**一、软件测试的原则**

1. 软件测试为了发现错误而执行程序的过程
2. 尽量避免编码人员测试自己的程序
3. 好的测试用例能够对未发现的错误高度敏感
4. 成功的测试用例能够发现未知的错误
5. 成功的测试需要仔细的定义输入输出的期望值
6. 成功的测试需要仔细研究分析测试结果

**二、代码检查、走查和评审**

**01 代码检查（用于代码检查的错误列表）**

数据引用错误 数据生命错误

运算错误 控制流程错误

接口错误 输入/输出错误

其他检查

**02 代码走查（小组形式）**

**03 同行评审**

一种依据程序整体质量、可维护性、可扩展性、易用性和清晰性对匿名程序进行评价的技术

**三、测试用例的设计**

推荐使用特定的面向黑盒测试的测试用例设计，而后使用白盒测试方法对程序的逻辑结构进行检查以补充测试用例

|  |  |
| --- | --- |
| 黑盒测试 | 白盒测试 |
| 等价类划分 | 语句覆盖 |
| 边界值分析 | 条件覆盖 |
| 因果图分析 | 判定覆盖 |
| 错误猜测 | 判定/条件覆盖 |
|  | 多重条件覆盖 |

**01 白盒测试**

逻辑覆盖测试

**1.1 语句覆盖**

路径每条语句都覆盖到

**1.2 判定覆盖或者分支覆盖**

判定覆盖可以满足语句覆盖

每个判断都至少有一个真和假的输出结果

**1.3 条件覆盖**

每个判断中的每个条件的所有可能结果至少执行一次

**1.4 判定/条件覆盖**

将每个判断的每个条件的所有结果至少执行一次

将每个判断的所有可能结果至少执行一次

将每个入口点都至少调用一次

缺点：某些特定的条件会屏蔽其他条件

**1.5 多重条件覆盖**

每个判定中所有可能结果的组合

**02 黑盒测试**

**2.1 等价类划分**

设计用例的步骤：①确定等价类 ②生成测试用例

**2.2 边界值分析**

输入值和输出的等价类中恰好处于边界或超过边界、或者再边界以下的状态

（边界值和等价类划分的一个弱点:未对输入条件的组合进行分析）

**2.3 因果图**

2.3.1 有利于指出规格说明的不完整性和不明确处

2.3.2 生成测试用例采用过程：

①将规格说明分解为可执行的片段

②确定规格说明的因果关系

③分析规格说明的语义内容，并将其转为连接因果关系的布尔图

④给图加上注解符号

⑤仔细跟踪图中的状态变化

⑥将判定表中的列转换成测试用例

**2.4 错误猜测**

凭直觉和经验猜测出错的可能类型

**2.5 测试策略**

①规格说明包含输入条件组合 -> 首先使用**因果图分析法**

②任何情况下都使用**边界值分析法**

③应为输入和输出确定**有效和无效等价类**

④使用**错误猜测**技术增加更多的测试用例

⑤针对上述测试用例集检查程序的**逻辑结构**

**四、单元测试**

单元测试是对程序中的单个子程序、子程序或过程进行测试的过程

**01 测试用例设计**

①需要两种类型的信息：模块的规格说明和模块的源代码

②模块测试总体面向白盒测试

③模块测试用例的设计过程：

使用一种或多种**白盒测试**方法分析模块的逻辑结构，然后使用**黑盒测试**方法对照模块的规格说明比补充测试用例

**02 增量测试**

►非增量测试：将每个模块进行独立测试，最后将这些模块组装成完整的程序

►增量测试：首先将下一个要测试的模块组装到已经测试的模块集合中去

►驱动模块和桩模块：



**总结：**

①非增量测试所需的工作量多

②增量测试可以较早的发现模块中与不匹配的接口、不正确的假设相关编程错误

③增量测试，调试会进行容易一些

④增量测试会将测试进行的更彻底

⑤非增量测试占用的机器时间显得少一些

⑥模块测试阶段开始时，如果使用的是非增量测试，就会有更多的机会进行操作

（增量测试优于非增量测试）

**03 自顶向下测试/ 自底向上测试**

►增量测试：自顶向下测试/ 自底向上测试

►自顶向下测试：从程序的顶部或初始模块开始

