第七章　实现

7.1　编码

7.2　软件测试基础

7.6　白盒测试技术

7.7　黑盒测试技术

7.3　单元测试

7.4　集成测试

7.5　确认测试

7.8　调试

7.3 单元测试

- 单元测试集中检测软件设计的最小单元——模块。
- 在源程序代码通过编译程序的语法检查后，可以用详细设计描述作指南，对重要的执行通路进行测试，以便发现模块内部的错误。
- 单元测试主要使用白盒测试技术，辅以黑盒测试，而且对多个模块的测试可以并行地进行。
- 可用人工测试和计算机测试两种不同类型的测试方法

一、测试重点

1.模块接口

- 实参的数目、次序、属性、单位与形参是否一致；
- 是否修改了只作输入用的变量；
- 全局变量的定义和用法在各个模块中是否一致。

2.局部数据结构

- 应该仔细设计测试方案，以便发现局部数据说明、初始化、默认值等方面的错误。

3.重要的执行通路

- 选择最有代表性、最可能发现错误的执行通路进行测试。

4.出错处理通路

好的设计应能预见出现错误的条件，并且设置适当的处理错误的通路。

- 应着重测试下述可能发生的错误
  - 对错误的描述是难以理解的；
  - 记下的错误与实际遇到的错误不同；
  - 在对错误进行处理之前，错误条件已经引起系统干预；
  - 对错误的处理不正确；
  - 描述错误的信息不足以帮助确定造成错误的位置。

5.边界条件

- 边界测试是单元测试中最后的也可能是最重要的任务。
- 软件常常在它的边界上失效
  - 例如，处理n元数组的第n个元素时，或做到i次循环中的第i次重复时，往往会发生错误。
- 使用刚好小于、刚好等于和刚好大于最大值或最小值的数据结构、控制量和数据值的测试方案，非常可能发现软件中的错误。

二、代码审查

- 由审查小组正式对源程序进行人工测试。它是一种非常有效的程序验证技术，对于典型的程序来说，可以查出30%～70%的逻辑设计错误和编码错误。
- 审查小组最好由下述4人组成。

(1) 组长，很有能力的程序员，且没有直接参与这项工程；

(2) 程序的设计者；

(3) 程序的编写者；

(4) 程序的测试者。

7.3 单元测试

- 审查会上由程序的编写者解释他是怎样用程序代码实现设计的，通常是逐个语句地讲述程序的逻辑，小组其他成员仔细倾听他的讲解，并力图发现其中的错误。
- 审查会上需要对照程序设计常见错误清单，分析审查这个程序。当发现错误时由组长记录下来，审查会继续进行(**审查小组的任务是发现错误而不是改正错误**)。
- 代码审查比计算机测试优越的是：一次审查会上可以发现许多错误；用计算机测试的方法发现错误之后，通常需要先改正这个错误才能继续测试。

