1. 测试概述

危机原因：缺乏规范化工程约束，缺陷积累放大

软件缺陷原因：需求说明书，设计，编码，其它

意义：保证软件质量的重要手段，深入每个开发阶段确保产品符合用户需求

定义：（1950S测试被视为发现缺陷的活动，与调试区分开）

Bill Hetzelt: 测试是建立一种信心，认为程序能够按预期执行

Myers: 测试是为发现错误而执行的过程

Myers: 测试是为了证明程序有错，好的测试用例与成功的测试均为了发现未知错误（核心：发现尽可能多的程序错误）

测试目的：确保软件质量，确保开发过程方向正确性

**十大原理**：用户需求至上、测试是有计划的活动（贯穿整个开发）、缺陷有**集群性**、测试应从小走向大规模、不可能穷尽测试、第三方独立进行测试、测试无法找到所有缺陷、缺陷对已进行测试有免疫力、测试是有风险的活动（最优测试量）、并非所有缺陷都需修复（bug, feature）

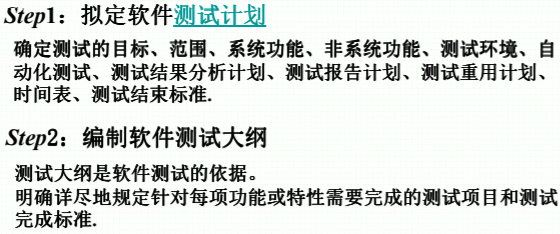
测试过程：测试计划、大纲、用例、实施测试、分析结果

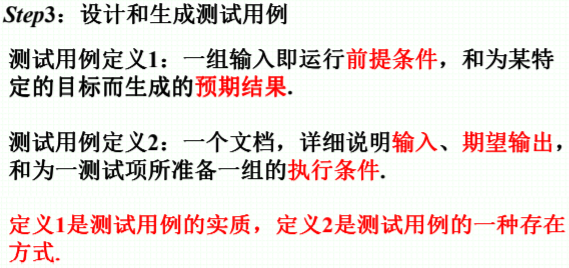
测试用例：输入、执行条件、期望输出（前提条件、预期结果）

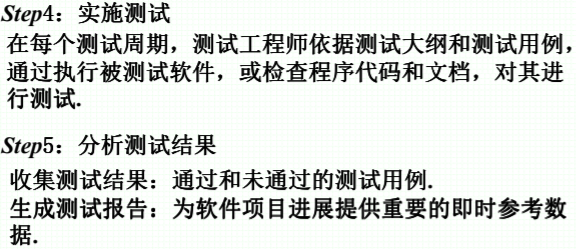
测试类型：

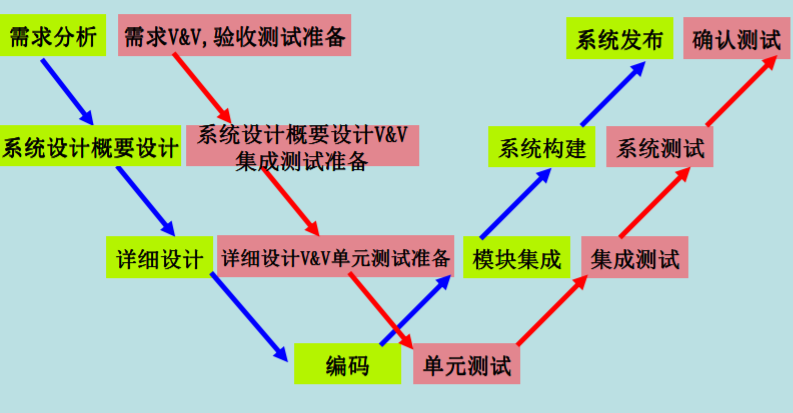
|  |
| --- |
| 测试技术：白盒，黑盒，灰盒 |
| 开发阶段：单元，集成，系统，确认，回归，性能，验收 |
| 执行状态：静态，动态 |
| 执行主体：开发，用户，第三方 |
| 特殊：国际化，兼容性，安全性 |

W模型：将静态分析与测试准备提前，形成测试驱动开发









测试工作量：45%

1. 白盒测试

定义：基于代码测试，根据程序逻辑行为，生成用例

静态白盒：不执行软件，审查软件设计、结构、代码（结构化分析）

桌面检查（代码编写者）：优点（编码者易理解自己的代码、开销小无进度限制、尽早发现缺陷），缺点（开发人员不是测试最佳人选、有效性难以保证），特点（团体小、非正式）

代码走查（团队会议）：特点（由团队通过会议完成、与会者充当计算机执行测试用例、开发人员及时回答问题）

代码审查（ALL）：优点（发现缺陷、提高质量、尽早定位缺陷集群位置），缺点（费时、涉及人员多、参与者不一定全部理解程序），步骤：计划-概述-准备-审查会议-审查报告-返工-跟进

