# 软件工程概论10

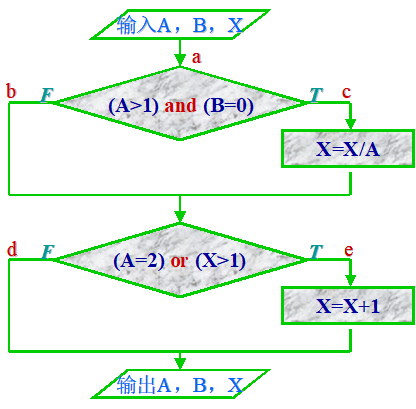
1. 十、软件测试方法
   1. 1.软件测试的目的和原则
      1. 基于不同的立场，存在着两种完全不同的测试目的。
         1. 从用户的角度出发，普遍希望通过软件测试检验软件中隐藏的错误和缺陷，以考虑是否可以接受该产品。
         2. 从软件开发者的角度出发，则希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。
      2. 软件测试目的提出了以下观点。
         1. 测试是程序的执行过程，目的在于发现错误。
         2. 好的测试用例在于能发现至今未发现的错误。
         3. 成功的测试是发现了至今未发现的错误的测试。
      3. 软件测试的原则如下：
         1. （1）应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。
         2. （2）测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。
         3. （3）程序员应避免检查自己的程序。
         4. （4）在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。
         5. （5）充分注意测试中的群集现象。把Pareto原理应用于软件测试。Pareto原理：测试发现的错误中的80%很可能是由程序中20%的模块造成的。
         6. （6）严格执行测试计划，排除测试的随意性。
         7. （7）应当对每一个测试结果作全面检查。
         8. （8）妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。
      4. 机器测试与人工测试
         1. 机器测试：在设定的测试数据上执行被测程序的过程。又称动态测试。
         2. 人工测试：采用人工方法进行，目的在于检查程序的静态结构，找出编译不能发现的错误。
      5. 人工测试的分类
         1. 代码审查：以小组会的形式，发现程序在结构、功能、编码风格等方面存在的问题。可查出30%~70%的错误
         2. 走查：以小组会的形式进行，把测试数据“输入”到被测程序，并在纸上跟踪监视程序的执行情况，让人代替机器沿着程序的逻辑走一遍。
         3. 桌前检查：设计模块时，程序员自己检查。
      6. 机器测试的分类
         1. 黑盒测试
            1. 黑盒测试是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。
            2. 黑盒穷举测试：对所有输入数据的各种可能值的排列组合都进行测试，来检查程序是否都能产生正确的输出。实际上这是不可能的。
         2. 白盒测试
            1. 白盒测试是对软件的过程性细节做细致的检查。
            2. 这一方法是把测试对象看做一个打开的盒子或透明的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。
            3. 通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。
            4. 因此，白盒测试又称为结构测试或逻辑驱动测试
            5. 白盒测试主要是对程序模块进行检查：

