# 如何在 mantis 中增加需要统计的字段

作者：吴佳祥

**摘　要：**如何在 mantis 中增加一些需要统计的字段，导出报表进行统计。

**关键词：**mantis；自定义段

在使用测试缺陷管理工具 mantis 时，可能根据实际情况统计一些需要的数据。这就要添加字定义自段，并添加一些代码，导出之后很容易统计。我举个实例来进行操作。假如说要统计 BUG 错误类型（用户接口、环境、赋值、性能、标准、语法、功能、接口、联编打包、文档）、错误阶段（需求分析、概要设计、详细设计、编码、集成测试、系统测试、交付、维护）。

操作方法如下：

## 一、 添加自定义段

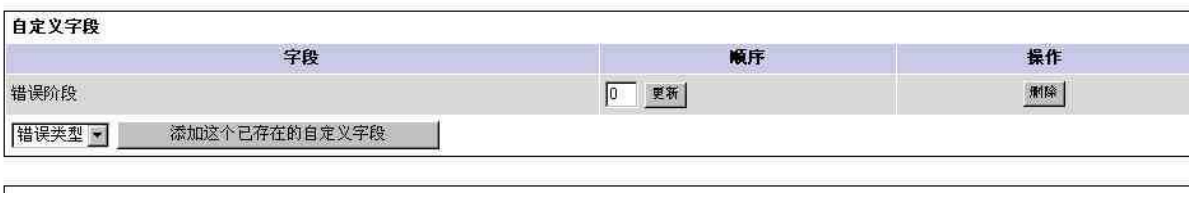
管理员进入系统----点管理----点自定义段管理，输入字段名“错误阶段”----点新建自定义段。 如下图：


对字段进行设置，（枚举用|隔开）。如下图：


同种方法，可添加错误类型的字段。

## 二、 把自定义段加入项目中

点管理-点项目管理-选要添加字段的项目进入，找到自定义段项，可增减项目中的自定义字段。如下图：



三、 **验证字段是否成功加入项目**

点报告问题，查看是否有自定义的字段。如果存在，说明添加成功。如下图：


四、 **添加代码**（代码来源于网络）

找到 Core 里面的 custom_function_api.php 文件，将以下代码添加进去，添加到末尾的 ?> 前面即可

```
# --------------------
function
```

```
custom_function_override_get_columns_to_view(  $p_columns_target
=
    COLUMNS_TARGET_VIEW_PAGE ) {
    $t_columns = array();

    if ( $p_columns_target == COLUMNS_TARGET_CSV_PAGE ) {
    $t_columns[] = 'id'; // localized: 'id',
    $t_columns[] = 'project_id'; // 'email_project'
    $t_columns[] = 'reporter_id'; // 'reporter'
    $t_columns[] = 'handler_id'; // 'assigned_to'
    $t_columns[] = 'priority'; // 'priority'
    $t_columns[] = 'severity'; // 'severity'
    $t_columns[] = 'reproducibility'; // 'reproducibility'
    $t_columns[] = 'version'; // 'version'
    $t_columns[] = 'projection'; // 'projection'
    $t_columns[] = 'category'; // 'category'
    $t_columns[] = 'date_submitted'; // 'date_submitted'
    $t_columns[] = 'eta'; // 'eta'
    $t_columns[] = 'os'; // 'os'
    $t_columns[] = 'os_build'; // 'os_version'
    $t_columns[] = 'platform'; // 'platform'
    $t_columns[] = 'view_state'; // 'view_status'
    $t_columns[] = 'last_updated'; // 'last_update'
    $t_columns[] = 'summary'; // 'summary'
    $t_columns[] = 'status'; // 'status'
    $t_columns[] = 'resolution'; // 'resolution'
    $t_columns[] = 'fixed_in_version'; // 'fixed_in_version';
```

```
    # if viewing only one Project, Add all custom fields linked to this
project
    if ( helper_get_current_project() != ALL_PROJECTS ) {
    $t_custom_fields                                                =
custom_field_get_linked_ids(helper_get_current_project());

    foreach( $t_custom_fields as $t_field_id ) {
    $t_desc = custom_field_get_definition( $t_field_id );

    $t_columns[] = 'custom_' . $t_desc['name'];
    }
    }



    if ( OFF == config_get( 'enable_relationship' ) ) {
    $t_columns[] = 'duplicate_id'; // 'duplicate_id'
    }
    } else {
    $t_columns[] = 'selection';

    if ( $p_columns_target == COLUMNS_TARGET_VIEW_PAGE ) {
    $t_columns[] = 'edit';
    }

    $t_columns[] = 'priority';
    $t_columns[] = 'id';

    $t_enable_sponsorship = config_get( 'enable_sponsorship' );
    if ( ON == $t_enable_sponsorship ) {
```

```
        $t_columns[] = 'sponsorship';
        }

        $t_columns[] = 'bugnotes_count';

        $t_show_attachments=
config_get( 'show_attachment_indicator' );
        if ( ON == $t_show_attachments ) {
        $t_columns[] = 'attachment';
        }

        $t_columns[] = 'reporter_id';

        $t_columns[] = 'category';
        $t_columns[] = 'severity';
        $t_columns[] = 'status';
        $t_columns[] = 'last_updated';
        $t_columns[] = 'summary';
        }

        return $t_columns;
        }
```

## 五、 导出统计

这个统计是不能在 mantis 统计里直接出现的，要 CVS 导出。如下图：


查看问题 (1 - 22 / 22) [ 打印报告 ] [ CSV 导出 ]

| | | P | 编号 | # | 报告人 |
|---|---|---|---|---|---|
| ☐ | ✎ | | 0000038 | | administrator |
| ☐ | ✎ | ⌃⌃ | 0000037 | | administrator |

导进 Excel，里面错误类型，错误阶段都很容易统计出来。如下图

| S | T | U | V | W |
|---|---|---|---|---|
| 状态 | 完成度 | 修正此问题 | 错误类型 | 错误阶段 |
| 新建 | 未处理 | | 功能 | 需求分析 |
| 新建 | 未处理 | | 功能 | 编码/单体 |
| 新建 | 未处理 | | 用户接口 | |
| 新建 | 未处理 | | 功能 | |
| 新建 | 未处理 | | 功能 | |
| 新建 | 未处理 | | 功能 | |

# 性能测试项目总结

## —虚拟数据的准备

作者：裴明哲

**摘　要：**本文主要是面向性能测试的工程师，从实际项目中总结经验、教训，并且提出一些改善的建议，希望大家能在以后的性能测试的项目中吸取和借鉴，本文尤其在性能测试的前期数据准备方面给出了解决方案。

**关键词：**测试用例；性能测试；测试流程

### 项目介绍

该项目为两年前的一个项目，目前该系统的性能在一定的条件下速度极慢，当用户量达到一定程度时，整个程序会无法响应，所以需要对该项目进行性能测试，找到系统的瓶颈，为以后的系统升级做充分的准备。

### 项目延期的原因

XXX项目已经结束,在整个项目的测试过程中遇到了不少困难，由于各种原因导致项目延期，其中虚拟数据的准备是其中一个重要环节。

由于第一次做这样的项目，前期的数据准备不合理，项目测试设计难免存在着一些问题，在项目进行过程中遇到了种种问题，比如说工具的使用问题，在测试执行过程中为了准备虚拟数据，设计SQL脚本就延误了项目大部分的时间，出现的问题如下：

1. 前期需求理解不充分（需求理解时间太短），测试计划中给予需要理解的时间不足，所以如果对于一些功能点理解的不充分，这样就会将问题遗留到测试执行过程中，然后你会在测试执行中把问题提出来，与客户交流，这必然导致项目的延期。

2. 工具使用不熟练，事实上，如果对一个项目进行性能测试，

人员配置方面一定要有(至少一位)有性能测试经验的工程师来参与项目，这样可以降低项目的风险，由于该项目组有经验的工程师出差，所以只好由我们无经验的人员在自学或培训的情况下参与该项目的测试工作，在这样的情况下，我们会有一段熟悉学习测试工具的时间，显然自己学习理解过程当中会有很多问题，未解决的问题就会带到项目执行过程当中，而且在项目执行过程当中也会遇到不预期的错误，问题解决就会耗去一部分时间。

3. 最重要的一个环节，就是虚拟数据的准备，当然第一次做这样的项目在这方面并没有太多的经验，在测试执行中，才进行 SQL 语句的设计，数据的添加，在测试执行过程中，SQL 语句的设计就会用掉大部分时间。

**改善建议**

据以上问题，结合在这个项目中的经验，给出以下几点性能测试方面改善的建议，在大家以后进行性能测试的项目中避免这样的问题再次发生，使项目能够按照进度顺利的完成，达到预期的测试目的。

1. 需求理解方面，建议针对一些准备测试的功能点一定要理解充分，若发现问题，尽早与测试负责人或客户进行沟通并解决问题，避免将问题遗留到测试执行中去解决；在做测试计划时，要根据项目的大小以及客户给予的工作量合理安排需求理解的时间。

2. 工具使用方面，大家可以在平时业余时间进行学习，一个项目结束与另一个项目启动之间一般会有一段时间空闲，在这段时间可以去学习测试工具，并且要具有针对性的学习，不要盲目的学习（既学 LR，也学 QTP），尽量学懂一门后再学一门（达到基本会用该工具做项目），不要急于求成，在学习的过程中将学到的东西做一个学习笔记，方便你以后的查阅；测试组也可以适当的为员工做培训。

3. 虚拟数据的准备方面，在性能测试的项目中一般都会分为两种情况：

1) 固定用户量，数据库中数据量的递增，测试该功能点的性能；

2) 固定数据库数据量，用户数量递增，测试该功能点的性能，其实，这样就会分为两个测试用例（当然这不是全场景测试），譬如：

TestCase 01：20 用户在线，共有 200 个项目，定制显示默认（20 条/页，8 列/页），在数据库中其他项目记录数不断增加的情况下，系统的相应时间。

TestCase 02：用例描述： 固定数据库问题数为 20 万条，使用的项目问题卡数量 1000，自定义显示（20 条/页，8 列/页），浏览用户不断增加的情况下响应时间；

情况一（建议后）：前提条件：TestCase 01 与 TestCase 02 浏览的数据（问题卡 ）访问的是数据库同一张表；

1> 当你添加 5 万条数据，执行 TestCase 01，记录结果；然后再添加 5 万条数据，数据量就是 10 万，再执行 TestCase 01，记录结果；

2> 当添加的数据等于 20 万的时候，也就是 TestCase 02 的固定使用数据量，你就可以将 TestCase 02 的测试场景设计的用户量设置为 10、15 等等依次执行完 TestCase 02 这个用例，记录结果；

这样就是全局考虑，简单的说就是考虑每个测试用例中数据会使用数据库中的哪一张表，也许会有很多测试用例使用同一个数据库表，这样就要考虑到表中的数据量递增到多大的时候，执行哪一个测试用例，不是一味的按照一个测试用例，添加虚拟数据，一直到执行完该用例后，等执行下一个用例时，将该表的数据全部删除，再继续添加该用例要求的数据量。

情况二（建议前）：如果你一直添加数据执行完 TestCase 01，你在执行 TestCase 02 的时候数据库该表的数据量已经到达 50 万，TestCase 02 的固定使用数据量为 20 万，你还需要写一个 SQL 脚本去删除 30 万的数据量，才能到达 TestCase 02 执行时需要的数据量，所以这样就比情况一多了一步写 SQL 脚本删除数据的过程，其实并不是多了一步，其中：T2（情况二的时间）＝T（书写调试 sql 的时

间）＋T（执行删除 30 万数据量）；对于 30 万的数据量的删除时间也会是比较长的一段时间，所以说改善后的方案 T1（情况一时间）<T2（情况二的时间）。

总之，当然这是对于两个用例访问的都是同一个表，如果多个用例都会访问到一个表时，就可以全局考虑，在测试设计的时候就应该将这些考虑进来，将执行步骤或者执行方案写成文档，来指导测试执行，这样就不会在测试执行添加数据时盲目的按照测试编号顺序的去添加数据，执行测试用例，不考虑测试用例之间数据准备时候的联系，而是可以根据设计好的测试大纲和文档进行有效的测试执行，测试数据的准备可以按照如下流程进行：一、书写测试用例；二、根据测试用例整体考虑，设计出 SQL 脚本，并且可以做一个文档，记录 SQL 编号与测试用例之间的对应关系，同时要写出执行测试用例的顺序，其中这个顺序并不是按照测试用例里面的编号顺序执行，三、测试执行，按整体设计出的"添加数据及测试流程"进行执行。下面是整个顺序的设计测试流程以及几个文档的模板，大家可以见解一下。

一、 书写测试用例：

Testcase 01 用例描述：固定用户为 30 人，页面显示（50 条/页，4 列/页）。每统计用户使用的日报数据不断增加的情况下响应时间。

Testcase 02 用例描述：固定日报数据为 30 万条。每统计用户使用数据固定为 1 万条。页面显示（50 条/页，4 列/页）。浏览用户不断增加的情况下的响应时间。

根据以上测试用例，就可以设计出添加虚拟数据得 SQL 脚本了，然后根据以下文档将设计的 SQL 脚本与测试用例对应记录，然后再设计出测试流程。

二、 书写 SQL 脚本

如：declare

```
p number;
q number;
```

```
begin
p :=15424;
q :=0;
while p<=15503 loop
select qmtools.PRPROBLEMSEQ.nextval into q from dual;
INSERT INTO PRPROBLEM ( PRPROBLEMID, PRRECORDID, PRPBTYPEID, PRBDESCRIBE, PRESENTER, SOLVE, STAFFID, FACTSOLVEDATE, FACTEFF, PRSTATEID, MODULEID, CONFIRMSTAFF, CONFIRMDATE, PREFLAG, MEMO, PREWORKLOAD,
POSITION, PBMOVEKIND, MILESTONEID, PBGRADE, PBPINCHEFF ) VALUES (
q, p, 42, 'ertre', 'pmz5', 'erteert'
, 5489, TO_Date( '12/16/2006 12:00:00 上午', 'MM/DD/YYYY HH:MI:SS AM'), 33, 2, 8903
, '                ', NULL, 0, 'ertret', 0, 'ert'
, '需求理解', 0, 'C', 3);
p :=p+1;
end loop;
end;
```

将该 SQL 脚本命名为 S1，记录到下表：

SQL脚本与测试用例对应表

项目编号: 项目名称: 项目软件经理: 测试负责人:

| 序号 | SQL脚本编号 | 用例编号 | 变量1 | 变量1描述 | 变量2 | 变量2描述 | 变量3 | 变量3描述 |
|---|---|---|---|---|---|---|---|---|
| 1 | S1 | Testcase01 | p | | q | | | |
| 2 | | Testcase02 | | | | | | |
| 3 | S2 | Testcase03 | | | | | | |
| | | | | | | | | |

该表主要为了使设计的 SQL 脚本和测试用例依次对应起来，而且将变量描述清楚，有利用设计测试流程，而且当执行这些 SQL 语句时会明白每个变量其中的含义。

三、 设计测试流程

根据以上脚本与测试用例对应表以及测试用例，当执行 SQL 脚本和 Testcase01 时，数据量达到 30 的时候，也就是 Testcase02 的固定数据库的数据量时，这时候，设置 Testcase02 的场景，就可以将 Testcase02 的所有情况执行完毕，也就是 Testcase02 的优先级最高，执行完后将执行状态打 √。所以说整个设计对于以后的执行是非常有好处的，正式开始执行测试用例的时候就可以根据以下设计的测试流程执行 SQL 脚本和测试用例。