结对编程

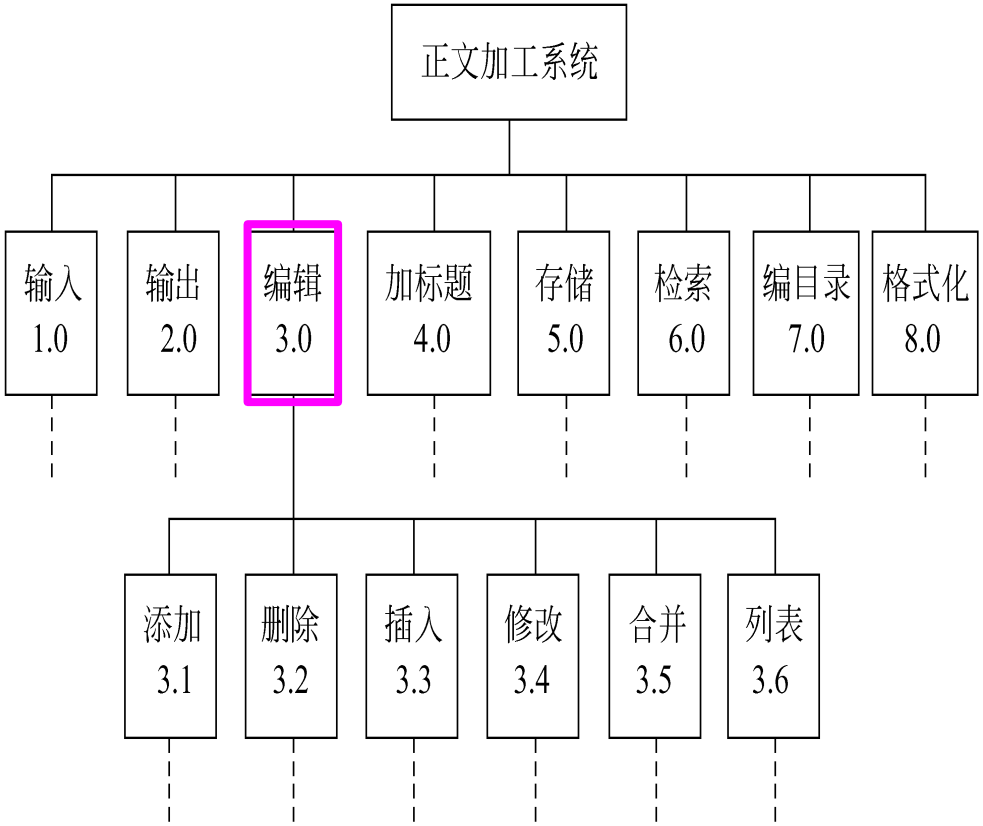
人工测试和计算机测试互相补充，相辅相成。

三、计算机测试

- 模块不是一个独立的程序，因此必须为每个单元测试开发**驱动程序**和(或)**存根程序**。
- **驱动程序**是一个“主程序”，它接收测试数据，启动被测模块，输出测试结果。 。
- **存根程序**代替被测试的模块所调用的模块，接收被测试模块的调用和输出数据。

7.3 单元测试

假定要测试编号为3.0的正文编辑模块。它不是一个独立的程序，需要有一个**测试驱动程序来调用它**。这个驱动程序说明必要的变量，接收测试数据(字符串)，设置正文编辑模块的编辑功能。并且需要有**存根程序**简化地模拟正文编辑模块的下层模块来完成具体的编辑功能。



7.4 集成测试

- 集成测试是测试和组装软件的系统化技术。
- 由模块组装成程序时有两种方法。
  - **非渐增式测试方法**：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序。
  - **渐增式测试**：把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。**这种方法实际上同时完成单元测试和集成测试**。

7.4 集成测试

- **非渐增式测试**把所有模块放在一起作为一个整体来测试。测试时会遇到许多错误，改正错误非常困难，因为在庞大的程序中想要诊断定位一个错误非常困难。
- **渐增式测试**与“一步到位”的非渐增式测试相反，它把程序划分成小段来构造和测试，在这个过程中比较容易定位和改正错误；对接口可以进行更彻底的测试。因此，**目前集成测试时普遍采用渐增式测试方法**。
- 当使用**渐增方式**把模块结合到程序中去时，有**自顶向下**和**自底向上**两种集成策略。

一、自顶向下集成

- 从主控制模块开始沿程序的控制层次向下移动，逐渐把各模块结合起来。在把附属于（及最终附属于）主控制模块的那些模块组装到程序结构中去时，使用两种策略：
  - **深度优先**的结合方法先组装在软件结构的**一条主控制通路**上的所有模块。选择一条主控制通路取决于应用的特点，并且有很大任意性。
  - **宽度优先**的结合方法是沿软件结构**水平**地移动，把处于同一个控制层次上的所有模块组装起来。

