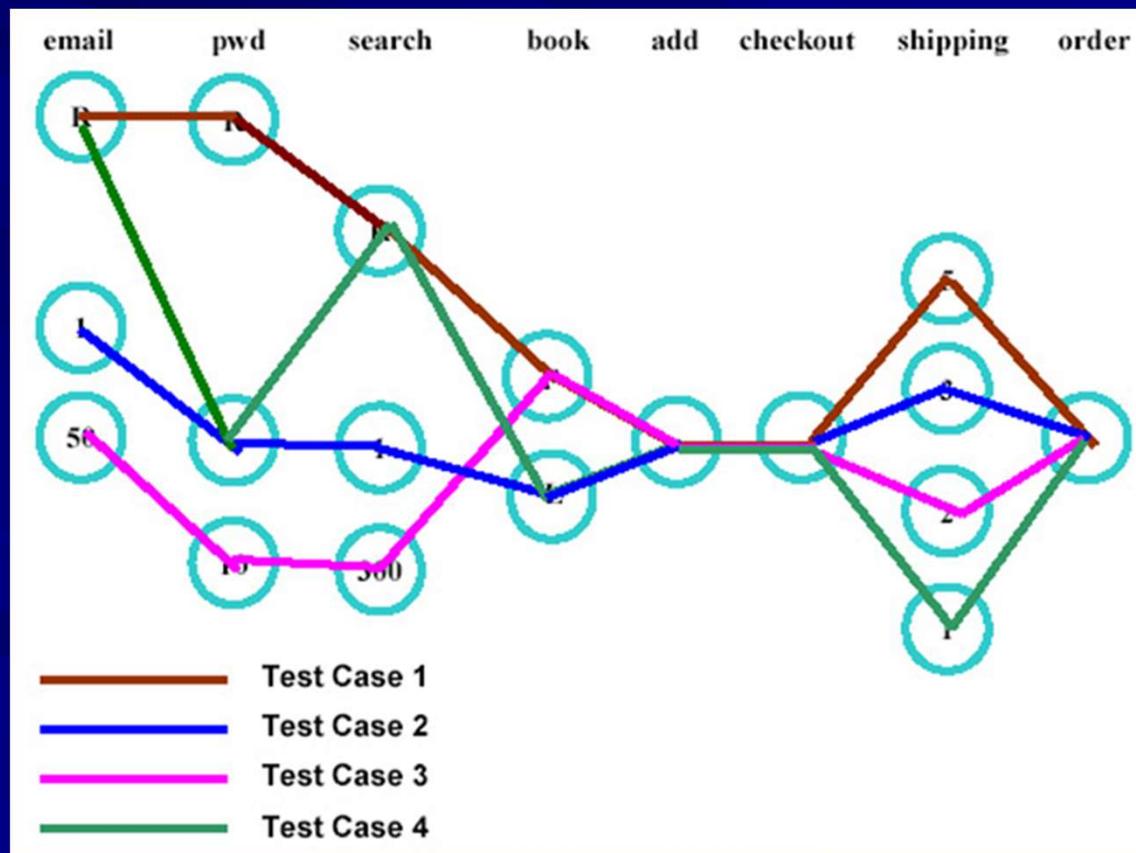
# 项目测试实践（2）

## ■ 测试计划（过程控制，1分，续）

- 针对这些测试内容，识别出其中的测试风险，并分析和列出主要的测试风险
- 一起讨论，看看能否针对这些测试风险，找到相应的测试对策（策略）
- 将前面的内容汇总起来，更新测试计划

# 3. 设计和维护测试用例

- 3.1 测试用例构成及其设计
- 3.2 测试用例的组织和跟踪



# 3.1 测试用例构成及其设计

- 3.1.1 测试用例的重要性
- 3.1.2 测试用例设计书写标准
- 3.1.3 测试用例设计的考虑因素
- 3.1.4 测试用例设计的基本原则

# 测试用例（1）

## ■ 什么是测试用例

- 测试用例是可以独立进行测试执行的最小单元
- 测试内容的一系列情景和每个情景中必须依靠输入和输出，而对软件的正确性进行判断的测试文档，称为测试用例
- 编写测试用例就是将软件测试的行为活动转化为规范化的文档

# 测试用例（2）

## ■ 测试用例实例

### 【测试用例 1】

测试目标：验证输入错误的密码是否有正确的响应。

测试环境：Windows XP 操作系统和浏览器 Firefox 3.0.3

输入数据：用户邮件地址和口令

步骤：

1. 打开浏览器
2. 点击页面右上角的“登录”链接，出现登录界面。
3. 在电子邮件的输入框中输入：test@gmail.com
4. 在口令后面输入：xxxabc
5. 点击“登录”按钮

期望结果：

登录失败，页面重新回到登录页面，并提示“用户密码错误”。

# 测试用例 (3)

## ■ 测试用例的元素

字段名称	注释
标识符	唯一标识该测试用例的值，自动生成
测试项	测试的对象，可以从软件配置库中选择
测试目标	从固定列表中选择一个
测试环境要求	可从列表中选择，如果没有，则直接输入新增内容
前提	事先设定、条件限制，如已登录、某个选项已选上
输入数据	输入要求说明、或数据列举
操作步骤	按 1., 2., ... 操作步骤的顺序，准确详细地描述。
期望输出	
所属模块	模块标识符。
优先级	1, 2, 3 (1 - 优先级最高)
层次	0, 1, 2, 3 (0 - 最高层)
关联的测试用例	上层(父)用例的标识符。
执行时间	分钟
自动化标识	Ture, False
关联的缺陷	缺陷标识符列表。

### 3.1.1 测试用例的重要性（1）

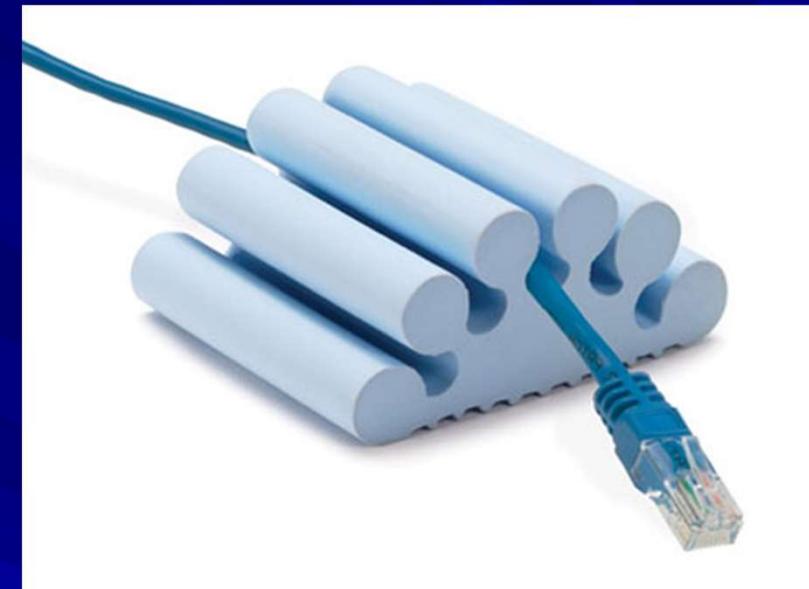
- 如何以最少的人力、资源投入，在最短的时间内完成测试，发现软件系统的缺陷，保证软件的优良品质是软件公司探索和追求的目标
- 软件测试是有组织性、步骤性和计划性的，为了能将软件测试的行为转换为可管理的、具体量化的模式，需要创建和维护测试用例

### 3.1.1 测试用例的重要性（2）

- 测试用例是测试工作的指导，是软件测试的必须遵守的准则，更是软件测试质量稳定的根本保障

# 测试用例的作用

- 有效性
- 可复用性
- 易组织性
- 客观性
- 可评估性和可管理性
- 知识传递
- 重要参考依据， 提高测试质量



## 3.1.2 测试用例设计书写标准

- 标识符 (Identification)
- 测试项 (Test Items)
- 测试环境要求
- 输入标准 (Input Criteria)
- 输出标准 (Output Criteria)
- 测试用例之间的关联

# 实例（1）

【示例 2：书写规范的测试用例】

**ID：**LG0101002

**用例名称：**验证输入错误的密码后是否提示正确。

**测试项：**用户邮件地址和口令

**环境要求：**Windows XP SP2 和 Firefox 3.0.3

**参考文档：**软件规格说明书 SpecLG01.doc

**优先级：**高

**层次：**2 （即 LG0101 的子用例）

**依赖的测试用例：**LG0101001

**步骤：**

1. 打开浏览器
2. 点击页面右上角的“登录”链接，出现登录界面。
3. 在电子邮件的输入框中输入：test@gmail.com
4. 在口令后面输入：xxxabc
5. 点击“登录”按钮

**期望结果：**

登录失败，页面重新回到登录页面，并提示“用户密码错误”。

# 实例（2）

用例编号	TC_F_QXLC_001
用例名称	缺陷提交至关闭业务流程
测试内容	测试缺陷提交至缺陷关闭是否正确实现闭环管理。
前置条件	<ol style="list-style-type: none"><li>系统运行正常；</li><li>测试人员、开发人员均可以正常登录系统，相关权限配置正确；</li><li>发现该缺陷的测试执行已经完成。</li></ol>

操作步骤		
序号	操作步骤/输入数据	预期结果
1	测试人员登录系统后，点击【缺陷】。进入缺陷列表页面。	
2	缺陷列表页面中，测试人员点击【新建缺陷】。进入新建缺陷界面，界面中，必填字段均以红色字体显示。	
3	新建缺陷界面中，正确输入缺陷总结、发现版本、发现活动、缺陷类型、严重程度、缺陷状态、优先级、缺陷描述等属性信息，点击【提交】。	缺陷提交成功，且该缺陷为“新建”状态。
4	开发人员登录系统，进入缺陷列表页面，选择【新建】状态的缺陷，点击【缺陷 Id】链接。	正确进入缺陷详细信息页面。
5	开发人员修复缺陷后，在缺陷详细信息页面中，开发人员修改缺陷状态为【已修正】。	缺陷状态修改成功，缺陷状态修改成功，该缺陷在列表页面中，状态被更新为【已修正】。
6	测试人员登录系统，进入缺陷列表页面，选择【已修正】状态的缺陷，点击【缺陷 Id】链接。	正确进入缺陷详细信息页面。
7	测试人员经回归测试验证缺陷修复后，在缺陷详细信息页面中，将缺陷状态修改为【已关闭】。	缺陷状态修改成功，缺陷状态修改成功，该缺陷在列表页面中，状态被更新为【已关闭】。

评判标准：实际结果与预期结果相符，判定该用例通过；否则为不通过。

# 良好测试用例的特征

- 可以最大程度地找出软件隐藏的缺陷
- 可以最高效率地找出软件缺陷
- 可以最大程度地满足测试覆盖要求
- 既不过分复杂、也不能过分简单
- 使软件缺陷的表现可以清楚的判定
- 测试用例包含期望的正确的结果
- 待查的输出结果或文件必须尽量简单明了
- 不包含重复的测试用例
- 测试用例内容清晰、格式一致、分类组织

### 3.1.3 测试用例设计的考虑因素

- 具有代表性、典型性
- 寻求系统设计、功能设计的弱点
- 测试用例需要考虑到正确的输入，也需要考虑错误的或者异常的输入，以及需要分析怎样使得这样的错误或者异常能够发生
- 考虑用户实际的诸多使用场景

## 3.1.4 测试用例设计的基本原则

- 尽量避免含糊的测试用例
- 尽量将具有相类似功能的测试用例抽象并归类
- 尽量避免冗长和复杂的测试用例

# 单个测试用例的质量要求

- 具有可操作性
- 具备所需的各项信息
- 各项信息描述准确、清楚
- 测试目标针对性强
- 验证点完备，而且没有太多的验证点
- 没有太多的操作步骤，例如不超过7步
- 符合正常业务惯例

# 测试用例的粒度

粗粒度

输入不同的用户名和口令，其结果要满足设定的要求  
如用户名、口令判断正确与否等

VS  
•

细粒度

3.1 用户名输入为空。

属性	值
编号	MSG0001
显示的页面	ErrorPage0001
出现条件	当用户输入的用户名为空而试图登录
提示信息	错误：请输入用户名。

3.2 密码为空。

属性	值
编号	MSG0002
显示的页面	ErrorPage0002
出现条件	当用户密码输入为空且没有出现 WMSG001 的提示信息。
提示信息	错误：请输入密码。

# 整体测试用例的质量要求（1）

## ■ 覆盖率

- 依据特定的测试目标的要求，尽可能覆盖所有的测试范围、功能特性和代码

## ■ 易用性

- 测试用例的设计思路清晰、组织结构层次合理，测试用例操作的连贯性好，使单个模块的测试用例执行顺畅

# 整体测试用例的质量要求（2）

## ■ 易维护性

- 应该以很少的时间来完成测试用例的维护工作，包括添加、修改和删除测试用例。易用性和易读性，也有助于易维护性

## ■ 粒度适中

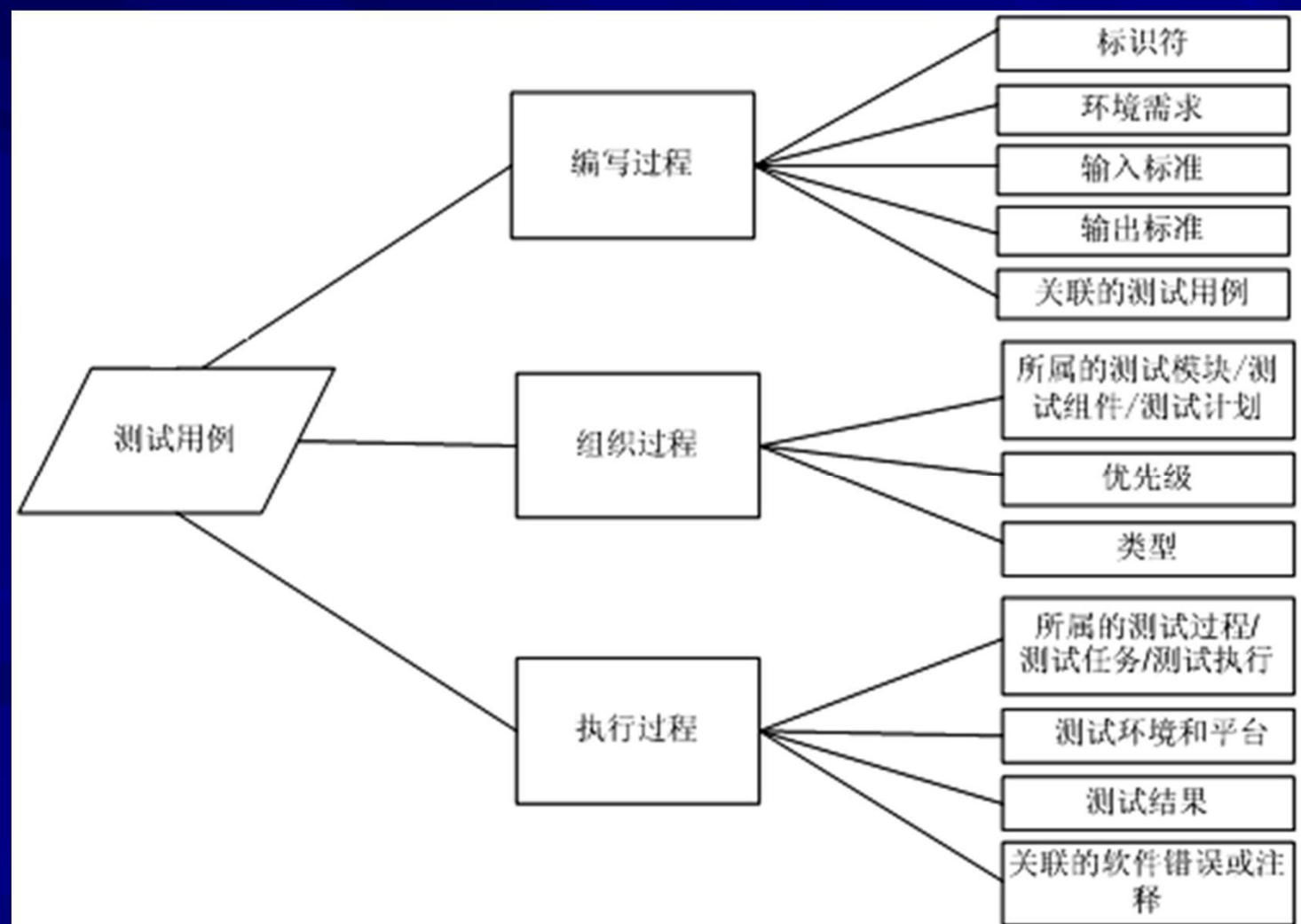
- 既能覆盖各个特定的场景，保证测试的效率；又能处理好不同数据输入的测试要求，提高测试用例的可维护性

## 3.2 测试用例的组织和跟踪

- 3.2.1 测试用例的属性
- 3.2.2 测试套件及其构成方法
- 3.2.3 跟踪测试用例
- 3.2.4 维护测试用例
- 3.2.5 测试用例的覆盖率

