第一章软件测试背景

缺陷累积：（个人理解）不同的开发阶段中产生的缺陷随着开发过程的进行而累积

软件危机：

1. 背景：落后的生产方式无法满足日趋复杂大型软件系统的开发需要
2. 表现：项目延期、经费超支、产品无法维护
3. 原因：缺乏规范化工程约束导致缺陷的不断累积与放大效应，缺陷修复的代价随着开发时间快速增加

软件测试第一个定义：测试就是建立一种信心，认为程序能够按照预期设想运行（错误）正确：

1979-1982面向缺陷发现的时代

Glenford J.Myers:测试是尽可能多的发现软件错误

软件测试定义：测试是为发现错误而执行一个程序或系统的过程

他的三个重要观点：

1. 测试是为了证明程序有错，而不是证明程序无错误
2. 一个好的测试用例是在于它能发现至今为止未发现的错误
3. 一个成功的测试用例是发现了至今未发现的错误的测试

第二章 软件测试基本概念

测试与调试之间区别：

测试是一个有计划的可重复的过程，目的是为了发现软件潜在的错误和缺陷；调试是一个随机的不可重复的过程，目的是为了找出错误的原因和位置并修复。

调试一般在测试之后但调试又要测试，二者经常交叉进行。

软件测试意义：

1. 保证软件质量的重要手段
2. 深入软件开发过程的每个阶段，在有限的开发条件下，最大程度地保证最终软件产品符合用户需要

软件测试定义：分析某个软件项以发现现存和要求的条件之差别并评价此软件项的特性（IEEE)

软件测试目的：

1. 确保软件质量

2. 确保软件开发过程方向的正确性

三个概念区别：

fault: a static defect in the software（软件存在的问题

error:an incorrect internal state that is the manifestation of some faults

failure: External, incorrect behavior with respect to the requirements or other description of the expected behavior

fault产生error，从而造成了failure

软件测试对象：程序、需求分析、设计工作、文档

测试原理：（9个）

1. 用户需求至上
2. 测试是有计划的活动：测试贯穿于全部软件生存周期
3. 缺陷出现的集群性
4. 测试应从“小规模”走向“大规模”
5. 穷尽测试是不可能的
6. 有效的测试应该由第三方独立进行
7. 测试无法揭示所有缺陷：测试只能说明缺陷的存在而不能说明软件没有缺陷
8. 测试的杀虫剂悖论
9. 测试是有风险的行为