- 模块结合进软件结构的具体过程由下述4个步骤完成：

① 对主控制模块进行测试，测试时用存根程序代替所有直接附属于主控制模块的模块；

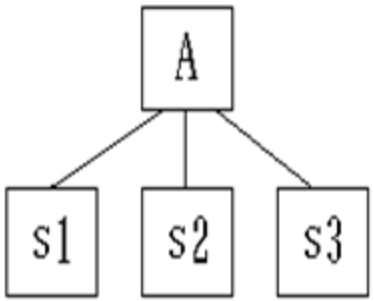
② 根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块替换一个存根程序（新结合进来的模块往往又需要新的存根程序）；

③ 在结合进一个模块的同时进行测试；

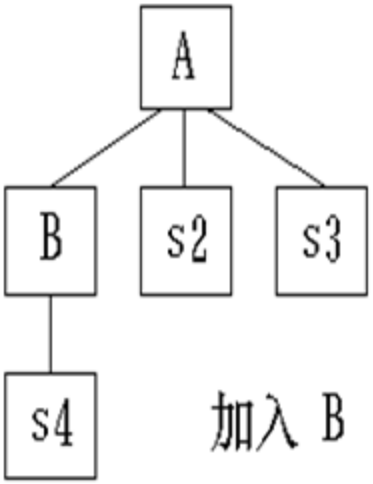
④ 为了保证加入模块没有引进新的错误，可能需要进行回归测试（即全部或部分地重复以前做过的测试）。

