单元测试大揭密

FILEID：	VINCETEST_002
VERSION：	1.0
AUTHOR：	Vince
DATE：	2006-6-6
FILE STATE：	[ ] DRAFT [ ] MODIFY [√] RELEASE

VINCETEST

未 经 授 权 严 禁 扩 散

目 录

1 单元测试的重要性 3

1.1 一些错误的认识 3
1.2 测试的重要性 3
1.3 具有的优点 4

2 单元测试的基本理论 5

2.1 基本概念 5
2.2 测试的内容 5
2.3 测试的环境构成 7

3 测试方法与过程 9

3.1 用例设计 9
3.2 用例执行 11
3.3 测试优化和策略 11
3.4 测试评估 12
3.5 测试过程 12
3.6 测试实施 13

4 常用测试工具介绍： 15

1 单元测试的重要性

1.1 一些错误的认识

在实际的单元测试过程中总会有一些错误的认识左右着我们，使之成为单元测试最大的障碍，在此将其一一分析如下：

- 它太浪费时间了，现在要赶进度，时间上根本不允许，或者随便做做应付领导。
- 我是一个很棒的程序员，我写的代码肯定是没有问题的。
- 做单元测试太烦了，直接集成，到时有问题在集成测试时肯定能发现的，实在不行在系统测试总该能发现吧。
- 它仅仅是证明这些代码做了什么。

对于以上错误认识的产生归根结底还是由于对单元测试的理解还是不够，没有真正认识到单元测试的重要性。

1.2 测试的重要性

单元测试是软件测试的基础，因此单元测试的效果会直接影响到软件的后期测试，最终在很大程度上影响到产品的质量。从如下几个方面就可以看出单元测试的重要性在何处。

✧ 时间方面：如果认真的做好了单元测试，在系统集成联调时非常顺利，因此会节约很多时间，反之那些由于因为时间原因不做单元测试或随便做做的则在集成时总会遇到那些本应该在单元测试就能发现的问题，而这种问题在集成时遇到往往很难让开发人员预料到，最后在苦苦寻觅中才发现这是个很低级的错误而在悔恨自己时已经浪费了很多时间，这种时间上的浪费一点都不值得，正所谓得不偿失。

✧ 测试效果：根据以往的测试经验来看，单元测试的效果是非常明显的，首先它是测试阶段的基础，做好了单元测试，在做后期的集成测试和系统测试时就很顺利。其次在单元测试过程中能发现一些很深层次的问题，同时还会发现一些很容易发现而在集成测试和系统测试很难发现的问题。再次单元测试关注的范围也特殊，它不仅仅是证明这些代码做了什么，最重要的是代码是如何做的，是否做了它该做的事情而没有做不该做的事情。

✧ 测试成本：在单元测试时某些问题就很容易发现，如果在后期的测试中发现问题所花的成本将成倍数上升。比如在单元测试时发现 1 个问题需要 1 个小时，则在集成测试时发现该问题需要 2 个小时，在系统测试时发现则需要 3 个小时，同理还有定位问题和解决问题的费用也是成倍数上升的，这就是我们要尽可能早的排除尽可能多的 bug 来减少后期成本的因素之一。

✧ 产品质量：单元测试的好与坏直接影响到产品的质量，可能就是由于代码中的某一个小错误就导致了整个产品的质量降低一个指标，或者导致更严重的后果，如果我们做好了单元测试这种情况是可以完全避免的。

综上所述，单元测试是构筑产品质量的基石，我们不要因为节约单元测试的时间不做单元测试或随便做而让我们在后期浪费太多的不值得的时间，我们也不愿意因为由于节约那些时间导致开发出来的整个产品失败或重来！

1.3 具有的优点

1. 它是一种验证行为。

程序中的每一项功能都是测试来验证它的正确性，为以后的开发提供支缓。就算是开发后期，我们也可以轻松的增加功能或更改程序结构，而不用担心这个过程中会破坏重要的东西。而且它为代码的重构提供了保障，这样，我们就可以更自由的对程序进行改进。