过程：

1. 拟定测试计划
2. 编制测试大纲
3. 设计测试用例
4. 实施测试
5. 分析测试结果

测试用例三要素：输入、执行条件、期望输出（设计原则：代表性、可判定性、可再现性）

软件测试类型：

1. 测试技术：白盒测试、黑盒测试、灰盒测试
2. 开发阶段：（7个）
    a. 单元测试：对最小的设计单元---模块的验证工作

b. 集成测试： 验证模块间的接口是否正确，多个模块是否能够协调一致的正确实现需求和功能

c. 系统测试：测试整个系统的行为和错误属性

d. 性能测试：评价系统属性并与不同版本或竞争产品进行比较

e. 确认测试：验证软件是否可以按照用户合理的期望方式工作

f. 验收测试：保证客户对所有需求都满意

i. α测试：开发公司内部人员模拟各类用户进行的软件产品测试

ii. β测试：用户进行的测试，目的在于帮助开发方在产品发布前做最后的改进。

iii. α测试调整的软件产品是β测试

g. 回归测试：保证增强型或改正型修改使软件正常运行，不影响现有功能

3. 执行状态：静态测试、动态测试

静态：不运行测试，通过人工对程序和文档进行分析和检查

动态：通过人工或利用工具运行程序进行检查，分析程序执行状态和外部表现

4. 执行主体：开发方测试、用户测试、第三方测试

5. 特殊测试：（7个）

a. 国际化测试：保证全球化软件产品符合不同国家的语言和使用习惯

b. 即兴测试：通过直觉和经验，不采用任何形式化的测试。针对有经验的工程师

c. 兼容性测试：确保软件在不同基础设施下都能够一致地发挥作用

d. 安全性测试：测试软件是否存在安全漏洞和隐患

e. 可用性与易获得性测试：确认产品的易用性、美感以及是否方便行动不便的用户

f. 面向对象系统测试：针对采用面向对象技术开发的软件所使用的测试技术

g. Web测试：发现存在于web应用中的内容、功能、性能、安全性等方面的错误的测试活动集

W模型：

在V模型的基础上，增加与开发阶段的同步测试，形成W模型；测试与开发同步进行，有利用尽早的发现问题

优点：有利于尽早地全面地进行测试，发现软件中存在的问题，有利于全过程的测试

缺点：无法支持迭代、自发性、变更调整

软件测试现状：

逐渐受到重视，开发：测试=1:1

第三章白盒测试

实施者：

单元测试阶段：开发人员

集成测试阶段：测试人员和开发人员共同完成

动态白盒测试步骤：

1. 程序逻辑分析
2. 生成测试用例
3. 执行测试
4. 分析覆盖标准
5. 判定测试结果

进入条件：编码开始阶段

退出条件：

1. 完成测试计划（满足一定覆盖率）
2. 发现并修正了错误
3. 预算和开发时间

静态白盒测试

定义：在不执行软件的条件下有条理地仔细审查软件设计、体系结构和代码，从而找出软件缺陷的过程，有时称为结构化分析

测试条件：只要求提供软件源代码，不要求提供可执行程序，即不用在计算机上执行程序，而由人阅读代码。

使用理由：

1. 尽早发现软件缺陷
2. 为后续测试中设计测试用例提供思路

静态白盒测试步骤：（规范化程度：弱-->强）

1. 桌面检查
    a. 实施者：代码编写者
    b. 特点：
        i. 无结构化或形式化方法保证
        ii. 不维护记录或检查单
        iii. 格式：
    c. 优点;
        i. 编码者容易理解和阅读自己代码
        ii. 开销小，没有指定进度
        iii. 尽早发现缺陷
    d. 缺点
        i. 开发人员不是事实最佳人员
        ii. 有效性难以保证
2. 代码走查/代码检查：以组为单位进行代码阅读的测试
    a. 特点：
        i. 由特定人员组成的团队通过会议完成
        ii. 与会者充当计算机执行测试用例
        iii. 开发人员及时回答与会者提出的问题
    b. 区别;
        i. 人员分工：代码检查有主持人、编码者、测试专家、程序员；代码走查还有其他项目成员、记录人

员、（高级程序员、程序语言专家、初级程序员）

ii. 规程上：代码检查：编码者逐条语句讲述程序逻辑结构、与会人员根据核对表发现错误； 代码走查：测试人员会先带着事先设计的测试用例参与会议、参与者使用“头脑”作为计算机模拟程序执行

3. 代码审查

a. 审查小组：主持人、作者、评论员、记录员

b. 审查步骤：计划--------概述-------准备-------审查会议--------审查报告-------返工----------跟进

i. 计划：主持人做计划

ii. 概述：描述技术背景

iii. 准备：评论员审查代码核对表

iv. 审查会议：阅读代码--讨论--提问---记录错误（类型、严重级别）--不讨论解决防范

v. 审查报告：缺陷列表，核对表基础

vi. 返工：分配缺陷并修复

vii. 跟进：监督、审查修复部分

c. 代码审查作用：

i. 发现代码缺陷

ii. 提高代码质量

iii. 及早定位缺陷群集位置

d. 挑战：

i. 耗费时间

ii. 设计多人参加

iii. 不能保证参与者全部理解程序

动态白盒测试

特点：不但要提供软件源代码，还要提供可执行程序，测试过程需要在计算机上执行程序

优点：

1. 检测代码中的判断和路径
2. 揭示隐藏在代码中的错误
3. 对代码的测试比较彻底

缺点：

1. 无法检测代码中不可达路径
2. 不验证需求规格

基于控制流的测试方法（注意逻辑短路的影响）六种基本+MC/DC+基本路径测试

判定 VS 条件：一个判定可以有多个条件组成

六种基本测试方法----绘制程序流程图（一个入口、一个出口）内部写语句

1. 语句覆盖：程序中每条语句都执行一遍，控制流图节点覆盖
    a. 100%的语句覆盖很困难：处理错误的代码片段； 不可达代码；小概率事件
    b. 100%语句覆盖无法发现某些严重问题，原因在于会遗漏某些不存在语句的边

2. 判定覆盖（分支覆盖）：每个判定都去T/F 至少一次，即判定的每个分支至少经过一次； 控制流图的边覆盖
    a. 优点：简单，避免语句覆盖的问题
    b. 缺点：忽略了表达式内部的条件，不能发现每个条件的错误

3. 条件覆盖：每个判定中的每个条件的取值都有T、F至少一次
    a. 对有逻辑短路机制的现代语言：同时也满足了语句、判定覆盖
    b. 对无逻辑机制的早期语言：覆盖率不稳定（50%~100%）；同时可能遗漏语句和分支

4. 判定条件覆盖：每个判定且每个判定中的每个条件T、F至少取一次

a. 故障敏感：

b. 错误屏蔽：原子条件取值不会影响判定结果，因此该条件上的取值错误不可见----------100%覆盖，但不能检测出测试用例错误

5. 条件组合覆盖：每个条件的取值组合都至少出现一次

a. 代价昂贵：一个判定中有n个原子条件，不考虑逻辑短路则有2的n次方个组合

b. 某些条件组合不可能

6. 路径覆盖：覆盖程序中所有可能路径，如果有环路，每条环路至少经过一次

a. 如果判定串联：本质是判定的组合

b. 判定非串联：具体分析

c. 优点：较为彻底

d. 缺点：

i. 路径分支指数增加

ii. 不可达路径的存在

iii. 不能替代设计条件测试的方法

在考虑逻辑短路情况下，单个判定中，条件覆盖=判定条件覆盖

条件覆盖！=条件组合覆盖

局限性总结：

- 语句覆盖：可能遗漏控制流边
- 判定覆盖：未考虑判定中的条件
- 条件覆盖：可能遗漏语句和控制流边
- 判定条件覆盖：覆盖能力强，但不能替代条件组合和路径覆盖；实际应用中，测试用例选择应注意错误屏蔽；
- 条件组合覆盖：可能产生较多测试用例，不能覆盖所有路径
- 路径覆盖：不能替代涉及条件的测试;路径数