**总结**

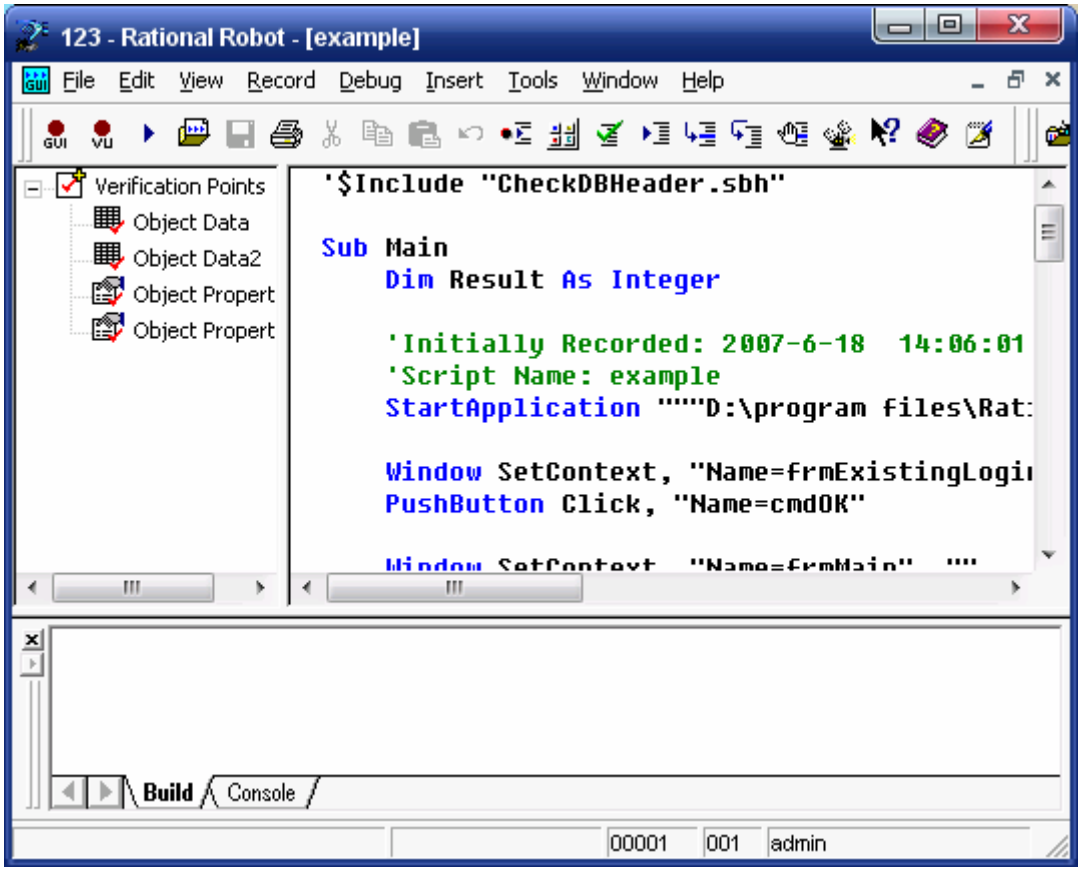
性能测试首先要将这些必要的设计在前期设计好，以免等到测试执行时候进行设计，这样就会延长项目的进度，同时也会造成不预期的风险，希望这篇文章能给大家一些好的借鉴，同时也希望大家能给该方法提出更好的意见和建议，提高我们的过程改善的质量。由于时间仓促，难免会有笔误，恳请大家批评、指正。

# Rational 在自动化测试中的应用

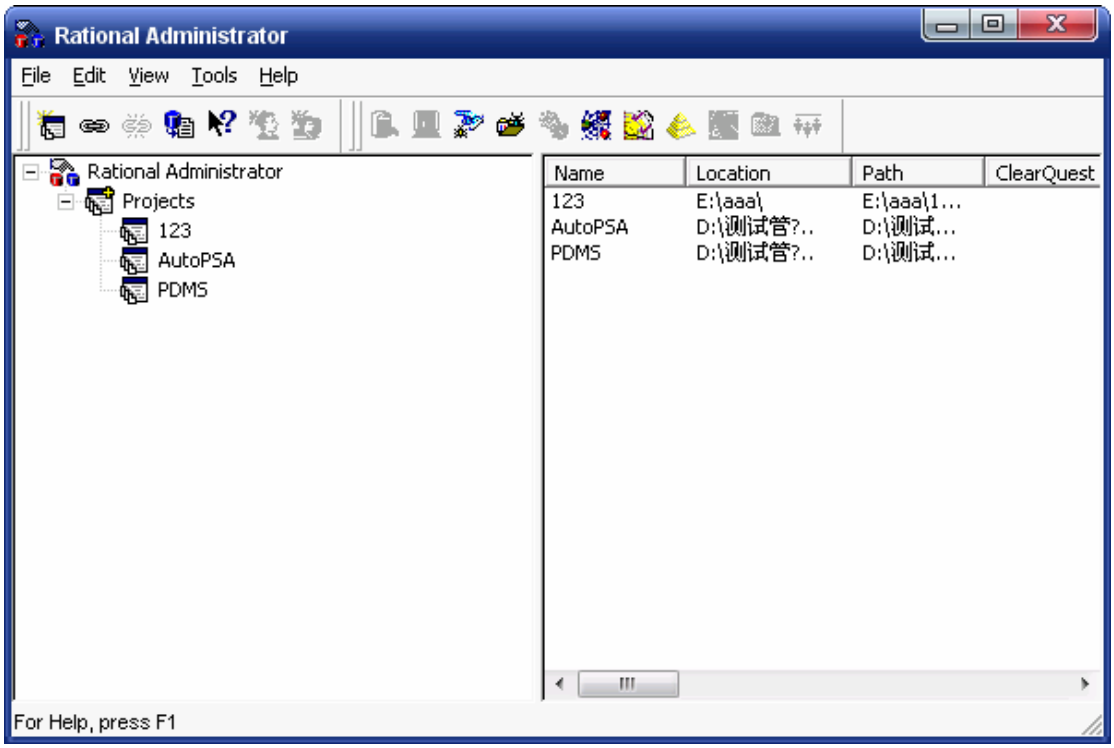
作者：刘英

## 成功的功能测试途径

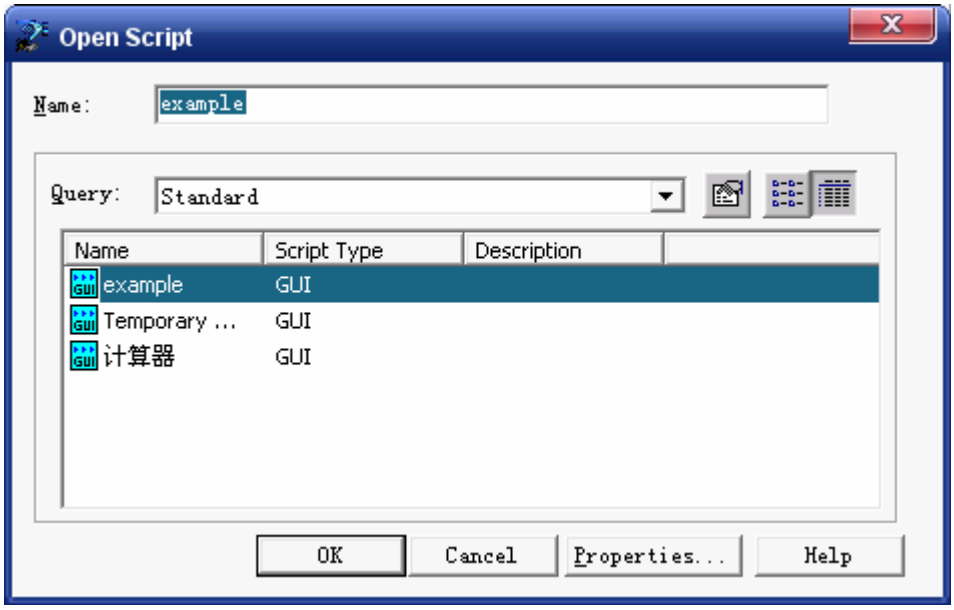
1. 安装Rational Robot，可通过Rational Suite Test Studio，Rational TeamTest or Rational Robot



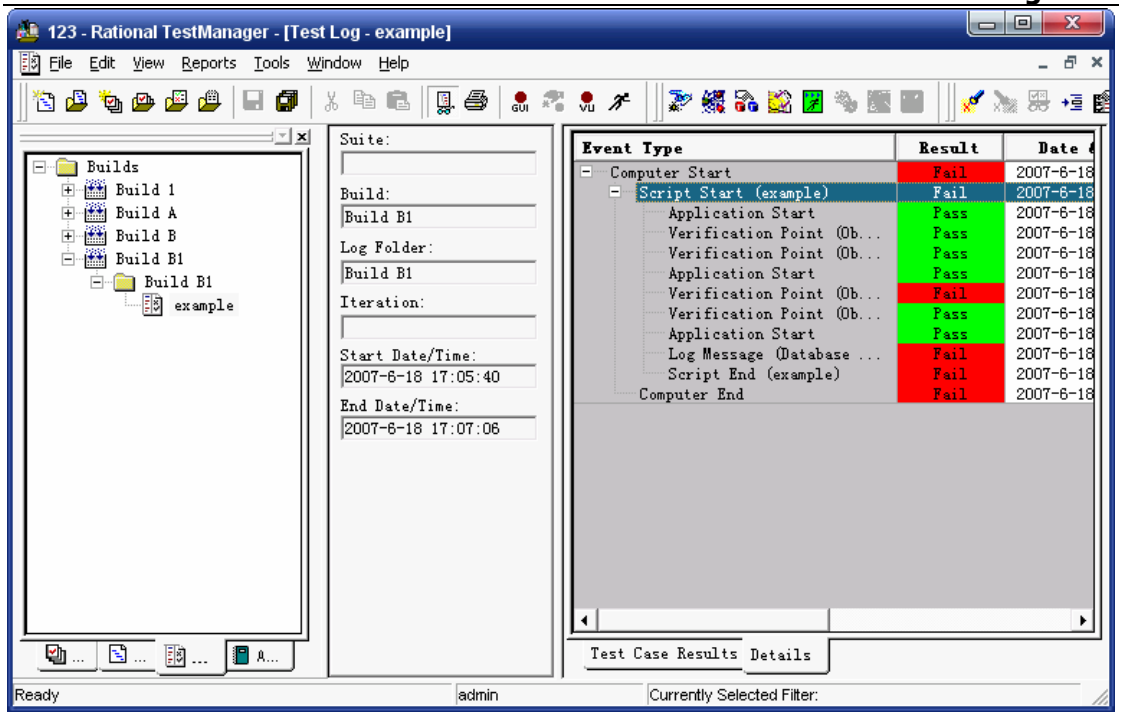

2. 通过Rational Administrator创建测试资产，来储存测试资产



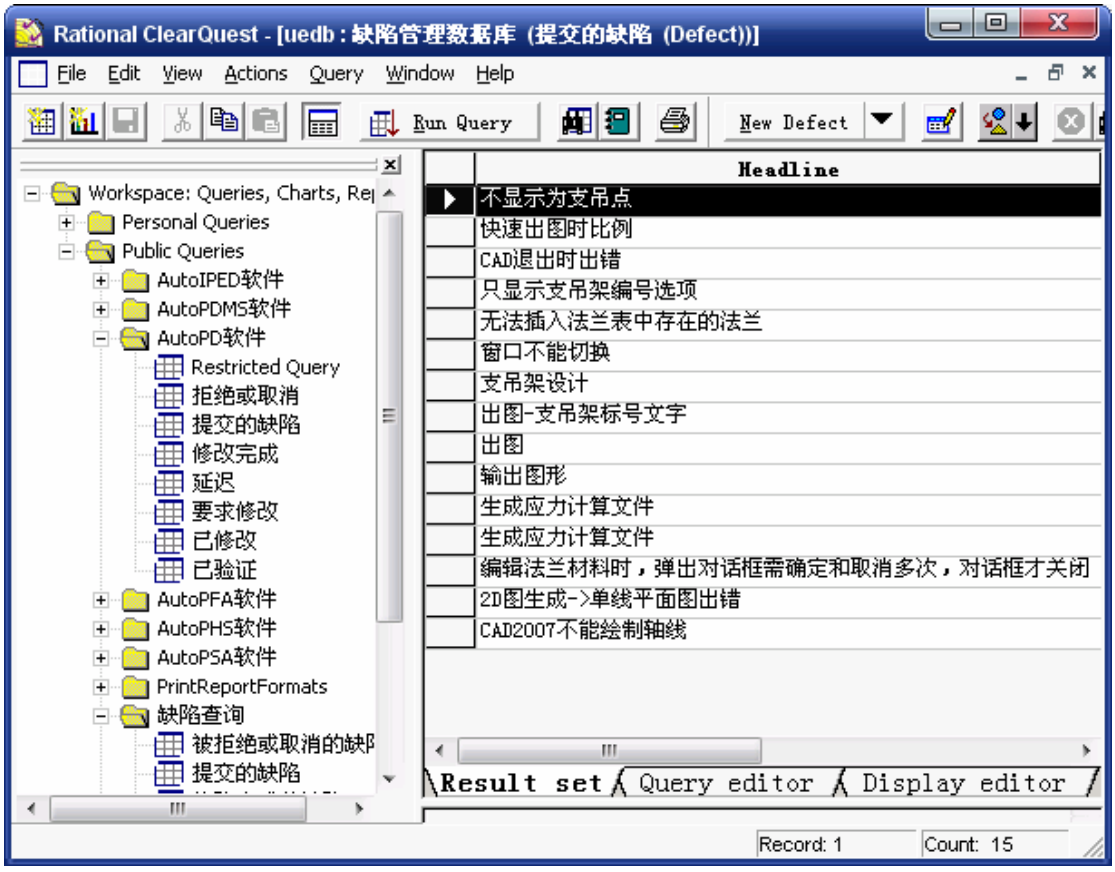

3. 通过Rational Robot来录制和回放测试脚本




4. 通过Rational Log Viewer 和 Comparators 来记录回放的测试结果

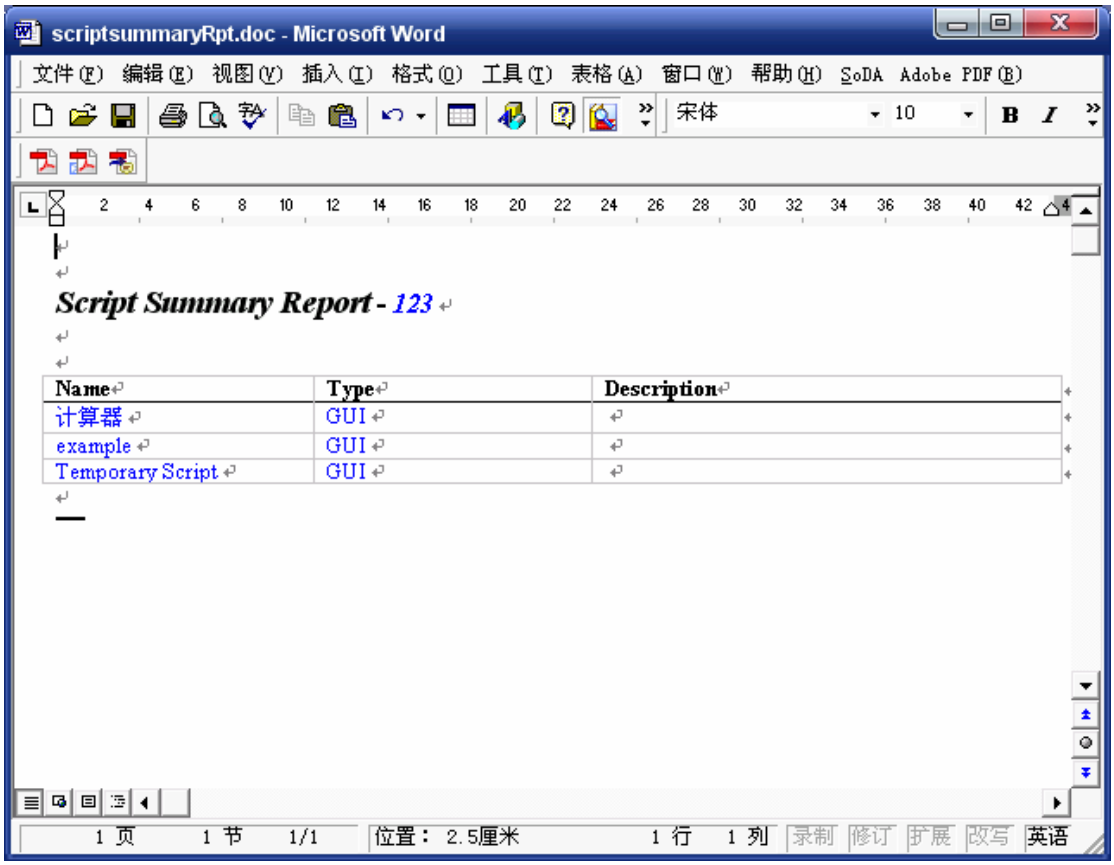



5. 通过Rational ClearQuest 来管理缺陷




6. 通过 TestManager、LogView 和 ClearQuest 的 report 来分析测

试结果




## 关于 Sample 的应用

本文通过一个 Sample 来介绍整个 Rational 在自动化测试中的应用，该 Sample 是 Rational 自带的例子----------Classic Online，是**通过 VB 开发的。**

