软件测试

第一章 软件测试引论（2022）

1.1 软件缺陷的表现?

- **软件没有实现产品说明书中描述的功能**

> 计算器应能准确无误地进行加、减、乘、除运算。如果按下加法键，没反应，或者计算结果出错，都属于缺陷。

- **软件实现了产品说明书中描述的不应有的功能**

> 如一个学生成绩管理系统，规格说明指明老师才具有修改学生成绩的权利，学生不能修改。如果学生用户账号也可以修改学生成绩，这是个缺陷。

- **软件功能超出了产品说明书中指明的范围**

> 如学生成绩管理系统规定每个学生只能看到自己的成绩,如果能查询到其它学生的成绩,就是一个缺陷。

- **软件未达到产品说明书中虽未指出但应当达到的目标**

> 如一个学生成绩管理系统，规定学生成绩不超过100分，那么如果输入120分或者学生成绩部分输入“，。！……”这些字符也能提交， 就是缺陷。

- **软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良**

> 如一个学生成绩管理系统，作为用户的学生、老师觉得界面布局、颜色搭配比较丑陋，可以认为是缺陷。

1.2 为什么这些软件开发者不在发布前把软件缺陷都解决呢?

- **软件缺陷不能避免，只能尽量减少**

> 一个编程大师说:“任何一个程序，无论它多么小，总存在着错误。”初学者不相信大师的话，他问:“如果一个程序小得只能执行一个简单的功能，那会怎样？”“这样的程序没有意义，”大师说，“但如果这样的程序存在的话，操作系统最后将失效，产生一个错误。”初学者不满足，他问:“如果操作系统不失效，那么会怎么样？”“没有不失效的操作系统，”大师说，“但如果这样的操作系统存在的话，硬件最后将失效，产生一个错误。”初学者仍不满足，再问:“如果硬件不失效，那么会怎样？”大师长叹一声道:“没有不失效的硬件。但如果这样的硬件存在的话，用户就会想让那个程序做一件不同的事，这件事也是一个错误。”没有错误的程序世间难求。【James 1999”】

1.3 软件缺陷出现在哪个阶段

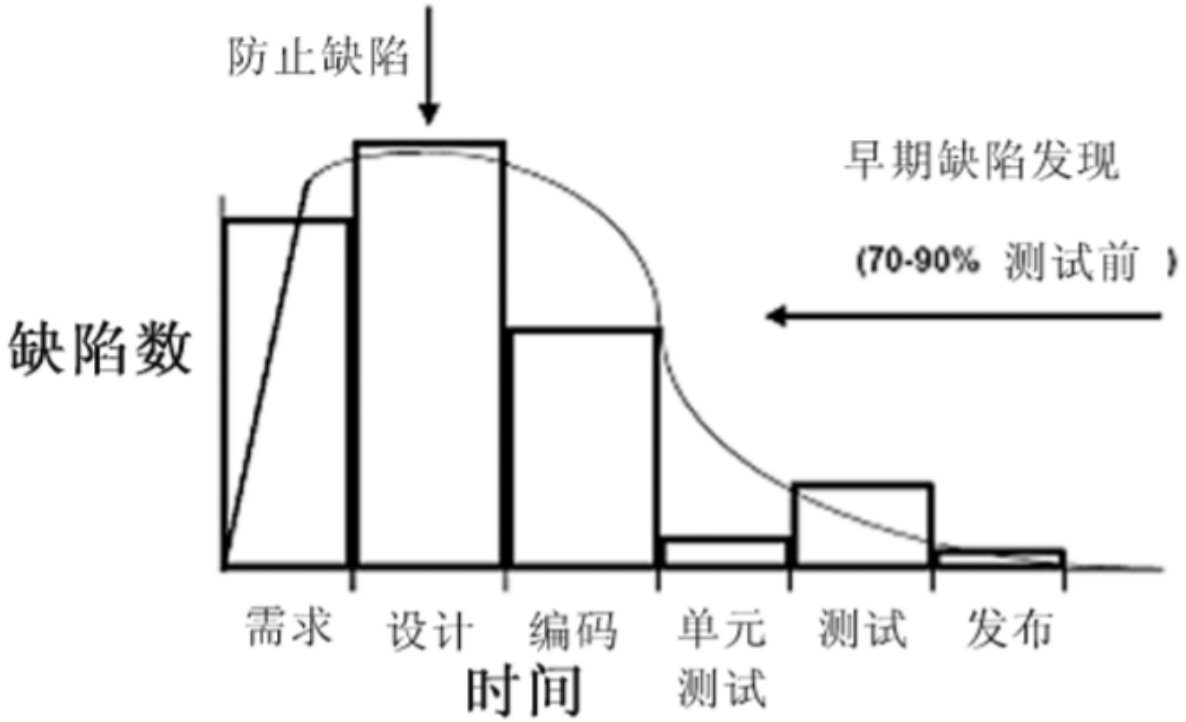
> 在软件开发的每一阶段都可能会引入缺陷

- **需求阶段**
- **设计阶段**
- **开发阶段**

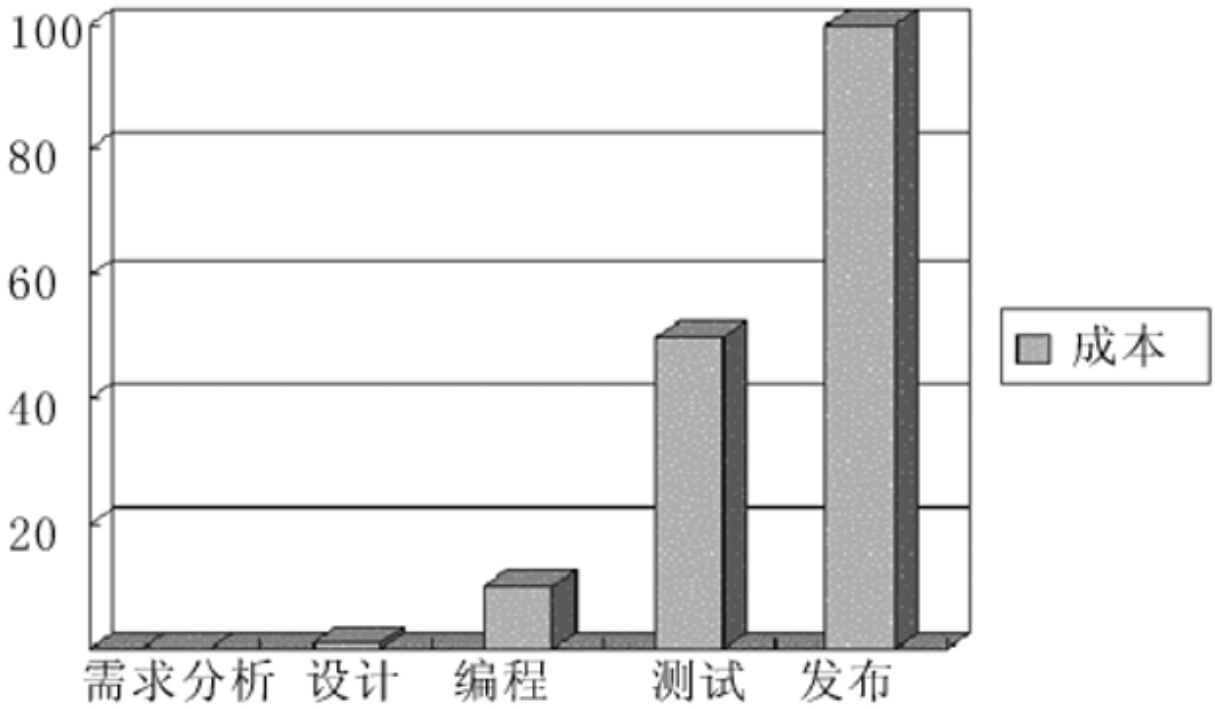
1.4 什么时候开始测试?