MC/DC覆盖：绘制程序流程图

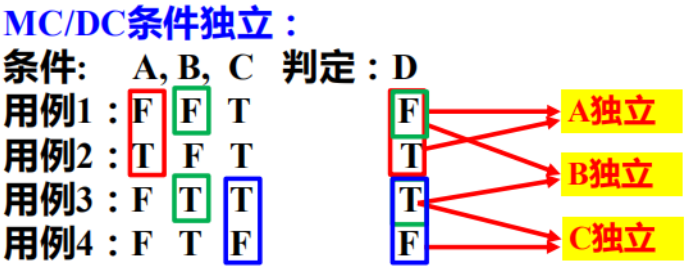
100%MC/DC覆盖所需测试用例：设n为条件数，最少：n+1，最多：2n

覆盖条件：

1.每个判定都去T、F至少一次

2.每个条件都取T、F至少一次

3.判定中的每个条件对判定取值独立



基本路径测试：绘制流图！！！（一个入口、一个出口）内写标号，标号代表原子语句

设计过程：流图-------环复杂度---------基本路径----------测试用例

基本路径：流图中由开始结点到结束结点

流图组成：

1. 结点：表示一个或多个过程语句
2. 边：表示控制流
3. 域：由边和结点限定的区间

绘制时：

1. 若判定由多个条件组成，每一个原子条件作为一个结点
2. 连续语句对应结点可以合并
3. 一样的语句作为一个节点，例如若出现多处return False，只有一个结点表示该语句

环复杂度=边数-结点数+2=域的数量

基本路径数=环复杂度

基本路径集合不唯一

基本路径集寻找算法？

Step1: 确认从入口到出口的最短基本路径

Step2:从入口到第1个未被先后评估为真和假两种结果的条件语句

Step3:改变该条件语句的判断值

Step4:按最短路径从这个条件语句到出口

Step5:重复步骤2-5,直到所有基本路径都被找到

优点：

1. 更细的控制流描述-------程序流图，一定要对复合条件进行处理
    a. 同时刻画判定和条件对程序控制流的影响
    b. 判定条件覆盖
2. 更少的覆盖路径
    a. 近似的路径覆盖
    b. 数学和算法理论支撑

对循环的处理：

基于数据流的测试方法-------数据流覆盖，程序流图

步骤：

1. 绘制程序流图
2. 分析变量的定义节点和使用节点
3. 可能的路径数
4. 列出DU路径数目=定义节点数*使用节点数
5. 化简DU路径

重点在于区分定义节点和使用节点：！！！！！重点复习掌握

！！！！指针操作：！！！！！！对*p或p操作都要考虑p和*p的节点类型

1. 针对指针*p时，非声明位置出现*p都一定有p的使用节点，*p是什么节点取决于语句
2. 而单独出现++p或者--p时，p一定是定义使用节点，*p是定义节点

定义节点：变量声明、初始化、分配存储空间、变量值改变，（定义节点中不一定出现对应变量）

使用节点：从内存中读取变量值并使用

1. 谓词使用，存在于条件判断语句中
2. 运算使用，计算表达式中
3. 输出使用，输出到屏幕
4. 定位使用，变量值用于定位数组位置，例如a[x]中x则是定位使用
5. 迭代使用，用于控制循环次数

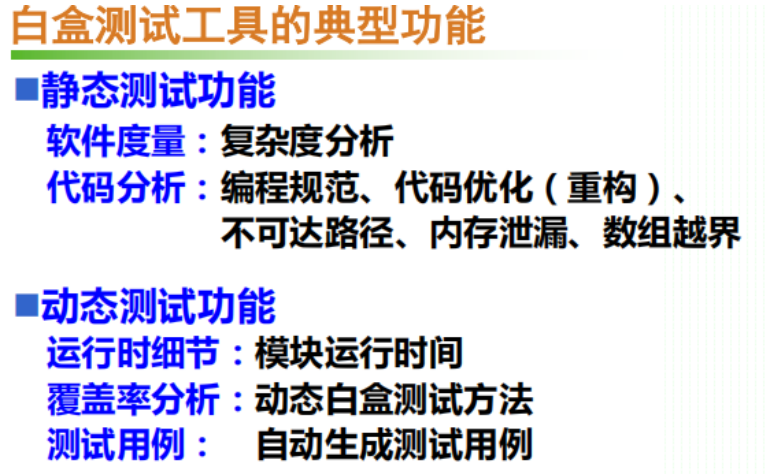
定义节点DEF(v,n)和使用节点USE(v,n)

```
int a=5;                                    (a的定义节点)
int *p=&a;          (p和*p的定义节点，a的使用节点)
*p=3;      (p的使用节点,*p的定义节点,a的定义节点)
p++;                 ( p的定义使用节点；*p的定义节点 )
```