从②开始不断重复上述过程，整个程序结构构造完毕。

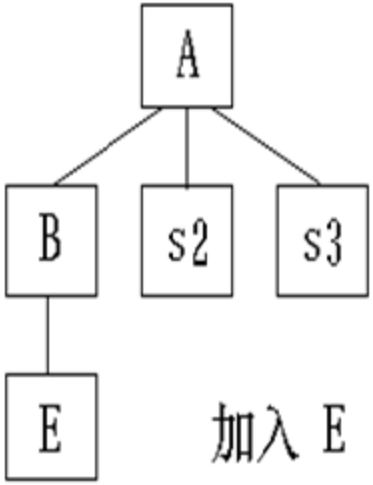
7.4 集成测试



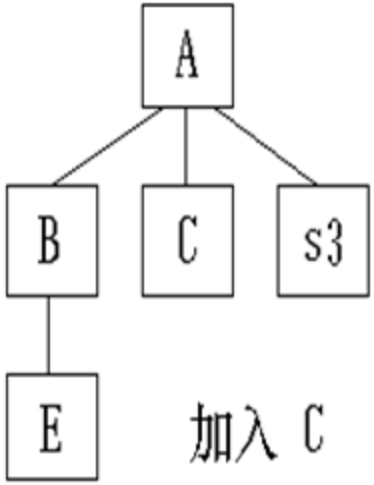
测试 A



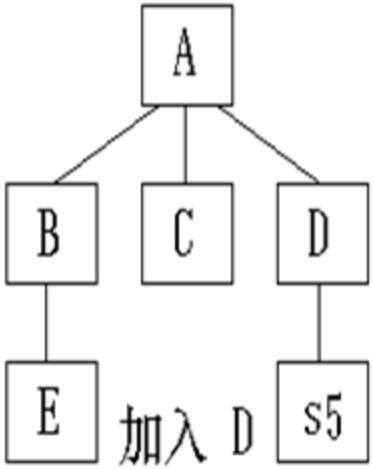
加入 B



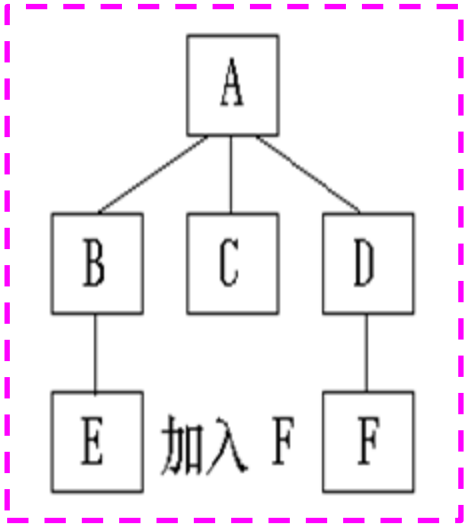
加入 E



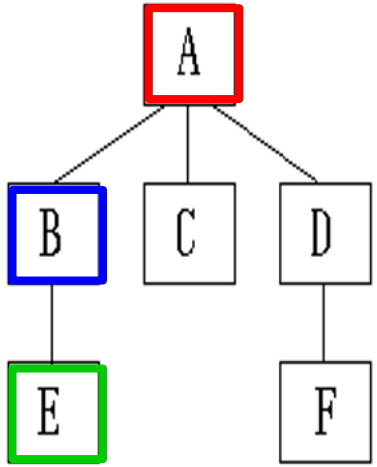
加入 C



加入 D

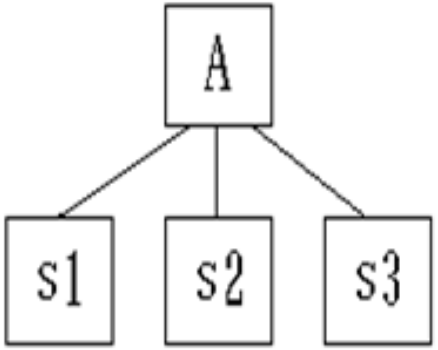


加入 F

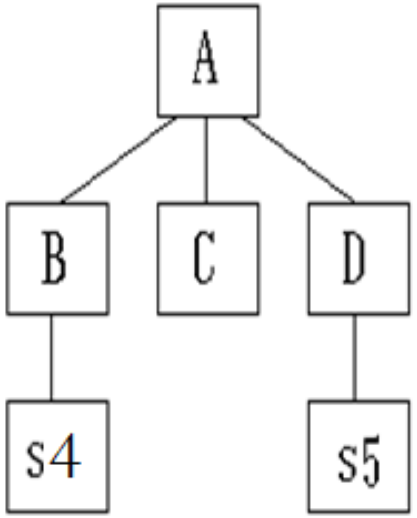


按深度方向组装的例子

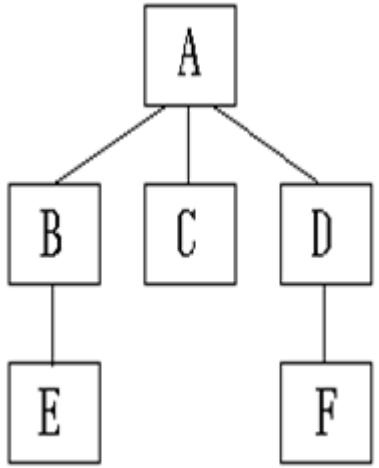
7.4 集成测试



测试 A

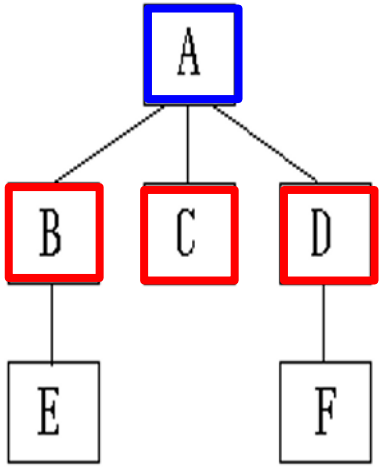


加入B、C、D



加入E、F

按广度方向组装



7.4 集成测试

- **自顶向下**的结合策略能够在测试的早期对主要的控制或关键的抉择进行检验。在一个分解得好的软件结构中，关键的抉择位于层次系统的较上层，因此首先碰到。
- 如果选择**深度优先**的结合方法，可以在早期实现软件的一个完整的功能并且验证这个功能。
- 测试较高层模块时，低层处理采用存根模块替代，这并不能够反映实际情况，因而测试并不充分和完善。为了解决这个问题，测试人员有两种选择：①把许多测试推迟到用真实模块代替了存根程序以后再进行；②从层次系统的底部向上组装软件。