### 3.2.1 测试用例的属性

#### ■ 各个阶段所表现的测试用例属性



# 一些属性说明 (1)

## ■ 目标性

- 包括功能性、性能、容错性、数据迁移等各方面的测试用例

## ■ 所属的范围

- 属于哪一个组件或模块

## ■ 关联性

- 和软件产品特性相联系

# 一些属性说明 (2)

## ■ 阶段性

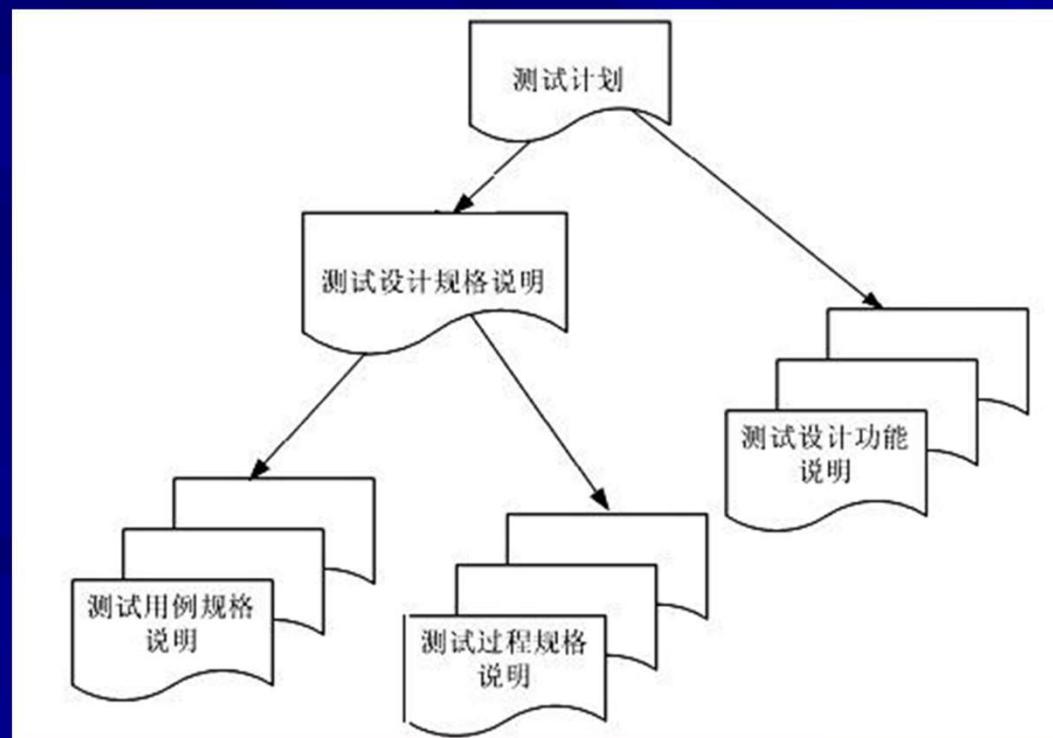
- 属于单元测试、集成测试、系统测试、验收测试中的某一个阶段

## ■ 时效性

- 不同的版本所适用的测试用例可能不相同

## 3.2.2 测试套件及其构成方法

- 建立合适的、可扩展的测试用例框架，从而借助这个框架能有效地组织众多的测试用例，包括对测试用例的分类、清晰的层次结构等



# 实例

## 示例：体现层次结构的测试用例

用户身份合法性验证 (父 case)

    ↳ 用户名验证 (子 case)

        ↳ 正常用户名的输入 (孙 case)

        ↳ 含有特殊字符的用户名

        ↳ 字母大小写无关性测试

        ↳ 非法用户名

    □ 令验证

        ↳ 正常口令

        ↳ 含有特殊字符的口令

        ↳ 字母大小写敏感性测试

        ↳ 口令有效期验证

        ↳ 口令保存

        ↳ 忘记口令后找回口令功能

# 测试用例套件

- 测试套件是由一系列测试用例并与之关联的测试环境组合而构成的集合，以满足测试执行的特定要求。通过测试套件，将服务于同一个测试目标、特定阶段性测试目标或某一运行环境下的一系列测试用例有机地组合起来
  - 按程序功能模块组织
  - 按测试用例的类型组织
  - 按测试用例的优先级组织

# 测试类型与测试用例设计

## 根据测试类型设计

功能测试

易用性测试

配置测试

压力测试

回归测试

界面测试

文档测试

国际化测试

- 测试用例1
- 测试用例2
- 测试用例3

## 根据程序功能模块设计

安装/卸载测试

联机注册测试

联机帮助测试

文件操作测试

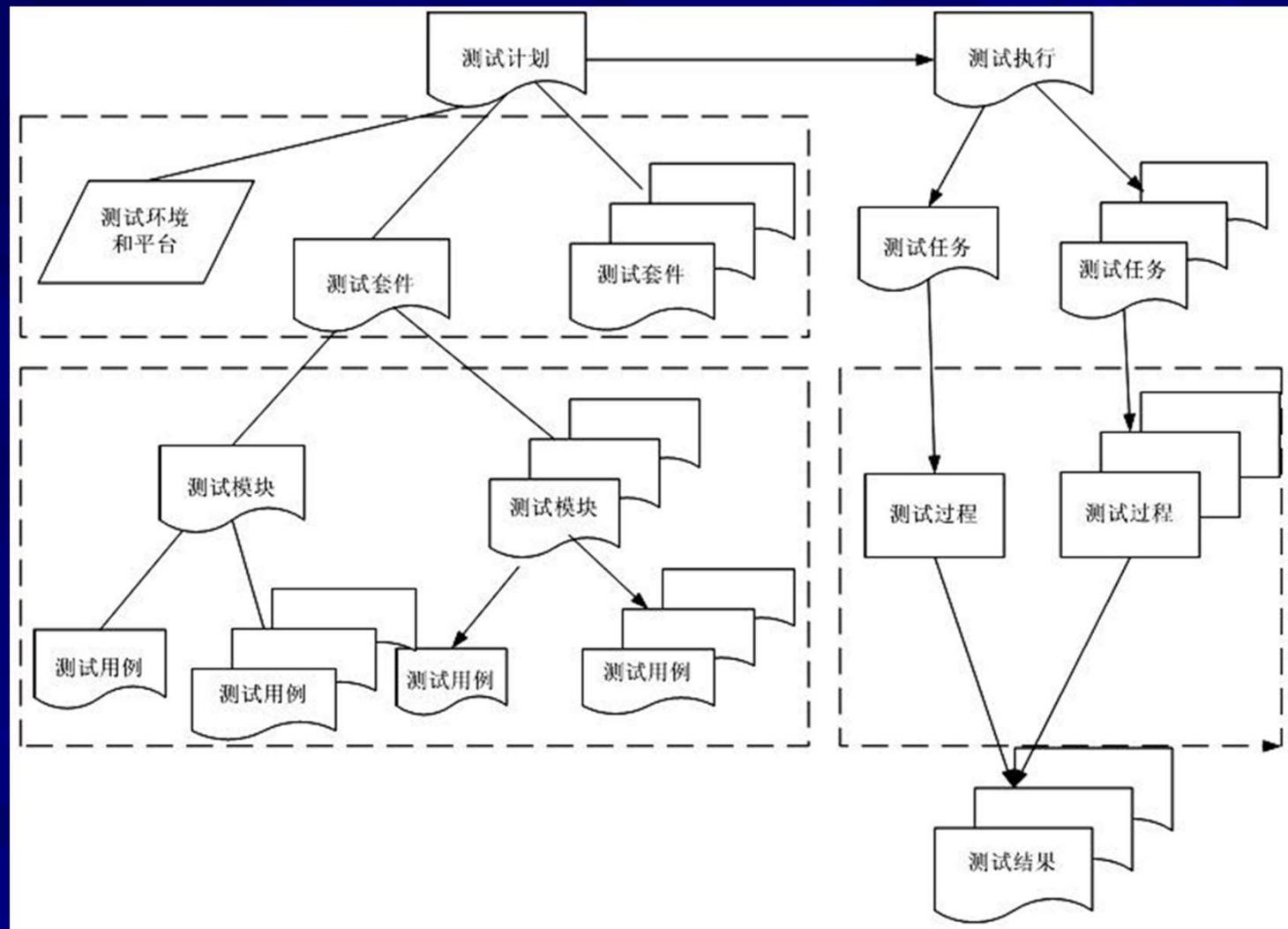
软件更新测试

数据备份测试

- 测试用例1
- 测试用例2
- 测试用例3

- 测试用例1
- 测试用例2
- 测试用例3

# 测试用例的组织和测试过程的关系



# 测试套件应用场景（1）

- 只是部分功能模块发生了变化，就可以创建由这些改动模块的测试用例构成的测试套件
- 在修改的模块中，不需要选择所有的测试用例，针对不同的优先级创建不同的测试套件
- 测试执行的第一阶段可以创建一个特定平台上的测试套件

# 测试套件应用场景（2）

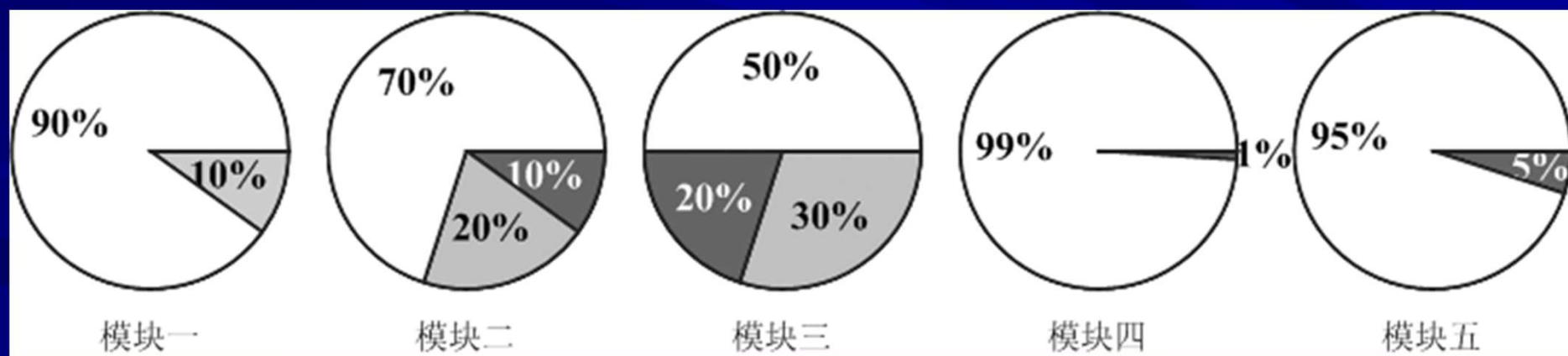
- 有必要为自动化测试、手工测试分别建立测试套件
- 还可以建立和测试人员相对应的、不同平台或不同模块的测试套件
- 回归测试中，可以先运行曾经发现缺陷的测试用例，然后再运行从来没有发现缺陷的测试用例

### 3.2.3 跟踪测试用例

#### ■ 用例执行的跟踪

- 跟上进度？测试人员每天能执行多少个测试用例？“通过、未通过以及未测试的”各占多少？不能被执行的原因是什么？

#### ■ 测试用例覆盖率的跟踪

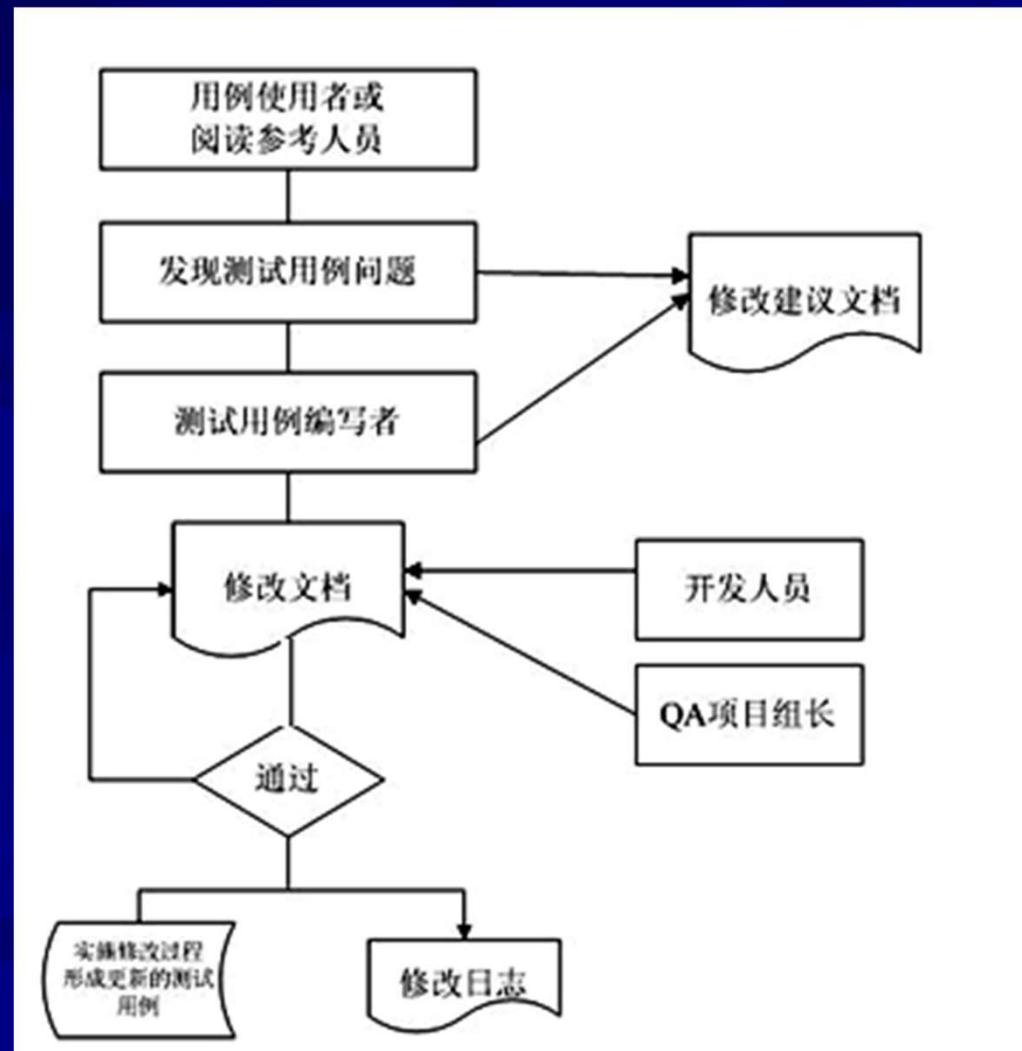


## 3.2.4 维护测试用例

### ■ 测试用例的维护是持续改进的过程

原因	更新时间	优先级
先前的测试用例设计不全面或者不够准确，随着测试过程的深入和对产品功能特性的更好理解，发现测试用例存在一些逻辑错误，需要纠正。	测试过程中	高，需要及时更新
所发现的、严重的软件缺陷没有被目前的测试用例所覆盖。	测试过程中	高，需要及时更新
新的版本中添加新功能或者原有功能的增强，要求测试用例做相应改动。	测试过程前	高，需要在测试执行前更新
测试用例不规范或者描述语句的错误。	测试过程中	中，尽快修复，以免引起误解
旧的测试用例已经不再使用，需要删除。	测试过程后	中，尽快修复，以提高测试效率。

# 测试用例维护流程实例



## 3.2.5 测试用例的覆盖率

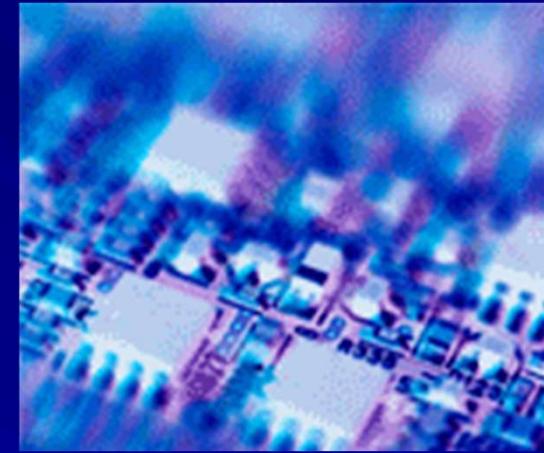
- 测试用例本身
  - 发现缺陷后补充的测试用例数/总的测试用例数
- 需求、功能点覆盖率
- 代码覆盖率

# 实际案例

- Adbook项目部分测试用例设计
  - ICS\_2766
  - ICS\_2968
- 课后练习
  - 针对UR\_3017进行测试用例设计

# 4. 部署测试环境

- 4.1 测试环境的重要性
- 4.2 测试环境要素
- 4.3 虚拟机的应用
- 4.4 如何建立项目的测试环境
- 4.5 自动部署测试环境
- 4.6 测试环境的维护和管理



# 4.1 测试环境的重要性

- 配置测试环境是测试实施的一个重要阶段，测试环境适合与否会严重影响测试结果的真实性和正确性
- 4.1.1 测试环境的定义
- 4.1.2 测试环境是测试的基础

# 4.1.1 测试环境的定义（1）

## ■ 软件测试环境包括

### - 设计环境

■ 编制测试计划、说明、报告及与测试有关的文件所基于的软、硬件设备和支持。在软件设计阶段，不仅要设计测试用例，绘制系统工作流程图、数据流程图等，需要一些设计工具支持，而且还有开发测试工具和测试脚本，需要集成开发环境（IDE）的支持，以及对技术讨论、沟通的必要的支持手段，如即时消息（IM）、邮件、在线会议系统等

## 4.1.1 测试环境的定义（2）

### ■ 软件测试环境包括（续）

#### — 实施环境

- 对软件系统进行各项测试所基于的软、硬件设备和支持。测试实施环境包括被测软件的运行平台和用于各项测试的工具。实施环境必须尽可能地模拟真实环境，以期望能够测试出真实环境中的所有问题，同时该环境是独立的，不受开发人员调试工作的影响。通常意义上所讨论的测试环境，主要就是指软件测试的实施环境

## 4.1.1 测试环境的定义（3）

### ■ 软件测试环境包括（续）

#### — 管理环境

- 管理测试资源所基于的软、硬件设备和支持。测试资源指测试活动所利用或产生的有形物质（如软件、硬件、文档）或无形财富（如人力、时间、测试操作等）。广义的测试管理环境包含测试设计环境、测试实施环境和专门的测试管理工具。例如，对 Bug 的跟踪、分析管理；对 Test case 的分类管理；对测试任务的分派、资源管理等