动态白盒：提供源代码与可执行程序，根据代码设计执行测试用例

|  |  |  |
| --- | --- | --- |
| 语句覆盖 | 执行每条语句 | 100%语句覆盖困难（处理错误，小概率，不可达），无法发现严重问题NULLPTR |
| 判定覆盖 | 每个判定TF均包含 | 优点（简单，包含语句覆盖）  缺点（忽略了条件表达式内部） |
| 条件覆盖 | 判定内每个条件TF均包含 | 不保证执行所有**分支**  可能比语句或判定覆盖弱 |
| 判定条件覆盖 | 判定与其内的条件TF均包含 | 错误屏蔽：原子条件取值对整体不可见  无法检查运算符错误 |
| 条件组合覆盖 | 判定内条件组合均出现 | 代价昂贵，为原子条件数目  **短路符**用于寻找最少测试用例集  某些条件组合内部互斥 |
| 路径覆盖 | 所有可能路径与环路 | 优点（相对彻底测试）  缺点（路径分支指数级增加，不可达路径，未测试分支中条件） |

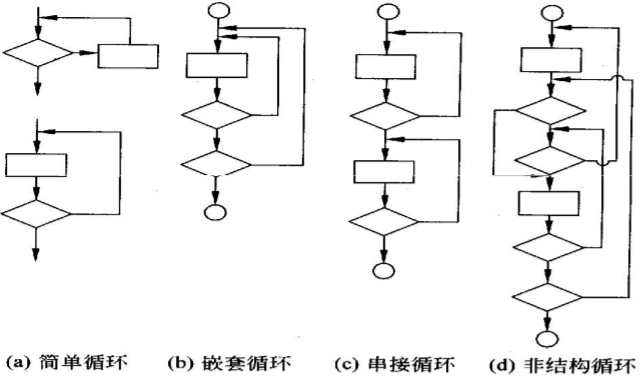
流图：抽象程序流程，通过对图问题的求解得到基本路径进行测试

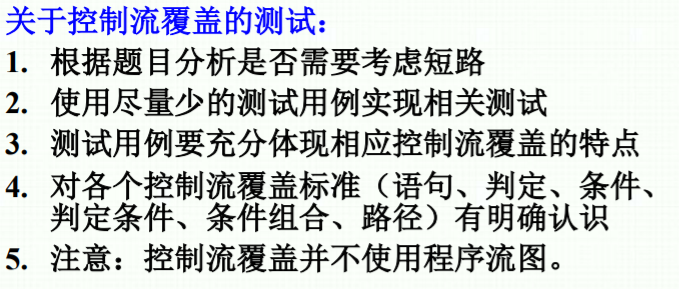
基本路径：**贯穿程序的**，至少引入一组**新语句**、**新判断**的程序通道

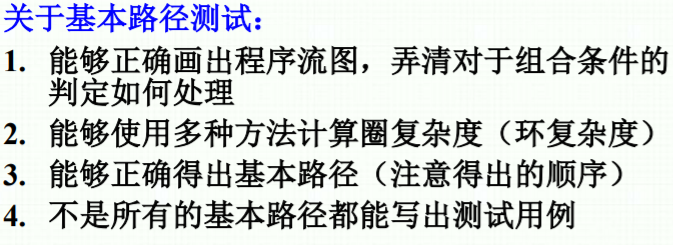
数学定义：其它路径的向量，可以通过基本路径的向量**线性运算组合**出来（**基本路径不唯一**）

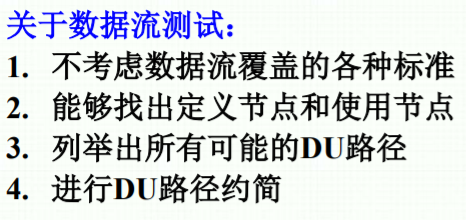
环复杂度：度量基本路径数，是语句覆盖所需测试用例上线（域数，E边-N节点+2，判定的条件数+1）