du-path：以定义节点为起点，使用节点为终点

dc-path：du-path的路径中没有该变量的其他定义节点

白盒测试工具：IBM Rational 、 ParaSoft、 开源Xunit测试开发框架



静态测试工具：LogiScope

具体功能：

1. 改进代码，并定位易出错的模块
2. 提供图形化软件度量信息，预测和诊断代码问题
3. 代码优化重构，降低维护成本

Purify

具体功能：
1. 检查内存泄漏
2. 检查数组越界

Quantify

具体功能：
1. 性能瓶颈分析：记录执行时间细节，包括语句和函数
2. 性能瓶颈解决：定位和修改性能问题
3. 性能比较：比较不同修改版本的性能

PureCoverage

具体功能：
1. 覆盖率统计：当前测试用例的覆盖率
2. 自动程序插桩：自动获取覆盖信息

第四章黑盒测试（功能测试、基于规格说明的测试）

出现原因：

1. 无法获得源代码；
2. 尽早黑盒测试的益处——发现软件功能缺陷；
3. 弥补遗漏的逻辑缺陷
4. 适用于各个测试阶段（单元测试、集成测试、系统测试、回归测试）

定义：一种基于规格说明，不要求考察代码，以用户视角进行的测试

意义：

1. 检查明确需求和隐含需求
2. 采用有效输入/输出 和无效输入/输出

3. 包含用户视角

步骤：

1. 阅读规格说明书
2. 设计测试用例
3. 执行测试
4. 分析测试结果

实施者：专门的软件测试部门；有经验的测试人员。

进入条件：编码开始：设计测试数据并执行测试。

退出条件：

1. 完成测试计划
2. 发现并修正了错误
3. 预算和开发时间

基于需求的测试

目的：确认软件需求规格说明书列出的需求

前提：需求已经过仔细评审、隐含需求明确化

RTM（需求跟踪矩阵）

需求跟踪矩阵样本

需求标识	需求描述	优先级	测试条件	用例标识	测试阶段
BR-01	插入号码为123的钥匙并顺时针转动，应能上锁	高	使用号码为123的钥匙	LOCK_001	单元 组件
BR-02	插入号码为123的钥匙并逆时针转动，应能开锁	高	使用号码为123的钥匙	LOCK_002	单元 组件
BR-06	锁受重物撞击也不能被打开	中	使用石头砸锁	LOCK_005 LOCK_006 LOCK_011	系统
BR-08	开锁和上锁转动方向可变，便于左手人士使用	低			未实现

作用：

1.可跟踪每个需求的测试状态而不会遗漏任何需求

2.优先执行优先级高的测试用例，尽早发现高优先级区域内缺陷

3.可导出特定需求对应的测试用例清单

4.评估测试工作量和测试进度的重要数据

正面测试：通过一组预期输入输出验证产品需求

目的：证明软件对于每条规格说明和期望都能通过

负面测试：输入非预期输入时产品没有失败

目的：使用产品没有涉及和预想到的场景，尝试使系统垮掉

	正面测试	负面测试
测试条件	根据需求说明规格	测试条件都在需求规格说明之外
目的	验证需求规格说明的正确性	通过未知条件搞垮软件
覆盖率计算	覆盖需求和条件	没有覆盖率
挑战性	根据规格说明设计测试用例	需要高度创造性产生尽可能多的未知条件

黑盒测试方法

等价类划分：有效等价类+无效等价类

总结：

1. 非冗余性：尽可能使等价类间没有交叉，保证测试用例不存在冗余
2. 完整性：等价类的并集等于输入/输出域
3. 局限：忽略边界位置的测试用例

边界值分析：

	边界值附近数据	测试用例设计思路
字符	Min-1, Min, Min+1 Max-1,Max,Max+1	文本框允许输入1-255个字符: 0,1,2,254,255,256
数值范围	Min-1, Min, Min+1 Max-1,Max,Max+1	输入数据允许1-999: 0，1，2，998，999，1000
集合空间	0，Min, Min+1 Max-1,Max,Max+1	图片大小不能超过50k: 0, 1k, 49k, 50k, 51k

因果分析法：

步骤：

1. 分析规格说明书，识别因果
2. 在因果图连接原因和结果
3. 标明原因之间和结果之间约束条件
4. 转化为因果图列表进而生成决策表
5. 决策表的规则转化为测试用例