## 4.1.2 测试环境是测试的基础

■ 测试环境是软件测试的基础，使用错误的测试环境，可能引起下列一系列的问题

- 得出完全错误甚至是相反的结果
- 得出的结果与实际使用中的结果有很大误差
- 忽略了实际使用可能会出现的严重错误，将严重的Bug遗留到客户的手中
- 项目返工，造成巨大的资源浪费
- 项目延期，信誉的损失

# 4.2 测试环境要素（1）

## ■ 测试环境

### — 硬件环境

- 测试必须的服务器、客户端、网络连接设备，以及打印机、扫描仪等辅助硬件设备所构成的环境

### — 软件环境

- 被测试软件运行时的操作系统、数据库及其他应用软件构成的环境

## 4.2 测试环境要素（2）

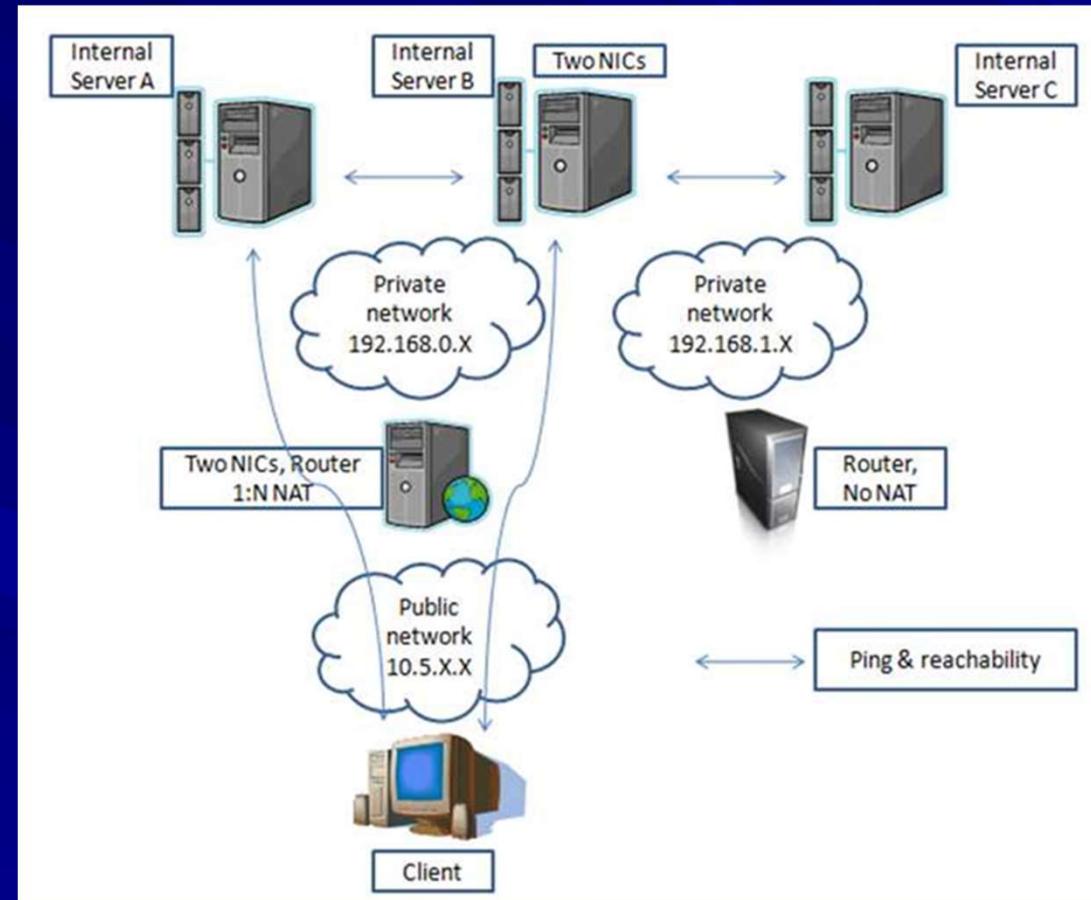
### ■ 测试环境的五要素

- 软件
- 硬件
- 网络环境
- 数据准备
- 测试工具

### ■ 专题介绍

# 4.2 测试环境要素 (3)

- 4.2.1 硬件
- 4.2.2 网络环境
- 4.2.3 软件
- 4.2.4 数据准备



## 4.2.1 硬件（1）

■ 软件测试中，最基本的硬件包括特定的网络设备、服务器和测试用机

- 网络设备种类非常丰富，包括交换机、路由器等，还有防火墙、负载均衡器相关的网络设备
- 服务器可分为PC服务器、专用服务器、小型机等。为了满足密集部署服务器的需要，开始普遍使用机架式服务器和刀片式服务器
- 测试用机又可分为普通PC、网络PC、苹果（Mac）机、Sun工作站、移动设备等

## 4.2.1 硬件 (2)

■ 硬件设备多种多样，完全根据产品的需求进行选择。选择时需要考虑其配置标准。通常有

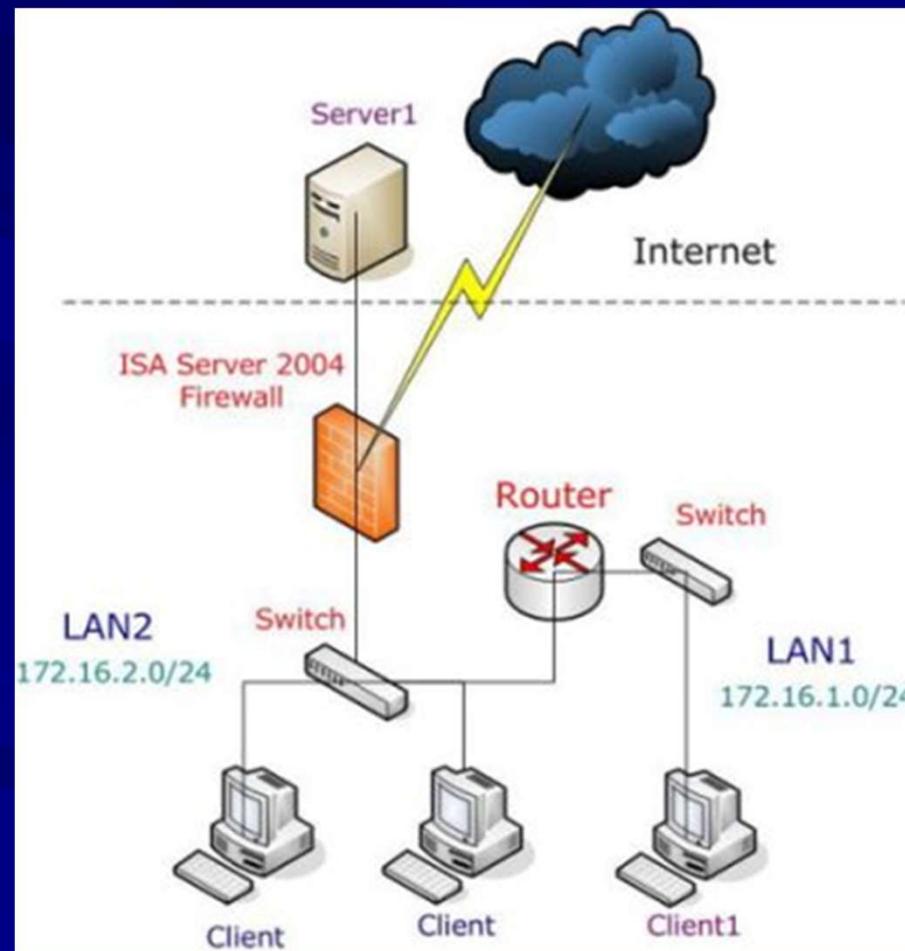
- 标准配置
- 最佳配置
- 最低配置

## 4.2.2 网络环境（1）

- 网络环境是由相关的网络设备、网络系统软件及其配置构成的综合环境，包括
  - 路由器、交换机、网线、网卡等硬件设备
  - 各种网络协议、代理、网关、防火墙、负载均衡器等配置
  - 网络工具的安装和配置，如网络限速器、带宽调度器等