- **要尽早的开始测试。所以一般从需求阶段就要开始测试**

1.5 为什么要尽早的开始测试?



软件缺陷在不同阶段的分布



修复软件缺陷成本

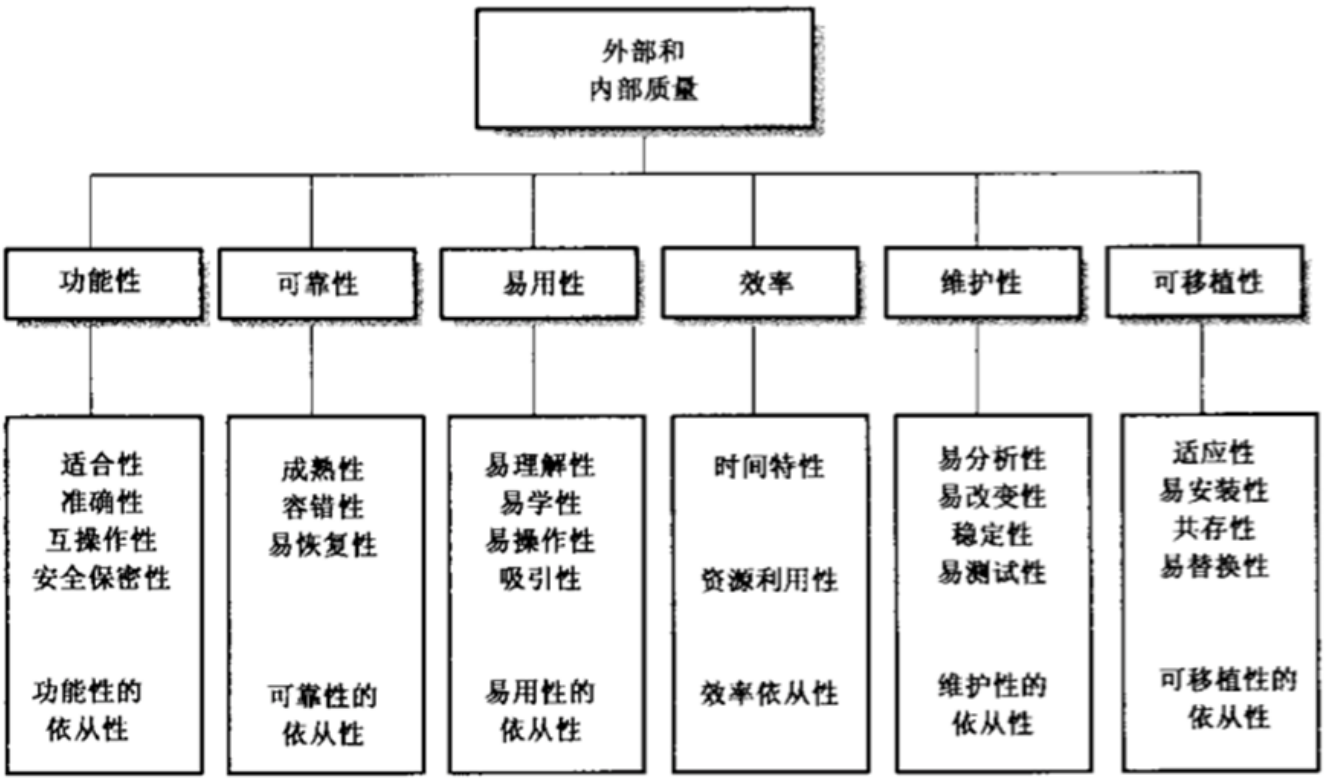
1.6 软件测试的目的

- 测试的目的就是尽可能早的发现缺陷。

1.7 什么是软件测试?

- 软件测试是为了尽快尽早地发现在软件产品中所存在的各种软件缺陷而展开的贯穿整个软件开发生命周期，对软件产品（包括阶段性产品）进行验证和确认的活动过程。

1.8 软件质量模型



1.9 测试能保证软件质量吗?

- 不能，软件测试是软件质量保证的重要手段

1.10 软件测试的价值

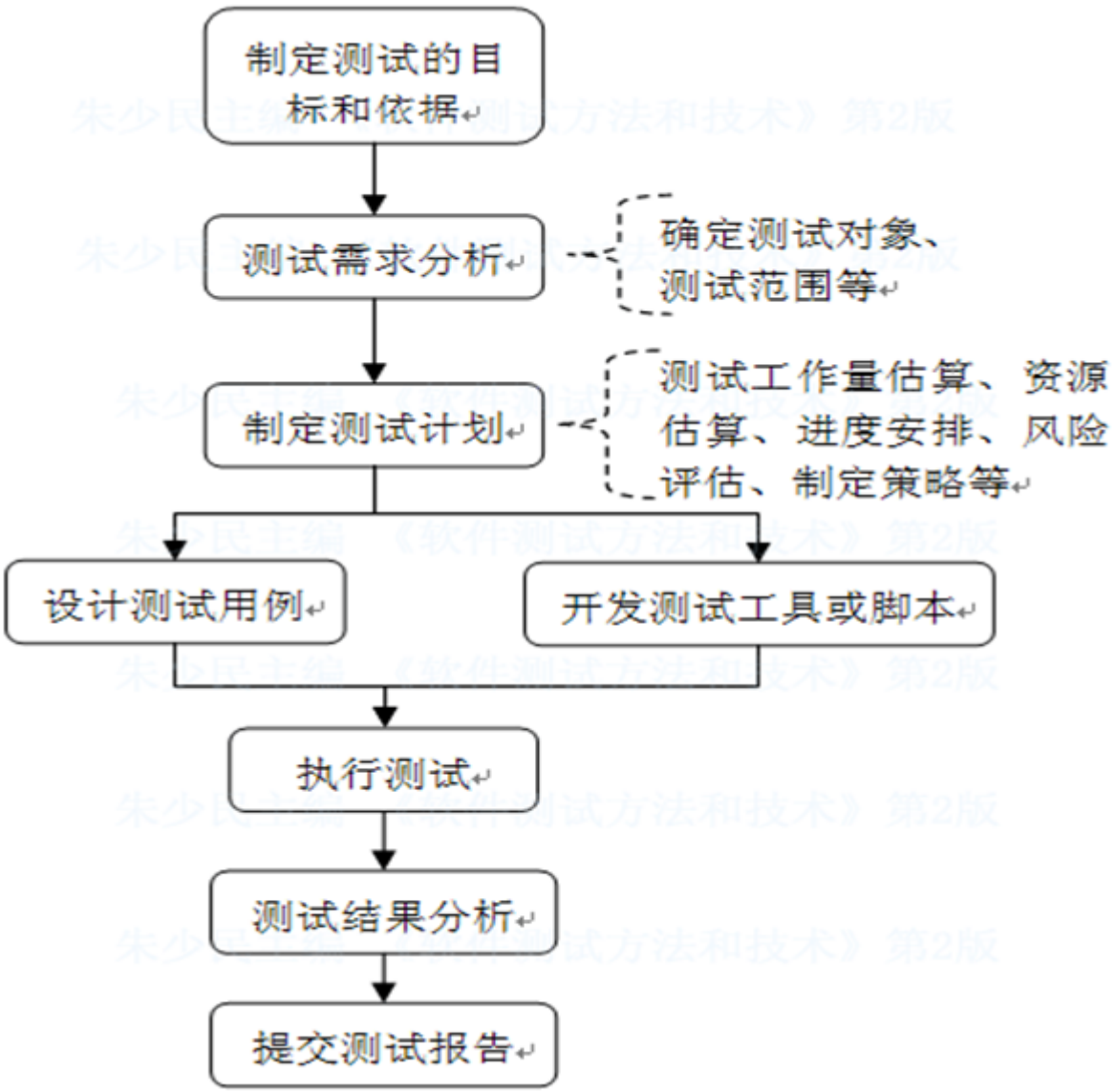
- 发现问题，督促问题解决，提高产品质量
- 持续提供质量反馈、及时揭示质量风险，有助于控制项目风险，提高构建的质量
- 全面评估产品质量，提供有关产品质量的全面、客观的信息
- 通过缺陷分析，获得缺陷模式，有助于缺陷预防

第2次课测试用例

2.1 如何进行软件测试?