二、自底向上集成

- 从软件结构最低层模块开始组装和测试，当测试到较高层模块时，所需的下层模块均已具备，故不再需要桩模块。

① 把低层模块组合成实现某个特定的软件子功能的族；

② 写一个驱动程序(用于测试的控制程序)，协调测试数据的输入和输出；

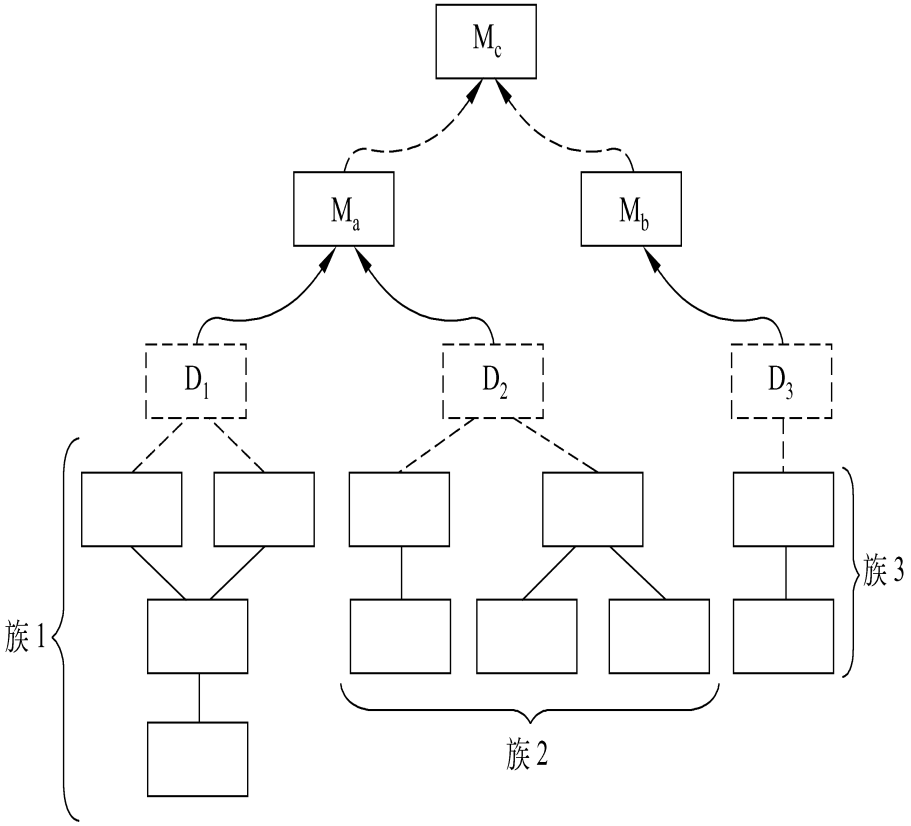
③ 对由模块组成的子功能族进行测试；

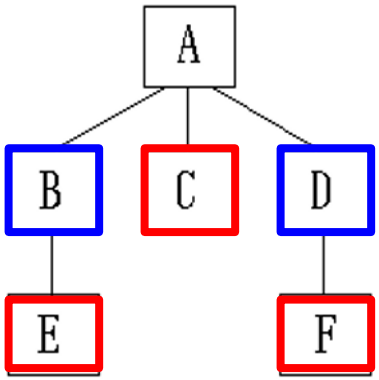
④ 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族。