## 4.2.2 网络环境 (2)

### ■ 基于防火墙的网络环境示例

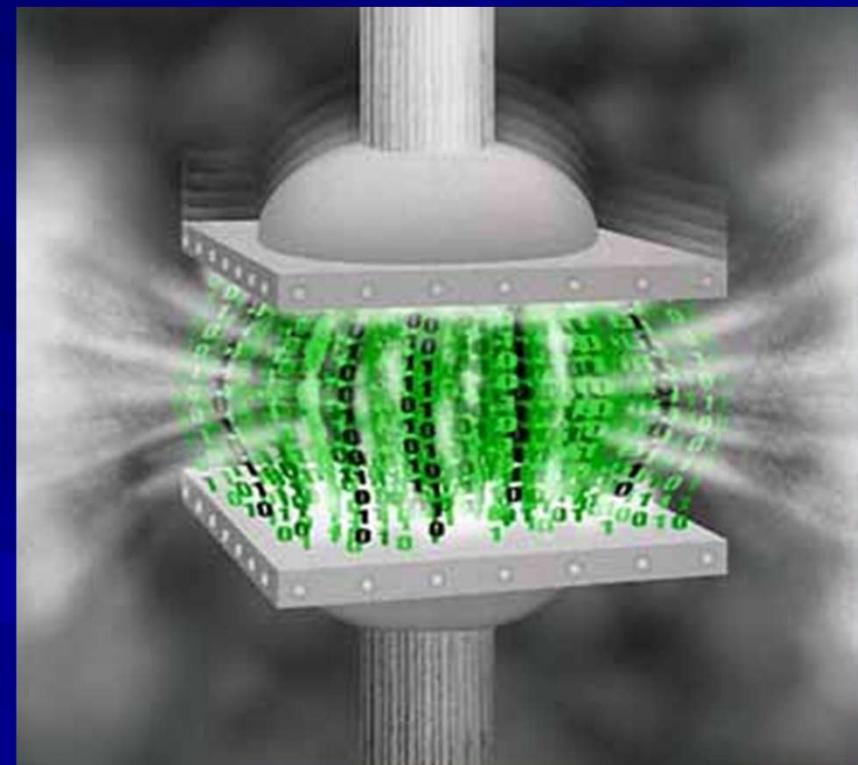


## 4.2.3 软件

- 软件环境包括操作系统、网络协议和应用程序。测试工具软件也是软件环境派生出来的一部分。建立软件测试环境的原则是选择具有广泛代表性的重要操作系统和大量应用程序

## 4.2.4 数据准备

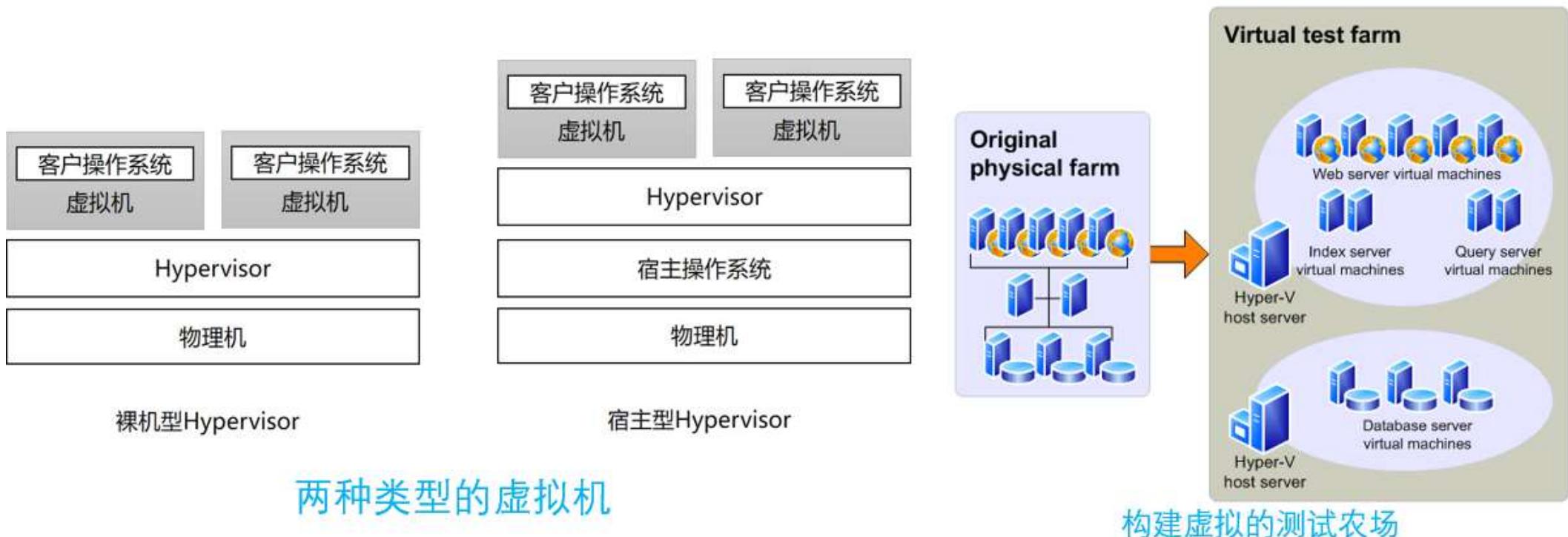
- 原有数据
- 正确数据和错误数据
- 真实的客户数据
- 大量的数据



# 4.3 虚拟机的应用

- 4.3.1 虚拟机技术
- 4.3.2 虚拟机软件

# 4.3.1 虚拟机技术



# 为什么使用虚拟机（1）

- 充分利用硬件资源

- 有70%的服务器利用率只有20%~30%，借助虚拟机技术提高到85%~95%

- 节约能源和空间

- 例如如果内存加大到16G或更高，一台机器可以虚拟4~8台服务器

# 为什么使用虚拟机（2）

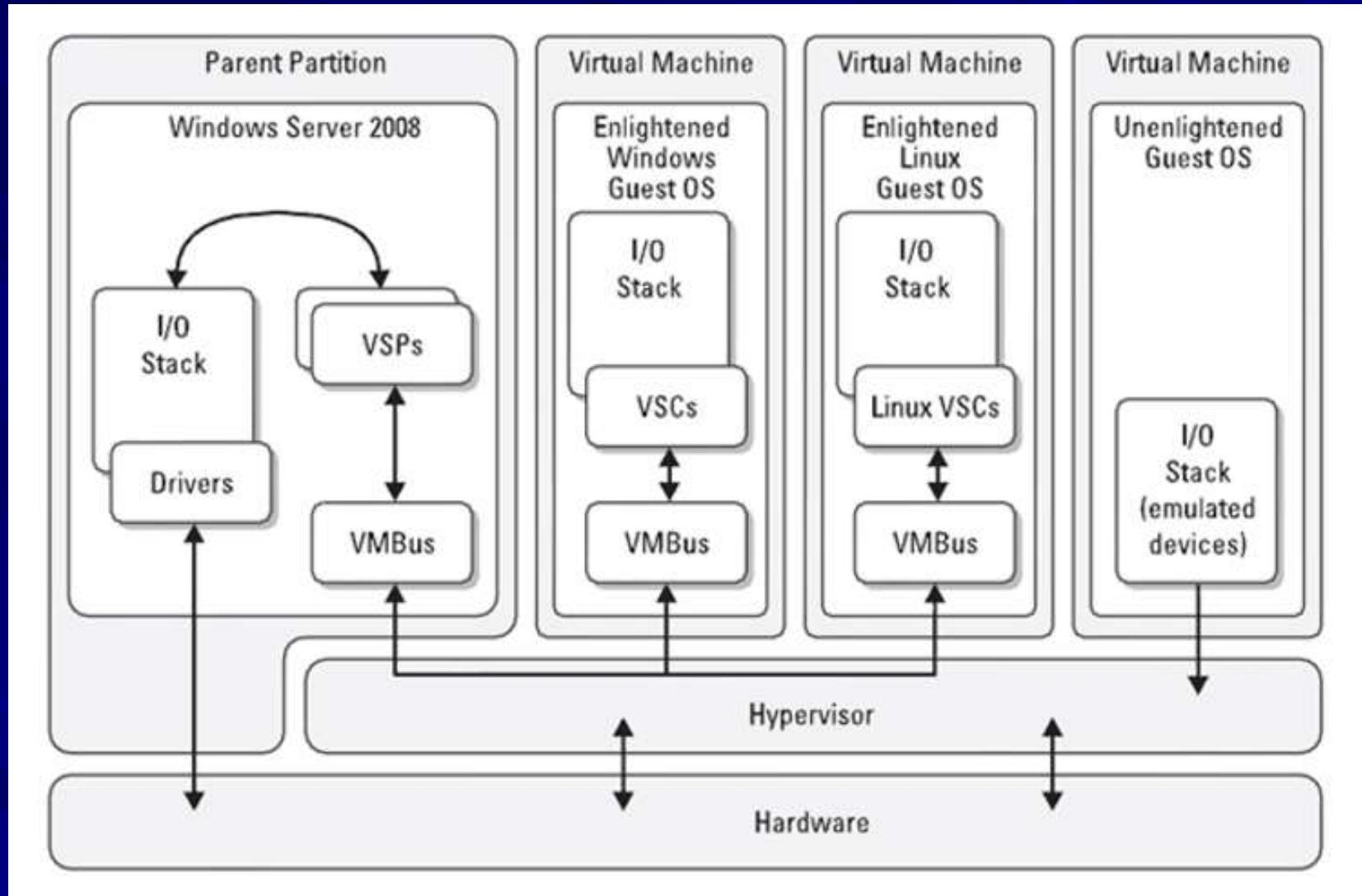
- 提升运作效率

- 几分钟就可装载所需的系统镜像文件

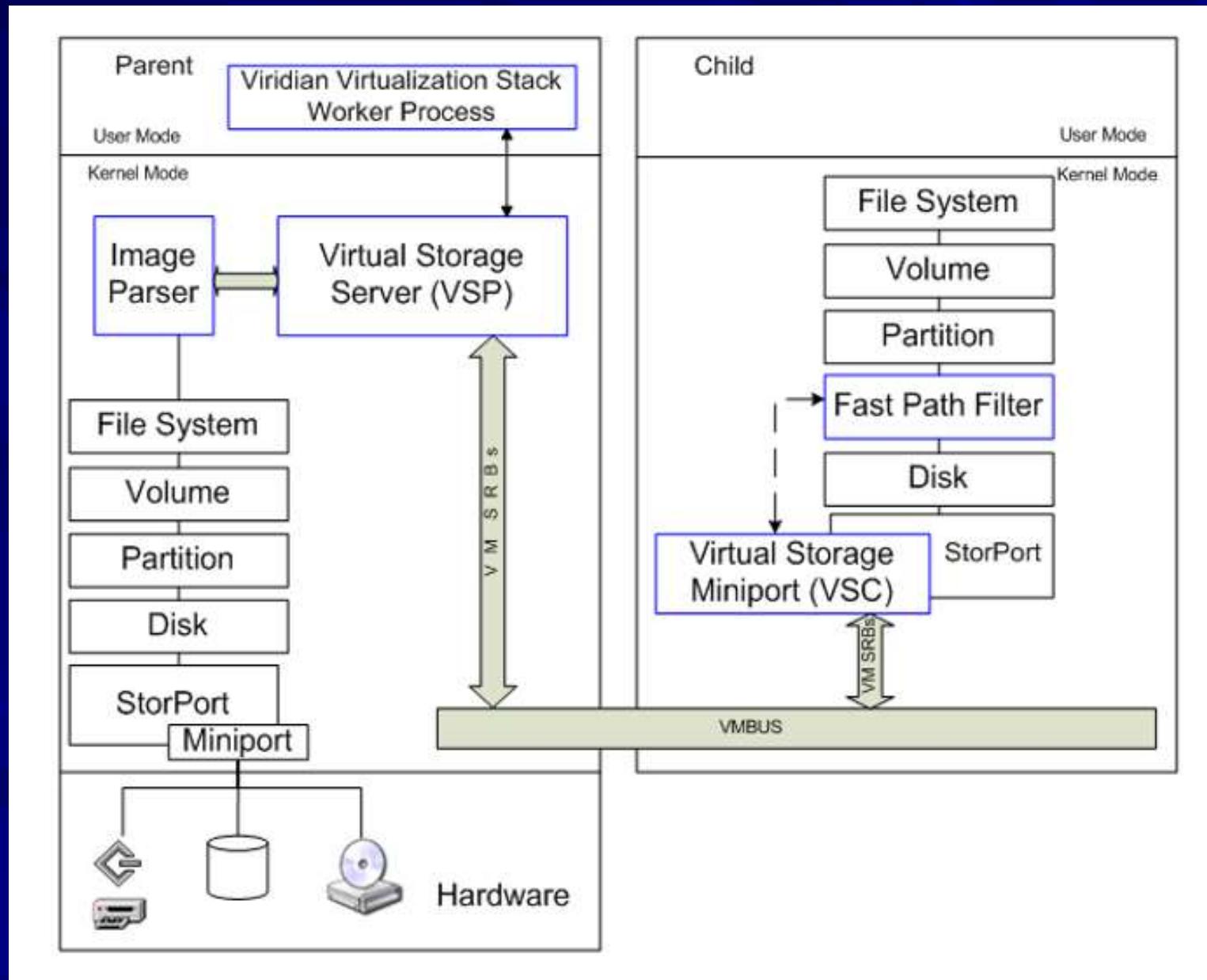
- 有利于环境的建立和维护

- 容易实现添加、移动、变更和重置服务器的操作

# 虚拟机的工作原理 (1)



# 虚拟机的工作原理 (2)



## 4.3.2 虚拟机软件（1）

- KVM (Kernel-based Virtual Machine, Linux)
- 开源软件QEMU
- VMware Workstation/Fusion
- 完全免费和开源的Oracle VM VirtualBox
- Parallels (macOS)

## 4.3.2 虚拟机软件（2）

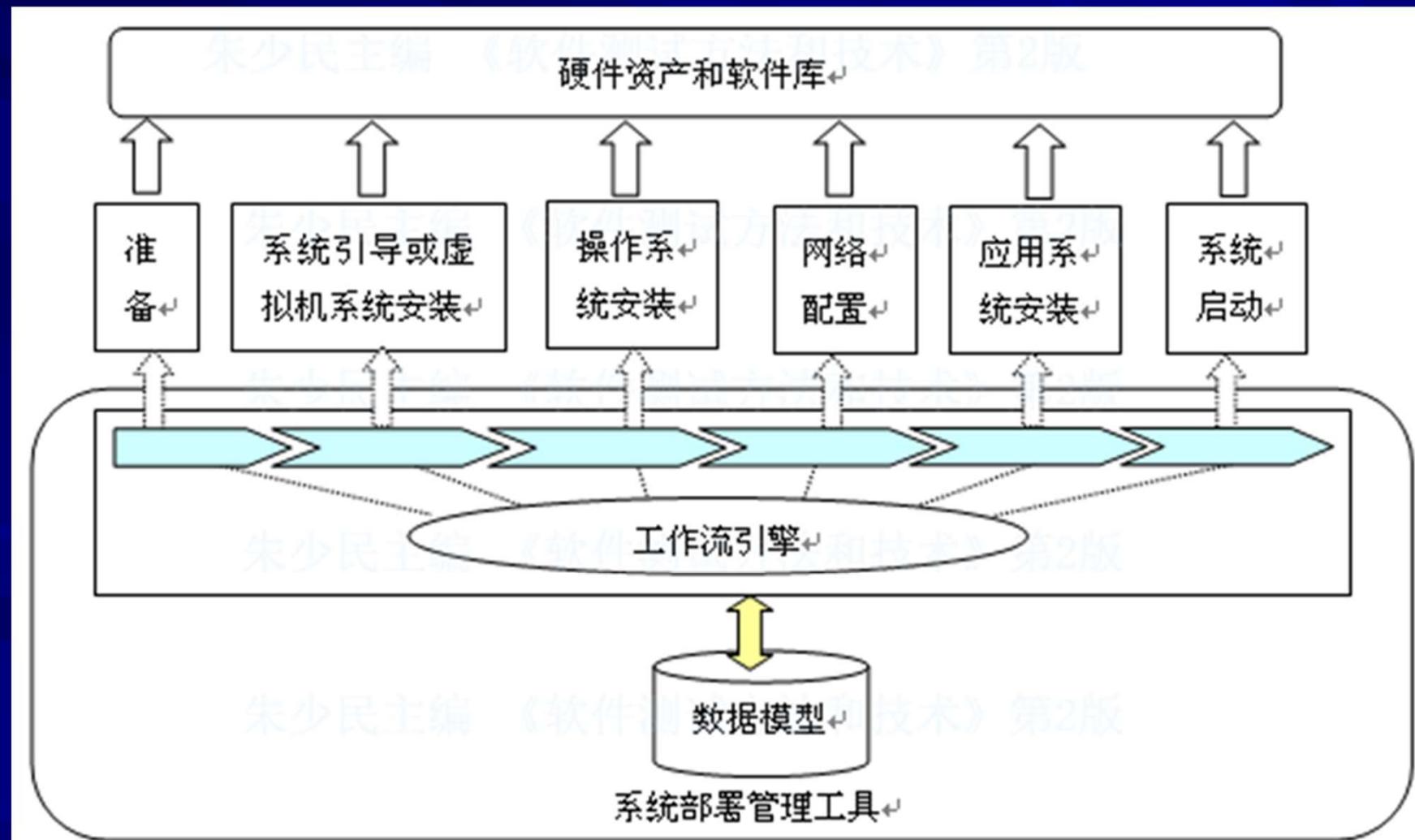
- 微软公司的Hyper-V Manager
- SW-soft公司的Virtuozzo
- 开源软件Xen（Linux）
- 开源软件Colinux（windows）

## 4.4 如何建立项目的测试环境

- 事先要清楚项目的要求，如软件构架文档、部署模型、测试自动化架构、测试数据的要求和测试策略和测试方法
- 规划测试环境
- 列出设备清单
- 环境实施：如安装虚拟机系统、操作系统、网络配置、安装应用系统、配置并调试应用软件

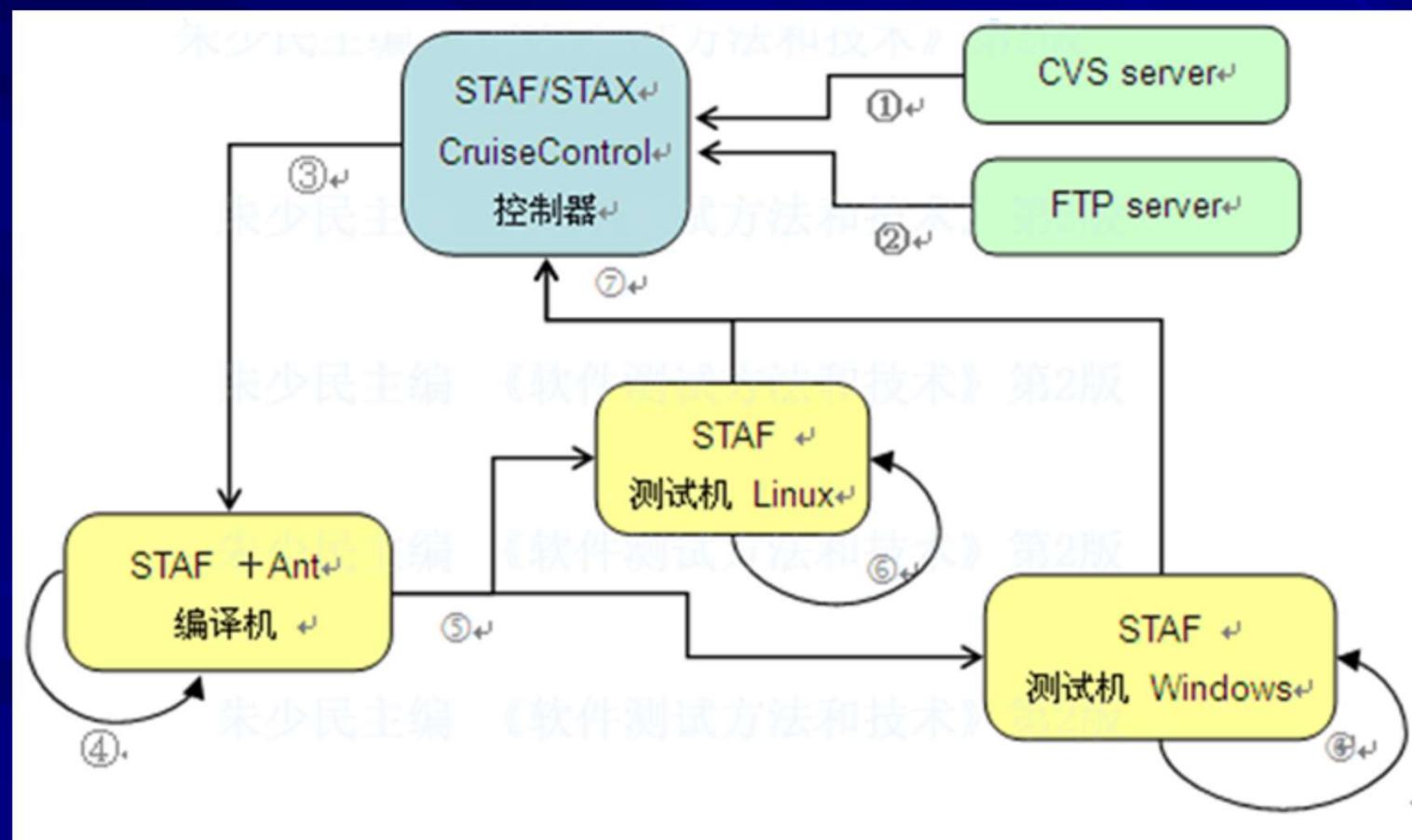
# 4.5 自动部署测试环境（1）

## ■ 测试环境自动化部署框架示意图



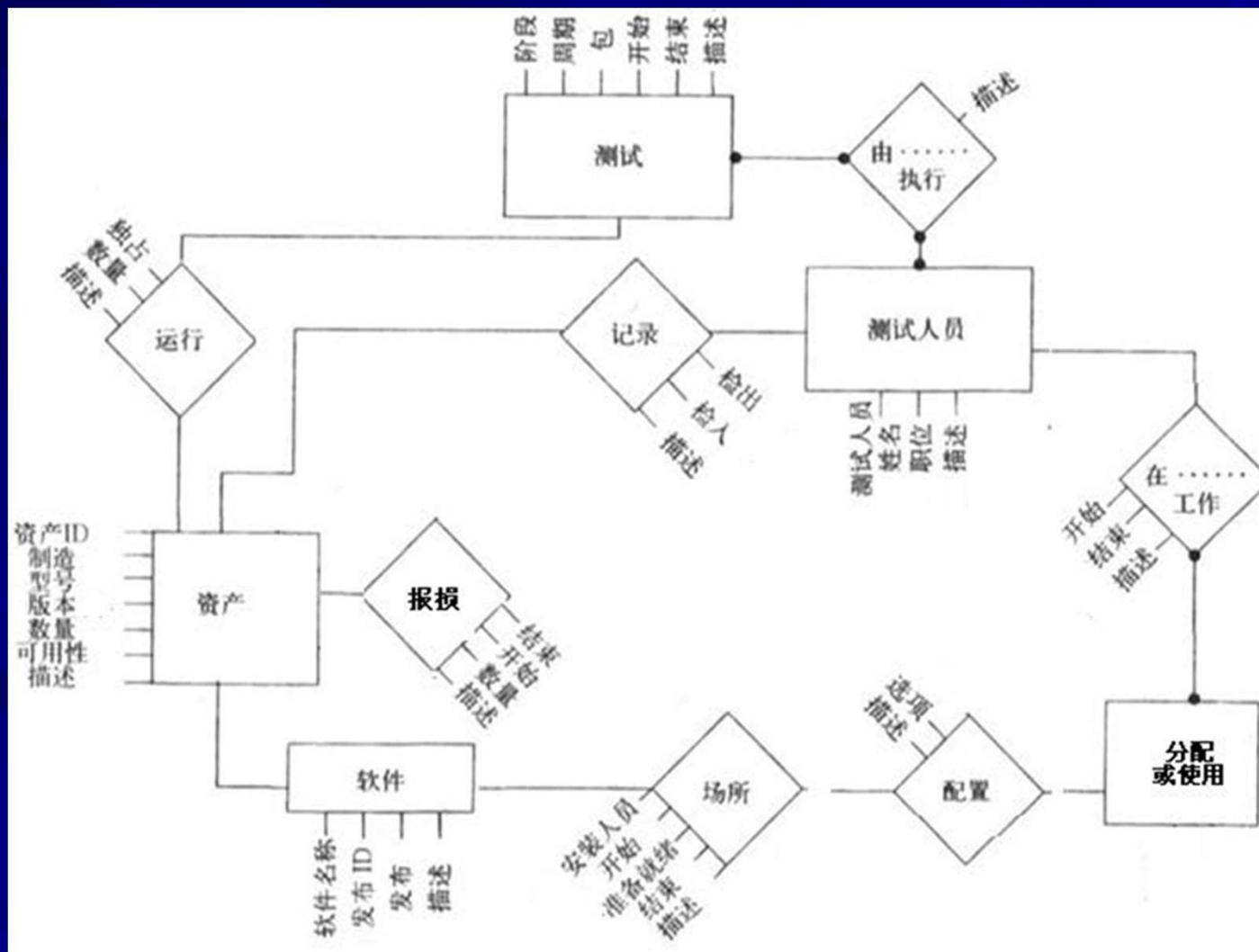
# 4.5 自动部署测试环境 (2)

## ■ STAF和STAX构成的自动部署体系实例



# 4.6 测试环境的维护和管理

## ■ 实验室设备管理流程示意图



# 5. 报告所发现的缺陷

- 5.1 软件缺陷的描述
- 5.2 软件缺陷相关的信息
- 5.3 软件缺陷跟踪和分析
- 5.4 软件缺陷跟踪系统



# 5.1 软件缺陷的描述

- 5.1.1 软件缺陷的生命周期
- 5.1.2 严重性和优先级
- 5.1.3 缺陷的其他属性
- 5.1.4 完整的缺陷信息
- 5.1.5 缺陷描述的基本要求
- 5.1.6 缺陷报告的示例

## 5.1.1 软件缺陷的生命周期

- 软件缺陷生命周期指的是从一个软件缺陷被发现、报告到这个缺陷被修复、验证直至最后关闭的完整过程
- 缺陷生命周期是各类开发人员一起参与、协同测试的过程
- 软件缺陷一旦发现，便进入严密监控之中，直至软件缺陷生命周期终结，这样既可保证在较短的时间内高效率地关闭所有的缺陷，缩短软件测试的进程，提高软件质量，同时又减少开发、测试和维护成本