- 通过缺陷分析，获得缺陷模式，有助于缺陷预防*
- 不同的软件产品
- 不同的阶段
- 测试方法和测试技术不一样

2.2 测试工作流程



2.3 需求评审解决的问题

- 不正确的需求认识

- 丢掉的需求点
- 模糊的描述
- 多余的或没意义的需求
- 不一致的理解

2.4 需求评审的标准

- 正确性
- 完备性
- 易理解性
- 一致性
- 易测试性
- 易追溯性

2.5 测试人员在需求评审中作用

- 检查需求定义是否合理、清楚，是否具有可测试性
- 熟悉评审内容，为评审做好准备
- 针对问题阐述观点，而非针对个人
- 从客户角度想问题，多问几个为什么
- 在会前或会后提出自己建设性的意见
- 对发现的问题跟踪到底
- 针对需求文档等报告问题

2.6 静态测试方法之一：代码评审

> **静态测试：** 不运行被测试程序，对代码通过检查、阅读进行分析。

- 走查 （Walk Through）
- 审查 （Inspection）
- 评审 （Review）

2.7 实施标准和规范的原因

- 可靠性
- 可读性和可维护性
- 可移植性

2.8 静态代码检查工具

- CheckStyle
- PMD
- FindBug
- SourceMonitor
- JTest

2.9 动态测试的步骤

- **步骤1：设计测试用例**
- **步骤2：搭建测试环境**
- **步骤3：编写测试程序（脚本）**
- **步骤4：执行测试**
- **步骤5：分析测试结果**

2.10 测试用例案例

> 编写论坛登录的测试用例，要求用户名至少3位,密码至少6位,且至少有1个字符。

用例编号	测试步骤	输入数据	预期结果

第3次课 白盒测试

> - 通过分析被测单元的内部程序结构来设计测试用例
> - 白盒测试又叫结构测试，逻辑驱动测试或基于程序本身的测试。

	前置条件	输入数据	预期输出
用例1			
用例2			

- **语句覆盖**

> 设计若干条测试用例，使程序中每条可执行语句至少执行一次。

- **判定覆盖（分支覆盖）**

> 设计测试用例，使程序中的每个逻辑判断的取真和取假的分支至少经历一次。

- **条件覆盖**

> 设计若干测试用例，使程序的每个判定中的每个条件的可能取值至少满足一次。

- **判定–条件覆盖**

> - 使判定中每个条件的可能取值至少满足一次，并且使每个判定分支至少执行一次。
> - 判定–条件覆盖能同时满足判定、条件两种覆盖标准。

- **条件组合覆盖**

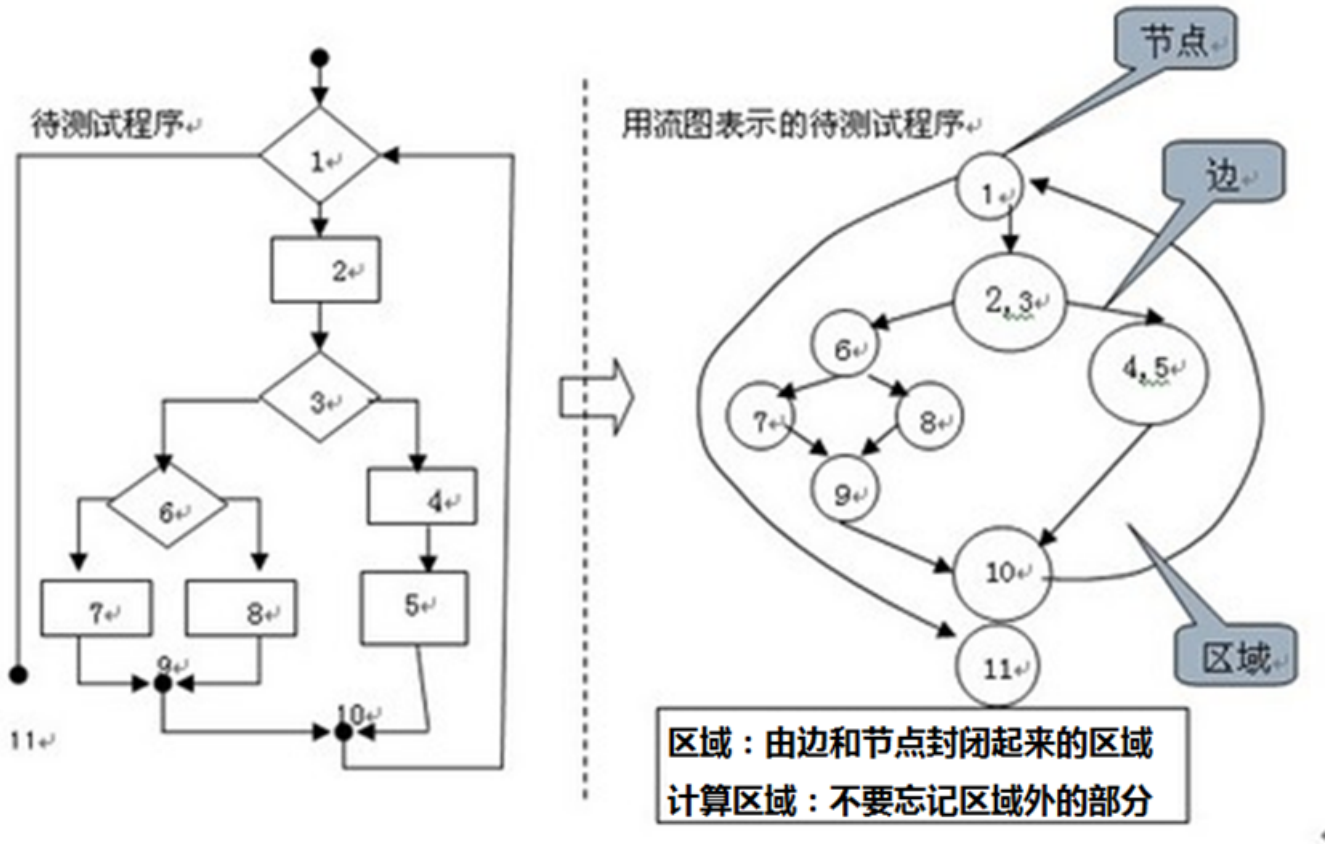
> 设计用例，使得每个判断表达式中条件的各种可能组合都至少出现一次;

- **路径覆盖**

> - 设计足够多的测试用例，覆盖程序中的每条可能路径。
> - 上述满足条件组合覆盖的测试用例不能覆盖路径。

3.1 控制流图

```
Procedure: process records
    Do While records remain
        Read record;
        If record field 1 = 0 Then
            store in buffer;
            increment counter;
        Else If record field 2 = 0 Then
                reset counter;
            Else store in file;
            End If
        End If
    End Do
End
```