2. 它是一种设计行为。

编写单元测试将使我们从调用者观察、思考，特别是先写测试（test-first），迫使我们把程序设计成易于调用和可测试的，即迫使我们解除软件中的耦合。另外还可以使编码人员在编码时产生预测试，将程序的缺陷降低到最小。

3. 它是一种编写文档的行为。

单元测试是一种无价的文档，它是展示函数或类如何使用的最佳文档。这份文档是可编译、可运行的，并且它保持最新，永远与代码同步。

4. 它具有回归性。

自动化的单元测试避免了代码出现回归，编写完成之后，可以随时随地的快速运行测试。

2 单元测试的基本理论

2.1 基本概念

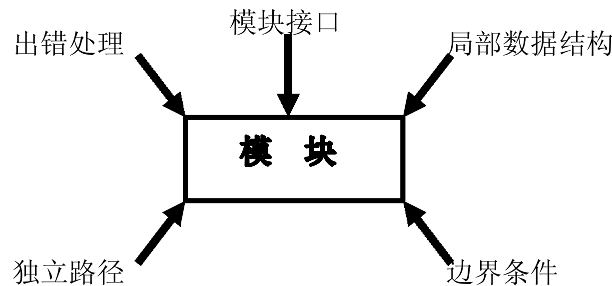
1. 单元测试：单元测试又称模块测试，属于白盒测试，是最小单位的测试。模块分为程序模块和功能模块。功能模块指实现了一个完整功能的模块（单元），一个完整的程序单元具备输入、加工和输出三个环节。而且每个程序单元都应该有正规的规格说明，使之对其输入、加工和输出的关系做出名明确的描述。
2. 测试驱动：驱动被测试模块正常运行起来的实体
3. 测试桩：代替被测模块调用的子模块的实体，该实体一般为桩函数。
4. 测试覆盖：评测测试过程中已经执行的代码的多少。
5. 覆盖率：代码的覆盖程度，一种度量方式。针对代码的测试覆盖率有许多种度量方式，定义如下：

- 语句覆盖（StatementCoverage）：也称为行覆盖（lin EC overage），段覆盖（segmentcoverage）和基本块覆盖（bASicblockcoverage）。它度量每一个可执行语句是否被执行到了。
- 判定覆盖（DecisionCoverage）：也被称为分支覆盖（branchcoverage），所有边界覆盖（all-edgescoverage），基本路径覆盖（basispathcoverage），判定路径覆盖（decision-decision-path 或 DDPtesting）。它度量是否每个 BOOL 型的表达式取值 true 和 false 在控制结构中都被测试到了。
- 条件覆盖（ConDItionCoverage）：它独立的度量每一个子表达式，报告每一个子表达式的结果的 true 或 false。这个度量和判定覆盖（decisioncoverage）相似，但是对控制流更敏感。不过，完全的条件覆盖并不能保证完全的判定覆盖。
- 路径覆盖（PathCoverage）：也称为断言覆盖（prEDIcatecoverage），它度量了是否函数的每一个可能的分支都被执行了。路径覆盖的一个好处是：需要彻底的测试。但有两个缺点：一是，路径是以分支的指数级别增加的，例如：一个函数包含 10 个 IF 语句，就有 1024 个路径要测试。如果加入一个 IF 语句，路径数就达到 2048；二是，许多路径不可能与执行的数据无关。
- 循环覆盖（LOOPCoverage）：这个度量报告你是否执行了每个循环体零次、只有一次还是多余一次（连续地）。对于 do-while 循环，循环覆盖报告你是否执行了每个循环体只有一次还是多余一次（连续地）。这个度量的有价值的方面是确定是否对于 while 循环和 for 循环执行了多于一次，这个信息在其它的覆盖率报告中是没有的。