循环测试：









9 　 char \*eptr = encoded; 　 　 　 　 　 　  
指针变量\*eptr和地址变量eptr在这里被定义，这点上没有问题。下面我们分析是否有USE节点  
语句9等同于：char \*eptr;  
              eptr = encoded;  
从而我们可以看到，eptr被定义了两次，隐含\*eptr的定义。eptr被使用的情况要看编译器的具体处理。  
  
12 　 while (\*eptr)  　 　 　 　 　 　 　  
这里显然有一个指针变量\*eptr的谓词使用，同时，由于指针运算符作用于eptr上，因此也是eptr的一个运算使用。  
  
22 　 　 　 int digit\_high = getHexValue(\*(++eptr)); 　 　  
++eptr等同于eptr=eptr+1，因此对eptr来用既有定义也有使用  
在eptr发生变化的同学，\*eptr的内存内容也变了，因此\*eptr被重新定义了  
\*eptr还作为一个参数被传给函数getHexValue，因此\*eptr还被使用  
  
33 　 　 ++dptr; 　 　 　 　 　 　 　  
这里同上，dptr既被定义又被使用。  
但需要注意的是，由于dptr指向的地址值变化了，因此\*dptr也发生了变化，也就是说，这条语句隐含了\*dptr被定义的事实。

1. 黑盒测试

定义：基于**规格说明**，不要求考察代码，以用户视角测试（功能测试）

意义：检查明确/隐含需求、包含有效/无效输入、包含用户视角

目的：有时无法获取源码、尽早发现软件功能缺陷、弥补逻辑缺陷、适用于各个测试阶段

实施者：专门的测试部门、测试人员

步骤：分析规格说明书、生成测试用例、执行测试、判定结果

进入退出条件：**进入**(编码开始)，**退出**(完成测试，修正错误，达到预算/开发时间)

基于需求测试RTM：目的（确定需求规格说明书列出的需求）、前提（需求规格已评审，隐含需求明确化）、作用（跟踪每个需求的测试状态，先执行优先级高的测试用例，导出特定需求测试用例清单，评估测试进度与工作量）

正面测试：测试用例通过一组预期输出验证产品需求、目的（证明软件对每条规格说明和期望都能通过，可给出预期错误）

负面测试：展示非预期输入时产品未失败、目的（尝试使用产品未预想到的场景使系统崩溃），**不能映射到需求**



**等价类划分**：原理（将输入数据域分为数据类便于导出测试用例，复用减少测试用例数量）

有效等价类：满足程序输入、一条用例可覆盖多条等价类

无效等价类：不满足程序输入要求、只违反一条规则、测试用例仅覆盖一条

划分步骤：选择划分准则、确定有效，无效等价类、选取样本数据、写出预期结果、加入**特殊值**（空字符串等）、执行测试

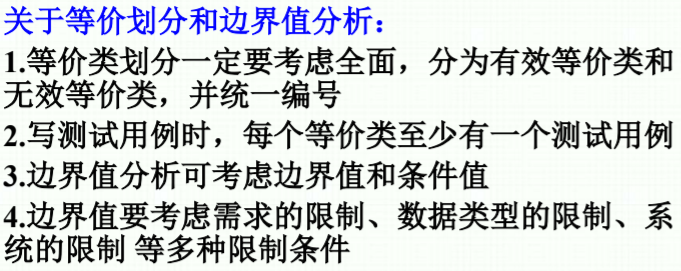


**边界值分析**：能有效捕捉出现在边界处缺陷的一种测试方法，利用并扩展了缺陷容易出现在边界处的概念

测试边界：测试接近边界的有效数据，最后个可能有效数据，刚超出边界数据（+1为全部参数取典型值一次）

|  |  |  |
| --- | --- | --- |
| 4n+1 | 基本边界测试 | 参数min、min+1、max-1、max |
| 6n+1 | 健壮边界测试 | 参数min、min+-1、max、max+-1 |
| 3m | 边界条件测试 | 条件自身、自身-1、自身+1 |

等价类划分往往先确定边界值，边界值分析是等价类划分的补充



**因果分析**：原理（软件输入输出存在逻辑关系，因果图从规格说明书中获得）、【分析规格说明书、因果图中连接、标明约束条件、生成决策表、生成测试用例】

因果关系：恒等（--），非（~），或（V），与（）

输入约束：互斥(E)，包含(I)，唯一(O)，要求(R)

输出约束：屏蔽(M)，E1为1，E2为0

**决策表**：输入输出较多且相互制约，将所有**输入组合**及**对应输出**罗列出来形成表格

条件桩：可能问题 条件项：问题条件取值

动作桩：可能动作 动作项：相应条件下采取的动作

决策规则：决策表中一列(最大总数)