- 流程图的圈复杂度

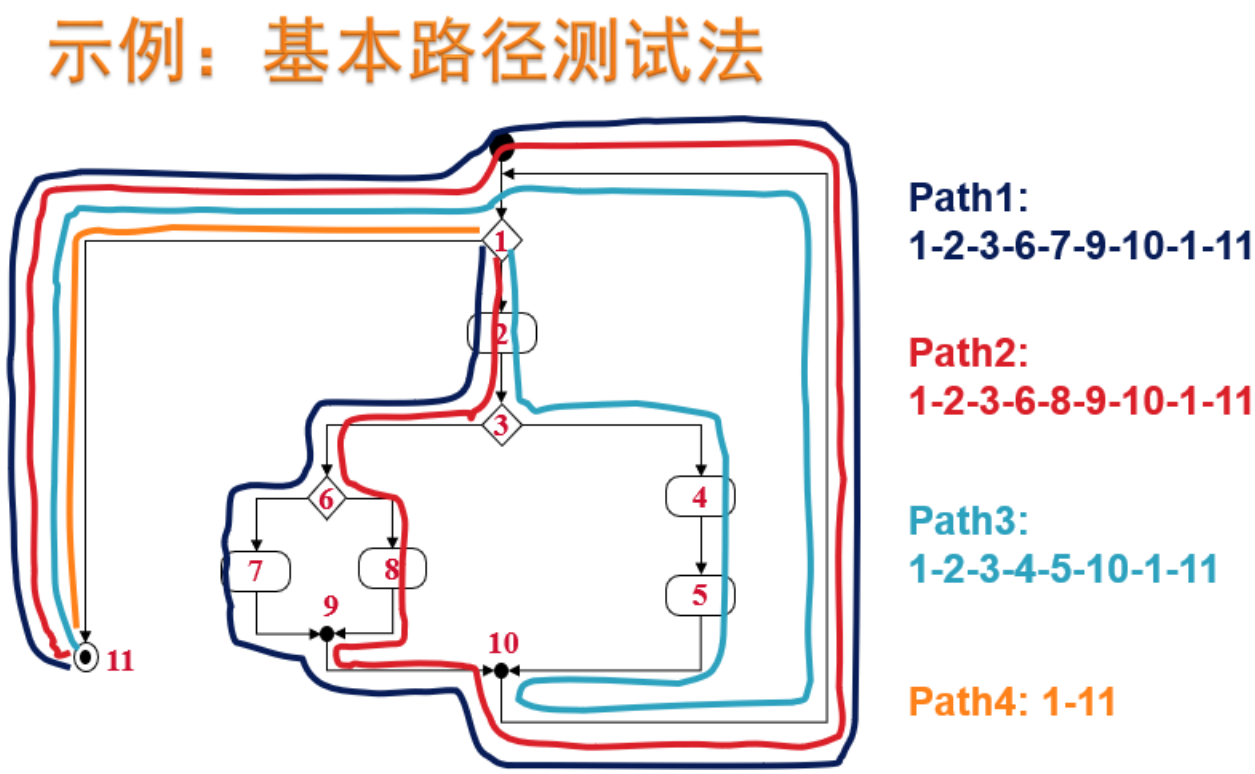
> 复杂度越高，出错的概率越大

> V(G) = 区域数量(由节点、连线包围的区域，包括图形外部区域)
>
> V(G) = 连线数量 – 节点数量 + 2
>
> V(G) = 分支节点+ 1

3.2 基本路径测试

> 1. 依据代码绘制流程图/控制流图
> 2. 确定流程图的圈复杂度
> 3. 确定线性独立路径的基本集合
> 4. 设计测试用例覆盖每条基本路径

示例：基本路径测试法



基本路径测试方法——例

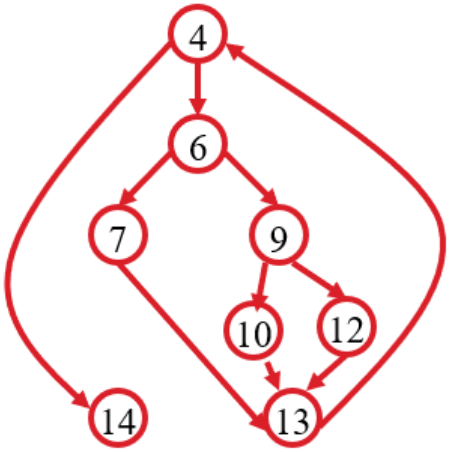
```
voidSort ( intiRecordNum,int iType ){
    int x = 0;
    int y = 0;
    while ( iRecordNum-- > 0 ){
        If ( iType == 0 )
            x = y + 2;
        else
            If (iType == 1 )
            x = y + 10;
        else
            x = y + 20;
    }
}
```

一、画出控制流图：

二、计算环形复杂度：

10（条边）- 8（个节点）+ 2 = 4

这是确定程序中每个可执行语句至少执行一次所必须的测试用例数目的上界。



三、导出独立路径（用语句编号表示）

路径1：4→14

路径2：4→6→7→13 →4→14

路径3：4→6→9→10→13→4→14

路径4：4→6→9→12→13→4→14

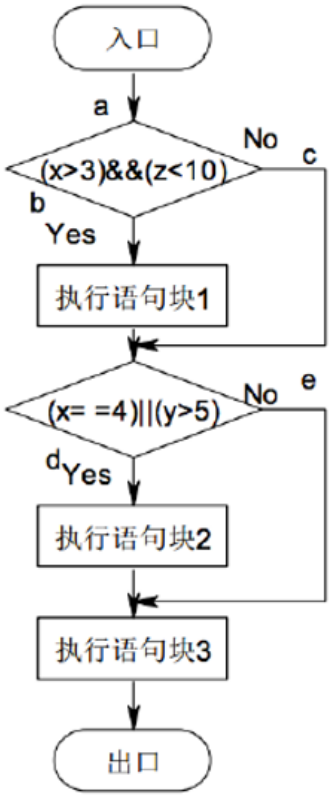
四、设计测试用例：

	输入数据	预期输出
测试用例1	iRecordNum = 0 iType = 0	x = 0 y = 0
测试用例2	iRecordNum = 2 iType = 0	x = 2 y = 0
测试用例3	iRecordNum = 2 iType = 1	x = 10 y = 0
测试用例4	iRecordNum = 2 iType = 2	x = 20 y = 0

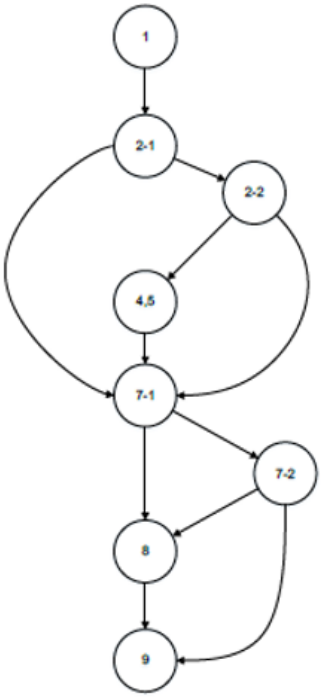
3.3 课堂练习

```
void DoWork (int x,int y,int z)
{
    int k = 0, j = 0;
    if (( x > 3 ) && ( z < 10 ))
    {
        k = x * y - 1;
        j = sqrt(k);
    }
    if (( x == 4 ) || ( y > 5 ))
    j = x * y + 10;
    j = j % 3;
}
//说明：程序段中每行开头的数字（1~10）是对每条语句的编号
```

1. 画出程序的控制流图（用题中给出的语句编号表示）并计算控制流图的圈复杂度V(G)
2. 写出所有的基本独立路径
3. 根据基本路径设计一组测试用例
4. 分析这组测试用例是否满足语句覆盖、判定覆盖、条件覆盖、条件组合覆盖



画出如题的控制流图如下：



其中 2-1、2-2 和 7-1、7-2 分别代表语句 2 和语句 7 中的两个判定条件

由控制流图知，圈复杂度=5。所以有5条基本路径。

Path1:1-2(2-1)-7(7-1)-8-9

Path2:1-2(2-1)-7(7-1)-7(7-2)-9

Path3:1-2(2-1)-7(7-1)-7(7-2)-8-9

Path4:1-2(2-1)-2(2-2)-7-8-9

Path5:1-2(2-1)-2(2-2)-4,5-7-8-9

注：基本路径集并不唯一

输入	预期输出	覆盖路径
不存在	不存在	path1
x=2,y=4,z=0	K=0,j=0	path2
X=2,y=6,z=0	K=0,j=1	path3

输入	预期输出	覆盖路径
X=4,y=1,z=11	K=3,j=2	path4
X=4,y=1,z=9	K=3,j=2	path5

> 注：path1对应的是：x>3为false,和x==4为true同时满足的情况，不存在

3.4 如何衡量测试效果?