2.2 测试的内容

单元测试的对象是软件设计的最小单位——模块或函数，单元测试的依据是详细设描述。测试者要根据详细设计说明书和源程序清单，了解模块的 I/O 条件和模块的逻辑结构。主要采用白盒测试的测试用例，辅之以黑盒测试的测试用例，使之对任何合理和不合理的输入都能鉴别和响应。要求对所有的局部和全局的数据结构、外部接口和程序代码的关键部分进行桌面检查和代码审查。在单元测试中，需要对下面 5 个方面的内容进行测试，也是构造

测试用例的基础。



1) 模块接口：测试模块的数据流。如果数据不能正确地输入和输出，就谈不上进行其他测试。因此，对于模块接口需要如下的测试项目：

- 调用所测模块时的输入参数与模块的形式参数在个数、属性、顺序上是否匹配；
- 所测模块调用子模块时，它输入个子模块的参数与子模块的形式参数在个数、属性、顺序上是否匹配；
- 是否修改了只做输入用的形式参数；
- 输出给标准函数的参数在个数、属性、顺序上是否匹配；
- 全局变量的定义在各模块中是否一致；
- 限制是否通过形式参数来传送。

2) 局部数据结构测试：模块的局部数据结构是最常见的错误来源，应设计测试用例以检查以下各种错误：

- 检查不正确或不一致的数据类型说明；
- 使用尚未赋值或尚未初始化的变量；
- 错误的初始值或错误的默认值；
- 变量名拼写错误或书写错误；
- 不一致的数据类型。

3) 路径测试：对基本执行路径和循环进行测试会发现大量的错误。根据白盒测试和黑盒测试用例设计方法设计测试用例。设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。

- 常见的不正确的计算有：
  - 运算的优先次序不正确或误解了运算的优先次序；
  - 运算的方式错误（运算的对象彼此在类型上不相容）；
  - 算法错误；
  - 初始化不正确；

  - 运算精度不够；
  - 表达式的符号表示不正确等。
- 常见的比较和控制流错误有：
  - 不同数据类型的比较；
  - 不正确的逻辑运算符或优先次序；
  - 因浮点运算精度问题而造成的两值比较不等；
  - 关系表达式中不正确的变量和比较符；
  - “差 1 错”，即不正确地多循环或少循环一次；
  - 错误的或不可能的循环终止条件；
  - 当遇到发散的迭代时不能终止循环；
  - 不适当地修改了循环变量等。

4) 错误处理测试：比较完善的模块设计要求能预见出错的条件，并设置适当的出错处理对策，以便在程序出错时，能对出错程序重新做安排，保证其逻辑上的正确性。这种出错处理也是模块功能的一部分。表明出错处理模块有错误或缺陷的情况有：

- 出错的描述难以理解；
- 出错的描述不足以对错误定位和确定出错的原因；
- 显示的错误与实际的错误不符；
- 对错误条件的处理不正确；
- 在对错误进行处理之前，错误条件已经引起系统的干预；
- 如果出错情况不予考虑，那么检查恢复正常后模块可否正常工作。

5) 边界测试：边界上出现错误上常见的。设计测试用例检查：

- 在 n 次循环的第 0 次、1 次、n 次是否有错误；
- 运算或判断中取最大最小值时是否有错误；
- 数据流、控制流中刚好等于、大于、小于确定的比较值时是否出现错误。