# 基本的缺陷生命周期

## ■ 发现——打开

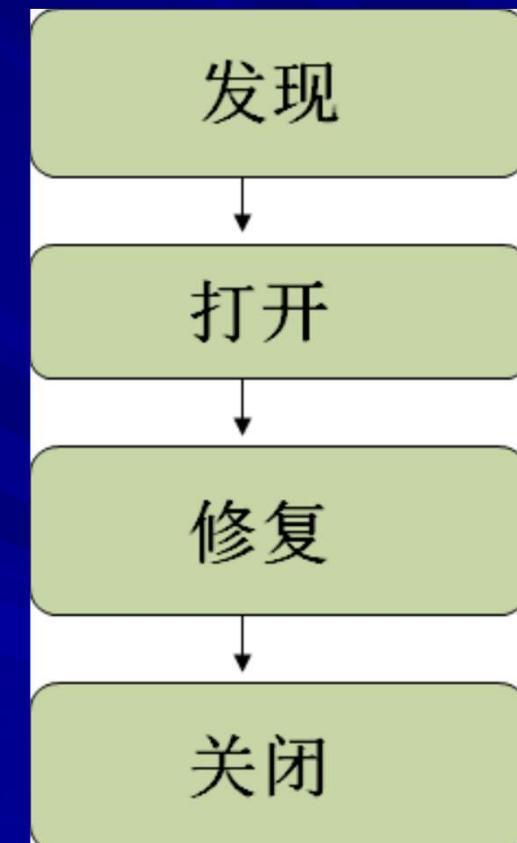
- 测试人员找到软件缺陷并将软件缺陷提交给开发人员

## ■ 打开——修复

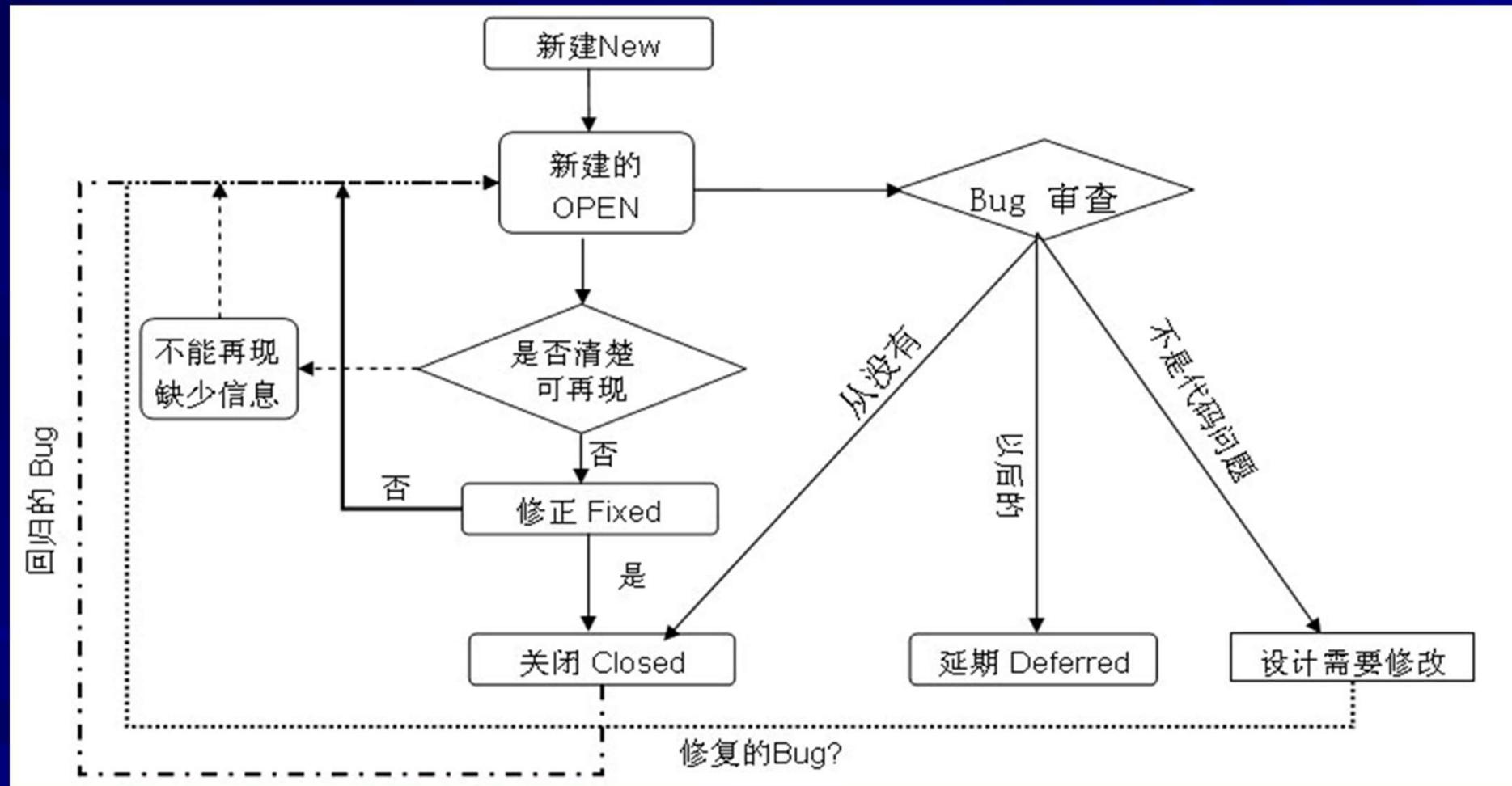
- 开发人员再现、修复缺陷，然后提交给测试人员去验证

## ■ 修复——关闭

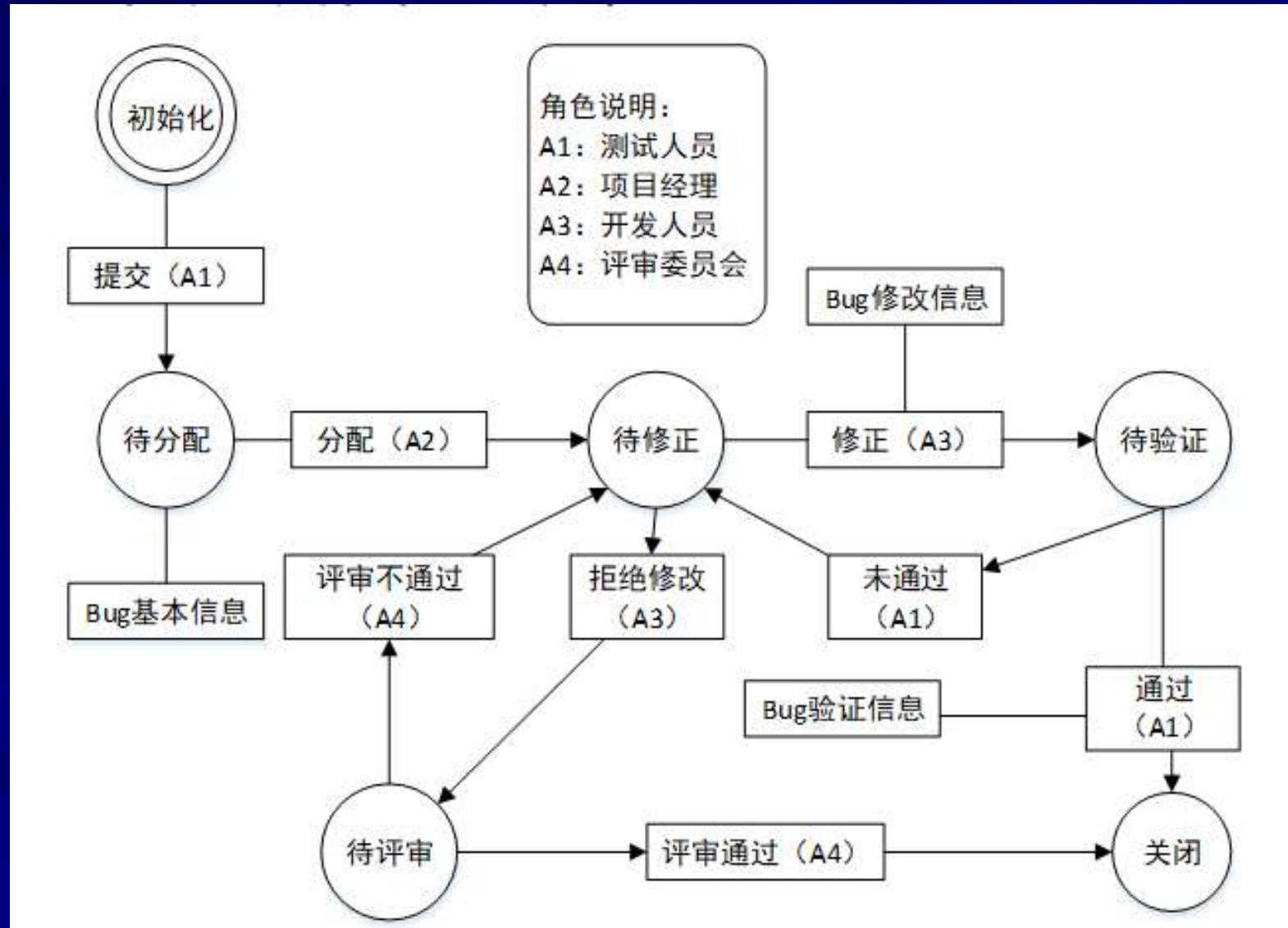
- 测试人员验证修复过的软件，关闭已不存在的缺陷



# 常见的软件缺陷生命周期



# 软件缺陷管理流程



# 缺陷管理流程中的角色

## ■ 测试人员A1

- 进行测试的人员，并且是缺陷的发现者

## ■ 项目经理A2

- 对整个项目负责，对产品质量负责的人员

## ■ 开发人员A3

- 执行开发任务的人员，完成实际的设计和编码工作，以及对缺陷的修复工作

## ■ 评审委员会A4

- 对缺陷进行最终确认，在项目成员对缺陷不能达成一致意见时，行使仲裁权力

# 缺陷管理流程中的缺陷状态

- 初始：缺陷的初始状态
- 待分配：缺陷等待分配给相关开发人员处理
- 待修正：缺陷等待开发人员修正
- 待验证：开发人员已完成修正，等待测试人员验证
- 待评审：开发人员拒绝修改缺陷，需要评审委员会评审
- 关闭：缺陷已被处理完成

# 缺陷管理流程描述（1）

- 测试小组发现新的缺陷，并记录缺陷，此时缺陷状态为“初始化”
- 测试小组向项目经理提交新发现的缺陷(包括缺陷的基本信息)，此时缺陷的状态为“待分配”
- 项目经理接收到缺陷报告后，根据缺陷的详细信息，确定处理方案，此时缺陷的状态为“待修正”

# 缺陷管理流程描述（2）

- 缺陷报告被分配给相应的开发人员，开发人员对缺陷进行修复，并填写缺陷的修改信息，然后等待测试人员对修复后的缺陷再一次进行验证，此时缺陷的状态为“待验证”
- 经测试人员验证后，发现缺陷未被修复，则重新交给原负责修复的开发人员，测试缺陷的状态为“待修正”

# 缺陷管理流程描述（3）

- 经测试人员验证后，认为缺陷被修复，则填写缺陷验证信息，缺陷修复完成，此时缺陷的状态为“关闭”
- 若测试人员验证缺陷未被修复，但是开发人员认为已修复完成拒绝再次修复，则将缺陷报告提交给评审委员会，等待评审委员会的评审，此时缺陷的状态为“待评审”

# 缺陷管理流程描述（4）

- 若评审委员会评审不通过，即软件缺陷未被修复，开发人员需继续修复，此时软件缺陷的状态为“待修正”
- 若评审委员会评审通过，即软件缺陷被修复，此时缺陷状态为“关闭”

## 5.1.2 严重性和优先级 (1)

### ■ 严重性 (Severity)

– 衡量缺陷对客户满意度的影响程度

- 致命的 (fatal)
- 严重的 (critical)
- 一般的 (major)
- 微小的 (minor)

### ■ 优先级 (Priority)

– 指缺陷被修复的紧急程度

## 5.1.2 严重性和优先级 (2)

缺陷优先级	描述
立即解决(P1级)	缺陷导致系统几乎不能使用或测试不能继续，需立即修复
高优先级(P2级)	缺陷严重，影响测试，需要优先考虑
正常排队(P3级)	缺陷需要正常排队等待修复
低优先级(P4级)	缺陷可以在开发人员有时间的时候被纠正。

### 5.1.3 缺陷的其他属性

- 缺陷标识 (ID)
- 缺陷类型 (type)
- 缺陷产生频率 (frequency)
- 缺陷来源 (source)
- 缺陷原因 (root cause)

## 5.1.4 完整的缺陷信息

- 前提
- 操作步骤
- 期望结果
- 实际结果
- 上述的各种缺陷属性

# 软件缺陷的详细描述

- “步骤” 提供了如何重复当前缺陷的准确描述，应简明而完备、清楚而准确。这些信息对开发人员是关键的，视为修复缺陷的向导
- “期望结果” 与测试用例标准或设计规格说明书或用户需求等一致，达到软件预期的功能。是验证缺陷的依据
- “实际结果” 实际执行测试的结果，不同于期望结果，从而确认缺陷的存在

## 5.1.5 缺陷描述的基本要求

- 单一准确
- 可以再现
- 完整统一
- 短小简练
- 特定条件
- 补充完善
- 不做评价

# 实例

## 优秀的缺陷报告

重现步骤：

- a) 打开一个编辑文字的软件并且创建一个新的文档（这个文件可以录入文字）
- b) 在这个文件里随意录入一两行文字
- c) 选中一两行文字，通过选择Font 菜单然后选择Arial字体格式
- d) 一两行文字变成了无意义的乱字符

期望结果：当用户选择已录入的文字并改变文字格式的时候，文本应该显示正确的文字格式不会出现乱字符显示。

实际结果：它是字体格式的问题，如果改变文字格式成Arial之前，你保存文件，缺陷不会出现。缺陷仅仅发生在Windows98并且改变文字格式成其它的字体格式，文字是显示正常的。

见所附的图片<有一个链接，点击即可看到>

# 散漫的缺陷报告实例

## 重现步骤:

- 在Window98上打开一个编辑文字的软件并且编辑存在文件
- 文件字体显示正常
- 我添加了图片，这些图片显示正常
- 在此之后，我创建了一个新的文档
- 在这个文档中我随意录入了大量的文字
- 在我录入这些文字之后，选择几行文字. 并且通过选择Font 菜单然后选择Arial字体格式改变文字的字体。
- 有三次我重现了这个缺陷
- 我在Solaris操作系统运行这些步骤，没有任何问题。
- 我在Mac操作系统运行这些步骤，没有任何问题。

**期望结果:** 当用户选择已录入的文字并改变文字格式的时候，文本应该显示正确的文字格式不会出现乱字符显示。

**实际结果:** 我试着选择少量的不同的字体格式，但是只有Arial字体格式有软件缺陷，不论如何，它可能会出现在我没有测试的其它的字体格式

# 5.2 软件缺陷的相关信息

- 5.2.1 软件缺陷的图片信息
- 5.2.2 使用WinDbg记录软件缺陷信息
- 5.2.3 使用Soft-ICE记录软件缺陷信息
- 5.2.4 分离和再现软件缺陷

## 5.2.1 软件缺陷的图片信息

- 软件缺陷相关的信息包括软件缺陷的图片、记录信息和如何再现和分离软件缺陷。缺陷报告提供的信息应能使开发人员和其他的测试人员更容易分离和重现缺陷
- 一些涉及用户界面（User Interface）的软件缺陷可能很难用文字清楚地描述，因此软件测试人员通过附上图片比较直观地表示缺陷发生在产品界面什么位置、有什么问题等

## 5.2.2 使用WinDbg记录软件缺陷信息（1）

- WinDbg是微软发布的源码级调试工具，用于Kernel模式调试和用户模式调试，可用于调试软件崩溃后形成Dump文件，包括操作系统的状态、进程运行的状态、时间和环境变量、汇编指令、调用堆栈等
- 安装、使用的具体操作方法，如提供了图形界面和命令行两种运行方式

## 5.2.2 使用WinDbg记录软件缺陷信息（2）

### ■ 调试方式

- 远程调试

- **!windbg -remote  
npipe:server=SERVER\_NAME,pipe=PIPE\_NAME**

- Dump调试

- **!windbg -z DUMP\_FILE\_NAME**

- 本地进程调试

- **!Windbg -p “process id”**

### ■ 常用命令

## 5.2.3 使用Soft-ICE记录软件缺陷信息（1）

- stack
- u eip-80
- 如果数据窗口是开启的状态，可以输入“wd”来关闭该窗口，然后再输入“dd esp-20”命令。stack、dd esp-20是为了标注跟踪信息

## 5.2.3 使用Soft-ICE记录软件缺陷信息（2）

- 通过输入"x"，退出 Soft-ICE的窗口；如果还是无法退出Soft-ICE，需要输入**faults off**，然后输入"x"
- 打开Soft-ICE应用程序，立即保存日志文件。一旦再次打开Soft-ICE，请输入“**faults on**”

## 5.2.4 分离和再现软件缺陷

- 分离和再现软件缺陷的步骤
- 分离和调试软件缺陷之间的区别

# 分离和再现软件缺陷的步骤

## ■ 常用方法和技巧

- 确保所有的步骤都被记录
- 特定条件和时间
- 压力和负荷、内存和数据溢出相关的边界条件
- 考虑资源依赖性包括内存、网络和硬件共享的相互作用等
- 不能忽视硬件

# 分离和调试软件缺陷之间的区别 (1)

- 讨论分离和调试软件缺陷之间的区别，是为了划清软件测试人员与开发人员的责任，增加界限的清晰度与测试资源的控制能力
  - 再现缺陷现象所需的最少步骤有哪些？这些步骤成功再现的可能性多大？
  - 缺陷是否成立存在？测试结果是否可能起源于测试因素或者测试人员自身的错误，还是影响顾客需求的、系统真正的故障？

# 分离和调试软件缺陷之间的区别 (2)

- 哪些外部因素产生软件缺陷？
- 哪些内部因素，是代码、网络、还是环境引起的软件缺陷？
- 怎样在不产生新的缺陷的条件下使这个软件缺陷得到修复？
- 这种修复是否经过调试，单元是否经过测试？
- 问题解决了吗？它是否通过了确认和回归测试，确定系统的其余部分仍工作正常？

# 5.3 软件缺陷跟踪和分析

- 5.3.1 软件缺陷处理技巧
- 5.3.2 缺陷趋势分析
- 5.3.3 缺陷分布分析
- 5.3.4 缺陷跟踪方法

# 软件缺陷的处理和跟踪（1）

- 确保每个被发现的缺陷都能够被解决，“解决”的意思不一定是被修正，也可能是其他处理方式（例如，延迟到下一个版本中修正或者由于技术原因不能被修正），总之，对每个被发现的BUG的处理方式必须能够在开发组织中达到一致