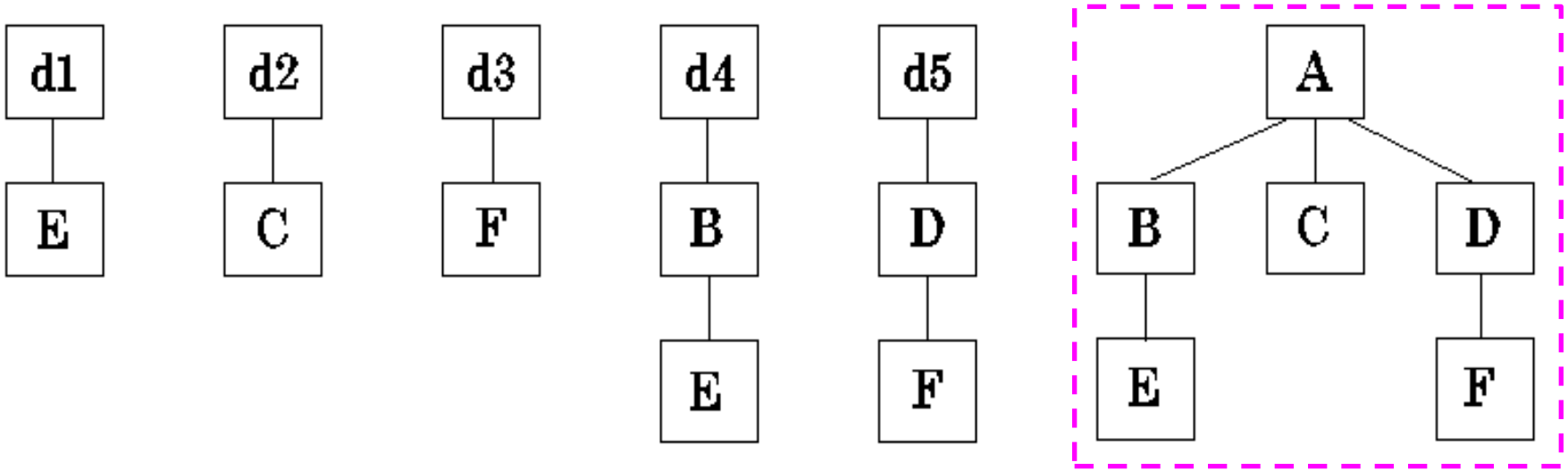
上述第②～④步实质上构成了一个循环。

7.4 集成测试

- 首先把模块组合成族1、族2和族3，使用驱动程序(图中用虚线方框表示)对每个子功能族进行测试。
- 族1和族2中的模块附属于模块M_a，去掉驱动程序D_1和D_2，把这两个族直接同M_a连接起来。
- 类似地在和模块M_b结合之前去掉族3的驱动程序D_3。最终M_a和M_b这两个模块都与模块M_c结合起来







自底向上结合测试过程

三、不同集成测试策略的比较

- 自顶向下测试方法的主要优点是不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。
- 自顶向下测试方法的主要缺点是需要存根程序，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力。
- 自底向上测试方法的优缺点与上述自顶向下测试方法的优缺点刚好相反。

7.4 集成测试

- 纯粹自顶向下或纯粹自底向上的策略可能都不实用，人们在实践中创造出许多混合策略。

(1) **改进的自顶向下测试方法**。基本上使用自顶向下的测试方法，但是早期使用自底向上的方法测试软件中的少数关键模块。一般的自顶向下方法所具有的优点在这种方法中也都有，缺点是测试关键模块时需要驱动程序。

(2) **混合法**。对软件结构中较上层使用的自顶向下方法与对软件结构中较下层使用的自底向上方法相结合。这种方法兼有两种方法的优点和缺点，当被测试的软件中关键模块比较多时，这种混合法可能是最好的折衷方法。

四、回归测试

- 集成测试过程中，每当一个新模块结合进来时，程序就发生了变化。在集成测试的范畴中，**回归测试**指重新执行已经做过的测试的某个子集，以保证上述这些变化没有带来非预期的副作用。
- **回归测试**集包括下述3类不同的测试用例。

(1) 检测软件全部功能的代表性测试用例。

(2) 专门针对可能受修改影响的软件功能的附加测试。

(3) 针对被修改过的软件成分的测试。

7.5 确认测试

- **确认测试**也称验收测试，目标是**验证**软件的有效性
- **软件有效性**的一个简单定义是：如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的
- 需求分析阶段产生的软件需求规格说明书，准确地描述了用户对软件的合理期望，因此是软件有效性的标准，也是进行确认测试的基础。