2.3 测试的环境构成

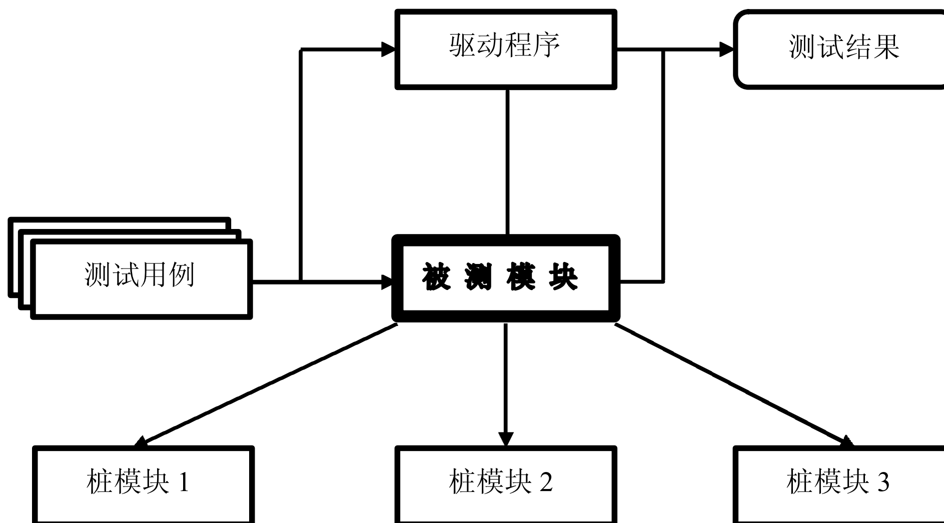
何时进行单元测试？单元测试在编码阶段进行。在源程序代码编制完成、经过评审和验证、确认没有语法错误之后，就可以开始进行单元测试的测试用例设计。要利用软件设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。

对于每一组输入，应该有预期的正确结果。在单元测试时，如果模块不是独立的程序，需要辅助测试模块，有两种辅助模块：

- 驱动模块（Driver）：所测模块的主程序。它接收测试数据，把这些数据传递给所测试模块，最后再输出测试结果。当被测试模块能完成一定功能时，也可以不要驱动模块。

- 桩模块（Stub）：用来代替所测模块调用的子模块。

被测试模块、驱动模块和桩模块共同构成了一个测试环境，如下图所示：



3 测试方法与过程

3.1 用例设计

1. 测试用例的组成（在单元测试中测试用例基本上由测试脚本组成）

- 用例运行前置条件
- 被测模块/单元所需环境（全局变量赋值或初始化实体）
- 启动测试驱动
- 设置桩
- 调用被测模块
- 设置预期输出条件判断
- 恢复环境（包括清除桩）

2. 测试用例的设计原则

- 一个好的测试用例在于能够发现至今没有发现的错误；
- 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；
- 在测试用例设计时，应当包含合理的输入条件和不合理的输入条件；
- 为系统运行起来而设计测试用例；
- 为正向测试而设计测试用例；
- 为逆向测试而设计测试用例；
- 为满足特殊需求而设计测试用例；
- 为代码覆盖而设计测试用例；

3. 用例设计方法

1) 规范（规格）导出发
2) 等价类划分法
3) 边界值分析法
4) 状态转移测试法
5) 分支测试法
6) 条件测试法
7) 数据定义－使用测试法（又名数据流测试法）
8) 内部边界值测试法
9) 错误猜测法

4. 特定的用例测试设计

1) 声明测试：检查模块中的所有变量是否被声明。经验表明，大量重要的错误都是由于变量没有被声明或没有被正确的声明而引起的。

2) 路径测试：要求模块中所有可能的路径都被执行一遍，属逻辑覆盖测试。

基本路径测试：由于实际中，一个模块中的路径可能非常多，由于时间和资源有限，不可能一一测试到。这就需要把测试所有可能路径的目标减少到测试足够多的路径，以获得对模块的信心。要测试的最小路径集就是基本测试路径集。基本测试路径集要保证：

- 每个确定语句的每一个方向要测试到；
- 每条语句最少执行一次。

3) 循环测试：重点检查循环的条件－判断部分以及边界条件。测试循环是一种特殊的路径测试，因为循环比其他语句都复杂一些。循环中错误的发生机会比其他代码构成部分多。因此，对于任何给定的循环测试应该包括测试下面每一条件的测试用例：

- 循环不执行；
- 执行一次循环；
- 执行两次循环；
- 反映执行典型的循环的执行次数；
- 如果有最大循环次数，最大循环次数减 1；
- 最大循环次数；
- 大于最大循环次数。