对程序模块的所有独立的执行路径至少测试一次；

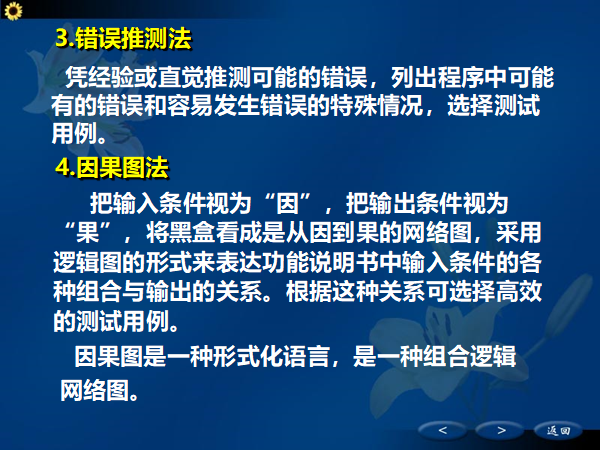
对所有的逻辑判定，取“真”与取“假”的两种情况都能至少测试一次；

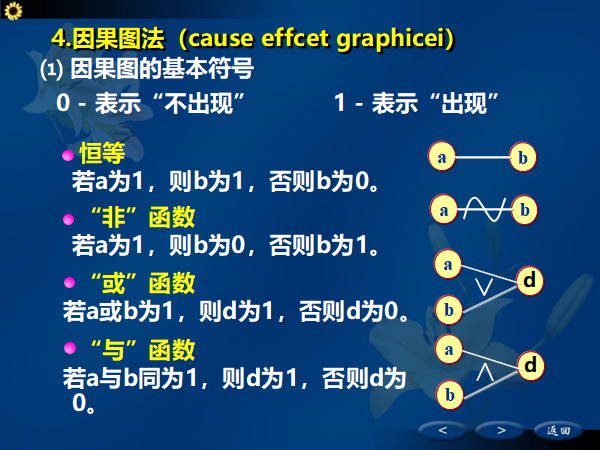
在循环的边界和运行界限内执行循环体；测试内部数据结构的有效性等。

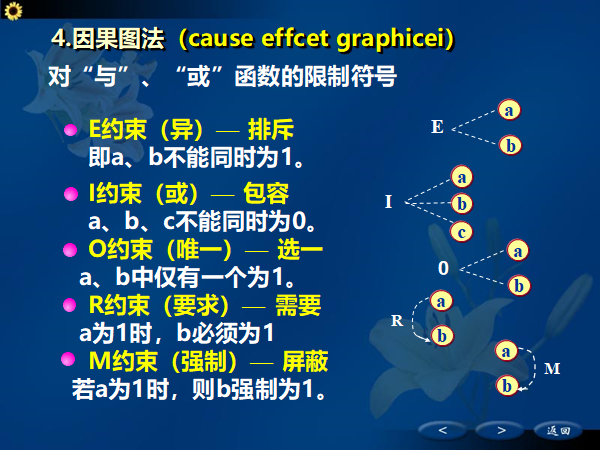
* + - * 1. 白盒穷举测试：对每条通路都应在每种可能的输入数据下执行一次。实际上这是不可能的。
  1. 2.白盒测试的测试用例设计
     1. 逻辑覆盖是以程序内部的逻辑结构为基础的设计测试用例的技术，它属于白盒测试。
     2. 由于覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖、判定覆盖、判定—条件覆盖、条件组合覆盖、路径覆盖。
     3. 最少测试用例覆盖最多的条件