一、确认测试的范围

- 必须有**用户积极参与**。用户应参与设计测试方案，使用用户界面输入测试数据并分析评价测试的输出结果。

- **通常使用黑盒测试法。**通过测试和调试要保证软件能满足所有功能和性能要求，文档资料准确完整。此外，还应该保证软件能满足其他预定的要求(如安全性、可移植性等)

- 具有代表性和典型性
- 寻求系统设计和功能设计的弱点
- 既有正确输入也有错误或异常输入
- 考虑用户实际的诸多使用场景

测试用例文档

标识符：1007

测试项：记事本程序的文件菜单栏——文件 / 退出菜单的功能测试

测试环境：Windows 7 Professional 中文版

前置条件：无

操作步骤：

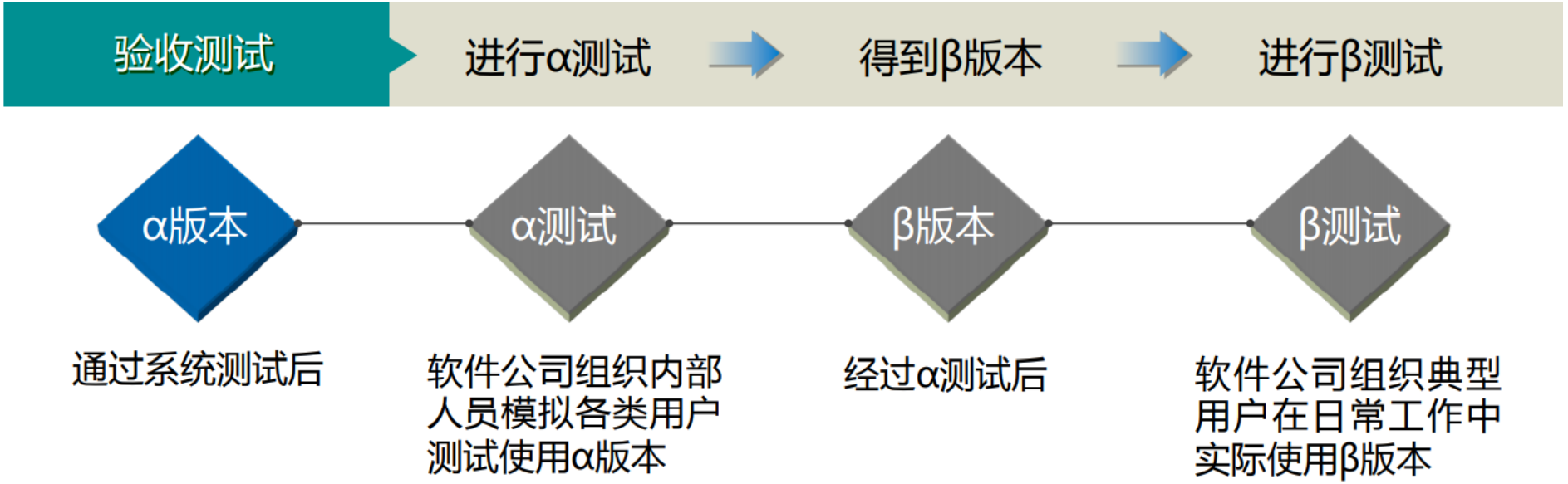
1. 打开记事本程序
2. 输入一些字符
3. 鼠标单击菜单“文件→退出”。

输入数据	期望输出	实际结果
空串	系统正常退出，无提示信息	
A	系统提示“是否将更改保存到无标题（或指定文件名）？”单击“保存”，系统将打开保存 / 另存窗口；单击“不保存”，系统不保存文件并退出；单击“取消”系统将返回记事本窗口。	