# 软件缺陷的处理和跟踪（2）

- 收集缺陷数据并根据缺陷趋势曲线识别测试处于测试过程中的阶段；决定测试过程是否结束，通过缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式
- 收集缺陷数据并在其上进行数据分析，作为组织过程改进的财富

## 5.3.1 软件缺陷处理技巧（1）

### ■ 审阅

- 可以由测试管理员、项目管理员或其他人来进行，审阅缺陷报告的质量水平

### ■ 拒绝

- 如果审阅者决定需要对一份缺陷报告进行重大修改，应该和测试人员一起讨论，由测试人员纠正缺陷报告，然后再次提交

## 5.3.1 软件缺陷处理技巧（2）

### ■ 完善

- 完整地描述了问题的特征并将其分离，那么审查者就会肯定这个报告

### ■ 分配

- 分配给适当的开发人员，如果不知道具体开发人员，应分配给项目开发组长，由开发组长再分配给对应的开发人员

## 5.3.1 软件缺陷处理技巧（3）

### ■ 验证

- 缺陷的修复需要得到测试人员的验证，同时还要进行回归测试，检查这个缺陷的修复是否会引起新的问题

### ■ 重新打开

- 重新打开一个缺陷，需要加注释说明、电话沟通等，否则会引起“打开-修复-再打开”多个来回，造成测试人员和开发人员不必要的矛盾

## 5.3.1 软件缺陷处理技巧 (4)

### ■ 关闭

- 只有测试人员有关闭缺陷的权限，开发人员没有这个权限

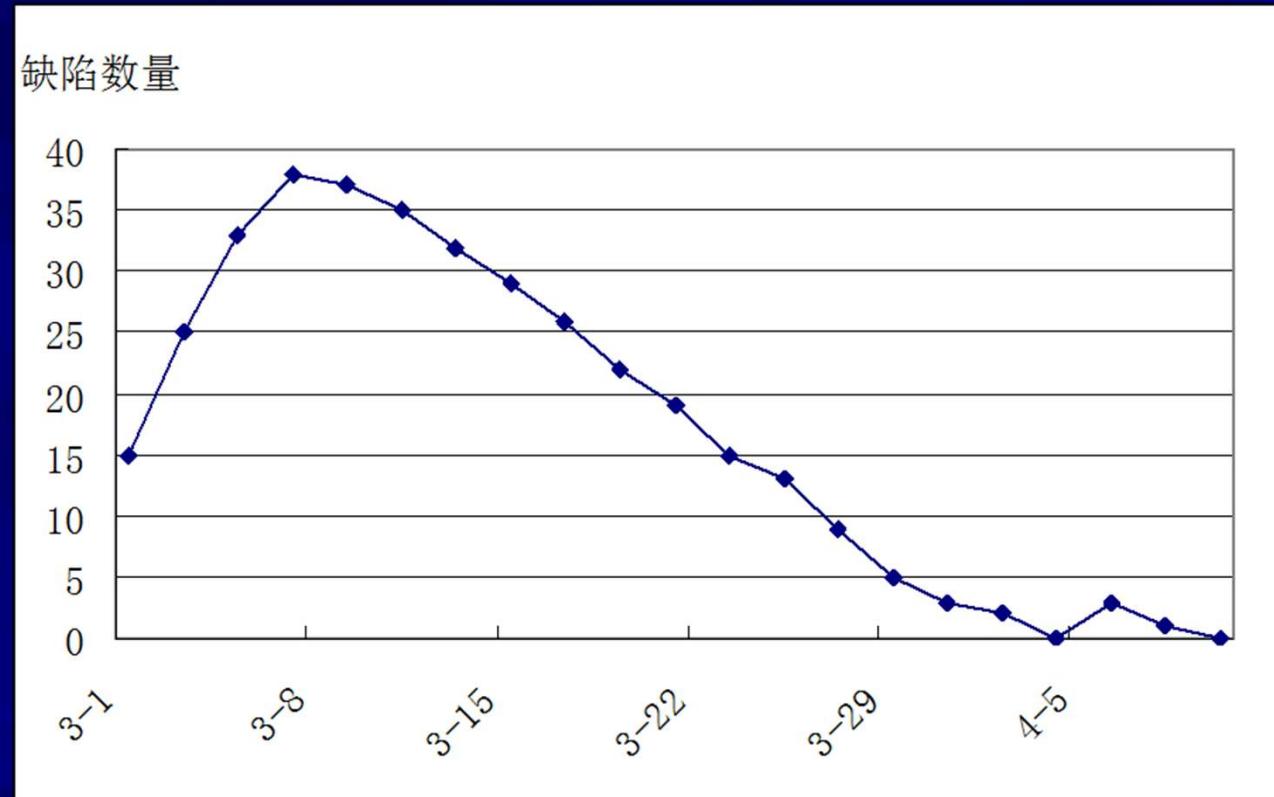
### ■ 暂缓

- 如果每个人都同意将确实存在的缺陷移到以后处理，应该指定下一个版本号或修改的日期。一旦新的版本开始时，这些暂缓的缺陷应该重新被打开

## 5.3.1 软件缺陷处理技巧（5）

- 监控（打开/关闭/已修正的）缺陷随时间的变化
  - 产品开发质量情况取决于累积打开/关闭曲线的趋势
  - 项目进度取决于累积关闭/打开曲线起点的时间差
  - 开发人员、测试人员的工作进度、效率也能得到反映

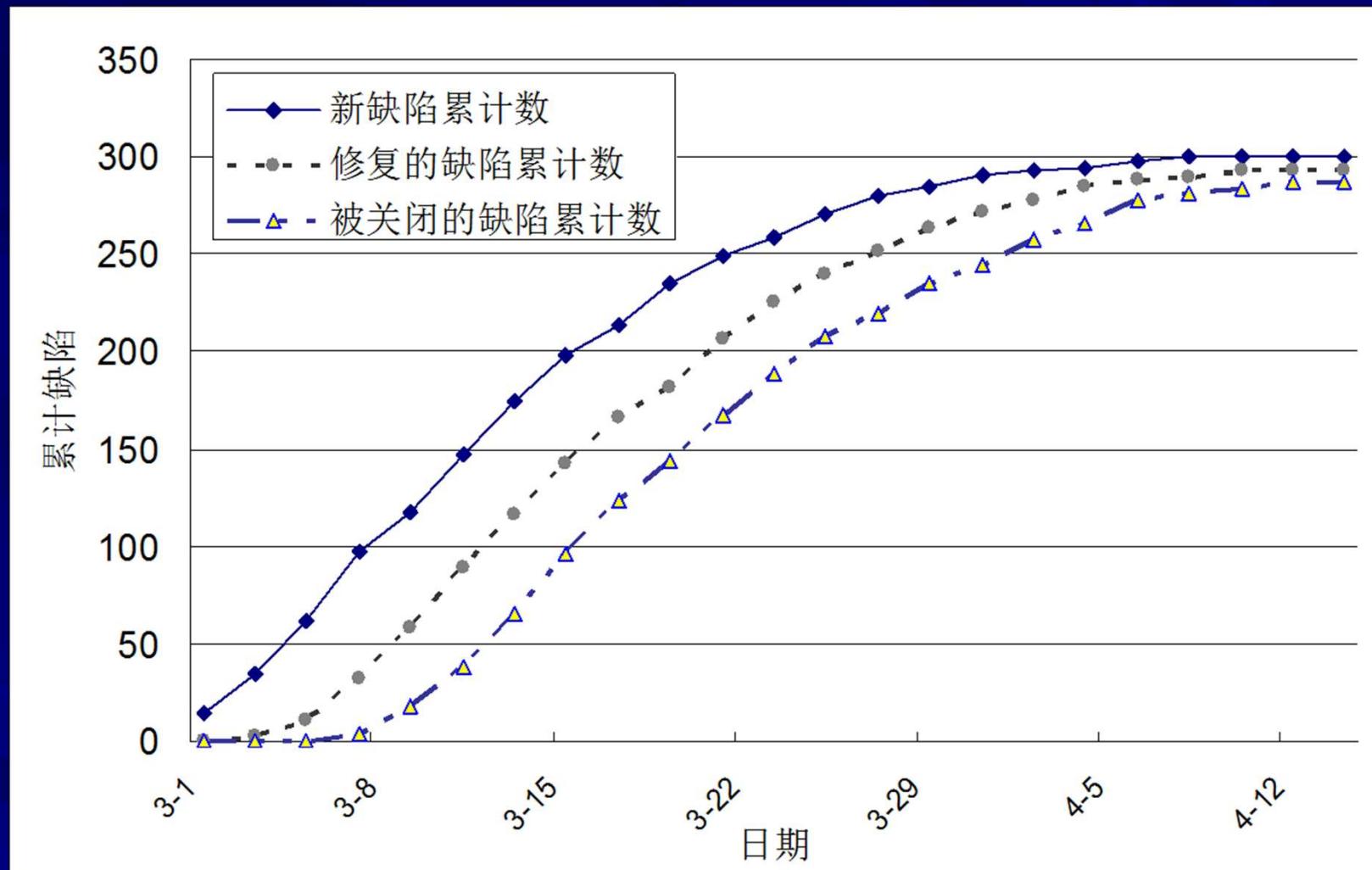
## 5.3.2 缺陷趋势分析



# 示例

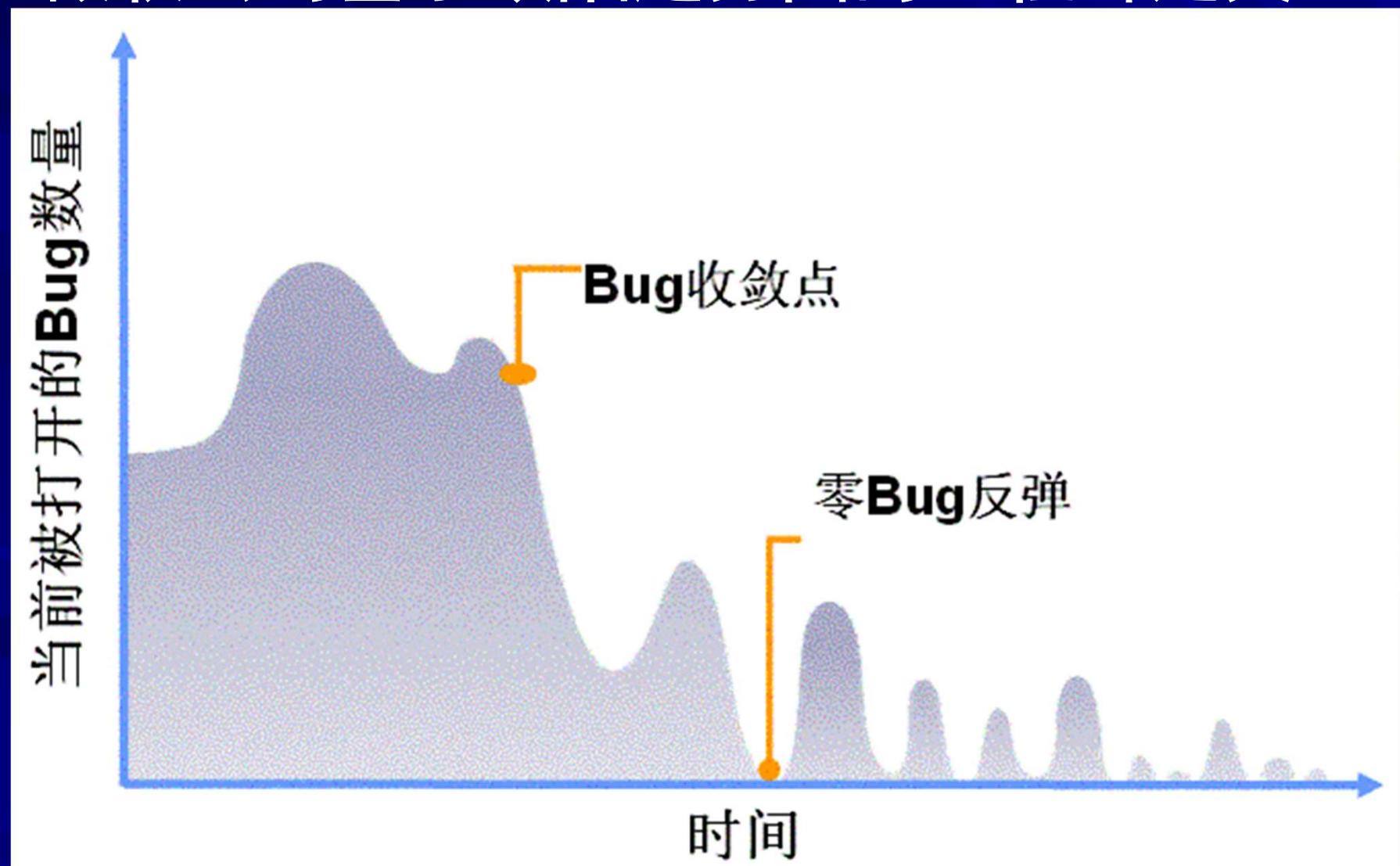


# 理想趋势图



# 示例

- 微软公司基于缺陷趋势图的里程碑定义



### 5.3.3 缺陷分布分析（1）

- 对缺陷进行分析，确定测试是否达到结束的标准，也就是判定测试是否已经达到用户可接受的状态

## 5.3.3 缺陷分布分析（2）

### ■ 最常用的缺陷分析方法有4种

#### — 缺陷分布报告

- 允许将缺陷计数作为一个或多个缺陷参数的函数来显示，生成缺陷数量与缺陷属性的函数。如测试需求和缺陷状态、严重性的分布情况等

#### — 缺陷趋势报告

- 按各种状态将缺陷计数作为时间的函数显示。趋势报告可以是累计的，也可以是非累计的

## 5.3.3 缺陷分布分析（3）

### ■ 最常用的缺陷分析方法有4种（续）

#### — 缺陷年龄报告

- 显示缺陷处于活动状态的时间，展示一个缺陷处于某种状态的时间长短，从而了解处理这些缺陷的进度情况

#### — 测试结果进度报告

- 展示测试过程在被测应用的几个版本中的执行结果以及测试周期

## 5.3.4 缺陷跟踪方法

### ■ 当前缺陷状态 – Bug Dashboard

级别	总数	未处理的	正在处理的	修正的	不是缺陷	重复的	暂不处理	关闭
致命的	2	0	0	0	0	0	0	2
严重的	216	18	7	5	1	4	20	161
一般的	31	23	1	0	0	0	0	7
微小的	5	2	0	0	0	3	0	0

### ■ 项目发展趋势: 每天的变化、差异，重点进行趋势分析

# 软件缺陷报告（1）

- 任何一个缺陷跟踪系统的核心都是“软件缺陷报告”，一份软件缺陷报告详细信息如表

分类	项目	描述
可跟踪信息	缺陷ID	唯一的、自动产生的缺陷ID，用于识别、跟踪、查询
软件缺陷基本信息	缺陷状态	可分为“打开或激活的”、“已修正”、“关闭”等
	缺陷标题	描述缺陷的最主要信息
	缺陷的严重程度	一般分为“致命”、“严重”、“一般”、“较小”等四种程度
	缺陷的优先级	描述处理缺陷的紧急程度，1是优先级最高的等级，2是正常的，3是优先级最低的
	缺陷的产生频率	描述缺陷发生的可能性1%-100%
	缺陷提交人	缺陷提交人的名字（会和邮件地址联系起来），一般就是发现缺陷的测试人员或其他人员
	缺陷提交时间	缺陷提交的时间

# 软件缺陷报告（2）

软件缺陷基本信息	缺陷所属项目/模块	缺陷所属的项目和模块，最好能较精确的定位至模块
	缺陷指定解决人	估计修复这个缺陷的开发人员，在缺陷状态下由开发组长指定相关的开发人员；也会自动和该开发人员的邮件地址联系起来，并自动发出邮件
	缺陷指定解决时间	开发管理员指定的开发人员修改此缺陷的时间
	缺陷验证人	验证缺陷是否真正被修复的测试人员；也会和邮件地址联系起来
	缺陷验证结果描述	对验证结果的描述（通过、不通过）
缺陷的详细描述	缺陷验证时间	对缺陷验证的时间
	步骤	对缺陷的操作过程，按照步骤，一步一步地描述
	期望的结果	按照设计规格说明书或用户需求，在上述步骤之后，所期望的结果，即正确的结果
	实际发生的结果	程序或系统实际发生的结果，即错误的结果
测试环境说明	测试环境	对测试环境描述，包括操作系统、浏览器、网络带宽、通讯协议等
必要的附件	图片、Log文件	对于某些文字很难表达清楚的缺陷，使用图片等附件是必要的；对于软件崩溃现象，需要使用Soft_ICE工具去捕捉日志文件作为附件提供给开发人员。

## 5.4 软件缺陷跟踪系统

- 基于缺陷数据库，可统一数据格式、完成数据校验，而且确保每一个缺陷不会被忽视
- 基于数据库系统，有利于建立各种动态的数据报表，用于项目状态报告和缺陷数据统计分析
- 基于系统可以随时得到最新的缺陷状态，消除沟通上的障碍
- 基于系统可以将缺陷和测试用例、需求等关联起来，更深度的分析，有利于产品的质量改进