- 覆盖率是测试覆盖质量的一个重要指标，通过测试覆盖率的统计，可以帮助改善测试用例，提高测试质量。
- 覆盖率检查的标准有逻辑覆盖、循环覆盖和基本路径覆盖等。
- 需要通过工具对软件测试覆盖结果进行考察。

3.5 最常用的覆盖率分析工具有哪些

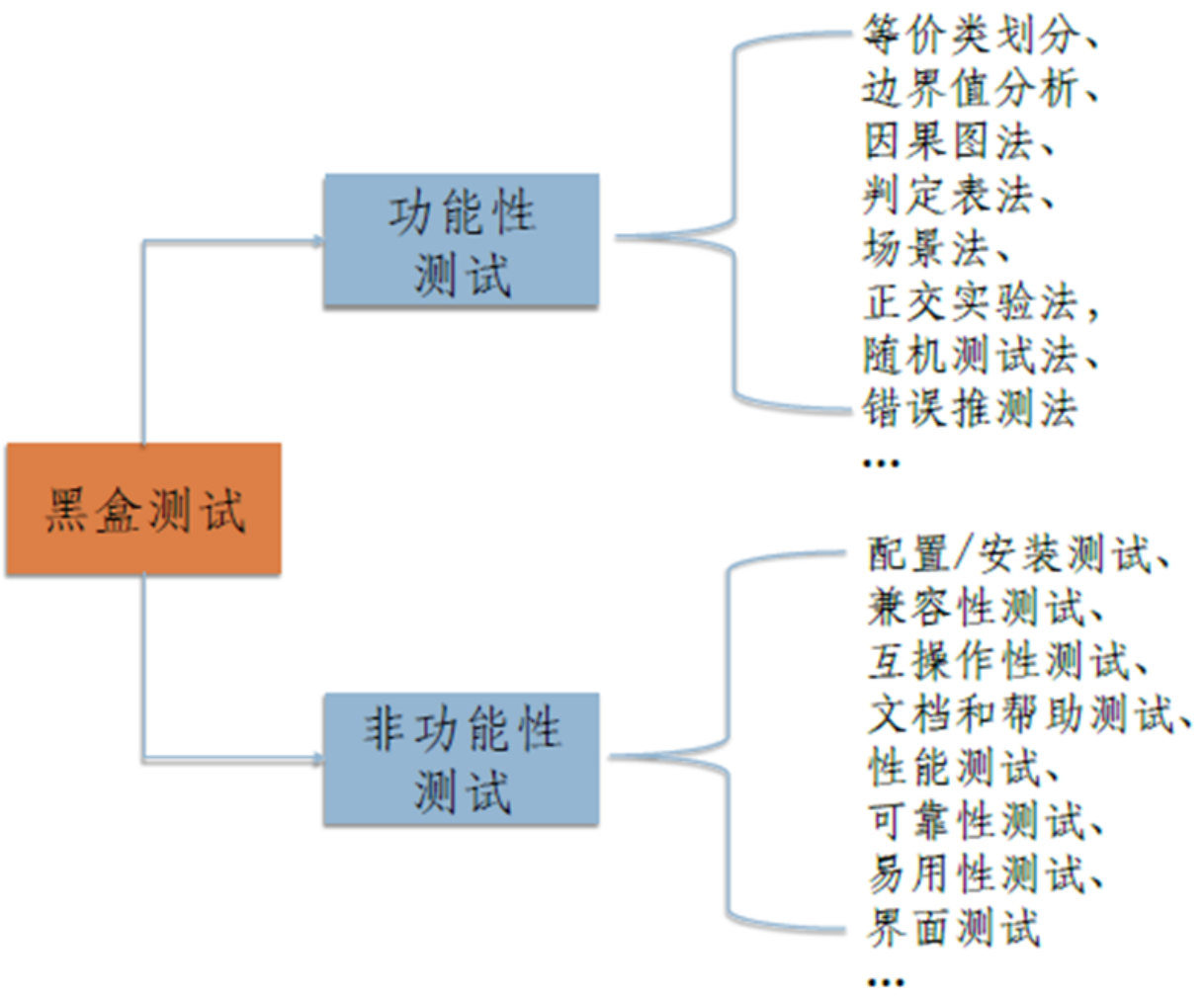
- JUnit框架下有EclEmma、Jcoverage、Jcoco等；
- 在NUnit框架下有Ncover；
- 针对C/C++语言有与GCC配套的Gcov工具；
- Eclipse IDE有Coverlipse插件；
- Maven在调用测试代码和代码覆盖率分析方面有强大的功能
- 其他的还有Cobertura、Quilt和JCoverage等

3.6 代码覆盖率检测工具

- Codecover
- JaCoCo(EclEmma)

黑盒测试(2022)

> 把被测程序视为一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下进行的测试 。



> 例：设计测试用例，完成对所有实数进行开平方运算（`y = sqrt(x)`）的程序的测试。

4.1 如何划分等价类?

> 先从程序的规格说明书中找出各个输入条件，再为每个输入条件划分两个或多个等价类，形成若干个互不相交的子集。

实践训练：请设计测试用例测试一个系统的登录功能

序号	输入条件	有效等价类	编号	无效等价类	编号
1	用户名	由4到20个中文、字母、数字、"-""__"组合的字符串	1	用户名长度小于4且不为空和空格	3
				用户名长度大于20	4
				用户名包含除中文、字母、数字、"-""__"之外的字符	5
				用户名为空	6
				用户名为空格	7
				用户名由中文、字母、数字、"-""__"组成的不正确的用户名	8
2	密码	由6到20个字母、数字和符号组合的字符串	2	密码长度小于6且不为空和空格	9
				密码长度大于20	10
				密码为空	11
				密码为空格	12
				不正确的密码	13

测试用例编号	输入		预期输出	覆盖的等价类
	用户名	密码		
001	sjmei	hznu001	登录成功	1，2
002	sjm	hznu001	用户名长度过短	3，2
003	qwertyuiopasdfghjklzx	hznu001	用户名长度过长	4，2
004	Sjmei#	hznu001	用户名包含非法字符	5，2
005	空	hznu001	用户名不能为空	6，2
006	空格	hznu001	用户名不能为空格	7，2
007	sjmui	hznu001	用户名不存在	8，2
008	sjmei	1234	密码过短	1，9
009	sjmei	12345678901234567890	密码过长	1，10
010	sjmei	空	密码不能为空	1，11
011	sjmei	空格	密码不能为空格	1，12
012	sjmei	hznu002	密码不正确	1，13

4.2 课堂练习

> 某城市电话号码由三部分组成，分别是：
>
> 地区码—— 空白或三位数字；
>
> 前 缀—— 非‘0’和‘1’开头的三位数字；
>
> 后 缀—— 4位数字。
>
> 假定被测程序能接受一切符合上述规定的电话号码，拒绝所有不符合规定的电话号码。
>
> 请用等价类法设计测试用例