结论：☐ 通过　☐ 不通过　　测试人：　　测试日期：

二、Alpha和Beta测试

- 如果一个软件是为许多客户开发的，那么绝大多数软件开发商都使用被称为Alpha测试和Beta测试的过程，来发现那些看起来只有最终用户才能发现的错误。
- Alpha测试由用户在开发者的场所进行，且在开发者“指导”下进行。开发者负责记录发现的错误和使用中遇到的问题。
- Alpha测试是在受控的环境中进行的。
- Beta测试由软件的最终用户们在一个或多个客户场所进行。与Alpha测试不同，开发者通常不在Beta测试的现场。
- Beta测试是软件在开发者不能控制的环境中的“真实”应用。


验收测试
进行α测试
得到β版本
进行β测试
α版本
α测试
β版本
β测试
通过系统测试后
软件公司组织内部人员模拟各类用户测试使用α版本
经过α测试后
软件公司组织典型用户在日常工作中实际使用β版本

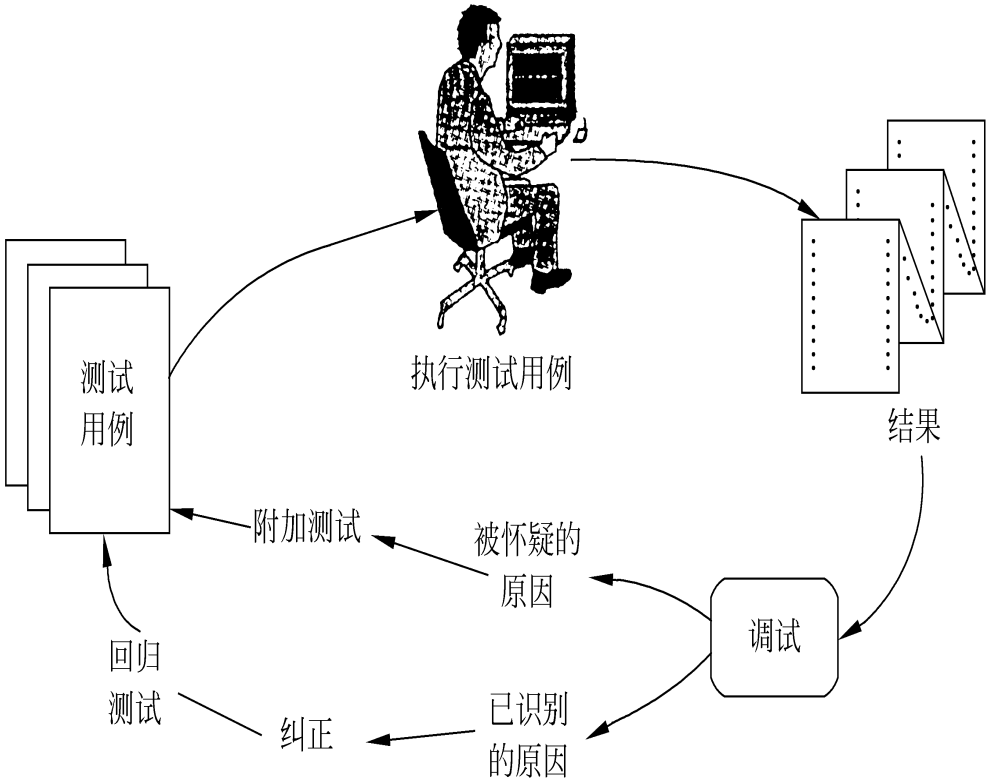
验收测试的流程

7.8 调试

口调试（也称为纠错）作为成功测试的后果出现，即调试是在测试发现错误之后排除错误的过程。

- 调试不是测试。
- 调试过程从执行一个测试用例开始，评估测试结果，如果发现实际结果与预期结果不一致，这种不一致就是一个症状，它表明在软件中存在着隐藏的问题。调试过程试图找出产生症状的原因，以便改正错误。

7.8 调试



调试过程

- 调试会有以下两种结果之一：

①找到了问题的原因并把问题改正和排除掉了；

②没找出问题的原因。调试人员可猜想一个原因，并设计测试用例来验证这个假设，重复此过程直至找到原因并改正了错误