约简：具有相同动作，条件项极大相似，则可合并(无关条件)

基于模型：将软件执行分解为对象与其间关系，验证关系是否满足期望（有限状态、工作流、数据流、时间 形成自动机）

**正交数组**：通过正交表从大量用例中挑选适量有代表性的进行测试

任意两**输入所有组合**均被包含，多因素水平，**均匀分散**、**整齐可比**

因子：考察的变量 水平：因子的取值 行数：测试用例个数

表格选择：水平数接近，因果数超过或等于，行数小

填表：因果超出则划掉列，水平超出相应水平设为典型值(--)



蜕变测试**（Metamorphic testing）：**对测试期望输出难以预测的程序，基于已知的期望输出以及相关领域知识和实现建立的用例之间、输出之间的关系来通过已知用例生成新的用例。通过多次生成的用例集来对程序进行测试就是蜕变测试（蜕变关系：等式、方程、关系）

R1: sin(x)=sin(x+2π) cos(x)=cos(x+2π)

R2: sin(x)=-sin(x+π) cos(x)=-cos(x+π)

R3: -sin(-x)=sin(x) cos(-x)=cos(x)

R4: sin(x)=sin(π-x) cos(x)=-cos(π-x)

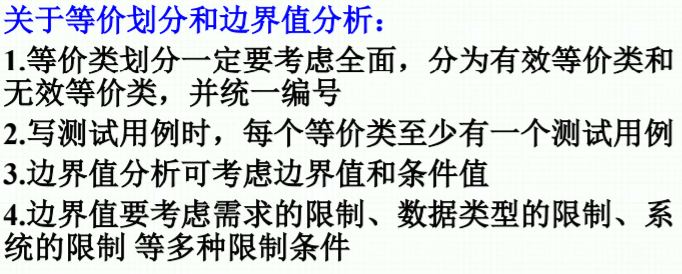
R5: sin(x)=-sin(2π-x) cos(x)=cos(2π-x)

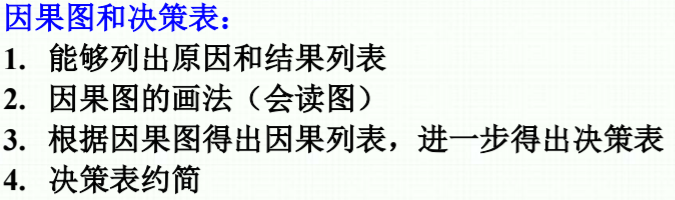
随机测试：不按照用例和说明书，根据经验对功能、性能进行抽查（缺陷分配不均、难以取到特定值）



测试工具功能：录制回放、检验（检验点）、可编程

原理：运行被测软件，捕获操作形成脚本（可修改回放）





1. 单元测试与集成测试

软件单元：应用程序最小可测部分

单元/模块测试：对最小的软件单元/模块验证（软件开发者）

目标与关注点：模块功能、内部逻辑、数据结构、边界、性能、安全

|  |  |  |
| --- | --- | --- |
| 事项 | 目标 | 关注点 |
| 模块/接口 | 数据流正确性 | 接口名称，参数，类型，顺序  输出返回值及类型正确性 |
| 局部数据 | 数据完整、正确性 | 定义使用正确性  局部对全局影响 |
| 边界条件 | 保证边界条件下正确执行 | 数据结构边界（数组）  控制流边界（循环，判断） |
| 基本（独立）路径 | 保证所有语句均被执行 | 路径选择性测试  基本路径+循环测试 |

驱动器：对子层模块测试时编写的调用程序，模拟上层模块

程序桩：对上层模块测试时，模拟被调用的子层模块

集成测试：将软件模块结合作为整体接受测试（验证单元间可协调工作、功能、性能、可靠性）**测试+开发人员**

接口：内部（内部模块间）、外部（第三方可见）、提供方法（API,SKD）、桩程序（模拟实际组件）、显式（文档化）、隐含（仅开发者知道）

瞬时集成：构件通过**单元测试**就组合成最终系统，缺点（错误多，原因难发现，接口与内部错误易混淆）

增量集成：将程序分为小部分构造和测试，优点（错误易**分离**，接口测试彻底），缺点（额外开销大）

自顶向下：主程序作为驱动器、新模块替换程序桩、测试、回归测试

自底向上：实现底层子构件、编写测试程序协调IO、测试子功能构件、向上依次组合

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 步骤 | 优点 | 缺点 |
| 自顶向下 | 主程序作为驱动器、新模块替换程序桩、测试、回归测试 | 尽早发现**高层**控制决策错误、一个驱动器、每步一个模块 | 底层验证晚、桩程序多且难、依赖关系复杂 |
| 自底向上 | 实现底层子功能构件、编写测试程序协调IO、测试、向上依次组合 | 尽早确认**底层**行为、无程序桩、树表示输入输出 | 高层认知晚、多驱动器、集成时元素多 |