Classic Online 是允许用户通过 Online 的目录和订购位置来浏览订购信息，当 Classic Online 完成，测试和成功配置，它将自动提供给顾客一个订购情况和订购完成的能力。

| **测试阶段** | Rational **的使用** |
|---|---|
| 测试计划------定义测试需求，计划和管理测试资产 | Rational Requisitepro<br>TestManager<br>See Example 1 |
| 测试开发-------录制，验证和编辑脚本 | Rational Robot<br>See Example 2、3、7 |

| | |
|---|---|
| 测试执行-------回放测试脚本 | Rational Robot<br>See Example 4 |
| 测试结果------检查和分析测试结果 | Rational LogViewer 和 Comparators<br>See Example 5 |
| 测试总结和分析--------运行测试报告来确认测试覆盖率和缺陷状态 | Rational TestManager 和 ClearQuest<br>See Example 6 |

开始前的准备：

- 安装 Rational Robot
- 安装 Classics Online sample application
- 连接 Sampl 的 Rational 管理
- 在每个阶段对每个任务都进行了描述

安装 Rational Robot：

| 安装 Rational Robot 的部分 | 参看部分 |
|---|---|
| Rational Robot package | 安装 Rational TeamTest 和 Rational Robot |
| Rational TeamTest package | 安装 Rational TeamTest 和 Rational Robot |
| Rational Suite TestStudio package | Rational Suite 安装引导 |

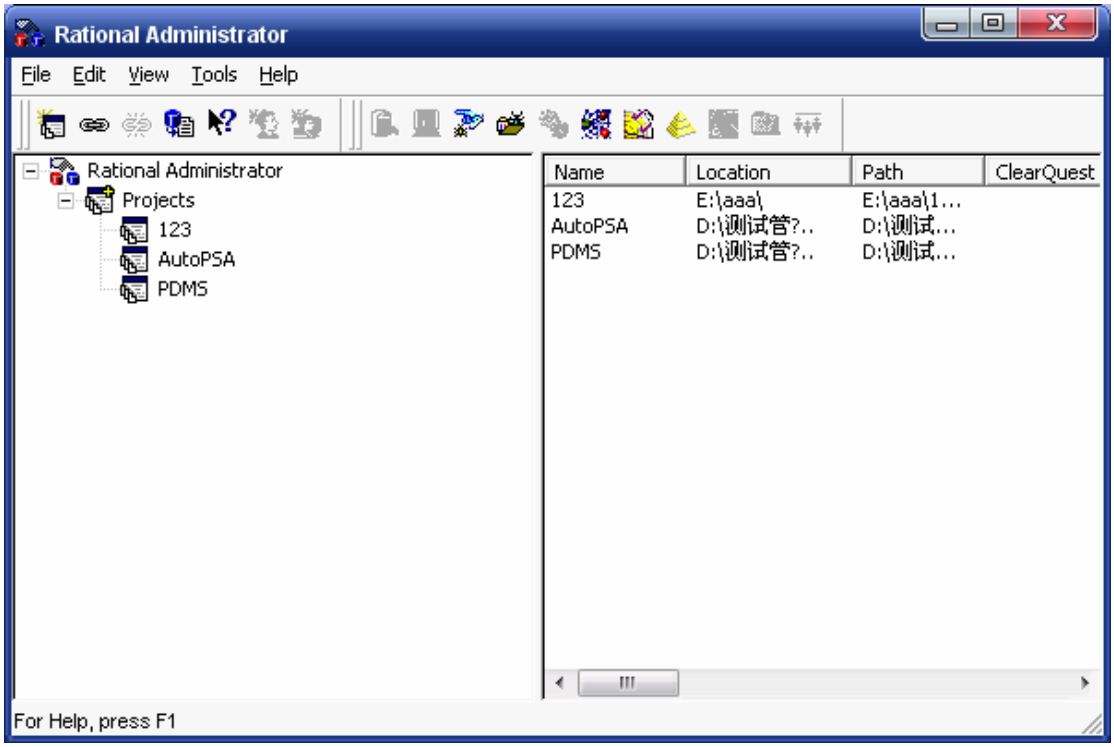
Sample 应用程序的安装

1. 连接到[开始]→[程序]→Rational Software→Rational Test→Setup Rational Test Sample.
2. 选择 Classics Online
3. 连接 Next，然后 Finsh

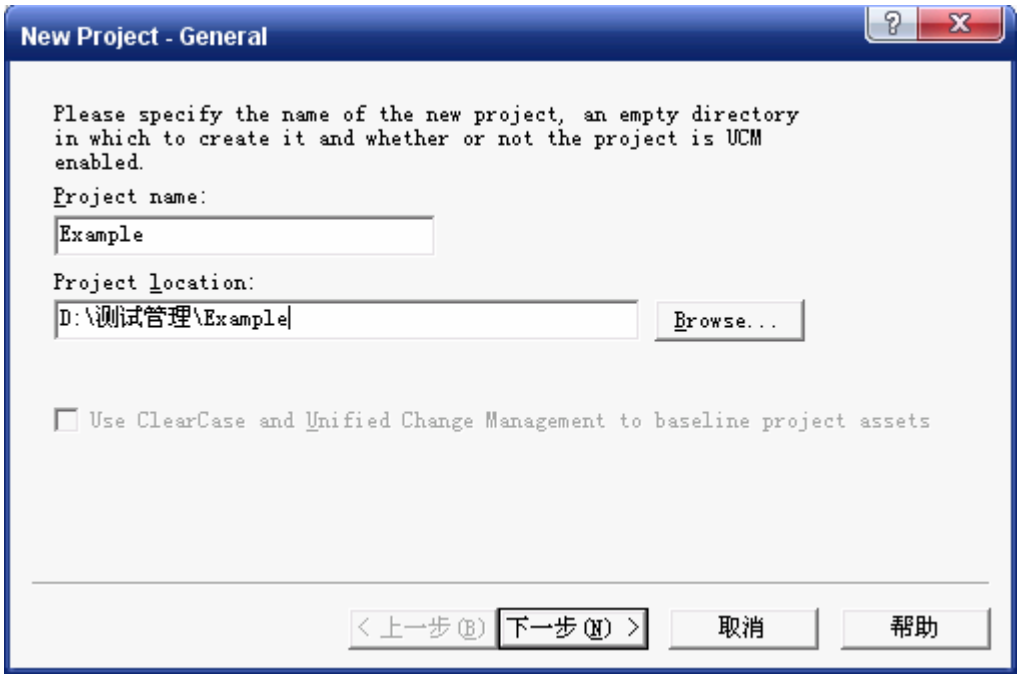
安装完成后，这个 Sample 将出现在[开始]→[程序]→Rational Test Samples.

## Sample 测试资产的建立

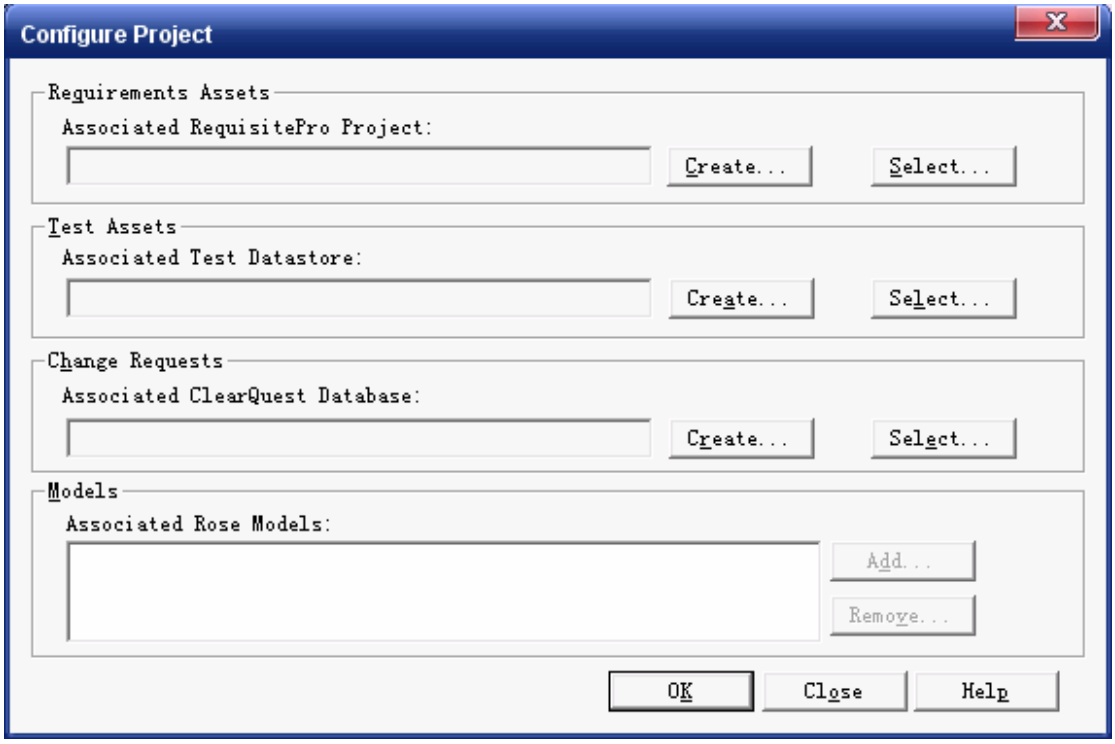
1. 连接[开始]→[程序]→ Rational Software→Rational Administrator



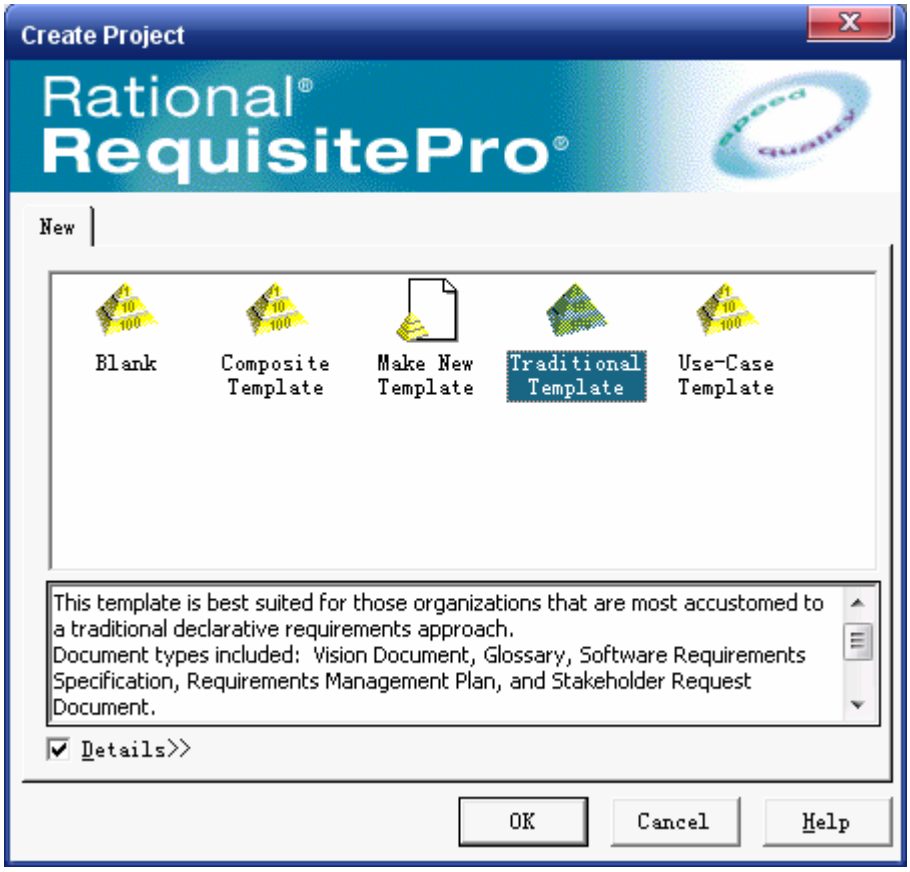

2. 选择 File→New Project 菜单命令，将弹出一个对话框，按要求输入相关内容。



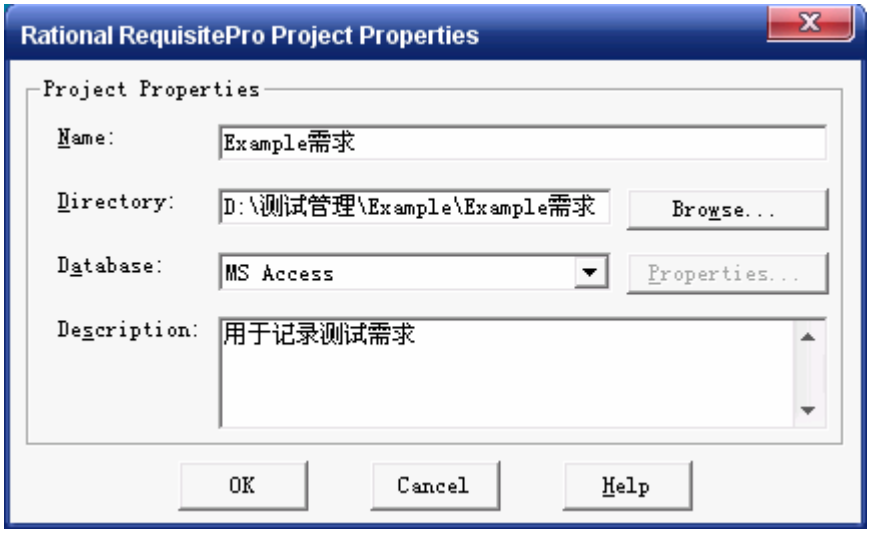
3. 单击“下一步”按扭，将出现设置本项目管理员口令的窗口，对该窗口可以不理会，继续后面操作，系统会弹出如下图所示窗口