因果图绘制：根据规格说明书

原理：软件输入和输出之间存在逻辑关系（因果图）

表示：Ci表示原因、Ei表示结果

（1）恒等（—）

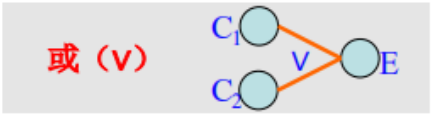
若C_i出现，则E_i出现；
若C_i不出现，则E_i也不出现

（2）非（ ~ ）

若C出现，则E不出现；
若C不出现，则E出现

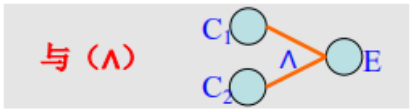
（3）或（ ∨ ）

若几个C_i中有一个出现，则E出现;
若几个C_i都不出现，则E不出现



（4）与（ ∧ ）

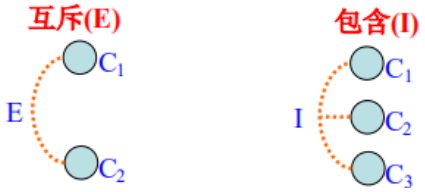
若几个C_i都出现，则结果出现;
若其中一个C_i不出现，则E不出现



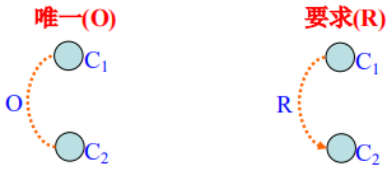
输入约束：

（1）互斥(E):多个原因不能同时成立，最多有一个能成立

（2）包含(I):多个原因中至少有一个必须成立；

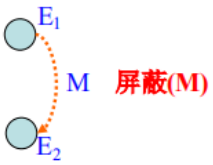


（3）唯一(O):多个原因中必须有一个且只有一个成立；
（4）要求(R):当C_1成立，C_2也必须成立；



输出约束：

（1）屏蔽(M):
当E_1是1时，E_2必须是0；
当E_1是0，E_2的值不定；



■因果分析法总结

1:适用于输入输出存在因果关系的测试情形
2:适用于输入输出参数取值为2值情形
3:因果分析法是对输入的组合测试
4:因果图保证各种组合不被遗漏
，5:因果图能消除违反约束的组合

决策表：

适用于：输入输出多且输入之间和输出之间互相制约条件较多

决策表：把作为条件的所有输入组合以及对应输出都罗列形成的表格

决策表组成

1. 条件桩：列出所有可能的条件（问题）
2. 条件项：列出所有条件的所有可能取值
3. 动作桩：列出可能采取的操作
4. 动作项：指出在条件项的各种取值情况下应采取的动作

决策规则：贯穿条件项和动作项的一列

构造步骤：依次填入四个组成----简化表

化简：

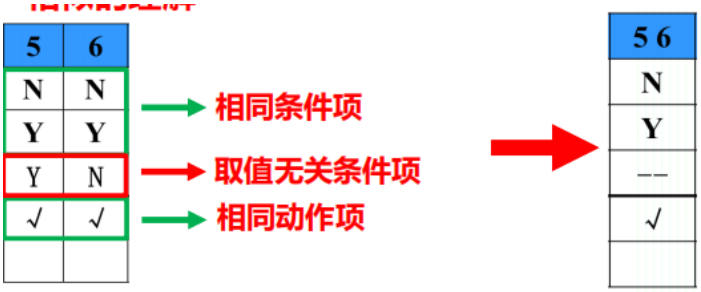
1. 目标：合并相似规则

2. 相似规则

a. 有两条或以上规则有相同动作

b. 且条件项之间粗在极大相似

3. “--”表示合并后该条件项与取值无关，称为“无关条件”



基于模型的测试

适用领域：有限状态、工作流状态、数据流状态、时间状态

原理：

1. 软件执行过程可分解为若干对象和连接对象间的关系
2. 测试序列可视为验证对象间所期望的关系是否满足

模型化软件：

1. 图论
2. 工作流
3. 语法模型
4. 其他：贝叶斯模型、Petri网

测试用例的生成：遍历图、覆盖图中所有边

正交数组测试：从大量测试用例中挑选适量的、有代表性的用例，从而合理地进行测试

适用：输入域相对较少而详尽测试次数又过大

特点：输入域有限，参数取值有限，需要将参数取值进行排列组合

1. 任意两个输入的所有组合均被包围
2. 所有可能取值出现次数相同，确保样本选择公正性

正交表构成：

1. 因子（输入参数）
2. 因子数：要测试的功能点数
3. 水平：因子的取值
4. 水平数：单个因子的取值个数
5. 行数：测试用例个数

记法：$L_{行数}(水平数^{因子数})$

$L8(2^7)$		列号							
			1	2	3	4	5	6	7
	行号	1	1	1	1	1	1	1	1
		2	1	1	1	0	0	0	0
		3	1	0	0	1	1	0	0
		4	1	0	0	0	0	1	1
		5	0	1	0	1	0	1	0
		6	0	1	0	0	1	0	1
		7	0	0	1	1	0	0	1
		8	0	0	1	0	1	1	0