有效等价类	编号	无效等价类	编号
地区码空白	(1)	地区码既不是空白也不是三位数	(6)
地区码三位数字	(2)		
前缀非0和1开头	(3)	前缀0开头	(7)
		前缀1开头	(8)
前缀三位数	(4)	前缀不是三位数	(9)
后缀四位数	(5)	后缀不是四位数	(10)
		号码不是纯数字	（11）

参考答案:使用等价类划分法

输入条件	有效等价类	编号	无效等价类	编号
地区码	空白	1	有非数字字符	5
			少于3位数字	6
	3位数字	2	多于3位数字	7
前缀	200~999	3	有非数字字符	8
			起始位为'0'的三位数字	9
			起始位为'1'的三位数字	10
			少于3位数字	11
			多于3位数字	12
后缀	4位数字	4	有非数字字符	13
			少于4位数字	14
			多于4位数字	15

测试用例编号	输入数据			预期输出	覆盖等价类
	地区码	前缀	后缀		
1	空白	234	1234	接受（有效）	1, 3, 4
2	123	234	1234	接受（有效）	2, 3, 4
3	20A	234	1234	拒绝（无效）	5,3,4
4	33	234	1234	拒绝（无效）	6,3,4
5	1234	234	1234	拒绝（无效）	7,3,4
6	123	2B3	1234	拒绝（无效）	2, 8,4
7	123	013	1234	拒绝（无效）	2,9,4
8	123	123	1234	拒绝（无效）	2,10,4
9	123	23	1234	拒绝（无效）	2,11,4
10	123	2345	1234	拒绝（无效）	2,12,4
11	123	234	1B34	拒绝（无效）	2,3,13
12	123	234	34	拒绝（无效）	2,3,14
13	123	234	23345	拒绝（无效）	2,3,15

4.3 怎样用边界值分析法设计测试用例?

1. 首先确定边界情况。通常输入或输出等价类的边界就是应该着重测试的边界情况
2. 选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值。

实践训练：电话号码

某城市电话号码由三部分组成，分别是：

地区码——空白或3位数字；

前缀——非“0”和“1”开头的三位数字；

后缀——4位数字。

> 假定被测程序能接受一切符合上述规定的电话号码，拒绝所有不符合规定的电话号码。
>
> 请用边界值法设计测试用例

4.4 判定表法

- （4）判定表

		1	2	3	4	5	6	7	8	9
条件	a, b, c构成三角形	N	Y	Y	Y	Y	Y	Y	Y	Y
	a = b?	—	Y	Y	Y	Y	N	N	N	N
	a = c?	—	Y	Y	N	N	Y	Y	N	N
	b = c?	—	Y	N	Y	N	Y	N	Y	N
动作	非三角形	✓								
	不等边三角形									✓
	等腰三角形					✓		✓	✓	
	等边三角形		✓							
	不可能			✓	✓		✓			

单元测试（2022）

- 单元测试的定义
  - 定义：单元测试是对软件基本组成单元进行的测试。
  - 时机：一般在代码完成后由开发人员完成,QA人员辅助
  - 概念： 模块, 组件, 单元
- 为何要进行单元测试：

    - 尽早发现错误
        - 错误发现越早,成本越低.
        - 发现问题比较容易
        - 修正问题更容易
    - 检查代码是否符合设计和规范，有利于将来代码的维护

- 单元测试的目标：单元模块被正确编码

如何进行单元测试

- 单元测试的任务
    - 模块接口测试
    - 模块局部数据结构测试
    - 错误处理检测
    - 路径测试
    - 模块边界条件测试

- 如何进行单元测试
    - 静态测试
    - 动态测试
        - 设计测试用例
        - 搭建测试环境
        - 编写测试程序（脚本）
        - 执行测试
        - 分析测试结果

测试内容

- Right-BICEP

    - Right-结果是否正确?
    - B-是否所有的边界条件都是正确的?
    - I-能查一下反向关联吗?
    - C-使用其它手段交叉检查一下结果吗?
    - E-你是否可以强制错误条件发生?
    - P-是否满足性能要求?
- 边界条件
    - 一致性（Conformance）
        - 值是否和预期的一致
    - 有序性（Ordering）
        - 值是否如应该的那样，是有序或无序
    - 区间性（Range）
        - 值是否位于合理的最小值和最大值之内
    - 引用，耦合性（Reference）
        - 代码是否引用了一些不在代码本身控制范围之内的外部资源
    - 存在性（Existence）
        - 值是否存在
    - 基数性（Cardinality）
        - 是否恰好有足够的值
    - 时间性（Time）

第八章 非功能性测试

- 性能的具体指标
    - 数据传输的吞吐量(Transactions)

    - 数据处理效率(Transactions per second)
    - 数据请求的响应时间（Response time）
    - 内存和CPU使用率

- 性能测试目标

    - **获取**系统性能某些指标数据
    - 为了**验证**系统是否达到用户提出的性能指标
    - **发现**系统中存在的性能瓶颈，优化系统的性能

- 性能测试类型

    - 性能验证测试，验证系统是否达到事先已定义的系统性能指标、能否满足系统的性能需求
    - 性能基准测试，在系统标准配置下获得有关的性能指标数据，作为将来性能改进的基准线
    - 性能规划测试，在多种特定的环境下，获得不同配置的系统的性能指标，从而决定在系统部署时采用什么样的软、硬件配置
    - 容量测试，也可以看作性能的测试一种，因为系统的容量可以看作是系统性能指标之一

压力测试与负载测试

- 负载测试

    - 负载测试的目标是测试在一定负载情况下，系统的性能

- 压力测试

    - 模拟实际应用的软硬件环境及用户使用过程的系统负荷，长时间或超大负荷地运行测试软件，来测试被测系统的性能、可靠性、稳定性等。

- 负载
    - 每次请求发送的数据量(Request Per Second, RPS)
    - 并发连接数(Simultaneous Connections)
    - 思考时间(thinking time），用户发出请求之间的间隔时间
    - 加载的循环次数或持续时间
    - 加载的方式或模式，如均匀加载、峰值交替加载等

性能测试的方法和技巧

- 两种负载类型
    - ”flat”测试
    - ramp-up测试

- 对于企业级的系统，性能测试的方法主要有
    - 基准测试
    - 性能规划测试
        - 要确定系统的容量，需要考虑几个因素:
            - 用户中有多少是并发与服务器通信的
            - 每个用户的请求时间间隔是多少。
        - 如何加载用户以模拟负载状态?
            - 最好的方法是模拟高峰时间用户与服务器通信的状况。
        - 什么是确定容量的最好方法?
            - 结合两种负载类型的优点，并运行一系列的测试
    - 渗入测试
    - 峰谷测试

性能测试对象

- 前端性能测试
- Web应用服务器
- 数据库服务器

安全性测试

安全性测试的范围

- 系统级别的安全性
- **应用程序级别的安全性**
- 安全功能测试：数据机密性、完整性、可用性、不可否认性、身份认证、授权、访问控制、审计跟踪、委托、隐私保护、安全管理等
- 安全漏洞测试：从攻击者的角度,以发现软件的安全漏洞为目的。安全漏洞是指系统在设计、实现、操作、管理上存在的可被利用的缺陷或弱点

安全测试的方法

- 测试方法一
  - 基于漏洞的方法
    - 从软件内部考虑其安全性，识别软件的安全漏洞。如借助于特定的漏洞扫描器。
  - 基于威胁的方法
    - 从软件外部考察其安全性,识别软件面临的安全威胁并测试其是否能够发生
  - 模拟攻击测试
    - 是一组特殊的、极端的测试方法，如Fuzzing，使用大量有效的数据作为应用程序的输入，以程序是否出现异常为标志，来发现应用程序中可能存在的安全漏洞
- 测试方法二（渗透测试方法）
  - 白盒测试
    - 用商业的统计分析工具

        - 对源码的内审
        - 借助工具Coverity /Fortify等
    - 黑盒测试
        - 错误注入(Fault injection)
        - Dumb Fuzzing
    - 灰盒测试
        - Smart Fuzzing / Intelligent fuzzing

静态安全性测试

- 对代码进行静态扫描，提前发现潜在的安全性问题，那么会避免后期因为漏洞而付出的代价。
- 从微观来看程序代码有哪些安全性问题
    - 语言级代码安全，如C/C++缓冲区溢出...
    - 逻辑级代码安全，如身份认证与授权...
    - 规范级代码安全，如内存泄漏.