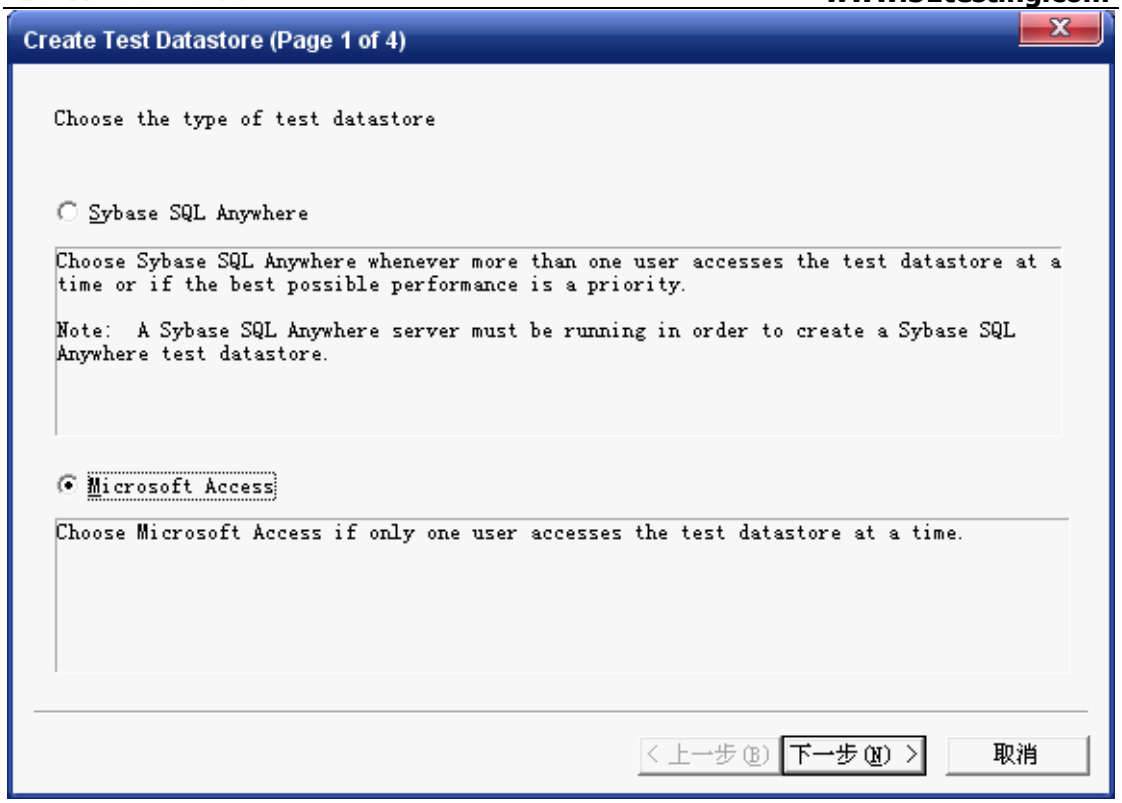

4. 在本项目中，希望能够关注对需求的跟踪，因此创建 RequisitePro 使用的数据库，方法是单击第一个 Create 按扭，将弹出如下图所示窗口，选择 Traditional Template（传统模板）



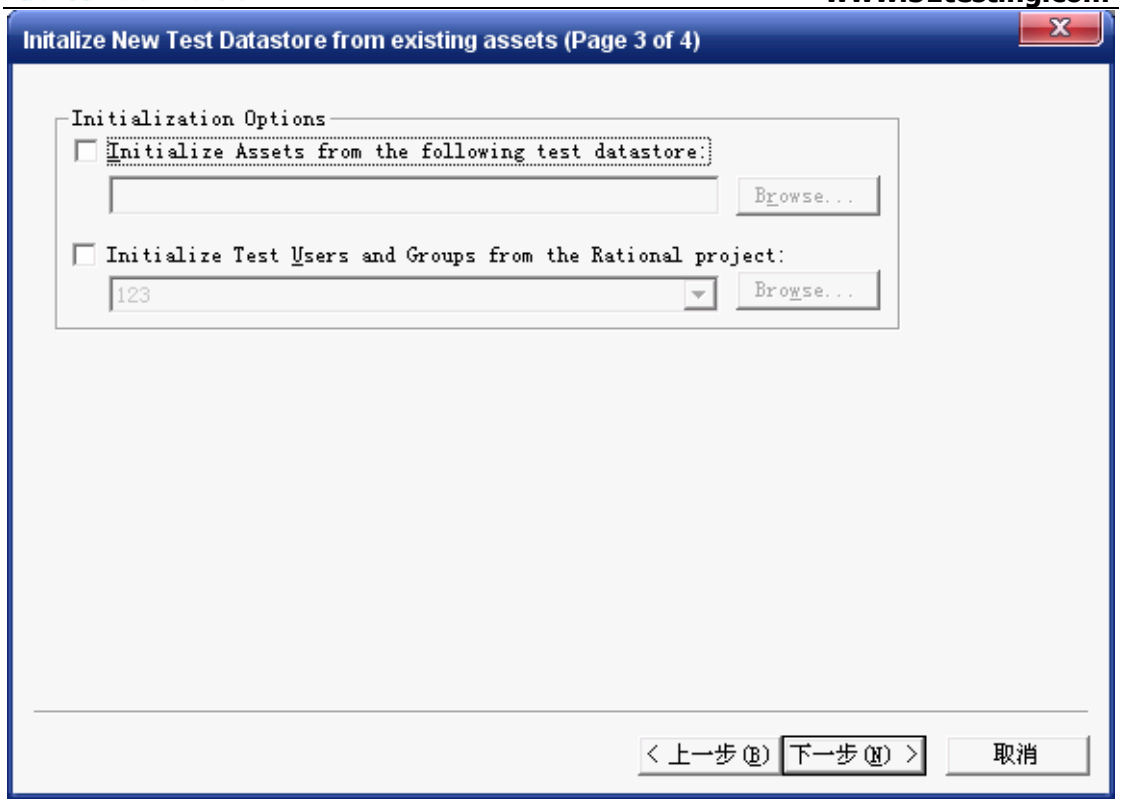

5. 确定需求库的名字和位置

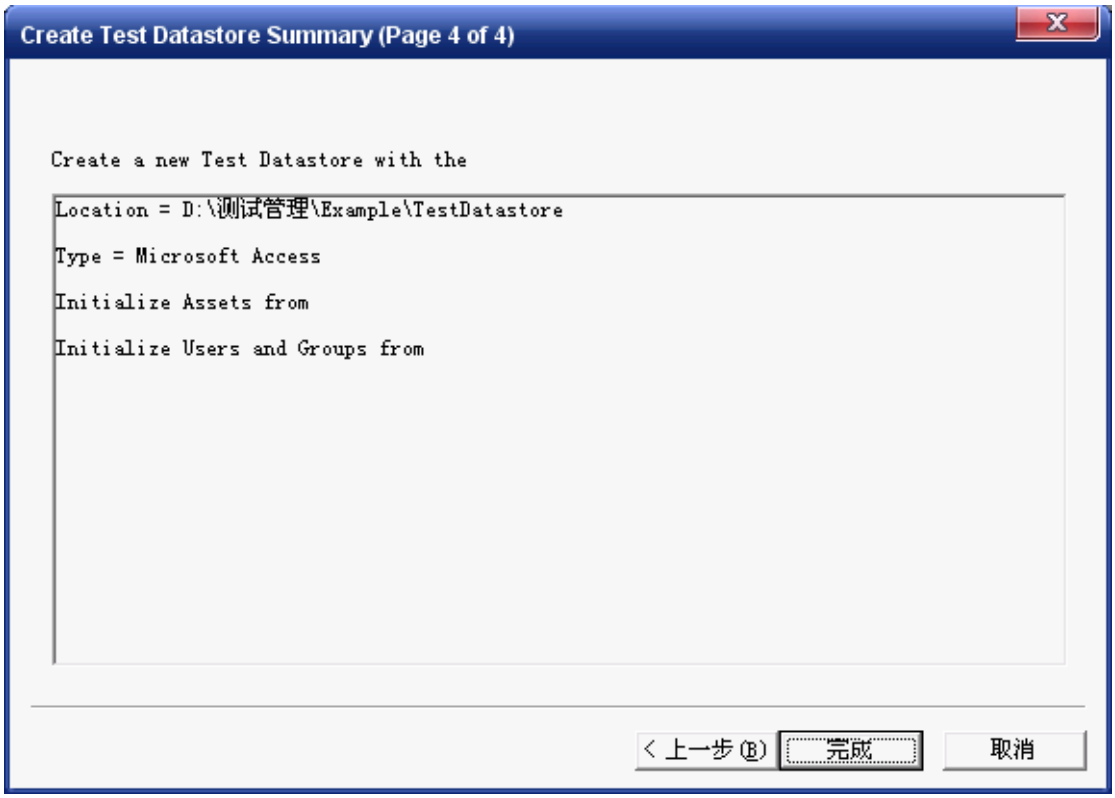



6. 单击第 4 步图中的第二个 Create 安扭，创建测试数据库

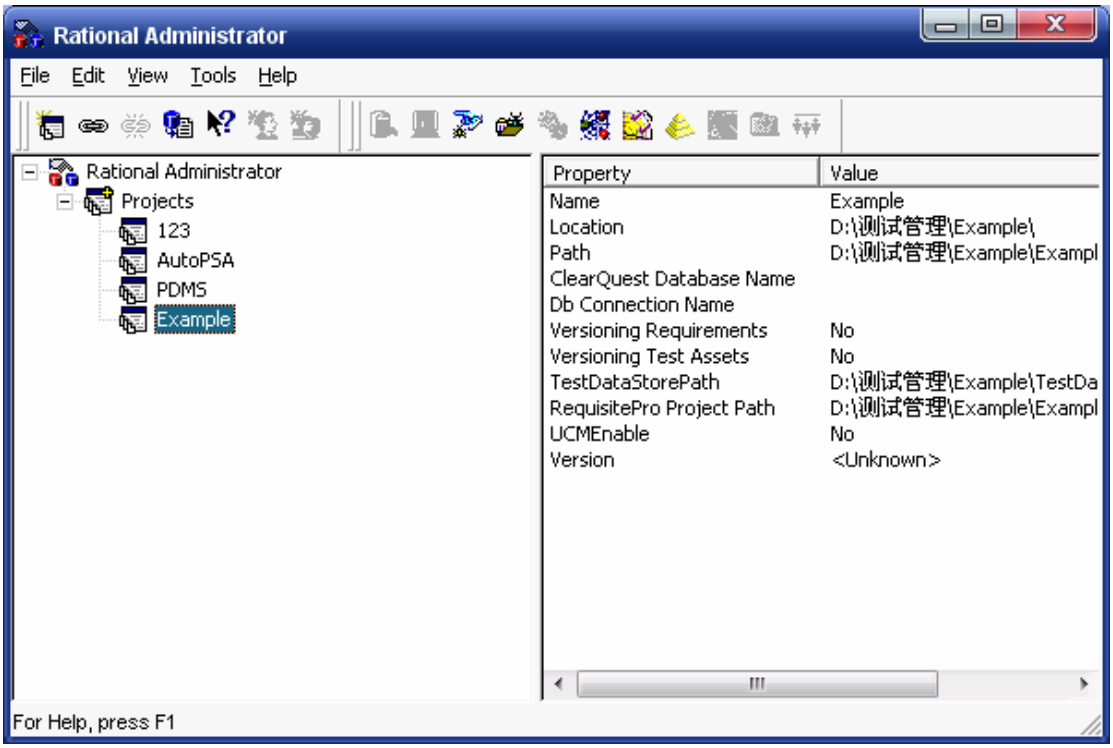

Create Test Datastore (Page 1 of 4)
Choose the type of test datastore
Sybase SQL Anywhere
Choose Sybase SQL Anywhere whenever more than one user accesses the test datastore at a time or if the best possible performance is a priority.
Note: A Sybase SQL Anywhere server must be running in order to create a Sybase SQL Anywhere test datastore.
Microsoft Access
Choose Microsoft Access if only one user accesses the test datastore at a time.
< 上一步(B)
下一步(N) >
取消


Microsoft Access Settings (Page 2 of 4)
New test datastore path
D:\测试管理\Example\TestDatastore
Browse...
< 上一步(B)
下一步(N) >
取消




7. 现在基本配置好了项目需求使用的自动化测试工程项目，在 Rational Administrator 下可以看到该项目的信息。



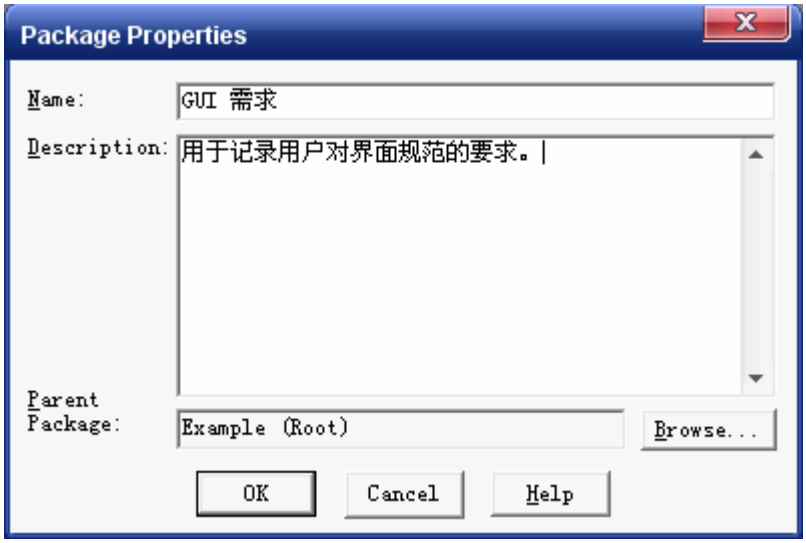

EXAMPLE 1

## 测试计划

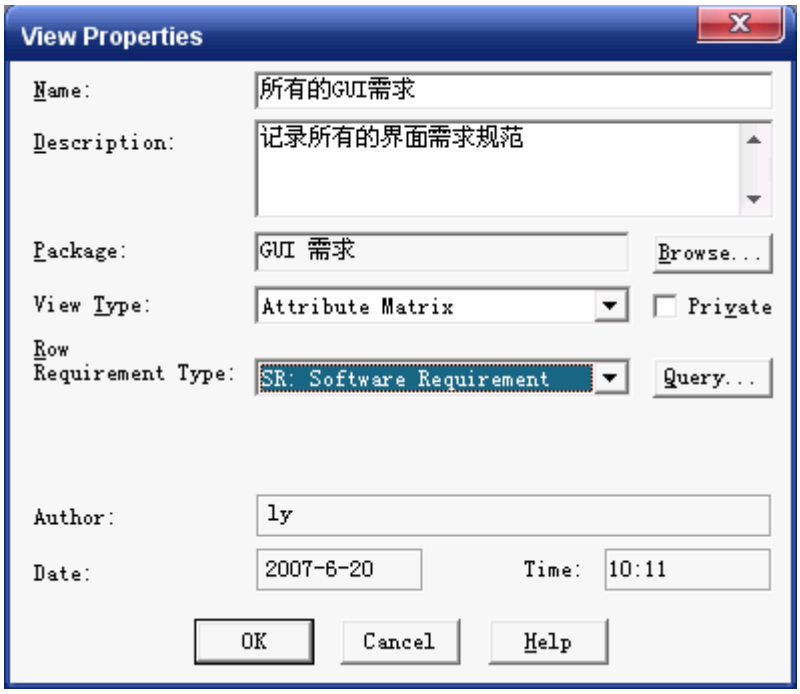
输入测试需求规格

下面将使用 RequisitePro 来建立测试需求规格，具体步骤如下：

1. 启动 Rational RequisitePro，打开“Example”选项
2. 选择 File→New→Package 菜单命令，创建一个用于记录 GUI 测试规格的包