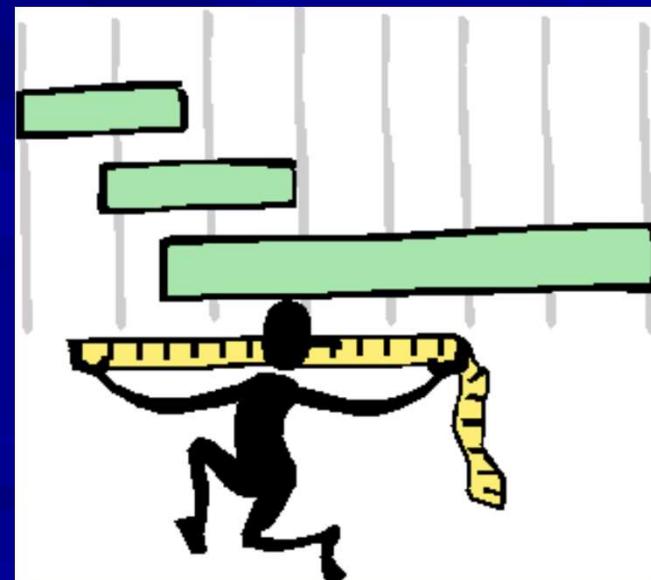
# 6. 软件测试和质量分析报告

- 6.1 软件产品的质量度量
- 6.2 评估系统测试的覆盖程度
- 6.3 基于缺陷分析的产品质量评估
- 6.4 测试报告的具体内容



# 6.1 软件产品的质量度量

- 6.1.1 软件度量及其过程
- 6.1.2 软件质量的度量
- 6.1.3 质量度量的统计方法



## 6.1.1 软件度量及其过程

- 软件度量就是对软件所包含的各种属性的量化表示
- 软件度量可以提供对软件过程和软件产品的深入了解的衡量指标，使组织能够更好地做出决策以达成目标，软件度量具有如下作用
  - 用数据指标表明验收标准
  - 监控项目进度和预见风险
  - 分配资源时进行量化均衡
  - 预计和控制产品的过程、成本和质量

# 软件度量的分类

- 测量（Measurement）、度量（Metric）和指标（Indicator）
- 软件度量的分类
  - 软件过程度量
  - 软件项目度量
  - 产品质量度量



# 软件度量的内容（1）

## ■ 规模度量

- 代码行数，功能点和对象点等

## ■ 复杂度度量

- 软件结构复杂度指标

## ■ 缺陷度量

- 帮助确定产品缺陷变化的状态，并指示修复缺陷活动所需的工作量，分析产品缺陷分布的情况

## ■ 工作量度量

# 软件度量的内容 (2)

- 进度度量
- 生产率度量
  - 代码行数 / 人·月
  - 测试用例数/人·日
- 风险度量
  - “风险发生的概率” 和 “风险发生后所带来的损失”

# 软件度量的分工 (1)

## ■ 度量工作小组

- 由专职的度量研究人员和项目协调人员组成，度量研究人员的主要职责是定义度量过程和指导进行度量活动，并对数据进行分析、反馈；项目协调人员的职责是为定义度量过程提供详细的需求信息，并负责度量过程在项目组的推行。

# 软件度量的分工 (2)

## ■ 数据提供者

- 一般是项目中的研发人员，有时还会包括用户服务人员和最终用户

## ■ IT支持者

- 确定数据提供的格式与数据存储方式，提供数据收集工具与数据存储设备

# 软件度量的过程（1）

## ■ 识别目标

- 分析出度量的工作目标和列表，并由管理者审核确认

## ■ 定义度量过程

- 定义其收集要素、收集过程、分析、反馈过程、IT支持体系，为具体的收集活动、分析、反馈活动和IT设备、工具开发提供指导

## ■ 搜集数据

- 应用IT支持工具进行数据收集工作，并按指定的方式审查和存储

# 软件度量的过程（2）

## ■ 数据分析与反馈

- 根据数据收集结果，按照已定义的分析方法进行数据分析，完成规定格式的图表，进行反馈

## ■ 过程改进

- 根据度量的分析报告，管理者基于度量数据做出决策

## 6.1.2 软件质量的度量

- 软件可靠性度量、复杂度度量、缺陷度量和规模度量
- $M_i = c_1 \times f_1 + c_2 \times f_2 + \dots + c_n \times f_n$ 
  - $M_i$ 是一个软件质量因素（如SQRC层各项待计算值）， $f_n$ 是影响质量因素的度量值（如SQDC层各项估计值）， $c_n$ 是加权因子

## 6.1.3 质量度量的统计方法（1）

- 说明不完整或说明错误（IES）
- 与客户交流不够所产生的误解（MCC）
- 故意与说明偏离（IDS）
- 违反编程标准（VPS）
- 数据表示有错（EDR）
- 模块接口不一致（IMI）

## 6.1.3 质量度量的统计方法（2）

- 设计逻辑有错（EDL）
- 不完整或错误的测试（IET）
- 不准确或不完整的文档（IID）
- 将设计翻译成程序设计语言中的错误（PLT）
- 不清晰或不一致的人机界面（HCI）
- 杂项（MIS）

## 6.1.3 质量度量的统计方法（3）

总计( $E_i$ )			严重( $S_i$ )		一般( $M_i$ )		微小( $T_i$ )	
错误	数量	百分比	数量	百分比	数量	百分比	数量	百分比
<b>IES</b>	296	<b>22.3%</b>	55	<b>28.2%</b>	95	18.6%	146	23.4%
<b>MCC</b>	204	<b>15.3%</b>	18	9.2%	87	17.0%	99	15.9%
<b>IDS</b>	64	4.8%	2	1.0%	31	6.1%	31	5.0%
<b>VPS</b>	34	2.6%	1	0.5%	19	3.7%	14	2.2%
<b>EDR</b>	182	<b>13.7%</b>	38	<b>19.5%</b>	90	17.6%	54	8.7%
<b>IMI</b>	82	6.2%	14	7.2%	21	4.1%	47	7.5%
<b>EDL</b>	64	4.8%	20	<b>10.3%</b>	17	3.3%	27	4.3%
<b>IET</b>	140	<b>10.5%</b>	17	8.7%	51	10.0%	72	11.6%
<b>IID</b>	54	4.1%	3	1.5%	28	5.5%	23	3.7%
<b>PLT</b>	87	6.5%	22	<b>11.3%</b>	26	5.1%	39	6.3%
<b>HCI</b>	42	3.2%	4	2.1%	27	5.3%	11	1.8%
<b>MIS</b>	81	6.1%	1	0.5%	20	3.9%	60	9.6%
<b>总计</b>	1330	100%	195	100%	512	100%	623	100%

# 6.2 评估系统测试的覆盖程度

- 6.2.1 对软件需求的估算
- 6.2.2 基于需求的测试覆盖评估
- 6.2.3 基于代码的测试覆盖评估



# 测试的评估

## ■ 软件测试评估主要有两个目的

- 量化测试进程，判断测试进行的状态和进度
- 为测试或质量分析报告生成所需的量化数据，如缺陷清除率、测试覆盖率等

# 示例

测试覆盖项	测试覆盖率指标测试描述	测试结果
界面覆盖	符合需求（界面图标、信息区、状态区）	
静态功能覆盖	功能满足需求	
动态功能覆盖	所有功能的转换功能正确	
正常测试覆盖	所有硬件软件正常时处理	
异常测试覆盖	硬件或软件异常时处理（不允许的操作）	测试结束判断

## 6.2.1 对软件需求的估算 (1)

- 假设有R个需求， 功能需求的数目为F， 非功能需求数为N，则
  - $R = F + N$
  - $Q_1 = M/R$
- 其中 $Q_1$ 表示需求的确定性，M是所有复审者都有相同解释的需求数目

## 6.2.1 对软件需求的估算 (2)

### ■ 功能需求的完整性Q2

$$- Q2 = Fu / (Ni \times Ns)$$

- 其中Fu是唯一功能需求的数目，Ni是由规格设计说明书定义的输入个数，Ns是被表示的状态的个数

### ■ 考虑非功能需求

$$- Q3 = Fc / (Fc + FnV)$$

- 其中Fc是已经确认为正确的请求的个数，FnV是尚未被确认的请求的个数

## 6.2.2 基于需求的测试覆盖评估

- 假定  $T_x$  是已执行的测试过程数或测试用例数,  $R_{ft}$  是测试需求的总数
  - 已执行的测试覆盖 =  $T_x/R_{ft}$
- 假定  $T_s$  是已执行的完全成功、没有缺陷的测试过程数或测试用例数
  - 成功的测试覆盖 =  $T_s/R_{ft}$

## 6.2.3 基于代码的测试覆盖评估

- 基于代码的测试覆盖评估是对被测试的程序代码语句、路径或条件的覆盖率分析。这种测试覆盖策略对于安全至上的系统来说非常重要
- 基于代码的测试覆盖通过以下公式计算
  - 已执行的测试覆盖 =  $Tc/Tnc$ 
    - 其中  $Tc$  是用代码语句、条件分支、代码路径、数据状态判定点或数据元素名表示的已执行项目数， $Tnc$  (**Total number of items in the code**) 是代码中的项目总数

# 6.3 基于缺陷分析的产品质量评估

- 6.3.1 缺陷评测的基线
- 6.3.2 经典的种子公式
- 6.3.3 基于缺陷清除率的估算方法
- 6.3.4 软件产品性能评估



## 6.3.1 缺陷评测的基线

- 为软件产品的质量设置起点，在基线的基础上再设置新的目标，作为对系统评估是否通过的标准

条目	目标	低水平
缺陷清除效率	>95%	<70%
原有缺陷密度	每个功能点 <4	每个功能点 >7
超出风险之外的成本	0%	>=10%
全部需求功能点	<1% 每个月平均值	>=50%
全部程序文档	每个功能点页数 <3	每个功能点页数 >6
员工离职率	每年1 to 3%	每年>5%

## 6.3.2 经典的种子公式 (1)

$$\frac{\text{已测试出的种子Bug (s)}}{\text{所有的种子Bug (S)}} = \frac{\text{已测试出的非种子Bug (n)}}{\text{全部的非种子Bug (N)}}$$

■ 则可以推出程序的总Bug数为

$$- N = S * n / s$$

■ 其中n是所进行实际测试时发现的Bug总数。如果  $n = N$ , 说明所有的Bug已找出来, 说明做的测试足够充分

## 6.3.2 经典的种子公式 (2)

- 这种测试是否充分，可以用一个信心指数来表示，即用一个百分比表示，值越大，说明对产品质量的信心越高，最大值为1。

$$C = \begin{cases} 1 & \text{if } n > N \\ S/(S-N+1), & \text{if } n \leq N \end{cases}$$

### 6.3.3 基于缺陷清除率的估算方法（1）

- F为描述软件规模用的功能点； D1为在软件开发过程中发现的所有缺陷数； D2为软件发布后发现的缺陷数； D为发现的总缺陷数。
  - 。因此，  $D = D_1 + D_2$
  - 质量= $D_2/F$
  - 缺陷注入率= $D/F$
  - 整体缺陷清除率= $D_1/D$

### 6.3.3 基于缺陷清除率的估算方法（2）

缺陷源	潜在缺陷	清除效率 (%)	被交付的缺陷
需求报告	1.00	77	0.23
设计	1.25	85	0.19
编码	1.75	95	0.09
文档	0.60	80	0.12
错误修改	0.40	70	0.12
合计	5.00	85	0.75

## 6.3.4 软件产品性能评估（1）

- 软件产品性能评估其技术性相对比较强，方法的基础是获取与性能表现相关的数据。
  - 。 性能评测一般和测试的执行结合起来做，或者是在执行测试时记录、保存各种数据，然后在评估测试活动中进行计算结果

## 6.3.4 软件产品性能评估（2）

- 主要的性能评测包括

- 动态监测
- 响应时间 / 吞吐量
- 百分比报告
- 比较报告
- 追踪报告



## 6.4 测试报告的具体内容（1）

- 在国家标准GB/T 9386-2008对测试报告有了具体要求，对测试记录、测试结果如实汇总分析，报告出来。测试报告应具有如下结构
  - 产品标识
  - 用于测试的计算机系统
  - 使用的文档及其标识
  - 产品描述、用户文档、程序和数据的测试结果

## 6.4 测试报告的具体内容（2）

- 与要求不符的清单
- 针对建议的要求不符的清单，产品未作符合性测试的说明
- 测试结束日期

# 7. 软件测试项目管理

- 7.1 测试项目管理的特点
- 7.2 如何做好测试项目管理
- 7.3 软件测试项目的资源管理
- 7.4 测试项目的进度管理
- 7.5 测试项目的风险管理
- 7.6 软件测试文档的管理
- 7.7 软件测试的误区
- 7.8 软件测试的原则

# 7.1 测试项目管理的特点

- 软件测试存在较大风险，质量标准定义不准确、任务边界模糊
- 软件测试项目的变化控制和预警分析要求高
- 软件测试管理要求更严格和细致
- 测试任务的分配难
- 测试要求人力资源十分稳定
- 测试人员在待遇、地位可能受到一些不公正的待遇