白盒插桩：位置（需求，紧凑精干），策略（语句、分支、条件覆盖），方法（assert），作用（生成状态检验可达性、读取内部数据、检测不变数据、监测前提条件、人为触发时间，监测事件时间）

黑盒插桩：独立于被测系统、确认输入是否有对应给定输出；特点（无源码，有API，灰盒测试），作用（检查函数正确性，系统安全性）

测试插桩：在程序特定部位附加操作，检验运行结果及执行特性

1. Junit(无)
2. 系统测试，确认测试，回归测试

**系统测试**：定义（对完整集成后产品和解决方案测试，评价系统对需求规格说明书的**功能**和**非功能**符合性）、目的（发现难以直接与模块或接口关联的缺陷，设计、体系和代码的基础**产品级**问题）独立团队

唯一既测试功能也测试非功能的阶段





**功能测试**：产品级测试，覆盖遗漏测试，从新视角发现缺陷

设计/体系结构：对照设计/体系结构开发阶段整理**产品级测试用例**，关注整体产品行为，静态（体系结构分析，特征模型提取），动态

业务垂直：原理（针对不同**纵深产品业务**制定用例，验证业务运作使用）、应用范围（通用系统在不同领域使用）、方法（模拟，复制：获取处理过程，改变一般工作流定制）

部署测试：验证系统是否满足客户部署需求、离场（开发组织内）、现场（客户场地与环境）

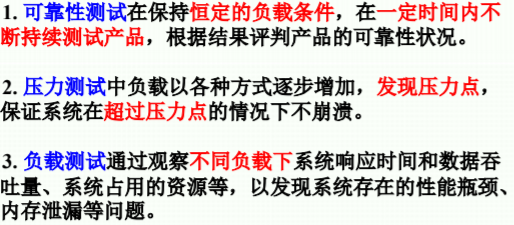
Alpha：用户在开发环境下受控测试，特点（由用户在开发者指导下完成 发布会），达到一定程度进行Beta测试

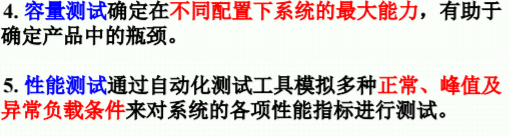
Beta：用户实际使用测试，特点（开发者不在场），挑战（客户数量，客户了解产品）

符合性：产品通过主流环境等，并符合相关法规

**非功能测试**：验证系统**质量因素**，挑战（设置配置：模拟、真实环境）

可伸缩性：确认产品**参数最大值**、主要瓶颈



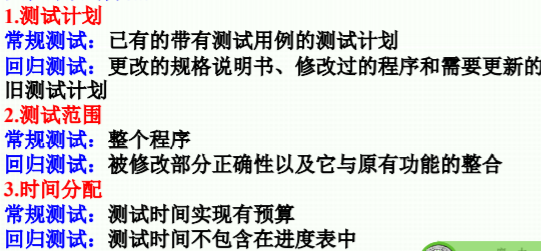


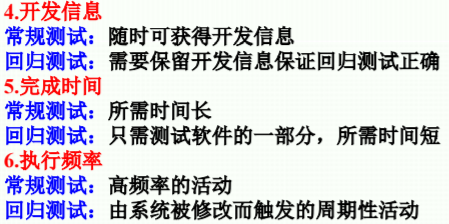
互操作/兼容性：产品、版本间交互

可用性易获得：用户视角，易用性、速度、美感（可理解，一致，导航，响应）

**确认测试**：检查是否满足需求阶段的**确认准则**，能否真实使用、**实施者（客户代表）**、目的（验证产品可接受）

**回归测试**：验证已变更系统完整与正确性、定义（对已测试过修改后程序重新测试保证无新错误，保证修改不影响已有功能）、过程（提出需求，修改，选择测试用例，执行测试，识别失败，识别错误，排除错误）





组测试：找出一个模块单独工作正常，多模块集成时错误

波及效应：发现所有和潜在受影响部分，保证发生改变后一致完整性、技术（字符串匹配，程序切片）、种类（需求，设计，代码，测试）