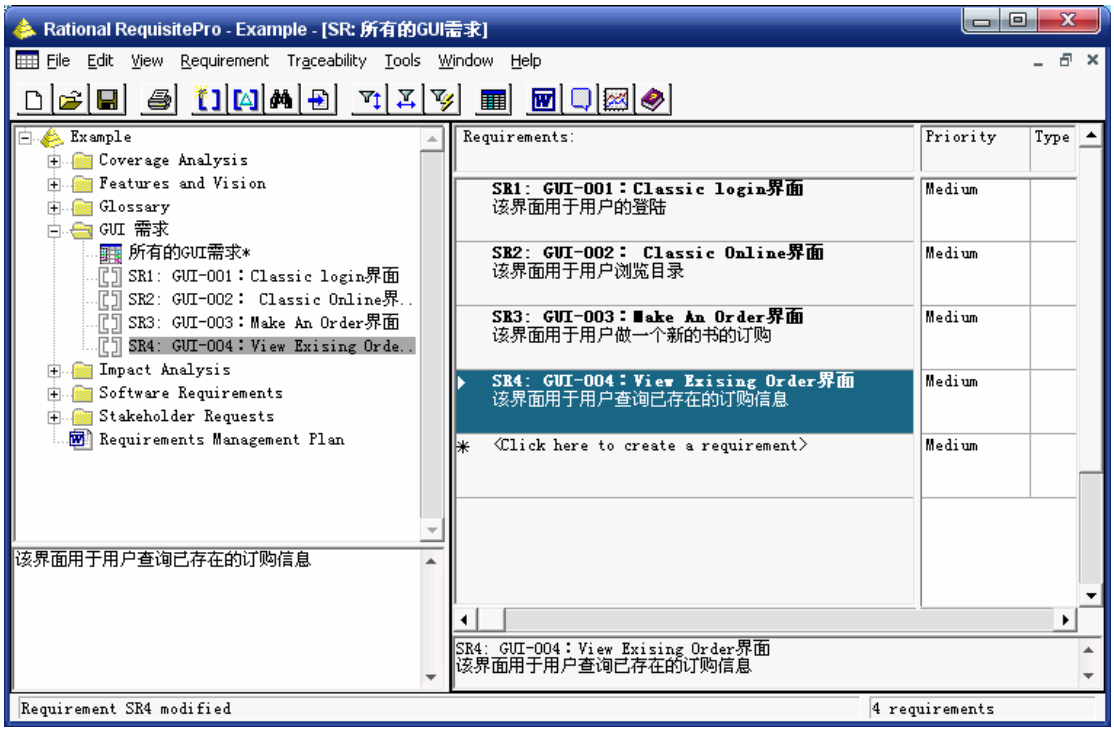



3. 在 GUI 需求下创建一个 View，命名为所有的 GUI 需求




4. 在 GUI 需求下创建第一个测试需求规格


5. 把系统测试所需要测试的需求规格全部输入到库中，最后可以看到如下图所示：

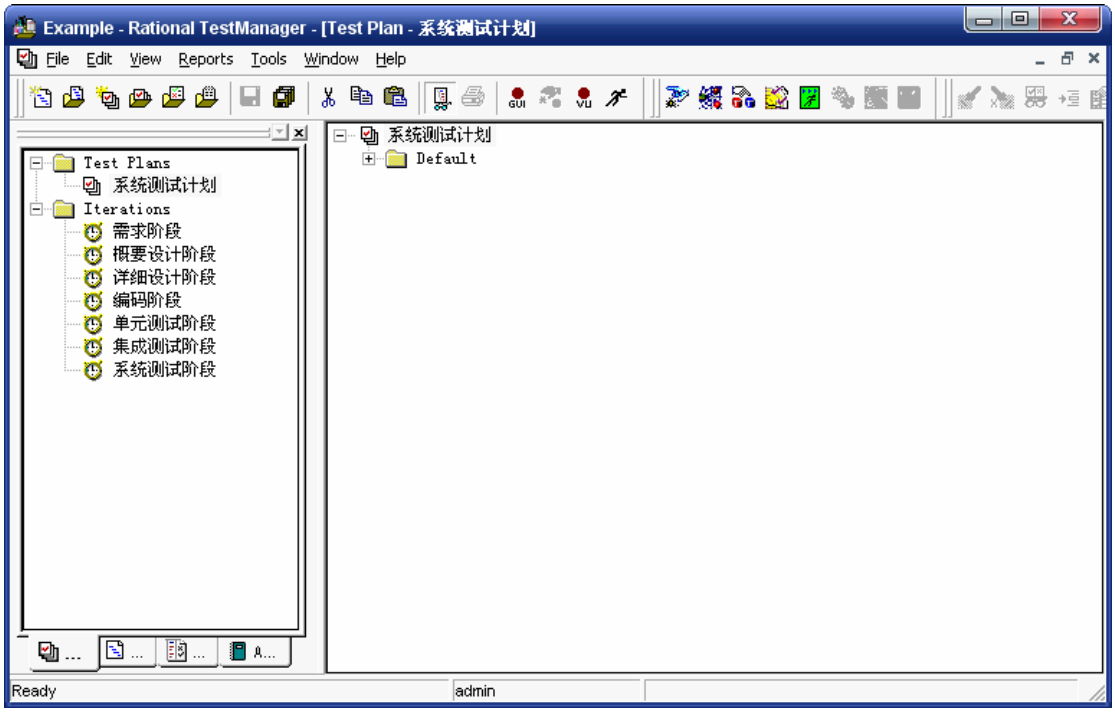



## 创建测试的管理

这里将使用 Rational TestManager 工具来管理测试，具体步骤如下：

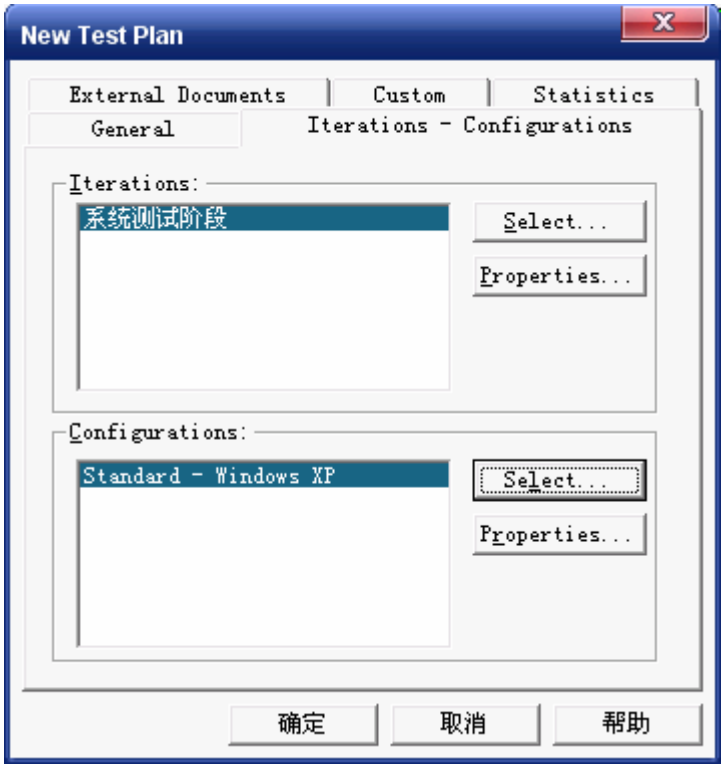
1. 启动 Rational TestManager，并打开“Example”选项。

2. 把左边工作区的迭代各个阶段改成瀑布模型中的各个阶段。

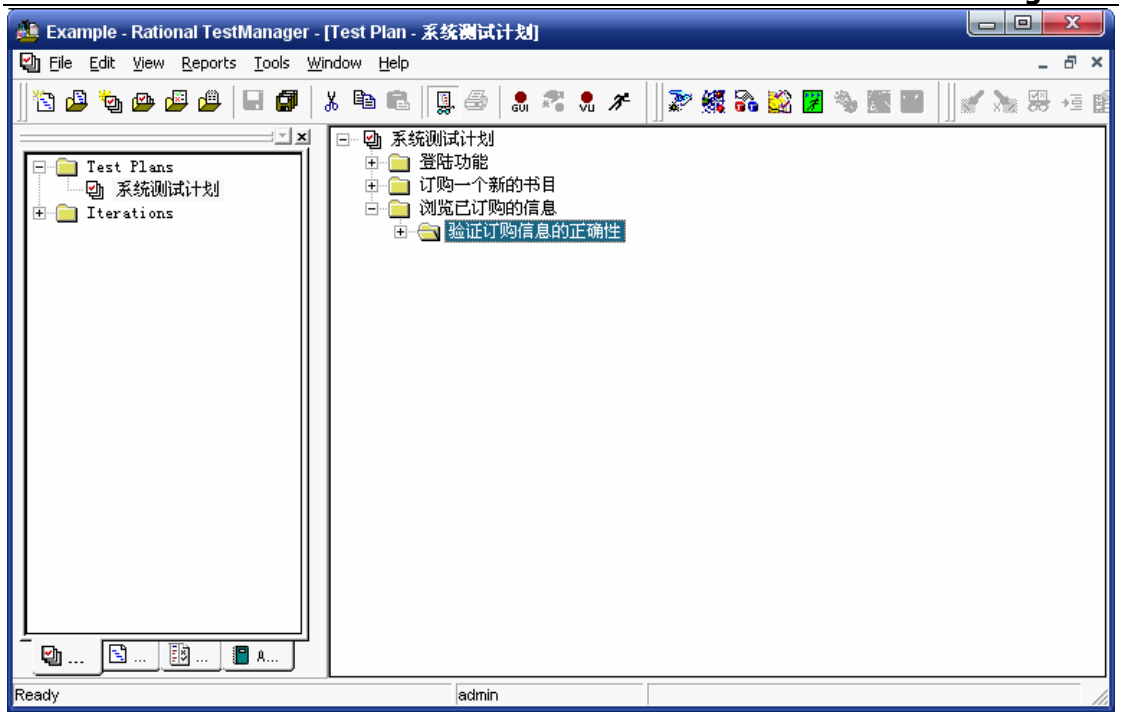



3. 创建一个测试计划，把它命名成系统测试计划，并且在

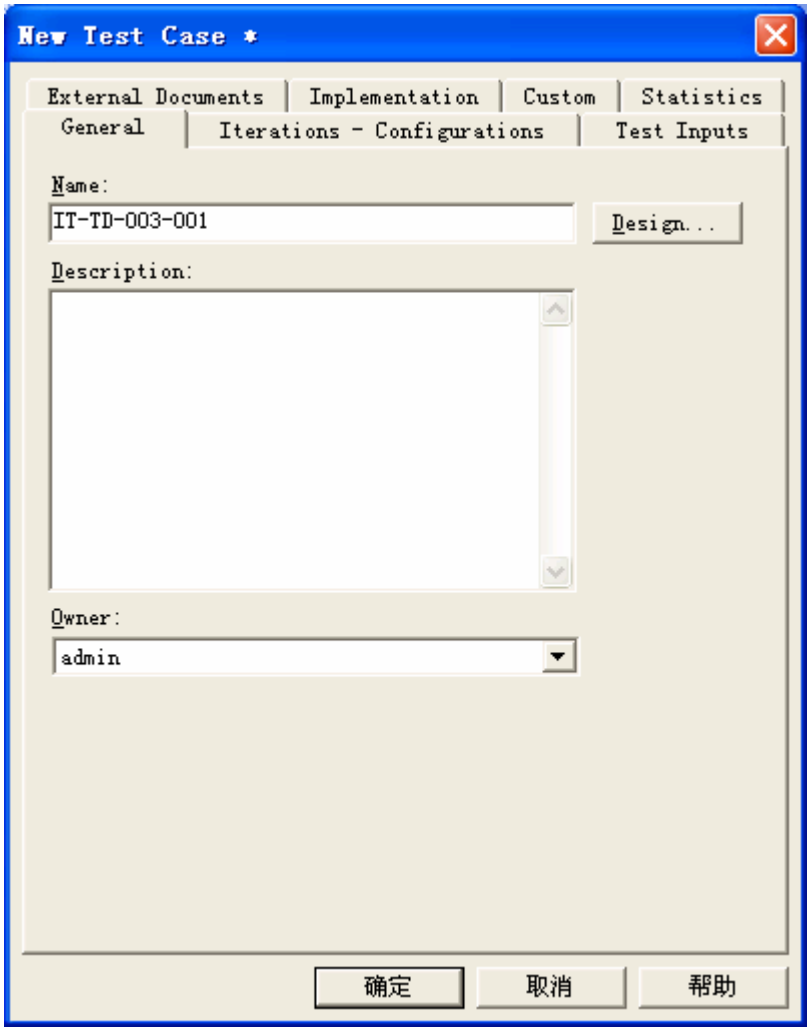
Iterations-Configurations 页把阶段定位成：“系统测试阶段”，测试配置定位为 Standard-WinXP