性质：

1. 整齐可比性：每个因子的每个水平出现次数完全相同
2. 均衡分散性：任意两列的水平搭配是完全相同的，有很强的代表性

黑盒测试总结：

特点	**被测软件内部逻辑未知的测试方法总称**
目的	**功能的正确和遗漏；界面错误；数据访问；性能；初始化和终止**
技术	**基于需求的测试；正面测试和负面测试； 边界值分析；因果分析法； 决策表；等价类划分； 基于图/状态的测试；正交数组测试**
插桩	**实现白盒测试和黑盒测试的支持技术**

	白盒测试	**黑盒测试**	**灰盒测试**
代码依赖	**需要源代码**	**不需要源代码**	**需要了解实现的有限细节**
实施时机	**单元测试**	**单元测试后**	**集成测试前期**
实施者	**开发人员**	**测试人员**	**测试人员**
关注对象	**内部实现**	**功能需求**	**二者兼有**
测试方法	**代码审查和代码分析**	**设计并执行测试用例**	**编写测试代码**
是否必须	**不能被取代**	**不能被取代**	**可被黑盒测试取代**

黑盒测试工具：==功能测试工具

多用于系统测试、确认测试和回归测试

以自动化测试工具为主

第五章单元测试与集成测试

单元测试

软件单元：一个应用程序中最小可测部分

单元测试定义：对最小的软件设计单元（模块/源程序单元）的验证工作

实施者：软件开发人员

意义：

1. 消除软件单元本身的不确定性
2. 其他测试阶段的必要基础环节

测试关注点：（5个）

1. 模块或构件接口
    a. 目标：进出模块/构件的数据流正确
    b. 关注点：
        i. 接口名称、参数个数、类型、匹配顺序
        ii. 输出或返回值及其类型是否正确
2. 局部数据结构
    a. 目标：数据在模块执行过程中维持完整性和正确性
    b. 关注点
        i. 数据结构定义和使用过程的正确性
        ii. 局部数据结构对全局数据结构的影响
3. 边界条件
    a. 目标：保证模块在边界条件张能够正确执行
    b. 关注点：
        i. 数据结构中的边界

ii. 控制流中的边界（循环次数、判断条件

4. 独立路径：

a. 目标：模块中每条独立路径（基本路径）都被覆盖

b. 关注点：对路径的选择性测试

5. 处理错误的路径

a. 目标：保证错误处理的正确性，软件的健壮性

b. 关注点：

i. 异常条件的正确性

ii. 异常的可发生性

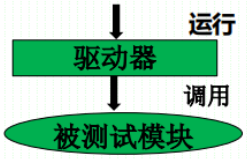
iii. 异常处理的逻辑正确性

在测试模块时，必须为每个被测模块开发一个驱动器和若干个桩程序

驱动器（Driver）：

对底层或子层模块进行测试时所编制的调用被测模块的程序，用以模拟被测模块的上级模块。

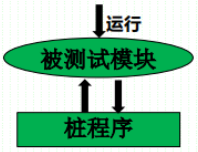
接收测试数据，并把数据传送给被测模块，最后输出相关结果。



桩程序（Stub）：

对上层模块进行测试时，所编制的替代下层模块的程序，用以模拟被测模块工作过程中所调用的模块.

模拟未被测试模块/隶属模块的功能



驱动器结构一般如下：

数据说明；
初始化；
输入测试数据；
调用被测模块；
输出测试结果；
停止

桩程序结构一般如下：

数据说明；
初始化；
输出提示信息（表示进入了哪个桩模块）
验证调用参数；
打印验证结果；
将模拟结果送回被测程序；
返回

集成测试

瞬时集成测试

通过单元测试的所有构件组合成一个最终系统，观察其是否正常运转

缺点：

1. 错误很多，修复可能，无休止的错误
2. 很难找出错误原因
3. 接口错误和其他错误容易混淆

适用：小型软件开发

增量集成测试

优点：错误容易分离和修正、接口容易进行彻底测试

缺点：会有额外开销，但能大大减少发现和修正错误的时间

1. 自顶向下集成：首先集成主模块，然后按照控制层次向下集成
    a. 方式：深度优先、广度优先

集成步骤：

*Step*1.主程序作为测试驱动器

*Step*2.根据集成顺序，程序桩逐渐被各模块替换

*Step*3.每个模块集成时都需要进行测试

*Step*4.每完成一次测试后，将新模块替换程序桩

*Step*5.利用回归测试来保证没有引进新的错误

优点：

1.尽早发现高层控制和决策错误
2.只需要一个驱动器
3.每步只增加一个模块
4.支持深度优先和广度优先

实施的主要难点：

高层测试需要在低层测试完成后才可进行的情形

解决方法：

1.推迟测试，直到低层模块完成测试；
2.设计程序桩，模拟低层模块；
3.自底向上测试

缺点：

1.对低层模块行为的验证比较晚
2.需要编写额外程序模拟未测试的模块
3.测试用例的输入和输出很难明确表示

2.自底向上集成：原子模块---->造件----->应用软件系统

集成步骤：

*Step*1. 低层模块组成实现特定子功能的构件
*Step*2. 编写驱动器程序协调输入输出
*Step*3. 测试特定子功能的构件
*Step*4. 沿层次向上对构件进行组合