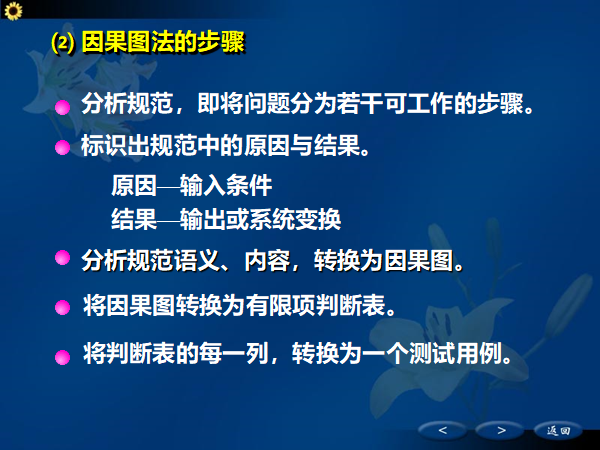


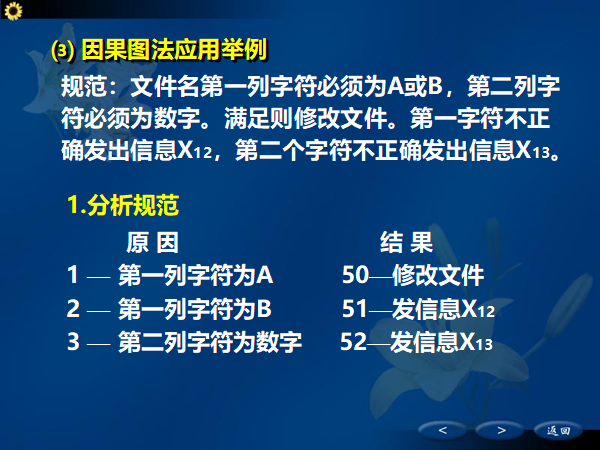
* + 1. 1.语句覆盖
       1. 语句覆盖就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。
       2. 在图例中，正好所有的可执行语句都在路径 ace上，所以选择路径ace设计测试用例，就可以覆盖所有的可执行语句。
    2. 2.判定覆盖
       1. 判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。判定覆盖又称为分支覆盖。(TT、FF)或者（TF、FT）
    3. 3.条件覆盖
       1. 条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。
    4. 4.判定—条件覆盖
       1. 所谓判定-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断本身的所有可能判断结果至少执行一次。
    5. 5.条件组合覆盖
       1. 条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。
    6. 6.路径覆盖
       1. 路径测试是设计足够的测试用例，覆盖程序中所有可能的路径。若仍以最初的图为例，则可以选择如下的一组测试用例，覆盖该程序段的全部路径。
    7. 基本路径覆盖
       1. 基本路径测试是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。
       2. 1. 由过程描述导出控制流图
       3. 2. 由过程描述导出控制流图 ：V（G）=（区域数）=（判定结点数）+1
       4. 3. 确定线性无关的基本路径集
       5. 4. 准备测试用例，确保基本路径集中的每一条路径的执行
  1. 3.黑盒测试的测试用例设计
     1. 1.等价类划分
        1. 等价类划分是一种典型的黑盒测试方法，也是一种非常实用的重要测试方法,它是用来解决如何选择适当的子集，使其尽可能多地发现错误。
        2. 使用这一方法设计测试用例要经历划分等价类（列出等价类表）和选取测试用例两步。
        3. 等价类的划分有两种不同的情况:
           1. （1）有效等价类：是指对于程序的规格说明来说，是合理的、有意义的输入数据构成的集合。利用它，可以检验程序是否实现了规格说明预先规定的功能和性能。
           2. （2）无效等价类：是指对于程序的规格说明来说，是不合理的、无意义的输入数据构成的集合。程序员主要利用这一类测试用例检查程序中功能和性能的实现是否有不符合规格说明要求的地方。
           3. 在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。
     2. 2.边界值分析
        1. 1. 边界值分析方法的考虑
           1. 边界值分析也是一种黑盒测试方法，是对等价类划分方法的补充。人们从长期的测试工作经验中得知，大量的错误是发生在输入或输出范围的边界上，而不是在输入范围的内部。这里所说的边界是指，相当于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。
        2. 2. 选择测试用例的原则
           1. (1)如果输入数据规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。 例如，若输入值的范围是“−1.0～1.0”，则可选取“−1.0”，“1.0”，“−1.001”，“1.001”作为测试输入数据。
           2. (2) 如果输入数据规定了值的个数，则用最大个数、最小个数、比最大个数多1、比最小个数少1的数作为测试数据。例如，一个输入文件有1～255个记录，设计测试用例时则可以分别设计有1个记录、255个记录以及0个记录和256个记录的输入文件。
           3. （3）根据规格说明的每个输出数据，使用前面的原则(1)。
           4. （4）根据规格说明的每个输出数据，使用前面的原则(2)。
           5. （5）如果程序的规格说明给出的输入域或输出域是有序集合（如有序表，顺序文件等），则应选取集合的第一个元素和最后一个元素作为测试用例。
           6. （6）如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。例如，如果程序中定义了一个数组，其元素下标的下界是0，上界是100，那么应选择达到这个数组下标边界的值，如0与100，作为测试用例。
           7. （7）分析规格说明，找出其他可能的边界条件。
        3. 3. 应用边界值分析方法设计测试用例的实例
     3. 3.因果图







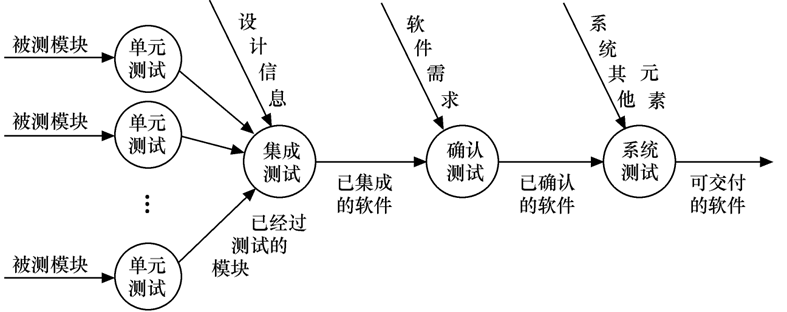








* 1. 4.软件测试的策略
     1. 通常软件测试过程按4个步骤进行，即单元测试、组装测试、确认测试和系统测试。如下图所示。



* + 1. 1.单元测试
       1. 单元测试（unit testing）又称模块测试，是针对软件设计的最小单位—程序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。
       2. 1．单元测试的内容
          1. 单元测试主要采用白盒测试方法设计测试用例，辅之以黑盒测试的测试用例，使之对任何合理的输入和不合理的输入，都能鉴别和响应。在单元测试中进行的测试工作如下图所示，需要在5个方面对被测模块进行检查。

（1）模块接口测试。

（2）局部数据结构测试。

（3）路径测试。

（4）错误处理测试。

（5）边界测试。

* + - 1. 2. 单元测试的步骤
         1. 模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块。这些辅助模块分为如下两种。