动态安全性测试

- 渗透测试
- 安全功能的测试

- 渗透测试实施策略
    - **全程监控**:采用类似ethereal/wireshark的嗅探软件进行全程抓包嗅探
    - **择要监控**:对扫描过程不进行录制，仅仅在安全工程师分析数据后，准备发起渗透前才开启软件进行嗅探
    - **主机监控**:仅监控受测主机的存活状态，避免意外情况发生
    - **指定攻击源**:该源地址的主机由用户进行进程、网络连接、数据传输等多方监控

    - **对关键系统**，可以采用对目标的副本进行渗透测试

- 模糊测试：巨大优势是，测试设计极其简单，系统的行为先入为主
- Web安全性测试—静态测试和渗透测试

可靠性测试

- 可靠性(Reliability)是产品在规定的条件下和规定的时间内完成规定功能的能力，它的概率度量称为可靠度。
    - 规定的时间
    - 规定的环境条件
    - 规定的功能

容错性测试

- 容错性测试是检查软件在异常条件下自身是否具有防护性的措施或者某种灾难性恢复的手段。
- 容错性测试包括两个方面
    - 输入异常数据或进行异常操作，以检验系统的保护性。
    - 灾难恢复性测试。

故障转移测试

- Failover 测试:故障转移(Failover)和故障恢复(Failback)
- 服务器的Failover测试的目的: 检查系统是否具备某种灾难性恢复的手段.
- 在服务器的Failover测试中, 将包括多种情况
    - 客户机或服务器掉电

- 客户机与服务器网络中断
- 服务器相关的程序CRASH
- 系统中全部或部分CORE SERVER出现掉电/网络中断情况.

兼容性测试

- 软件兼容性测试是指验证软件之间是否正确地交互和共享信息
- 兼容性包括：
  - 硬件兼容
  - 软件之间兼容
  - 数据之间兼容
- **向后兼容**是指可以使用软件的以前版本
- **向前兼容**指的是可以使用软件的未来版本

第九章 集成测试

- 集成测试：一种旨在暴露**单元接口之间**、**组件/系统间**交互或协同工作时所存在的缺陷的测试
  - 非渐增式测试模式：先分别测试每个模块，再把所有模块按设计要求放在一起组成所要的程序，如大棒模式(Big-bang Integration)
  - 渐增式测试模式：把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个已测试的模块结合进来测试
- 测试方法：
  - 自顶向下法
  - 自底向上法
  - 三明治方法：是一种混合增殖式测试策略，综合了自顶向下和自底向上两种集成方法的优点，因此也属于基于功能分解集成。

集成测试重在**接口测试**

接口

- 在计算机系统中，也可分为**硬件接口**和**软件接口**。
  - 硬件接口常见的有USB接口、耳机接口、HDMI接口
  - 软件接口常见的有API接口
- 接口测试
  - 接口中所有的参数的不同类型的有效值都要被测试
  - 参数的每个错误类型都要准备一个异常用例

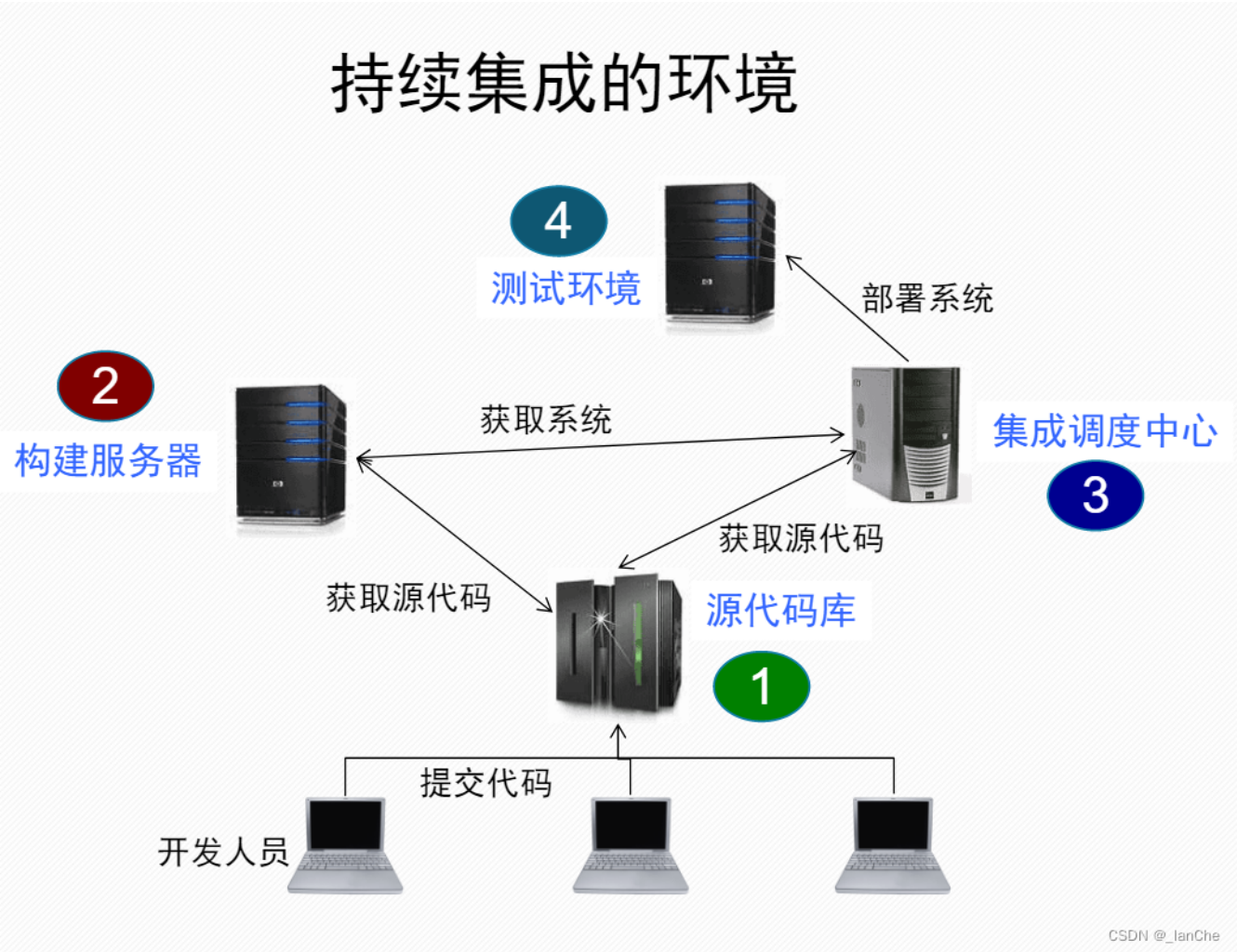
    如缺省值、类型错误、范围错误、参数超过最大位数、无效值、参数的关联性检查等
  - 把每个参数单独作为条件来进行测试，再进行多条件关联组合测试
- 接口测试工具
  - Postman
  - Apifox