优点：

1. 尽早确认低层行为
2. 无需编写程序桩
3. 对实现特定功能的树容易表示输入输出

缺点：

1.推迟确认高层行为
2.需编写驱动器
3.组合子树时，有许多元素要进行集成

混合式集成

特点：

开发小组对各自的低层模块向上集成；
专门的集成小组进行自顶向下集成；

步骤：

*Step*1:用程序桩独立测试上层模块；
*Step*2:用驱动器独立测试低层模块；
*Step*3:集成时对中间层进行测试；

端到端集成测试

特点：

从最终用户的角度出发，强调系统的功能测试

不关注接口

适用领域：

大型软件系统

测试插桩

定义：在程序特定部位附加一定操作或功能，用来检验程序运行结果以及执行特性，以便支持测试

技术：

1. 白盒测试插桩技术---探测什么信息、插桩位置、探测点数越少越好

    a. 目标代码插桩

优点：

- 目标代码的格式主要和操作系统相关，和具体的编程语言及版本无关，应用范围广；
- 特别适合需要对内存进行监控的软件中

缺点：

- 目标代码中语法、语义信息不完整

    b. 源代码插桩

优点：

- 词法分析和语法分析的基础上进行的，这就保证对源文件的插桩能够达到很高的准确度和针对性。

缺点：

- 工作量较大，而且随着编码语言和版本的不同需要做一定的修改

探针的植入常见位置：

a. 程序的第一条语句；
b. 分支语句的开始；
c. 循环语句的开始；
d. 下一个入口语句之前的语句；
e. 程序的结束语句；
f. 分支语句的结束；
g. 循环语句的结束；

常用插桩策略：

语句覆盖插桩（基本块插桩）： 在基本块的入口和出口处，分别植入相应的探针，以确定程序执行时该基本块是否被覆盖。

分支覆盖插桩： 分支由分支点确定。对于每个分支，在其开始处植入一个相应的探针，以确定程序执行时该分支是否被覆盖。

条件覆盖插桩： if, swich,while, do-while, for 几种语法结构都支持条件判定，在每个条件表达式的布尔表达式处植入探针，进行变量跟踪取值，以确定其被覆盖情况。

黑盒测试插桩

1. 测试预言：确认对于给定输入，系统是否有一个给定输出
    a. 调用API等信息，是一种灰盒测试
    b. 作用：检查函数正确性、系统安全性
2. 随机数据生成：在可能输入集中国选取一个任意子集进行测试
    a. 作用：避免只测试所知道的将奏效场景

有效测试用例生成方法：

（1）软件输入域进行等价划分；

（2）各子域边界进行边界值选取；

（3）各子域内进行随机测试选择输入样本；

第六章JUnit测试工具

第七章系统测试、确认测试与回归测试

系统测试=功能测试+非功能测试

定义：对完整集成后的产品和解决方案进行测试，用来评价系统对具体需求规格说明的功能和非功能的符号性测试

目的/作用：

1. 发现可能难以直接与模块或接口关联的缺陷
2. 发现产品设计、体系和代码的基础问题（产品级缺陷）

实施人员：独立测试团队

特点：是既测试产品功能也测试产品非功能的唯一测试阶段

功能系统测试

方法

1. 设计/体系结构测试

原理：

对照设计和体系结构开发和检查测试用例，从而整理出产品级测试用例

测试用例特征	建议
测试用例关注代码逻辑、数据结构和产品单元	单元测试
测试用例关注组件接口	集成测试
测试用例关注的不能为客户所看到	单元测试/集成测试
测试用例综合了客户使用和产品实现	系统测试