对于增量和减量不是 1 的 FOR 语句，要特别注意，因为程序员习惯于增量 1。

4) 循环嵌套：循环嵌套使逻辑的次数呈几何级数增长，设计测试嵌套循环的测试用例应该包括的测试条件有：

- 把外循环设置为最小值，并运行内循环所有可能的情况；
- 把内循环设置为最小值，并运行外循环所有可能的情况；
- 把所有的循环变量都设置为最小值运行；
- 把所有的循环变量都设置为最大值运行；
- 把外循环设置为最大值，并运行内循环所有可能的情况；
- 把内循环设置为最大值，并运行外循环所有可能的情况；

5) 边界值测试：指程序内部边界测试。检查确定代码在任何边界情况下都不会出差错。重点检查小于、等于和大于边界条件的情况。边界值测试是指专门设计用来测试当条件语句中引用的值处在边界或边界附近时系统反映的测试。被测试语句的最好的例子就是“IF-THEN…ELSE-ENDIF”部分。这样语句的例子如：

```
IF a <= 123 THEN
```

```
  b = 1
ELSE IF a >= 123 THEN
      b = 2
ELSE b = 3
END IF
```

上面例子中的边界值测试用例应该至少包括 a 的以下值：122，123，124。当 a=123 时，b=1 还是 2。（找出逻辑判断的矛盾）

6) 接口测试：检查模块的数据流（输入、输出）是否正确。检查输入的参数和声明的自变量的个数，数据类型和输入顺序是否一致。检查全局变量是否被正确的定义和使用等。

7) 确认测试：是否接受有效输入数据（操作），拒绝无效数据（操作）。

8) 事务测试：输入－>输出，错误处理。

3.2 用例执行

一般来说，做单元测试均采用的是商用的测试工具或自行开发的测试工具，用例的编写都是在测试工具上完成，测试用例都是一些测试脚本，都以文件的方式来保存，故其用例的执行过程主要是由测试工具根据所编写的具体的测试用例脚本来完成，这样对于用例的管理和执行也非常灵活。

在特定场合，比如某种压力测试或极限测试，对于测试执行过程时间很长时（几个小时以上），一般都预先编写好用例（确保用例无误），使用空闲机或非工作时间执行测试用例，这样操作起来较节约时间。

在用例的执行过程中务必注意如下事项：

- 程序的执行过程－－－便于构造发散用例
- 不要放过任何细节－－－这种细节可能就是问题

3.3 测试优化和策略

在测试的过程中为了提高测试效率和效果，不断的减少冗余劳动，也为后期的回归测试和测试管理带来很大的方便，不至于感到测试很混乱无序。因此我们要对测试用例和测试执行进行不断的优化，以测试策略为指导方针进行测试。

- 测试用例的优化

  测试用例的优化主要是指用例的合并、修改和删除，减少冗余的无价值的测试，其优化依据来源于测试后的测试数据分析和评估，其中测试覆盖也是用例优化的主要参考。

- 测试执行的优化

  测试执行的优化主要是指测试步骤的优化，减少测试人员的手工操作，因为太多的手工操作会导致测试人员很厌倦，直接影响测试效果，优化依据来源于测试总结。

- 测试策略

在测试过程中由于时间或资源的原因可能会使测试处于紧张的局面，在此情况下我们要采取一定的策略来解决此局面。策略来源于测试数据的分析，主要的方法是：为各模块制定测试优先级，其优先级的划分依据如下：

- ✧ 哪些是重点模块？
- ✧ 哪些程序是最复杂、最容易出错的？
- ✧ 哪些程序是相对独立，应当提前测试的？
- ✧ 哪些程序最容易扩散错误？
- ✧ 哪些程序是开发者最没有信心的？
- ✧ 80-20 原则：80％的缺陷聚集在 20％的模块中，经常出错的模块改错后还会经常出错，这种应该列入测试重点。