持续集成

- 持续集成：经常集成团队的工作，通常每天新完成的代码至少集成一次，也就意味着每天可能会发生多次集成
- 如何做好持续集成
  - 维护单一的代码库
  - 每人每天及时将自己的代码提交到产品代码主干上
  - 应在产品运行的克隆环境上运行测试，尽量确保测试环境和产品运行环境的一致性
  - 版本构建速度很快，1小时或2小时
  - 造成版本无法构建的缺陷要立刻修复，确保构建成功才能进行版本验证

    - 每人能较容易地拿到最新的可执行版本，及时浏览团队实现的产品功能特性
- CI环境是支撑CI实施的基础设施

持续集成的环境



持续集成系统的组成

- 一个代码存储库
- 一个自动构建过程
    - 包括：自动编译、分发、部署和测试等。

- 一个持续集成服务器

持续集成的原则

- 需要版本控制软件保障团队成员提交的代码不会导致集成失败。
- 开发人员必须及时向版本控制库中提交代码，也必须经常性地从版本控制库中更新代码到本地;
- 需要有专门的集成服务器来执行集成构建。根据项目的具体实际,集成构建可以被软件的修改来直接触发，也可以定时启动,如每半个小时构建一次;
- 必须保证构建的成功。如果构建失败,修复构建过程中的错误是优先级最高的工作。一旦修复，需要手动启动一次构建。

持续集成的内涵

- 持续检查（用工具扫描分析代码）
- 持续构建（Build）
- 持续部署
- 持续验证、测试
- 持续集成环境(基础设施)
- 持续报告

持续集成的核心价值

- 持续集成中的任何一个环节都是**自动**完成的,有利于**减少重复过程**以节省时间、成本等;
- 任何时间点都能**第一时间发现软件的集成问题**,使随时（快速、可靠、低风险)发布可部署的软件成为了可能;做到持续集成,才能做到**持续交付**
- 有利于软件本身的发展趋势，在需求不明确或频繁变更的情景中尤其重要,能够**更好、更快地满足用户的需求**
- 能帮助团队进行有效决策、提高开发产品的信心
- 尽早、尽快进行集成,有助于**尽早发现Bug**,避免集成中大量Bug涌现,**容易定位和修正Bug**,这样能够提高软件开发的质量与效率

第十章 缺陷管理

缺陷的定义

- 缺陷：最终产品同用户的期望不一致。
  - 功能错误
  - 功能遗漏
  - 超出需求的部分
  - 性能不符合要求

为什么要对缺陷进行管理

如何进行缺陷的管理

缺陷的区分

- 为了更好地分析缺陷，需要对缺陷在**严重程度**、**优先级**以及**状态**上加以区分。
  - 缺陷严重程度
    - 致命的（fatal）

      造成系统获应用程序奔溃、死机、系统悬挂或造成数据丢失、主要功能完全丧失。

    - 严重的（critical）

      指功能或特性没有实现，主要功能部分丧失，次要功能完全丧失，或致命的错误声明

- 一般的（major）

  不影响系统的基本使用，但没有很好地实现功能，没有达到预期效果。如次要功能丧失，提示信息不准确，或用户界面差，操作时间长等。

- 微小的（minor）

  对功能没有影响，产品及属性仍可使用，如有个别错别字、文字排版不整齐等。

- 缺陷优先级

  - 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。

- 缺陷状态

  - 新建（new）

    由测试人员发现并提交

  - 已分配（assigned）

```
开发人员接收了该Bug
```

- 解决（fixed）

```
开发人员解决了该Bug,并给测试人员回归测试
```

- 关闭（closed）

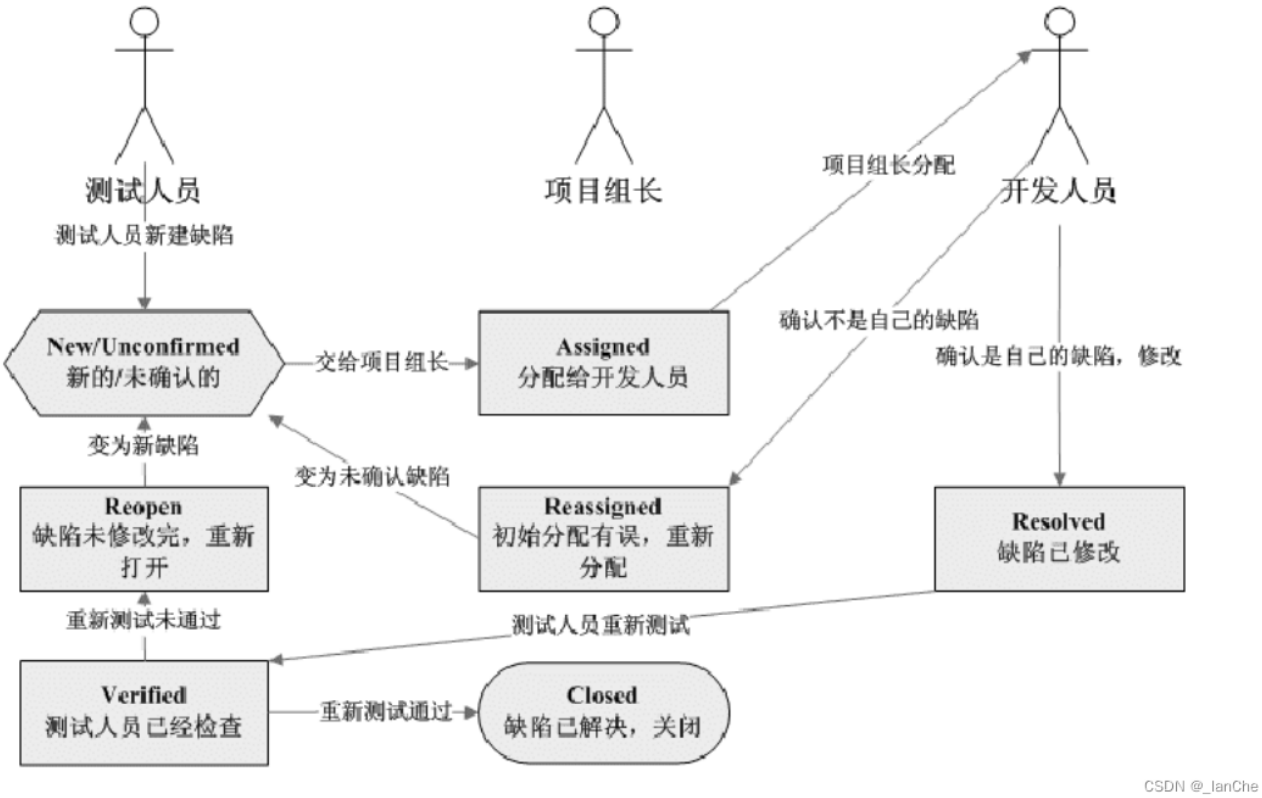
```
测试人员回归测试确认已经解决
```

- 重新打开（reopen）

缺陷管理的角色

- 开发负责人（项目组长）
  - 负责**制定缺陷管理计划和流程**
  - 将测试工程师发现的问题**指派**给指定开发工程师
  - **协调**缺陷管理流程中的问题

- 测试工程师
  - 将发现的问题**提交**到缺陷管理系统中
  - 写明问题的描述、严重程度，问题重现方法
  - 负责**重新测试**开发工程师修改过的缺陷。



- 开发工程师
  - **确认并修改**指定给自己的软件缺陷。

缺陷管理工具介绍

- 禅道
- TestIn
-