2. 业务垂直测试

    a. 原理：针对不同业务纵深的产品，根据业务定制测试用例，验证业务运作和使用

    b. 方法：模拟和复制、场景测试

3. 部署测试：验证新的系统是否能有效替代老大系统

    a. 目的：特定产品版本短期内是否能成功使用

    b. 过程：

        i. 采集实际系统真实数据，建立镜像测试环境，重新执行用户操作

        ii. 引入新产品，进行新业务操作

4. α测试和β测试

    a. β测试是在α测试达到一定程度后进行

    b. α测试不由开发人员或测试人员完成，是用户完成但在开发环境下的受控测试

    c. β测试完全不受控制

5. 符合性的认证、标准和测试

    a. 主流基础设施：操作系统、硬件、数据库

    b. 约定和法律要求：质量行业标准、法规

非功能系统测试

挑战：设置配置

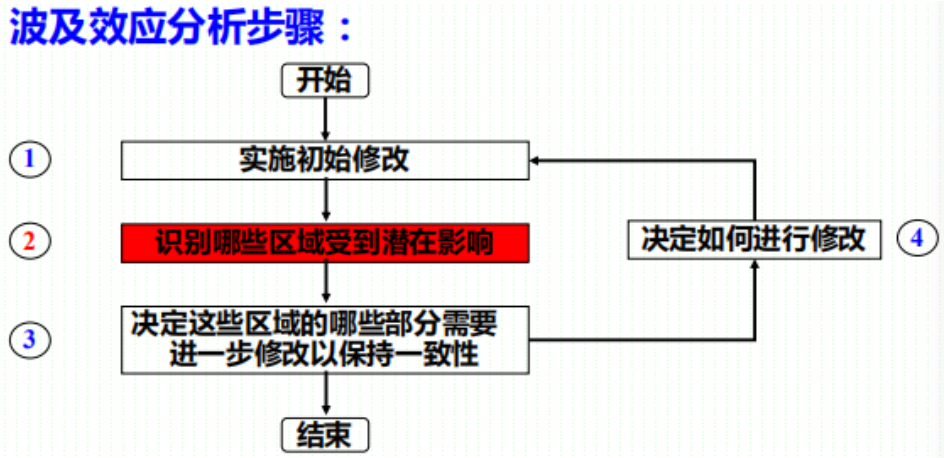
方法：

1. 性能测试
2. 互操作测试/兼容性测试
    a. 目的：保证两个或多个产品可以交换和使用信息，并恰当地在一起运行
    b. 目标：OS、数据库、网络、浏览器、APP
    c. 前向兼容/后向兼容
3. 可使用性和易获得性测试
    a. 可使用性：以用户视角确认产品的易用性、速度、美感的测试
        i. 质量因素：可理解性、一致性、导航、响应
    b. 易获得性：检查产品对于行动不便的用户也可使用的测试
        i. 基本：粘贴键、声响键、多种模拟鼠标功能、朗读器
4. 国际化测试
    a. 目的：确保软件支持不同国家语言的测试手段
    b.

回归测试

波及效应：为了发现所受影响部分和发现潜在的受影响部分，以保证软件发生改变后仍然保持一致性与完整性

1. 需求的波及效应
2. 设计的
3. 代码的
4. 测试用例的



第八章性能测试

性能测试广义定义：

1. 压力测试：通过对程序施加越来越大的负载，直到发现程序性能下降的拐点-----强调压力大小变化
2. 负载测试：不断增加压力或增加一定压力下持续一段时间，直到系统性能指标达到极限--------压力增加和持续时间
3. 强度测试：迫使系统在异常资源下运行，检查系统对异常情况的抵抗力-------强调极端的运行条件
4. 并发测试：多用户同时访问或操作数据是否存在死锁或其他性能问题----强调对多用户操作的承受能力
5. 大数据量测试：在多业务中结合并发操作测试系统的数据处理极限能力---强调数据处理能力
6. 配置测试：通过测试找到系统各项资源的最优分配原则-----强调资源优化配置
7. 可靠性测试：在系统加载一定压力情况下，运行系统一段时间，测试系统是否稳定-----强调系统稳定性

的总称

指标：

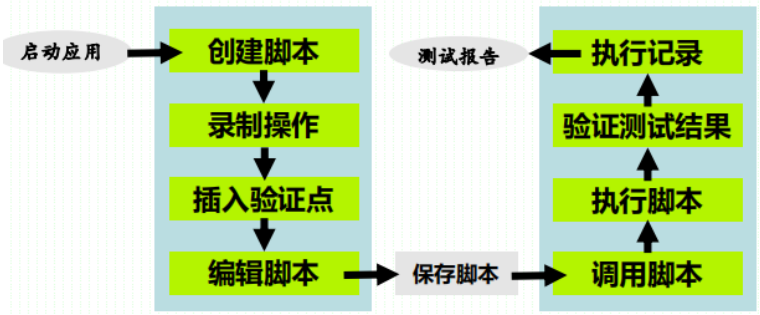
1. 并发用户数量
2. 响应时间
3. 吞吐量
4. 每秒处理事务数（TPS）
5. 点击率
6. 资源利用率

自动化测试

优点：

1.测试速度快
2.测试结果准确
3.高复用性
4.永不疲劳
5.可靠：
计算机不会弄虚作假
6.能力：
有些手工测试做不到的地方，自动化测试可以做到。例如，
在性能测试中模拟1000个用户同时访问网站

原理：


启动应用
创建脚本
录制操作
插入验证点
编辑脚本
保存脚本
调用脚本
执行脚本
验证测试结果
执行记录
测试报告