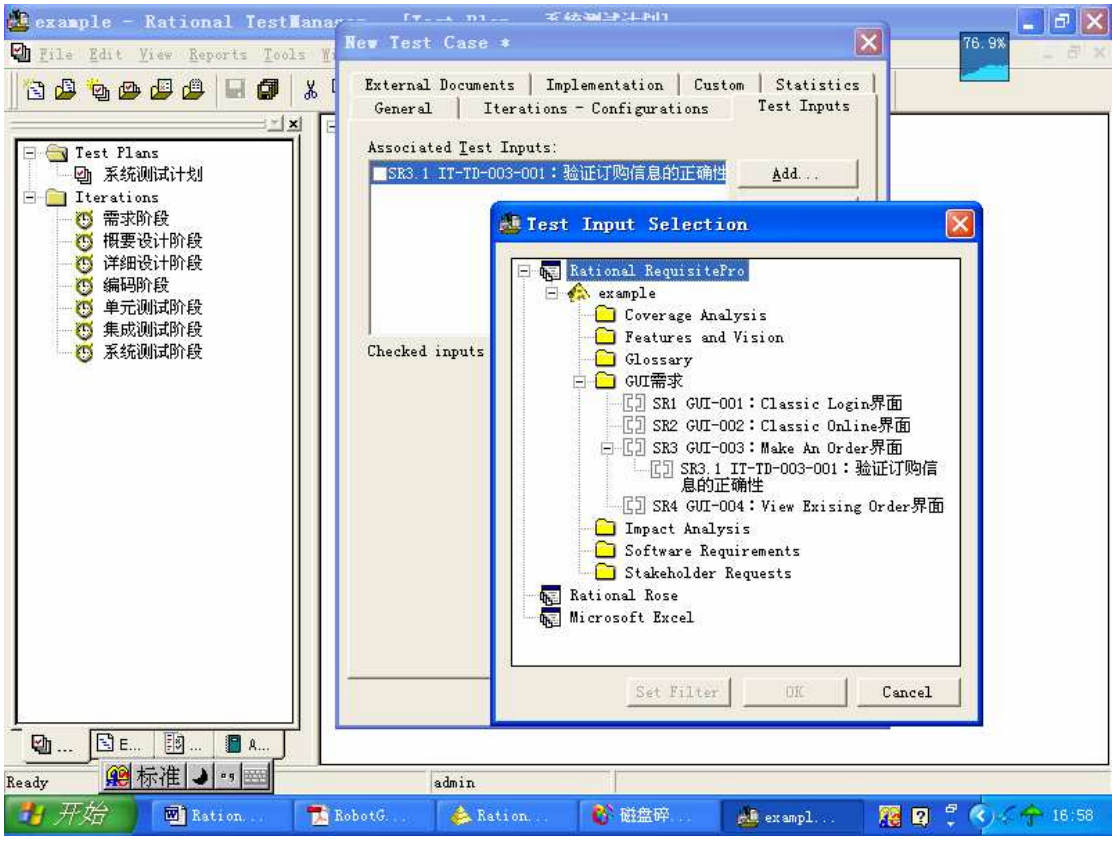

4. 创建“浏览已订购的信息”测试用例目录，并在该目录下创建“验证订购信息的正确性”子目录



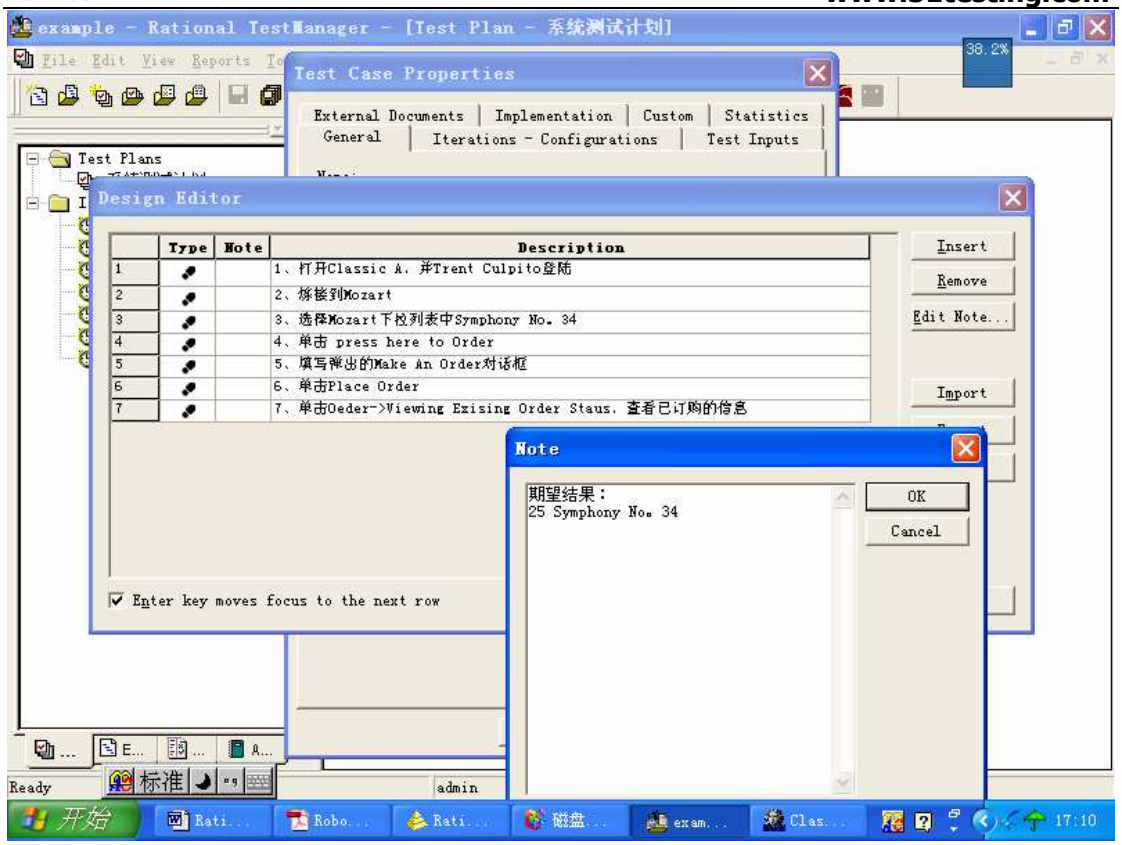

5. 在“验证订购信息的正确性”测试目录下创建第一个测试用例，



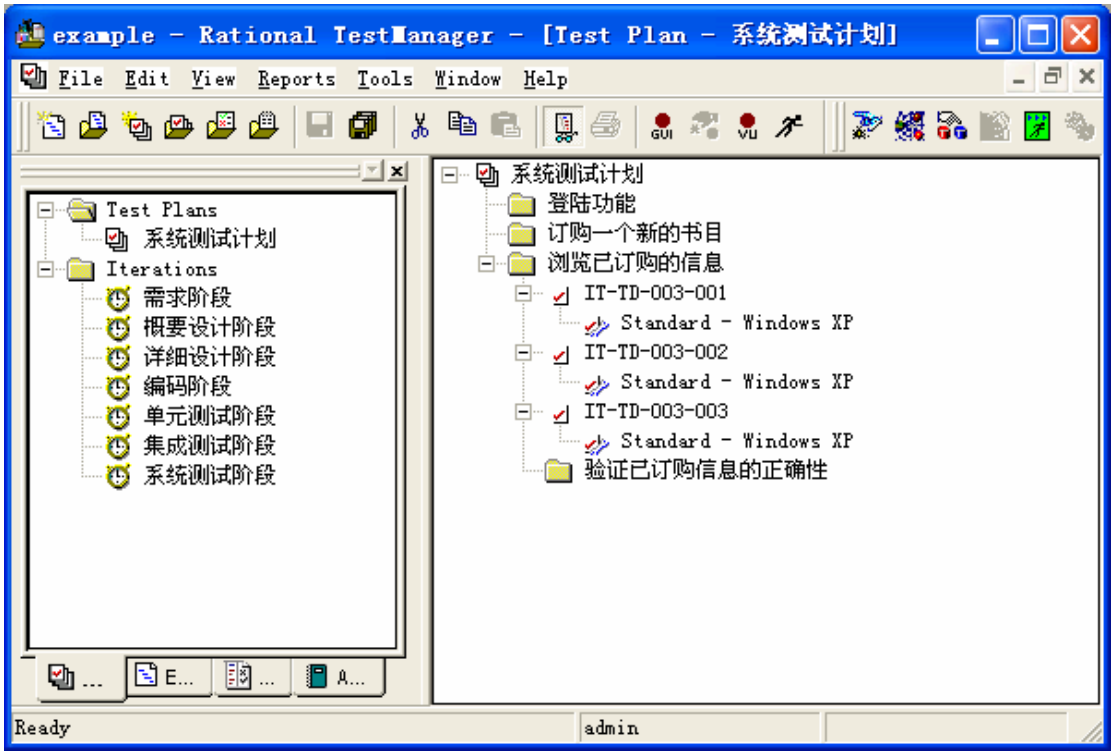
6. 选择 Test Inputs 页面，单击 Add 按扭，把测试用例和对应的测试需求规范链接起来




7. 选择 General 页面，单击 Design 按扭，把《Exampe 项目系统测试用例设计》中 IT-TD-003-001 用例的执行步骤输入到设计中，并把期望结果填写到最后一个步骤的备注中




8. 按前面的步骤，把《Exampe 项目系统测试用例设计》中的所有用例设计输入到 TestManager 中

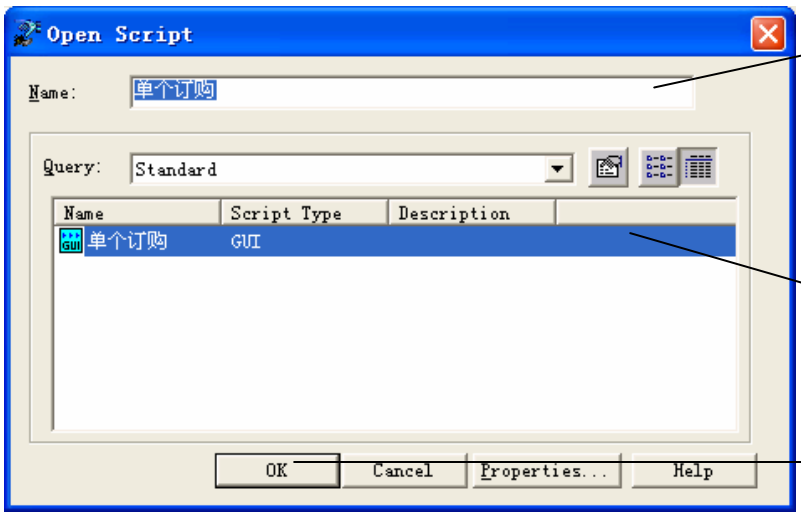



EXAMPLE 2

## 录制脚本

1. 当录制脚本开始的时候，最小化所有的窗口，除了 TestManager。
2. 在 TestManager 的工具栏中点击 Rational Robot 按扭，开始 Robot。

3. 最小化 TestManager
4. 从 Robot 窗口中，选择 File→Record GUI 




GUI Record 工具栏将出现在 Robot 的左上角，你可以根据自己的需求来插入不同的测试脚本。


点击 Display GUI Insert Toolbar 按扭来浏览 GUI 插入工具栏


5. 点击 Start Application 按扭 

注意：在你开始应用程序之前，让你不用开始菜单和桌面来开始一个应用程序，它将给你更多的练习在回放时控制你的测试环境。

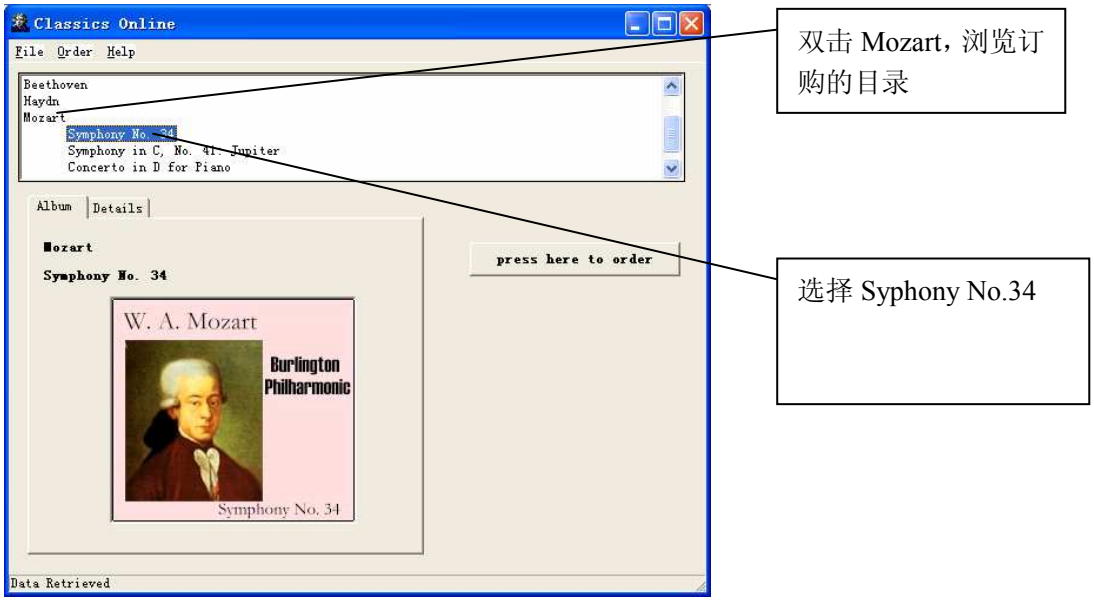
6. 点击 Browse，然后选择默认的 Classics Online sample 应用程序路径。

C:\Promgram Files\Rational\Rational Test\Sample Applications\Classic Online\ClassicsA.exe