（1）驱动模块（driver）——相当于被测模块的主程序，它接收测试数据，并把这些数据传送给被测模块，最后再输出实测结果。

2）桩模块（stub）——也叫做存根模块，用以代替被测

模块调用的子模块。桩模块可以做少量的数据操作，不需

要把子模块所有功能都带进来，但不允许什么事情也不

做。

* + 1. 2.组装测试
       1. 组装测试（integrated testing）也叫做集成测试或联合测试。通常，在单元测试的基础上，需要将所有模块按照设计要求组装成为系统，把模块组装为系统的方式有两种：一次性组装方式（big bang）和增值式组装方式。
       2. 1．一次性组装方式
          1. 它是一种非增值式组装方式，也叫做整体拼装。使用这种方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。
       3. 2. 增值式组装方式
          1. 这种组装方式又称渐增式组装，首先是对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统，在组装的过程中边连接边测试，以发现连接过程中产生的问题。最后通过增值逐步组装成为要求的软件系统。增值组装有以下3种做法。
          2. （1）自顶向下的增值方式。这种组装方式是将模块按系统程序结构，沿控制层次自顶向下进行组装，其步骤如下：

① 以主模块为被测模块兼驱动模块，所有直属于主模块 的下属模块全部用桩模块代替，对主模块进行测试。

② 采用深度优先（如下图）或宽度优先的策略，逐步用实际模块替换已用过的桩模块，再用新的桩模块代替它们的直接下属模块，与已测试的模块或子系统组装成新的子系统。

③ 进行回归测试（即重新执行以前做过的全部测试或部分测试），排除组装过程中引入新的错误的可能。

④ 判断是否所有的模块都已组装到系统中，若是则结束测试，否则转到②去执行。

* + - * 1. （2）自底向上的增值方式。这种组装方式是从程序模块结构的最底层的模块开始组装和测试。因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块（包括子模块的所有下属模块）已经组装并测试完成，所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以由直接运行子模块得到。

自底向上增值的步骤如下：

① 由驱动模块控制最底层模块的并行测试；也可以把最底层模块组合成实现某一特定软件功能的簇，由驱动模块控制它进行测试。

② 用实际模块代替驱动模块，与它已测试的直属子模块组装成为子系统。

③ 为子系统配备驱动模块，进行新的测试。

④ 判断是否已组装到达主模块。若是则结束测试，否则执行②。

* + - * 1. （3）混合增值式测试。自顶向下增值的方式和自底向上增值的方式各有优缺点。自顶向下增值方式的缺点是需要建立桩模块。自底向上增值方式的缺点是“程序一直未能作为一个实体存在，直到最后一个模块加上去后才形成一个实体”。也就是说，在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。鉴于此，通常是把以上两种方式结合起来进行组装和测试。下面简单介绍3种常见的综合增值方式测试。

① 衍变的自顶向下的增值测试：它的基本思想是强化对输入/输出模块和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增值测试。

② 自底向上—自顶向下的增值测试：它首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统作自顶向下的组装与测试。

③ 回归测试：这种方式采取自顶向下的方式测试被修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否适配。

* + - 1. 3．组装测试的组织和实施
         1. 组装测试是一种正规测试过程，必须精心计划，并与单元测试的完成时间协调起来。
         2. 在制定测试计划时，应考虑如下因素：

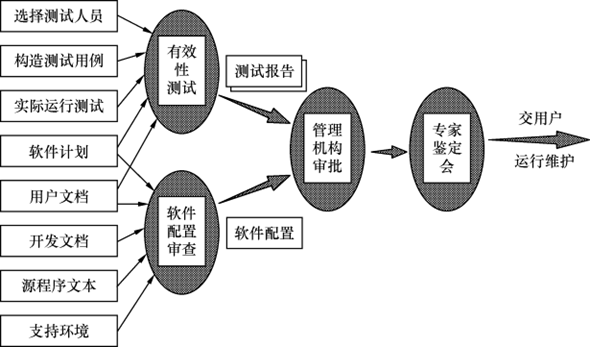
（1）采用何种系统组装方法进行组装测试。

（2）组装测试过程中连接各个模块的顺序。

（3）模块代码编制和测试进度是否与组装测试的顺序一致。

（4）测试过程中是否需要专门的硬件设备

* + 1. 3.确认测试
       1. 确认测试（validation testing）又称有效性测试。它的任务是验证软件的有效性，即验证软件的功能和性能及其他特性是否与用户的要求一致。在确认测试阶段需要做的工作如下图所示。