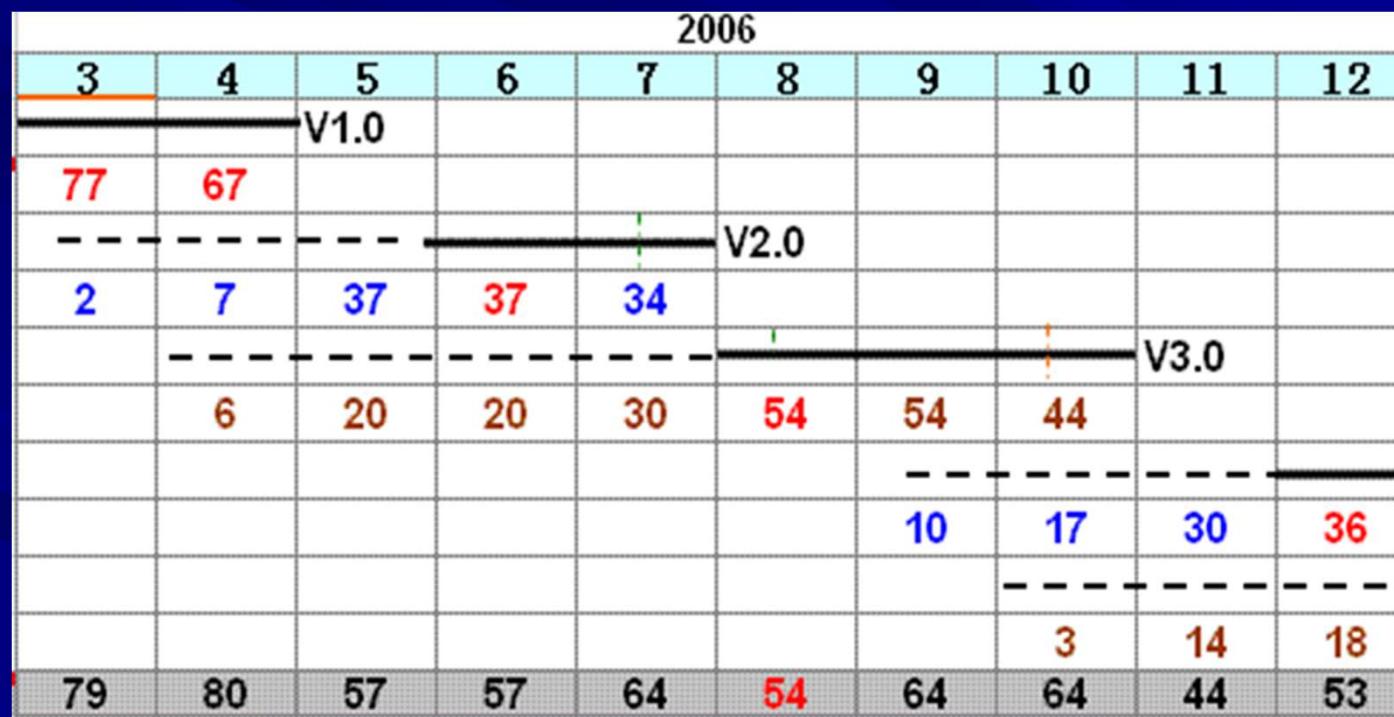
## 7.2 如何做好测试项目管理

- 测试项目的管理原则
- 测试计划先行
- 建立优先级
- 依靠团队的能力
- 建立客观的评价标准
- 软件测试项目管理者的责任



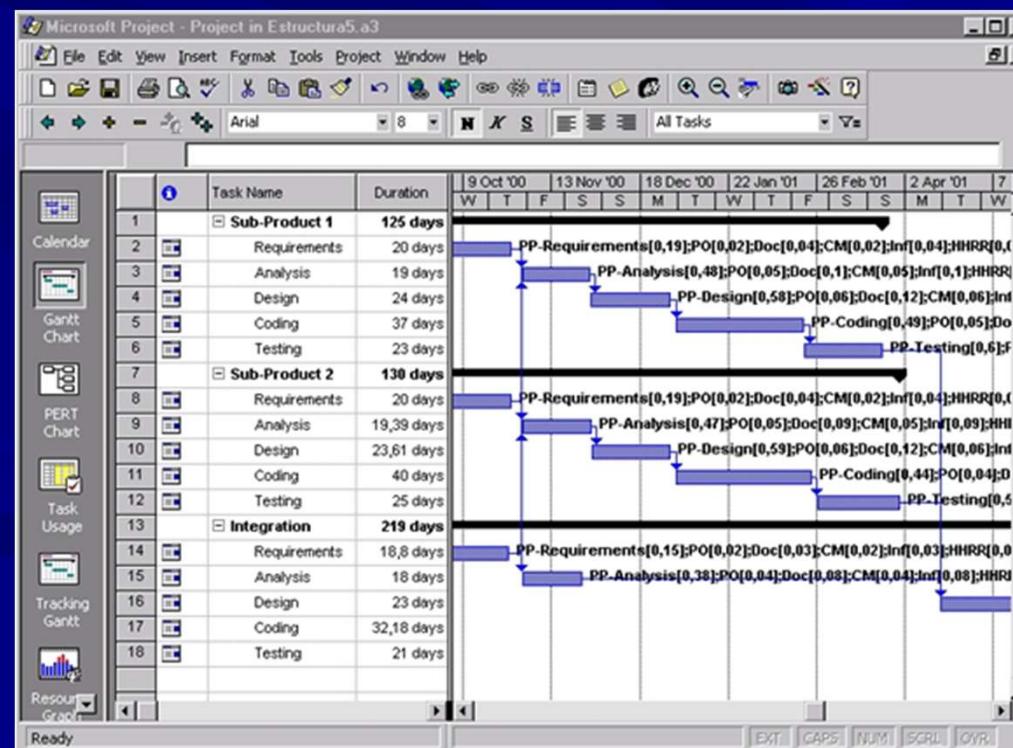
# 7.3 软件测试项目的资源管理

- 人力资源管理
- 测试环境资源
- 工作量的估计



# 7.4 测试项目的进度管理

- 7.4.1 测试项目的里程碑和关键路径
- 7.4.2 测试项目进度的特性及外在关系
- 7.4.3 测试项目进度的管理方法和工具



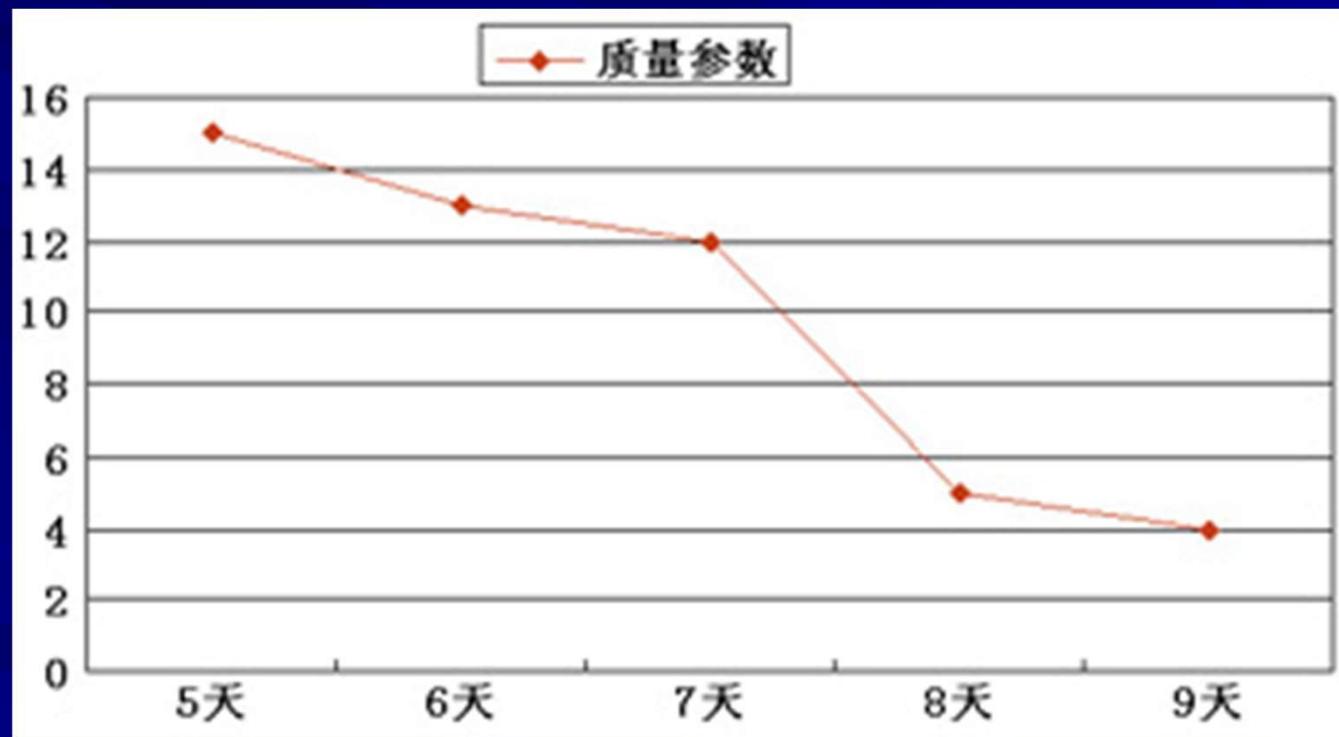
# 7.4.1 测试项目的里程碑和关键路径

## ■ 测试项目的里程碑

任务	天	任务	天	任务	天	任务	天
<b>M21: 测试计划制定</b>	<b>11</b>	<b>M23: 测试设计</b>	<b>12</b>	开发测试过程	5	验证测试结果	2
确定项目	1	测试用例的设计	7	测试和调试测试过程	2	调查突发结果	1
定义测试策略	2	测试用例的审查	2	修改测试过程	2	生成缺陷日记	1
分析测试需求	3	测试工具的选择	1	建立外部数据集	1	<b>M62: 测试评估</b>	3
估算测试工作量	1	测试环境的设计	2	重新测试并调试测试过程	2	评估测试需求的覆盖率	1
确定测试资源	1	<b>M26: 测试开发</b>	<b>15</b>	<b>M42: 功能测试</b>	9	评估缺陷	0.5
建立测试结构组织	1	建立测试开发环境	1	设置测试系统	1	决定是否达到测试完成的标准	0.5
生成测试计划文档	2	录制和回放原型过程	2	执行测试	4	测试报告	1

## 7.4.2 测试项目进度的特性及外 在关系

- 进度与质量关系
- 进度与成本的关系

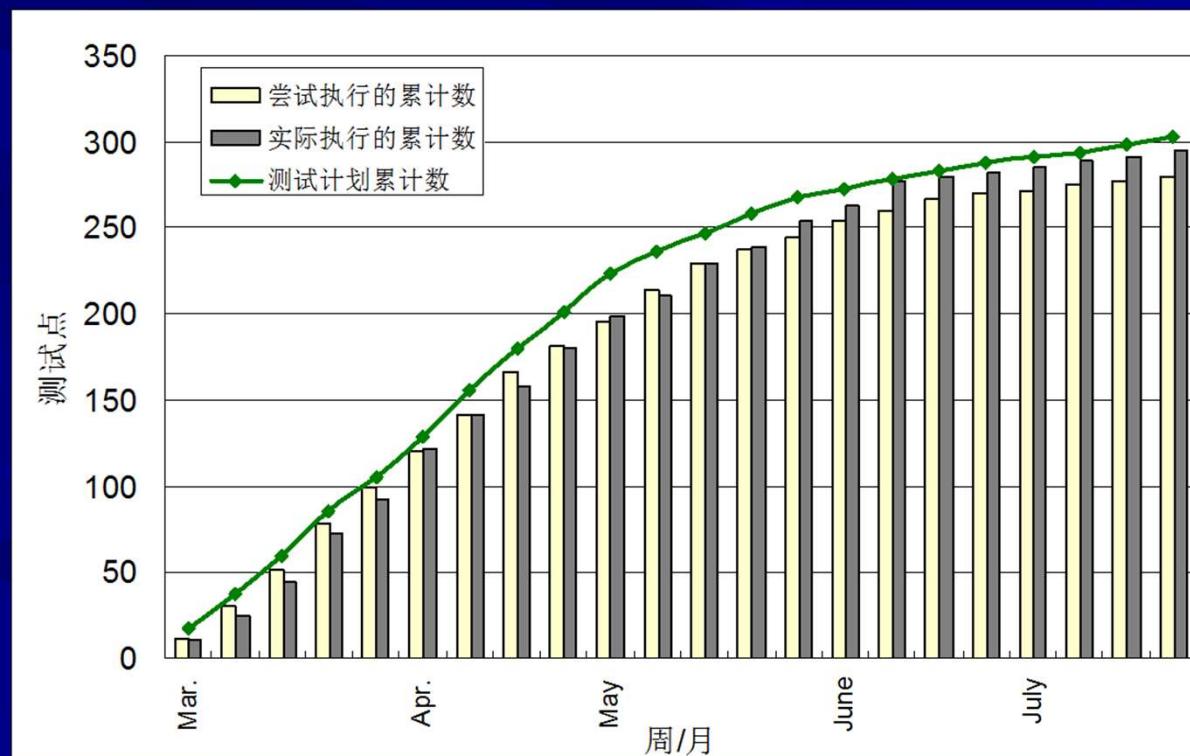


## 7.4.3 测试项目进度的管理方法 和工具

- 测试进度S曲线法
- 测试进度NOB曲线法

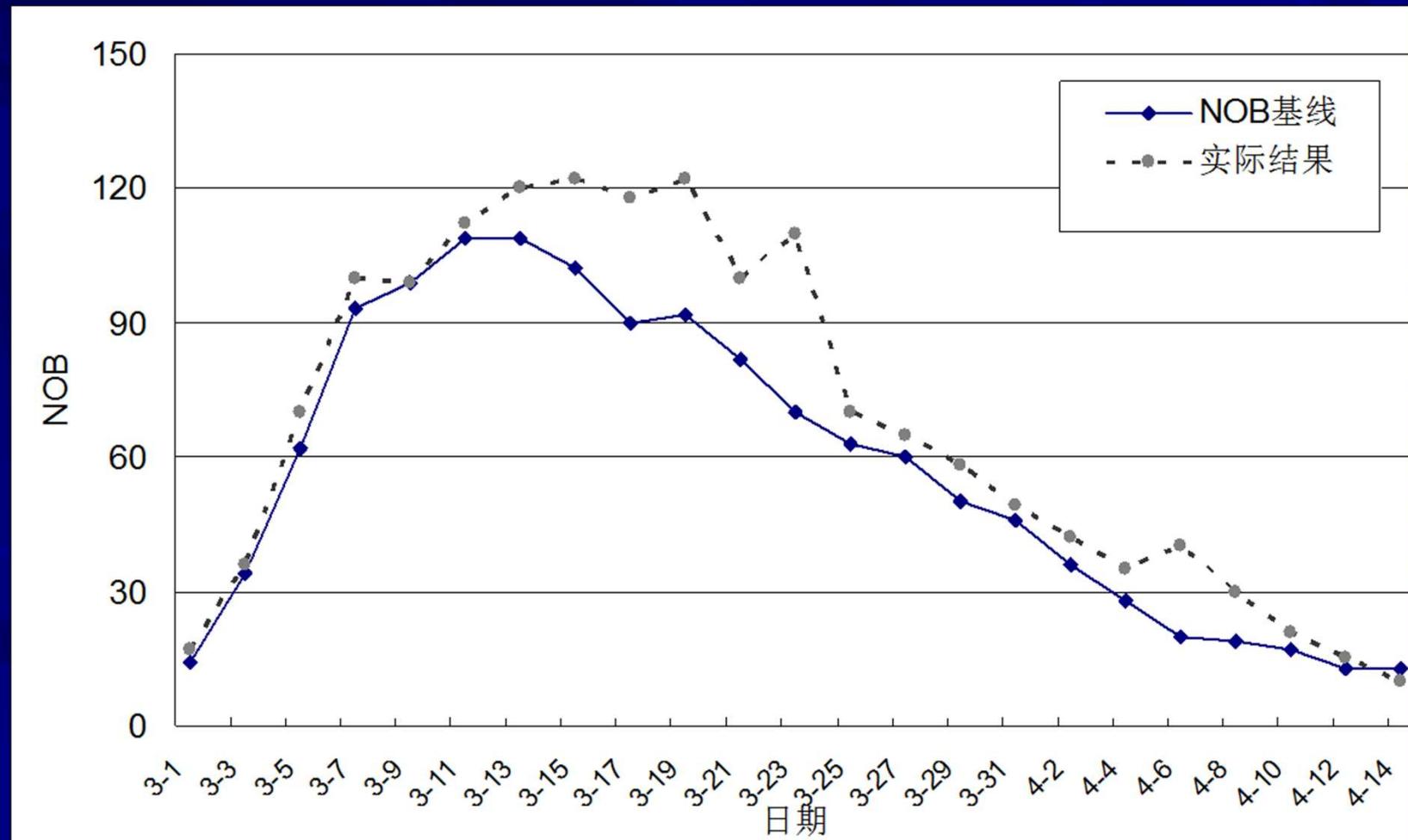
# 测试进度S曲线法

- 进度S曲线法通过对计划中的进度、尝试的进度与实际的进度三者对比来实现的，其采用的基本数据主要是测试用例或测试点的数量。



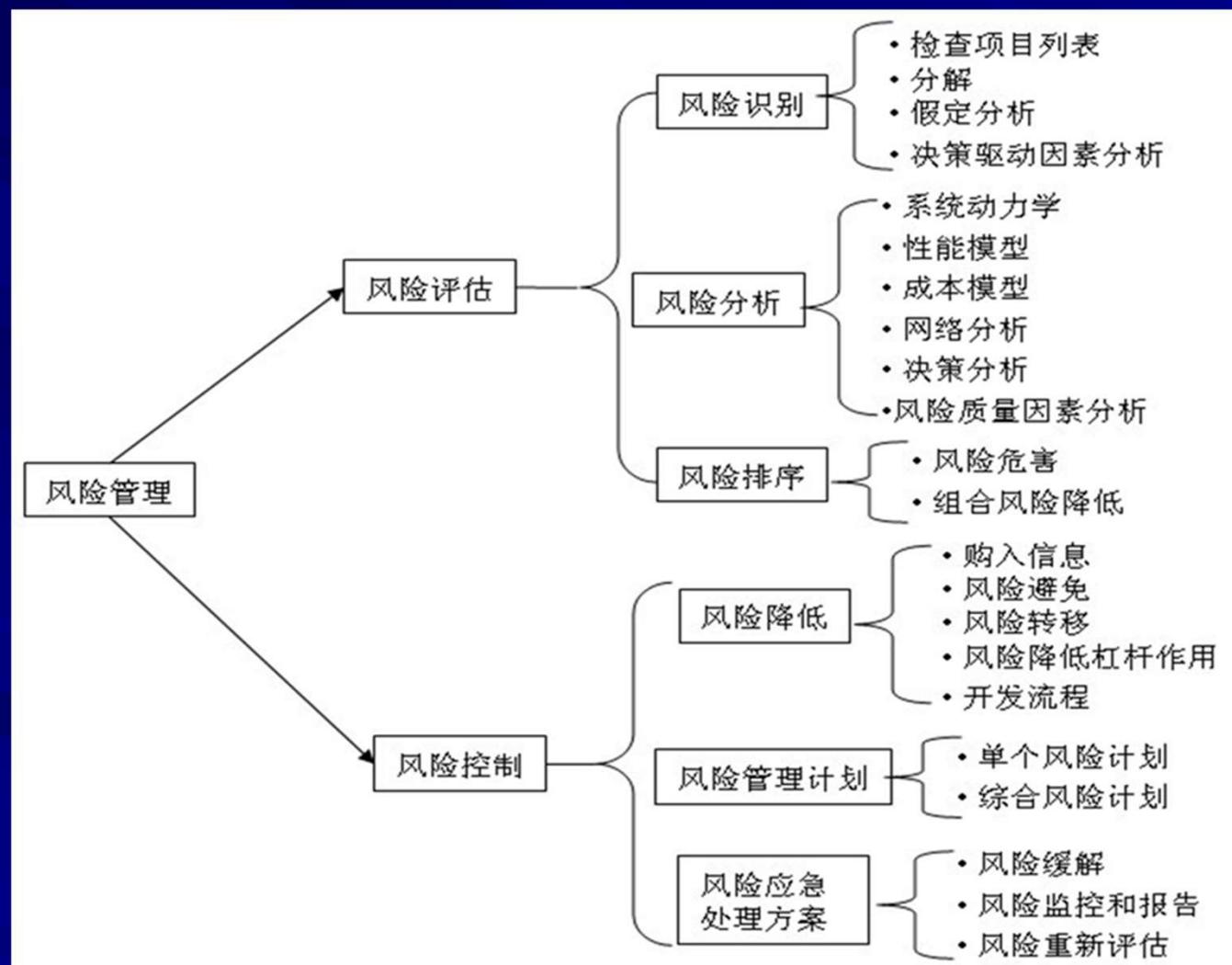
# 测试进度NOB曲线法

## ■ NOB, Number of Open Bug



# 7.5 测试项目的风险管理

## ■ 风险管理的内容和对策



# 7.6 软件测试文档的管理

- 文档的分类管理
- 文档的格式和模板管理
- 文档的一致性管理
- 文档的存储管理



# 7.7 软件测试的误区（1）

## ■ 误区一

- 如果发布出去的软件有质量问题，都是软件测试人员的错

## ■ 误区二

- 软件测试技术要求不高，至少比编程容易多了

## ■ 误区三

- 有时间就多测试一些，来不及就少测试一些

# 7.7 软件测试的误区（2）

## ■ 误区四

- 软件测试是测试人员的事，与开发人员无关

## ■ 误区五

- 根据软件开发瀑布模型，软件测试是开发后期的一个阶段

## 7.8 软件测试的原则（1）

- 所有测试的标准都是建立在用户需求之上
- 软件测试必须基于“质量第一”的思想去开展各项工作，当时间和质量冲突时，时间要服从质量
- 事先定义好产品的质量标准，只有有了质量标准，才能根据测试的结果，对产品的质量进行分析和评估

## 7.8 软件测试的原则（2）

- 软件项目一启动，软件测试也即开始，而不是等程序写完，才开始进行测试
- 穷举测试是不可能的。甚至一个大小适度的程序，其路径排列的数量也非常大，因此，在测试中不可能运行路径的每一种组合
- 第三方进行测试会更客观，更有效
- 软件测试计划是做好软件测试工作的前提

## 7.8 软件测试的原则（3）

- 测试用例是设计出来的，不是写出来的，所以要根据测试的目的，采用相应的方法去设计测试用例，从而提高测试的效率，更多地发现错误，提高程序的可靠性
- 对发现错误较多的程序段，应进行更深入的测试。一般来说，一段程序中已发现的错误数越多，其中存在错误的概率也就越大
- 重视文档，妥善保存一切测试过程文档（测试计划、测试用例、测试报告等）

*Thank You*