7. 点击 Open，然后点击 OK

8. 用 Trent Culpito 作为登陆，点击 OK，打开 Classics Online

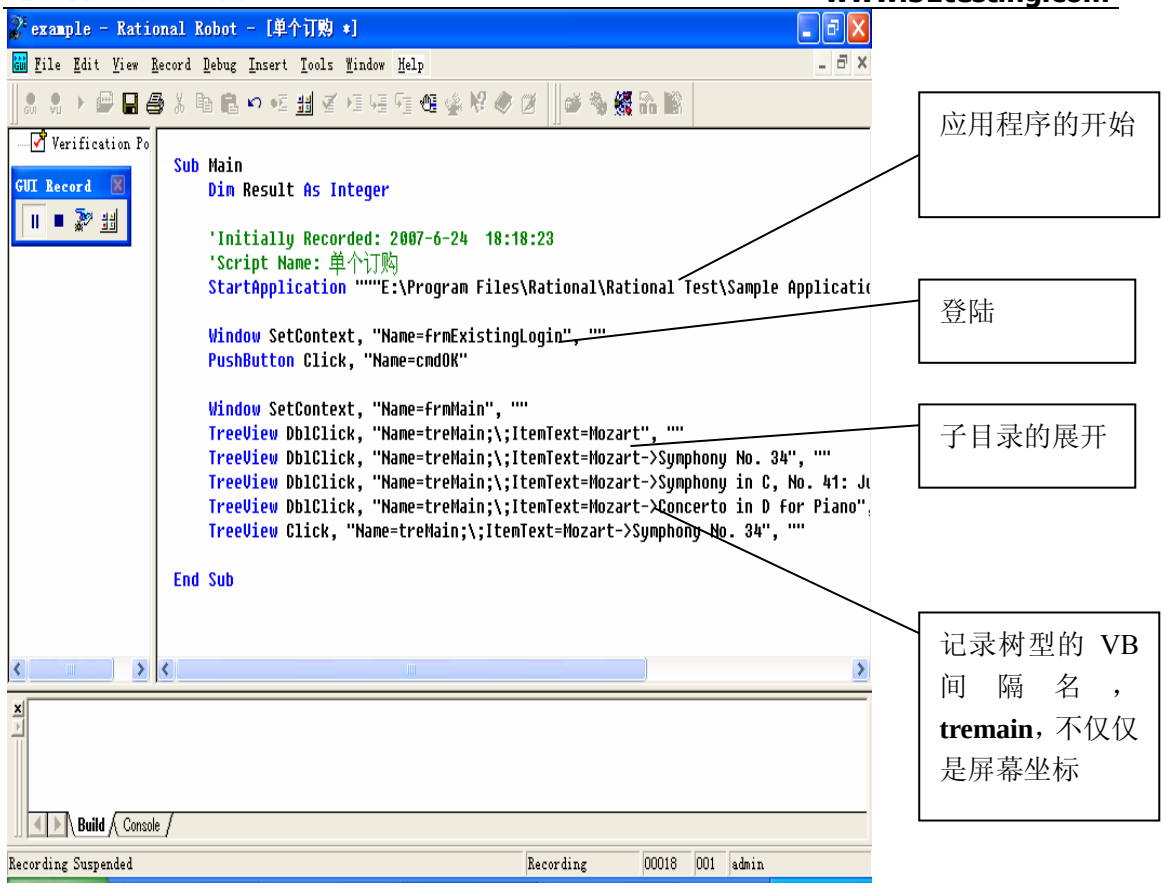
9. 点击弹出对话框的+，直到你看到 Mozart




## 浏览已录制的脚本

1. 在 GUI Record 工具栏中点击 Open Robot Window 按扭  。

2. 最大化你刚才录制的脚本。

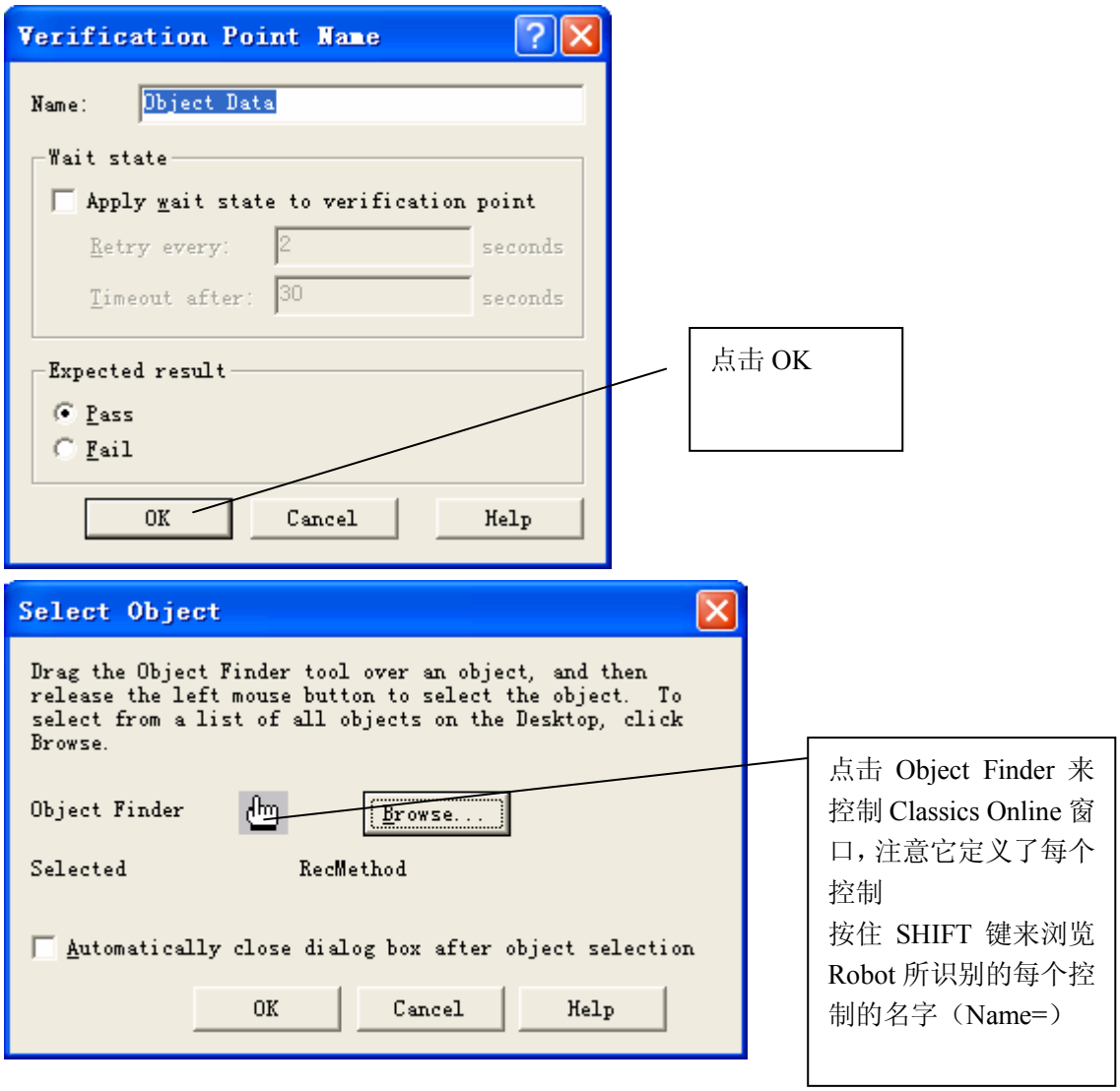



3. 你将看到脚本中记录了你刚才的行动。最小化 Robot 窗口

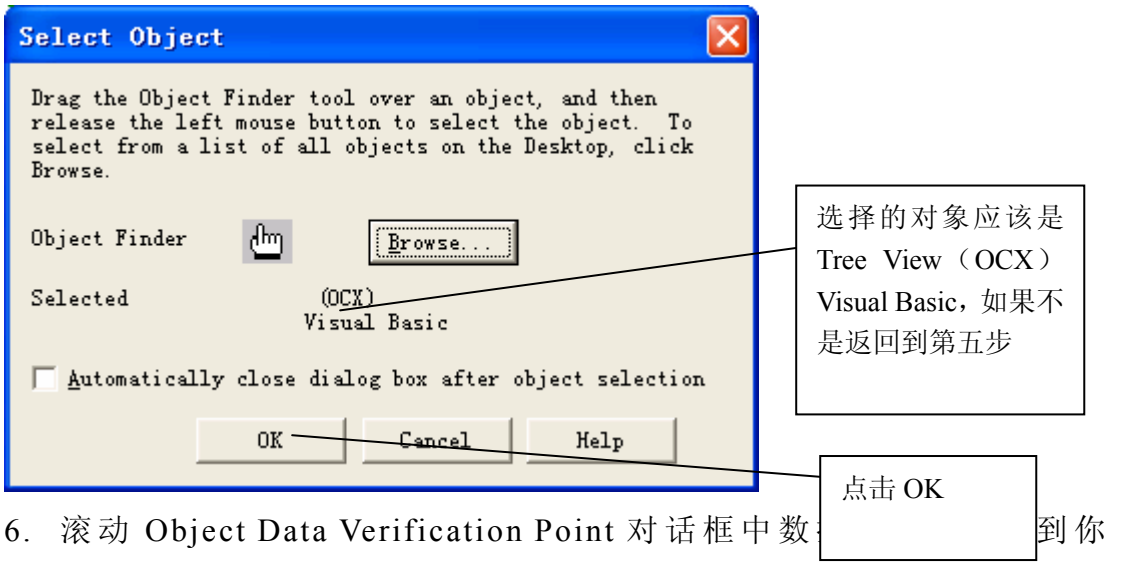
**增加验证点**

验证点 1：捕获树型控制数据

1. 如果有必要，点击 Display GUI Insert 工具栏按扭 
2. 如果有必要，最小化 TestManager 窗口
3. 点击 GUI Insert 工具栏右边的 Object Data 按扭 
4. 点击 OK，接受 Object Data 作为第一个验证点的名字

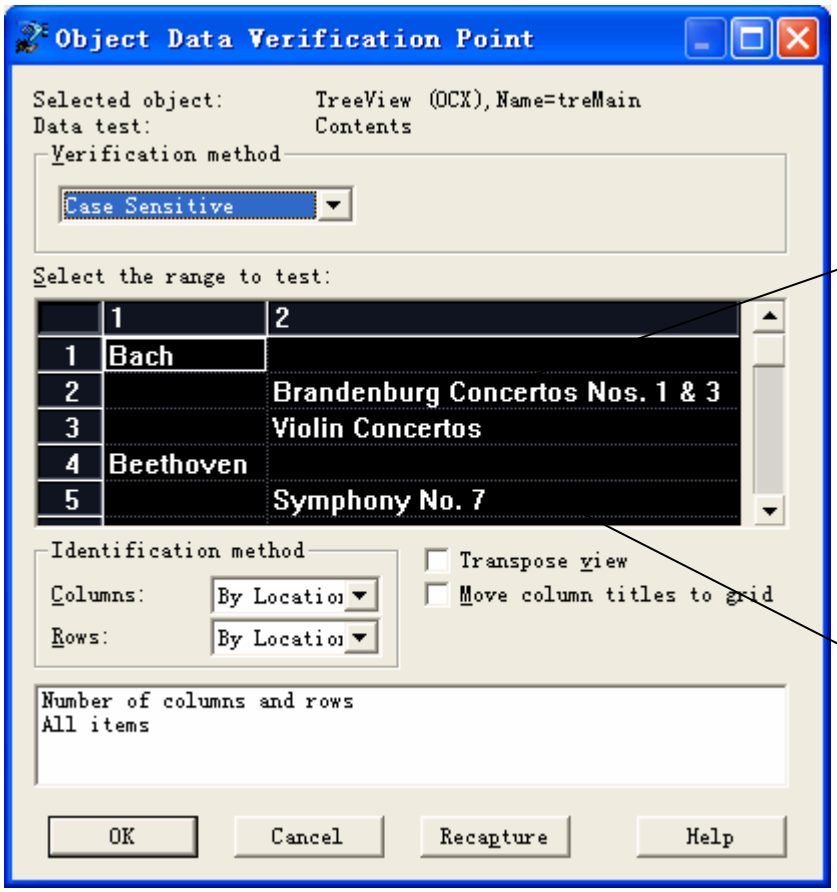



5. 移动 Object Finder 工具栏到应用程序的顶部的组成和目录列表框中，然后释放鼠标按扭。




6. 滚动 Object Data Verification Point 对话框中数[illegible]到你

刚才所捕获到的内容




Robot 将捕获所有的树型控制的数据，即使分支并没有展开

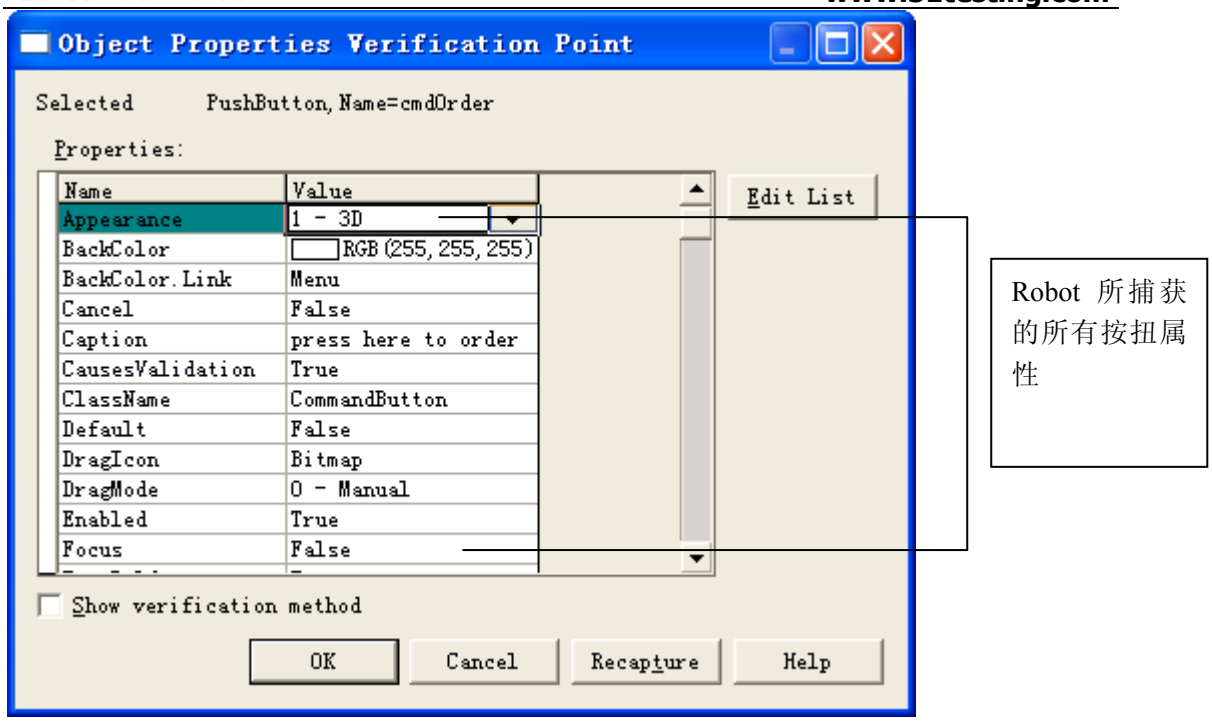
你可以选择或不选这些元素，Robot 只识别被选择的元素

确定所有的元素都已经被选择了

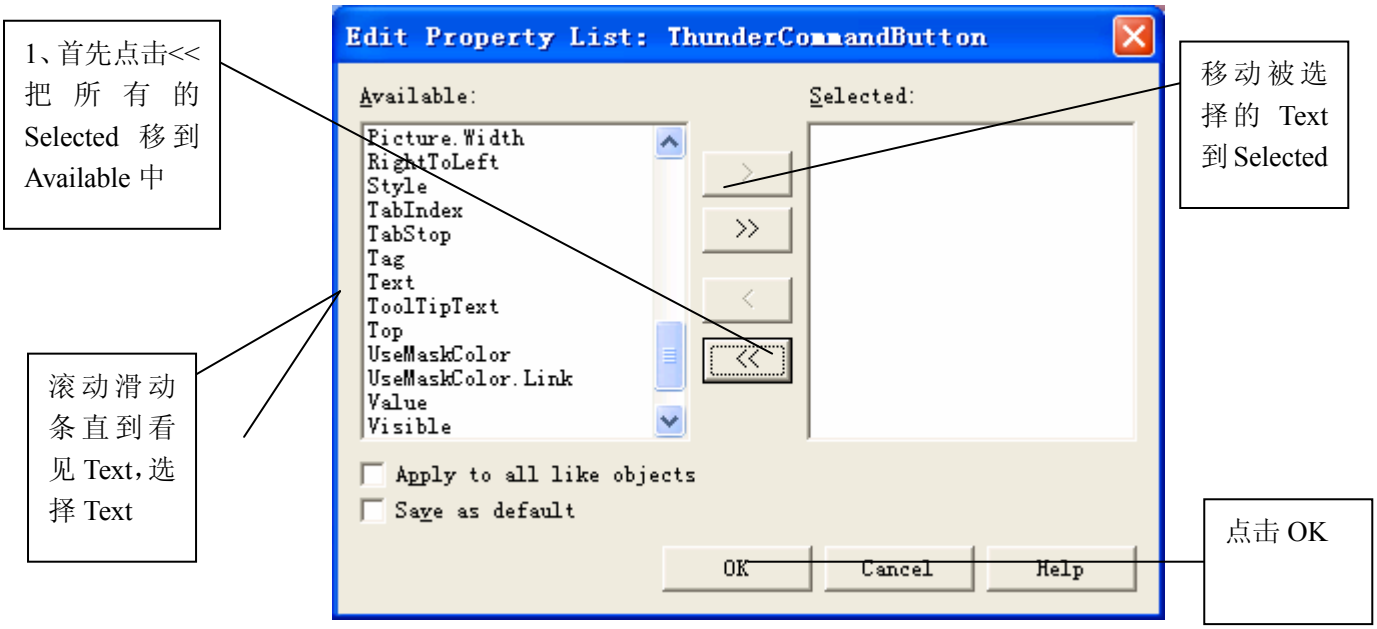
7. 点击 OK，就完成了一个验证点的捕获

验证点 2：捕获一个按扭的属性

1. 在 GUI Record 工具栏中点击 Open Robot Window 按扭 
2. 点击 Object Properties 按扭
3. 接受自动命名的验证点，点击 OK
4. 移动 Object Finder 工具栏，在 press here to order 按扭处释放
5. 确定被选择的对象是 PushButton VisualBasic，点击 OK




6. 点击 Edit List

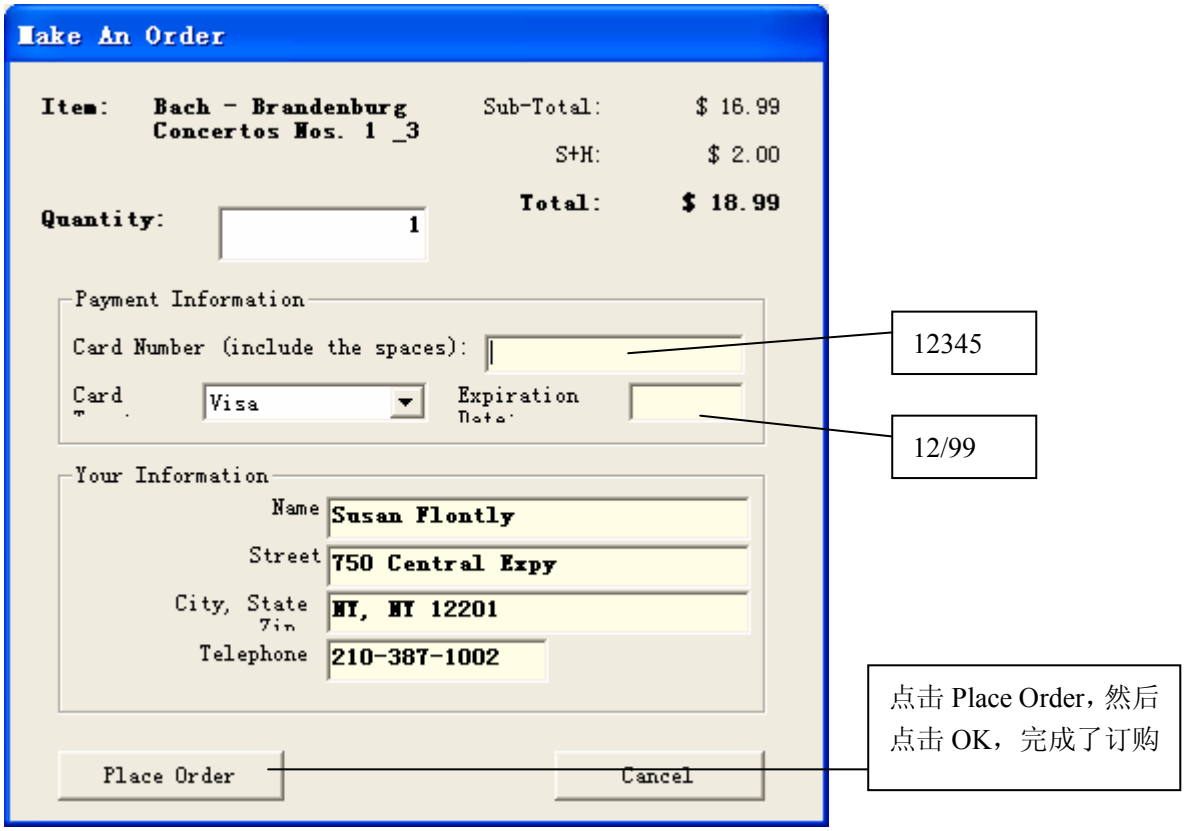



7. 点击 OK 完成了另一个验证点的插入

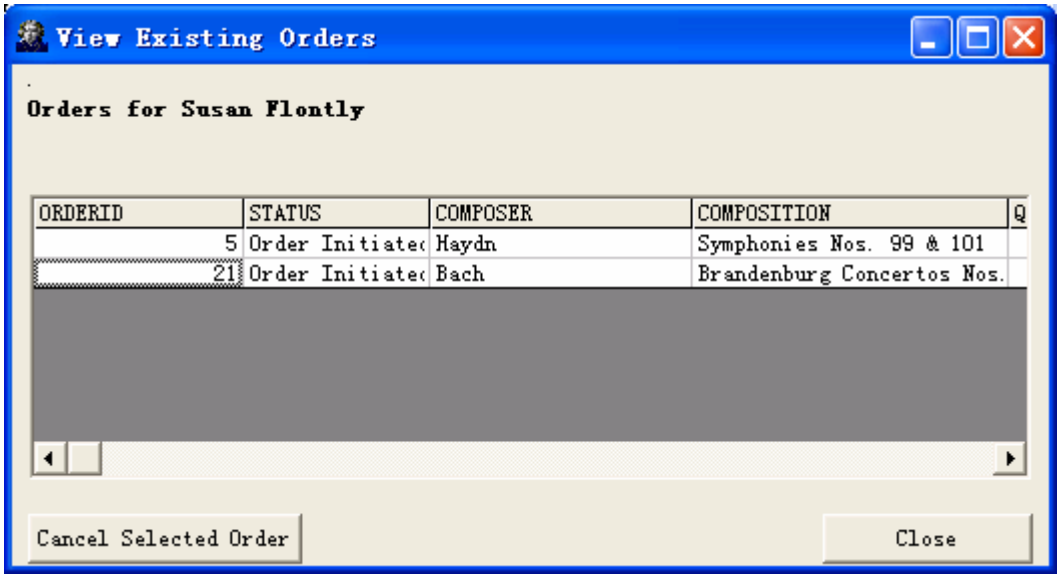
验证点 3：在数据限度控制中捕获数据

1. 以一个新的用户名登陆 File→Login As New User
2. 选择 Susan Flontly，点击 OK
3. 浏览 Susan 的订购信息，选择 Order→View Existing Order Status，注意到 Susan 已经订购了一本书
4. 点击 Close