3.4 测试评估

单元测试完成以后，需要对单元测试的执行效果进行评估，主要从以下几方面进行：

1) 测试完备性评估，主要检查测试过程中是否已经执行了所有的测试用例，对新增的测试用例是否已及时更新测试方案等。
2) 代码覆盖率评估，主要是根据代码覆盖率工具提供的语句覆盖情况报告，检查是否达到方案中的要求，公司要求语句覆盖达到 100％。但很多情况下，第一轮测试用例执行完后是很难达到的，这时在评估过程中要对覆盖率进行分析，主要从以下方面来考虑：
   - 不可能的路径或条件
   - 不可达的或冗余的代码
   - 不充分的测试用例
3) 从覆盖的角度看，测试应该覆盖：
   - 功能覆盖
   - 输入域覆盖
   - 输出域覆盖
   - 函数交互覆盖
   - 代码执行覆盖

大多数有效的测试用例都来自于分析，而不是仅仅为了达到测试覆盖率目标而草率设计测试用例。千万不要误解测试覆盖，测试覆盖并不是我们最求的目的，它只是评价测试的一种方式，为测试提供指导和依据。

3.5 测试过程

1. 测试过程中各种人员的作用
   - 系统分析设计人员

   进行需求跟踪，确保系统需求的实现和更新。进行软件单元可测性分析，确定单元测试

的对象、范围和方法。

- 软件开发人员

负责编码和单元测试过程，完成单元测试计划、方案和报告。

- 软件测试人员

参与单元测试计划、方案和报告的评审，对单元测试的计划、设计和执行质量进行监控。根据实际情况，可选择参与由开发人员负责的代码检视、单元测试等活动。

- 配置管理人员

对代码及单元测试文档进行配置管理。

- 质量保证（QA）人员

参与编码与单元测试评审，对编码和单元测试过程进行审计。

2. 单元测试输入
    - 《软件需求规格说明书》
    - 《软件详细设计说明书》
    - 《软件编码与单元测试工作任务书》
    - 《软件集成测试计划》
    - 《软件集成测试方案》
    - 用户文档
3. 单元测试的输出
    - 《单元测试计划》
    - 《单元测试方案》
    - 《需求跟踪说明书》或需求跟踪记录
    - 代码静态检查记录
    - 《正规检视报告》
    - 问题记录
    - 问题跟踪和解决记录
    - 软件代码开发版本
    - 《单元测试报告》
    - 《软件编码与单元测试任务总结报告》

3.6 测试实施

1. 单元测试实施步骤

    1) 制定测试计划和测试方案（包括测试工具的选择）

2) 根据计划和方案及相关输入文档编写测试用例

3) 搭建测试环境

4) 执行测试

5) 记录和跟踪问题

6) 编写测试报告和总结报告

2. 单元测试实施遵循的原则

- 精心制定测试计划
- 严格评审测试计划
- 严格执行测试计划
- 系统分析测试结果并提交报告

4 常用测试工具介绍

常用的 C 语言单元测试工具介绍如下：

1. VcTester

 1) 简介

 VcTester 是与 VC（注：Visual C++及 Visual Studio 开发套件是微软发布的产品）配套使用的新一代单元测试工具，分共享版与商用版两大系列，其主要功能包括：脚本化测试驱动（包括修改变量与调用函数）、脚本桩、支持持续集成测试、测试覆盖率统计（仅商用版本）、生成测试报告（仅商用版本）、测试消息编辑器（仅商用版本）等。

 2) 功能特性

 - 脚本化测试驱动
 - 脚本桩
 - 在线测试
 - 即时调测
 - 测试工程管理

 3) 价格

 共享版免费

 4) 相关网站

 www.ezTester.com

2. C++Test

 1) 简介

 C++Test 是一个功能强大的自动化 C/C++单元级测试工具，可以自动测试任何 C/C++函数、类，自动生成测试用例、测试驱动函数或桩函数，在自动化的环境下极其容易快速的将单元级的测试覆盖率达到 100%。

 2) 功能特性

 - 即时测试类/函数
 - 支持极端编程模式下的代码测试
 - 自动建立类/函数的测试驱动程序和桩调用
 - 自动建立和执行类/函数的测试用例
 - 提供快速加入和执行说明和功能性测试的框架
 - 执行自动回归测试
 - 执行部件测试(COM)

3) 价格

不详

4) 相关网站

http://www.parasoft.com