* + - 1. 1．进行有效性测试（黑盒测试）
      2. 2．软件配置复查
      3. 3．a测试和b测试
      4. 4．验收测试
      5. 5．确认测试的结果
    1. 4.系统测试
       1. 系统测试（system testing）是将通过确认测试的软件，作为整个计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据、人员等其他系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试。
       2. 系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统定义不符合或与之矛盾的地方。系统测试的测试用例应根据系统的需求分析说明书设计，并在实际使用环境下运行。
    2. 5.测试的类型
       1. （1）功能测试（function testing）：功能测试是在规定的一段时间内运行软件系统的所有功能，以验证这个软件系统有无严重错误。
       2. （2）回归测试（regression testing）：这种测试用于验证对软件修改后有没有引出新的错误，或者说，验证修改后的软件是否仍然满足系统的需求规格说明。
       3. （3）可靠性测试（reliability testing）：如果系统需求说明书中有对可靠性的要求，则需进行可靠性测试。通常使用平均失效间隔时间（MTBF）与因故障而停机的时间（MTTR）来度量系统的可靠性。
       4. （4）强度测试（stress testing）：也称压力测试，是要检查在系统运行环境恶劣的情况下，系统可以运行到何种程度的测试。因此，进行强度测试，需要提供非正常数量、频率或总量资源来运行系统。实际上，这是对软件的“超负荷”环境或临界环境的运行检验。
       5. （5）性能测试（performance testing）：是要检查系统是否满足在需求说明书中规定的性能。特别是对于实时系统或嵌入式系统，软件只满足要求的功能而达不到要求的性能是不可接受的，所以还需要进行性能测试。
       6. （6）恢复测试（recovery testing）：恢复测试是要证实在克服硬件故障（包括掉电、硬件或网络出错等）后，系统能否正常地继续进行工作，并不对系统造成任何损害。
       7. （7）启动/停止测试（startup/shutdown testing）：这类测试的目的是验证在机器启动及关机阶段，软件系统正确处理的能力。包括反复启动软件系统（例如，操作系统自举、网络的启动、应用程序的调用等），以及在尽可能多的情况下关机。
       8. （8）配置测试（configuration testing）：这类测试是要检查计算机系统内各个设备或各种资源之间的相互连接和功能分配中的错误。配置测试主要包括以下3种。
          1. ① 配置命令测试：验证全部配置命令的可操作性（有效 性）；特别对最大配置和最小配置要进行测试。软件配置和硬件配置都要测试。
          2. ② 循环配置测试：证明对每个设备物理与逻辑的、逻辑与功能的每次循环置换配置都能正常工作。
          3. ③ 修复测试：检查每种配置状态及哪个设备是坏的，并用自动的或手工的方式进行配置状态间的转换。
       9. （9）安全性测试（security testing）：检验在系统中已经存在的系统安全性和保密性措施是否发挥作用，有无漏洞。为此要了解破坏安全性的方法和工具，并设计一些模拟测试用例对系统进行测试，力图破坏系统的保护机构以进入系统。
       10. （10）可使用性测试（usability testing）：可使用性测试主要从使用的合理性、方便性等角度对软件系统进行检查，以发现人为因素或使用上的问题。
       11. （11）可支持性测试（supportability testing）：验证系统的支持策略对于公司与用户方面是否切实可行。它所采用的方法是试运行支持过程（如对有错部分打补丁的过程，热线界面等），对其结果进行质量分析，评审诊断工具、维护过程、内部维护文档；衡量修复一个明显错误所需的平均最少时间。还有一种常用的方法是，在发行前把产品交给用户，向用户提供支持服务的计划，从用户处得到对支持服务的反馈。
       12. （12）安装测试（installation testing）：安装测试的目的不是查找软件错误，而是查找安装错误。在安装软件系统时，会有多种选择。要分配和装入文件与程序库，布置适用的硬件配置，进行程序的连接。而安装测试是要查找出在这些安装过程中出现的错误。
       13. （13）互连测试（interoperability testing）：验证两个或多个不同的系统之间的互连性。这类测试对支持标准规格说明，或承诺支持与其他系统互连的软件系统有效。
       14. （14）兼容性测试（compatibility testing）：验证软件产品在不同版本之间的兼容性。有两类基本的兼容性测试：向下兼容和交错兼容。向下兼容测试是测试软件新版本，保留它早期版本的功能的情况；交错兼容测试是要验证共同存在的两个相关但不同的产品之间的兼容性。
       15. （15）容量测试（volume testing）：容量测试是要检验系统的能力最高能达到什么程度。
       16. （16）文档测试（documentation testing）：检查用户文档（如用户手册）的清晰性和精确性。用户文档中所使用的例子必须在测试中一一试过，确保叙述正确无误。