►自底向上测试：从程序的终端模块

►比较：

|  |  |  |
| --- | --- | --- |
| 测试类型 | 优点 | 缺点 |
| 自顶向下测试 | 1.主要的缺陷发生在程序的顶层将非常有利 | 1.必须开发桩模块 |
| 2.一旦引入I/O功能，提交测试用例会更容易 | 2.桩模块要比最初表现的更复杂 |
| 3.早期的程序框架可以进行演示，并可激发积极性 | 3.在引入I/O功能之前，向桩模块中引入测试用例比较困难 |
| 4.创建测试环境可能很难，甚至无法实现 |
| 5.观察测试输出很困难 |
| 6.使人误解设计和测试可以交迭进行 |
| 7.会导致特定模块测试的完成延后 |
| 自底向上测试 | 1.主要的缺陷发生在程序的底层将非常有利 | 1.必须开发驱动模块 |
| 2.测试环境比较容易建立 | 2.直到最后一个模块添加进去，程序才能形成一个整体 |
| 3.观察测试输出比较容易 |  |

**五、更高级别的测试**

测试周期是模仿软件开发周期建立起来的

**最终用户** **安装测试**

**验证 需求 验收测试**

**验证 目标 系统测试**

**验证 外部规格说明 功能测试**

**验证 系统设计 集成测试**

**验证 程序结构设计**

**验证 模块规格说明 模块测试**

**验证 代码**

**01 功能测试**

功能测试是一个试图发现程序与其外部规格说明之间存在不一致的过程

功能测试通常是一项黑盒操作

目的：为了暴露程序的错误以及发现程序与规格说明书中的不一致之处，而不是为了证明程序符合其规格说明书

**02 系统测试**

测试用例的15个分类

**能力测试**：确保程序的目标功能实现

**容量测试**：发现处理大容量数据时的程序异常

**强度测试**：发现在大规模负载、高强度不间断持续的数据处理中的异常

**可用性测试**：评估最终用户在使用软件并与软件交互时的可用性问题

**安全性测试**：试图攻破程序的安全防线

**性能测试**：评估程序的响应时间以及吞吐量

**存储测试**：

确保程序可以正确处理其对存储的需求，包括系统的存储和物理上的存储

**配置测试**：检查程序是否能在推荐配置上流畅运行

**兼容性/转换测试**：评估新版本是否能兼容旧版本

**安装测试**：确保能够在所有支持的平台上安装软件

**可靠性测试**：评估程序是否能达到规格说明中的运行时常和MTBF要求

**可恢复测试**：测试系统恢复相关的功能是否按照设计要求

**服务/可维护性测试**：

评估系统是否拥有良好的数据处理和日志机制，以备技术支持和调试之需

**文档测试**：校验所有的用户文档是否准确

**过程测试**：对软件系统操作或维护所需涉及的流程进行评估和确定

**03 验收测试**

将程序及其最初的需求及最终用户当前的需要进行比较的过程

**04 安装测试**

安装测试与设计过程中任何阶段都没有联系

目的是为了发现在安装过程中出现的错误

在安装软件系统期间会发生很多事件，作为示例的简短列表包括一下事件：

1.用户必须选择大量选择

2.必须分配并加载文件额库

3.必须进行有效的硬件配置

4.软件可能要求网络联通，以便与其他软件连接

安装测试应该由生产软件系统的机构来设计，作为软件的一部分来发布，在系统安装完成之后进行。

**05 测试的计划与控制**

一个良好的测试计划：

1.目标（必须定义每个阶段的测试目标）

2.结束准则（每个测试阶段何时可以结束）

3.进度（时间表）

4.责任（每个测试阶段由谁来设计、编写和验证测试用例等）

5.测试用例库及标准

6.工具（使用的测试工具）

7.计算机时间（每个测试阶段所需要的计算机时间等）

8.硬件配置

9.集成（集成测试）

10.跟踪步骤

11.调试步骤

12.回归测试

**06 测试结束准则**

**①根据的是特定的测试用例设计技术**

例子：

定义**模块测试**的结束准则:

测试用例来源：1.满足多重条件覆盖准则2.对模块接口规格说明进行边界值分析，产生的所有测试用例最终都是不成功的

满足下列情况时规定**功能测试结束**：

测试用例来源：1.因果图分析2.边界值分析3.错误猜测，产生的所有测试用例最终都是不成功的

缺点：

1.系统测试阶段没有特定方法，这类准则不起作用

2.依赖于主观的度量

**②以确切的数量来描述结束测试的条件**

**③单位时间内发现的错误数量，通过检查统计曲线的形状，来决定是继续还是结束**

**六、可用性（用户体验）测试**

**01 可用性测试基本要素**

可用性或基于用户的测试基本属于黑盒测试的范畴

可用性测试显然是任何软件开发都不能忽略的重要组成部分

1. 用户交互设计是否考虑到用户最终理解等

2. 程序的输出是否有意义，没有敏感词汇

3. 错误提示信息是否直白易懂

4. 用户界面是否保持慨念一致

5. 需要提高精确性和准确度的软件系统是否提供有效的输入验证？

6. 对用户的输入是否能够及时反应

7. 程序操作是否容易上手

8. 系统是否包含太多选项

9. 软件设计是否有助于用户准确输入

10.用户操作可以轻松重复使用？

11.用户是否能在众多的功能来回切换不发生意外？

12.软件的功能实现是否达到了规格设计要求？

**02 可用性测试流程**

**2.1测试用户的选择**

一份完整的可用性测试草案通过一组用户完成多个测试以及不同组用户完成多个测试（用户记忆）

**2.2需要多少用户进行测试**

Nielsen研究发现，测试中找到的可用性问题数量可以用数学公式表示：

E= 100 \* （1-（1-L）^n）

E=找到错误的比例

L=单个测试人员发现的可用性问题比例

n=测试人数

**2.3数据采集方法**

“发声思考”：录制测试

“眼球追踪”

**2.4可用性调查问卷**

是否/真假/某种程序的同意和反问

可以根据测试任务以及测试区域的不同

系统分析软件可以帮助捕捉和解释数据

**2.5何时收工、还是多多益善**

**七、调试**

调试是执行一次成功的测试之后所要进行的工作

包含了两个步骤：

1.确定程序中可疑错误的准确性质和位置 2.修改错误

（错误定位/错误修改）

**01 蛮力法调试**（最缺乏效率）

利用内存信息输出来调试

根据一般的“在程序中插入打印语句”建议来调试

自动化的调试工具进行调试

02 归纳法调试

步骤：1.确认相关数据 2.组织数据 3.作出假设 4.证明假设 5.解决问题

03 演绎法调试

步骤：1.列举出所有可能的原因或假设 2.利用数据排除可能的原因 3.提炼剩下的假设 4.证明剩下的假设 5.修复问题

04 回溯法调试

沿着程序的逻辑结构回溯不正确的结果，直到找出程序逻辑错误的位置

05 测试法调试

测试用例

常结合归纳测试法一起使用，以获得进行假设和/或证明假设所需的信息，也可以结合演绎法，以排除有嫌疑的原因，提炼剩下的假设，并/或证明假设

06 调试的原则

**6.1定位错误的原则**

1.动脑筋

2.如果遇到了僵局，就留到稍后解决

3.如果遇到了困难，就把困难描述给其他人听

4.仅将调试工具作为第二种手段

5.避免使用实验方法--仅将其作为左后的手段

**6.2修改错误的技术**

1.存在一个缺陷的地方，很有可能还存在其他缺陷

2.应修正错误本身，而不仅是其症状

3.正确纠正错误的可能性并非100%

4.随着程序规模的增加，正确修改错误的可能性反而降低

5.应意识修改错误会引入新错误的可能性

6.修改错误的过程也是临时回到设计阶段的过程

7.应修改源代码，而不是目标代码

07 错误分析

错误出现在什么地方？

谁制造了这个错误？

哪些做得不正确？

如何避免该错误的出现？

为什么错误没有更早的发现？

该如何更早的发现错误？

**八、敏捷开发模式下的测试**

01 敏捷开发的特征

敏捷开发提倡迭代和增量式开发模式

围绕以用户为中心、以客户需求为导向的开发过程

敏捷没有单一固定的开发方法和过程，很多快速开发模式都可以看作是敏捷

有三个共同点： 依赖客户的参与、测试驱动、紧凑的迭代开发周期

02 敏捷测试

03 极限编程与测试

**九、互联网应用测试**

01 电子商务的基本结构

互联网应用系统本质上是C/S模式的程序

客户端：web浏览器 服务端：web或应用服务器

C/S结构：

每一层都可作为一个具有清晰定义接口的黑盒

第一层：运行web网站

第二层：又称“业务层”，运行应用服务层

第三层：数据存储

02 测试的挑战

用户群庞大而且五花八门

业务环境

地点

安全性

测试环境

测试浏览器的兼容性

03 测试的策略

**十、移动应用测试**

01 移动环境

02 测试面临的挑战

03 测试方法