5. 选择 Mozart 的 Symphony No.34，点击 press here to order




6. 选择 Order→View Existing Order Status，将看到一个新的订购信息。

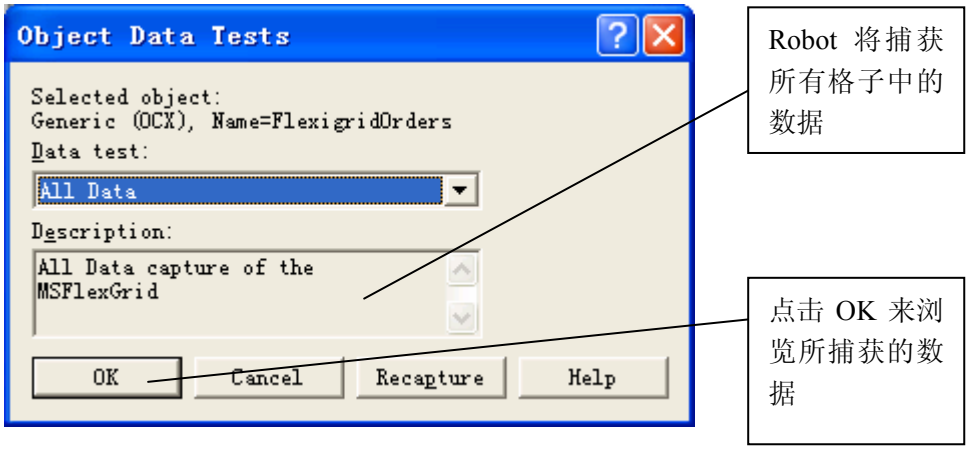



7. 保持该对话框打开，开始一个验证点的插入

8. 接受自动命名的验证点，点击 OK

9. 移动 Object Finder 到格子处，然后释放鼠标

10. 确定所选择的对象为 Generic （OCX）Visual Basic，点击 OK




11. 点击 OK，完成了验证点的插入

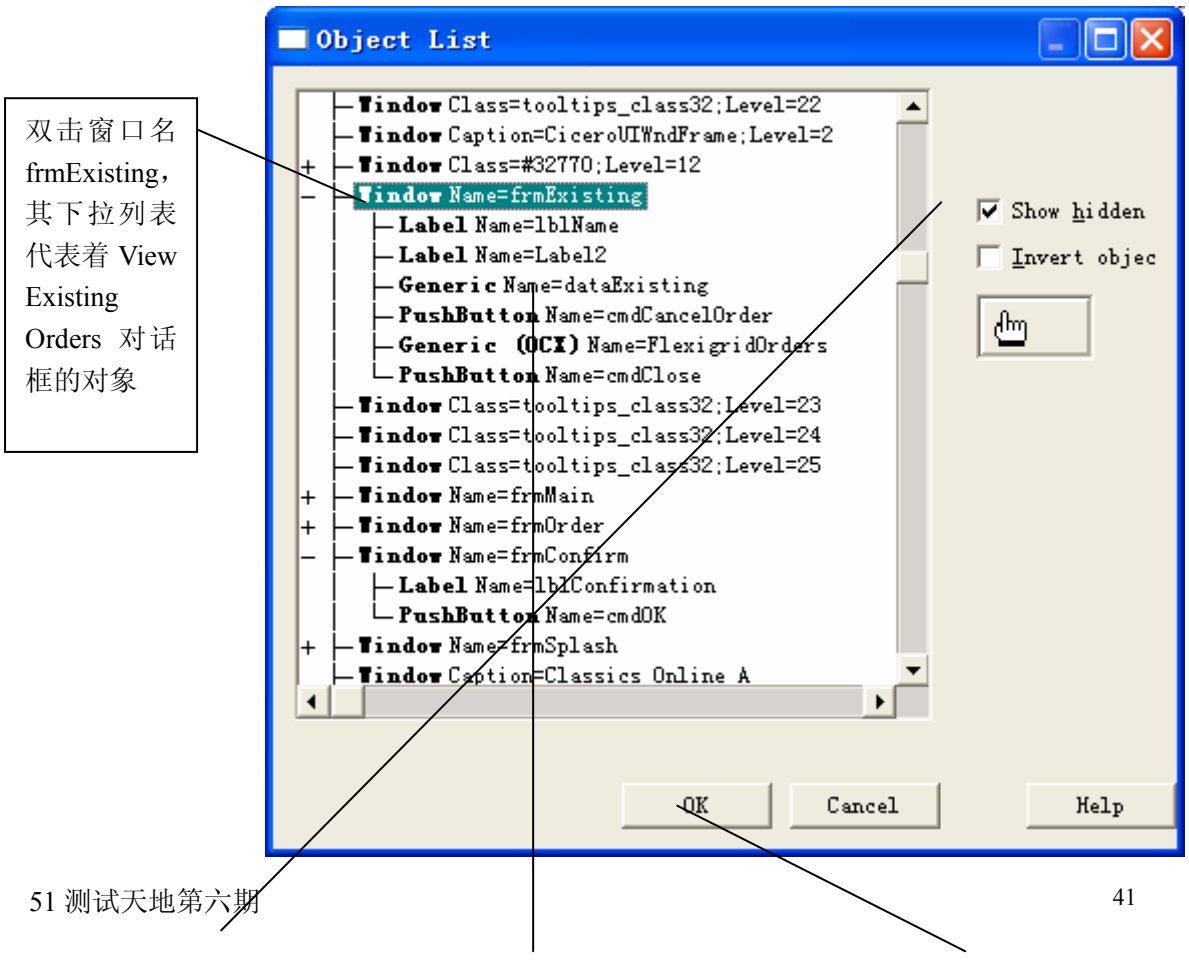
验证点 4：捕获一个不可视的数据控制属性

1. 确定 View Existing Orders 窗口的打开

2. 开始一个验证点对象属性的捕获  

3. 接受自动命名的验证点，点击 OK

4. 点击 Browse 按扭，Robot 将找到所有桌面上可视和不可视的对象



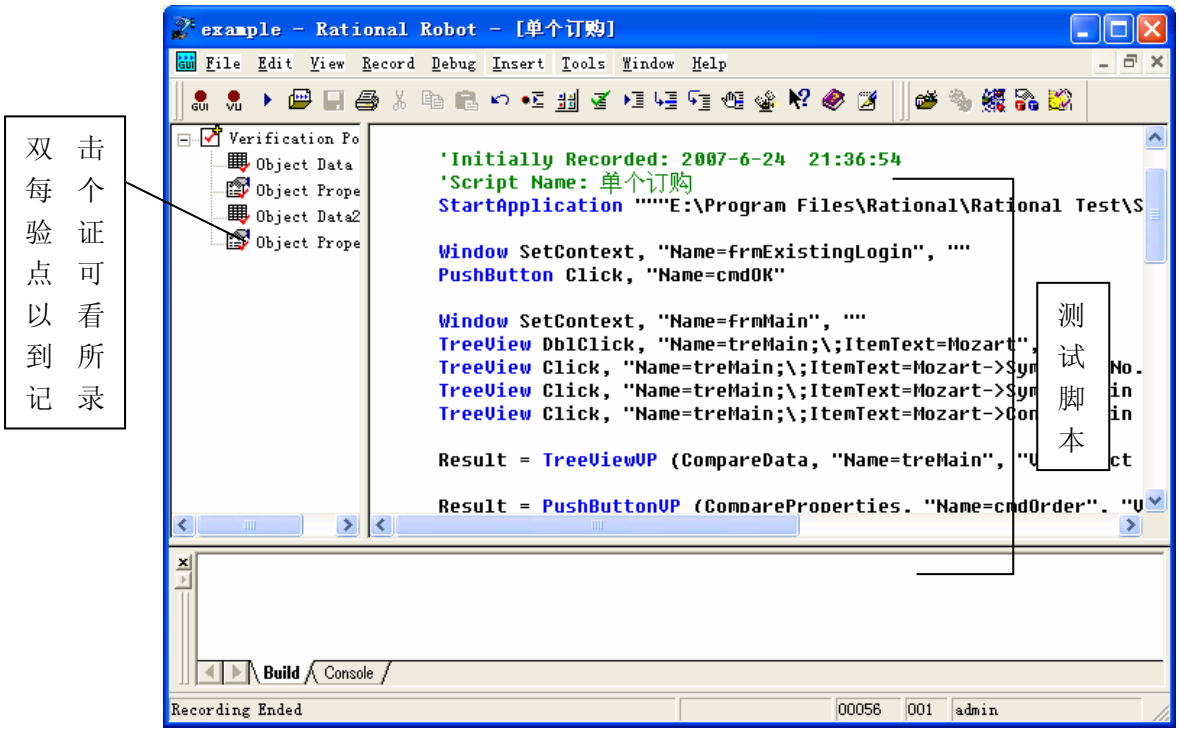
展示Robot所隐藏的所有对象

选择对象名 Generic Name=dataExisiting 不可视数据控制的订购进入数据库

点击OK捕获所有的不可视图的数据控制

5. 在验证点对象属性对话框中可以看到 Visible 的值为 False，这表明该数据控制是不可视的，但是却已经被 Robot 的对象测试技术所捕获。
6. 点击 OK 完成了测试
7. 点击 Close
8. 关闭 Classic Online
9. 点击 Stop Recording 按扭 

浏览所录制的脚本




脚本的解释：

Windows SetContext 作为每个行动的开始，并记录开始以(')

每个验证点作为单独的一行出现并开始为 Restult=

下面出现在脚本的命令为：

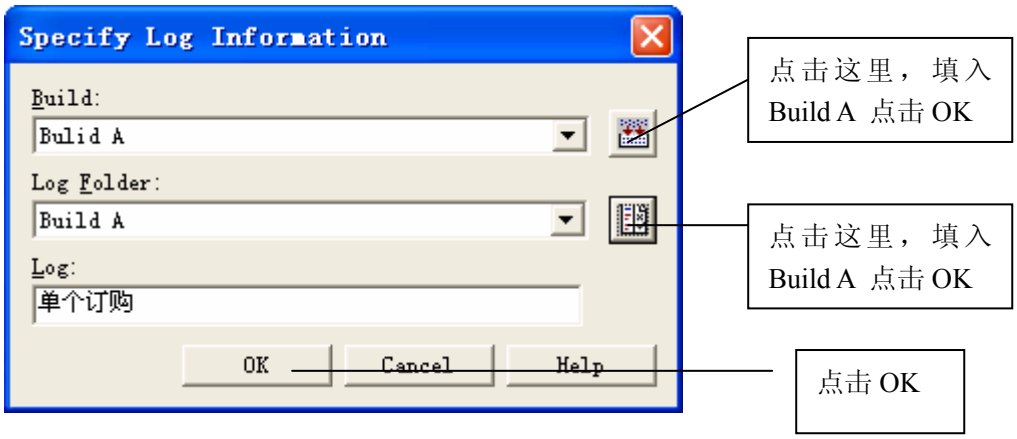
- Sub Main----------表明脚本的开始
- Dim Result as Integer---------声明验证点的值 Result 作为一个变量，这个 Rseult 变量定义的是主程序的
- 'Initially Recorded：----------自动记录所记录的脚本的日期和时间
- 'Script Name：----------定义脚本名
- Window SetContext----------定义用户行动发生的窗口
- Window SetTestContext--------定义一个验证点所插入的窗口
- Result = 定义脚本所记录的一个验证点。这个验证点表明了对象名和脚本命令识别方法的控制。
- Name = 定义了所有被测试对象名
- PushButton Click----表明一个用户的行动命令

总结：你现在已经记录了一个脚本和四个验证点的插入

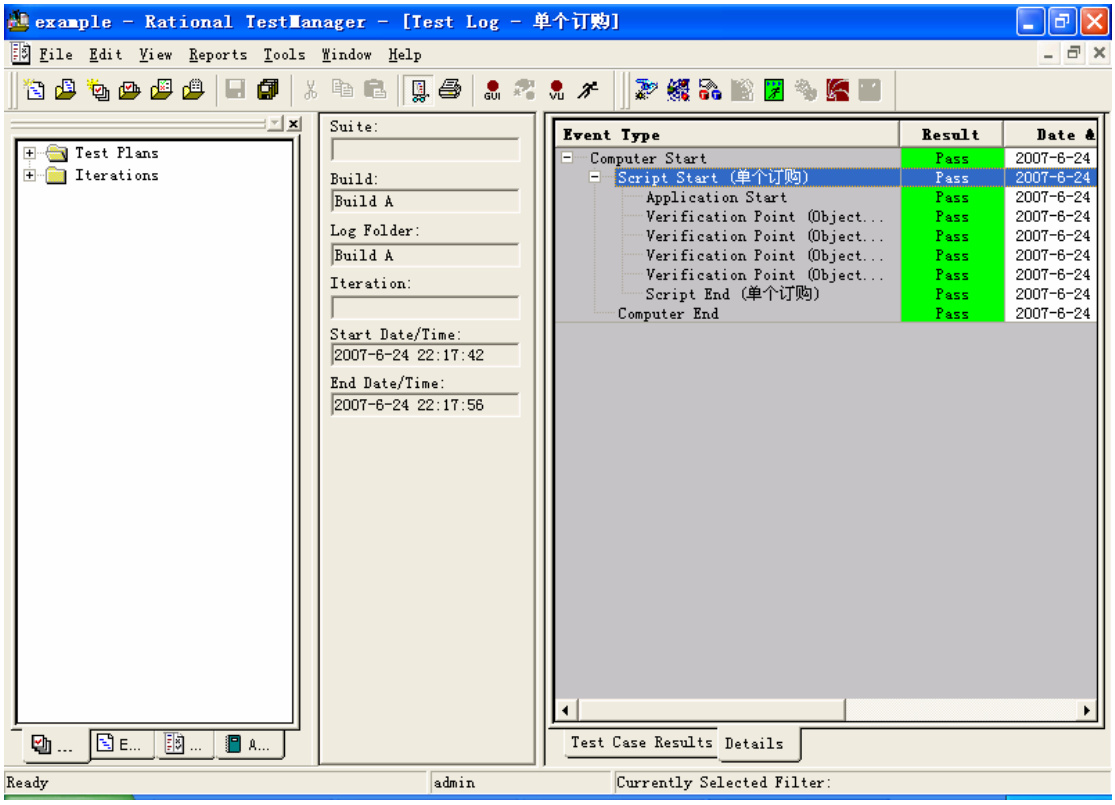
- 在树型控制的目录数据浏览
- 按扭的属性
- 订购进入数据库的数据浏览
- 隐藏的数据控制属性 EXAMPLE 3

**脚本的回放**

1. 确定 Robot 的打开
2. 点击 File→Playback 
3. 选择“单个订购“，点击 OK



你将会看到测试结果在 LogViewer




4. 点击 File→Exit 关闭 LogViewer，回到 RobotEXAMPLE 4

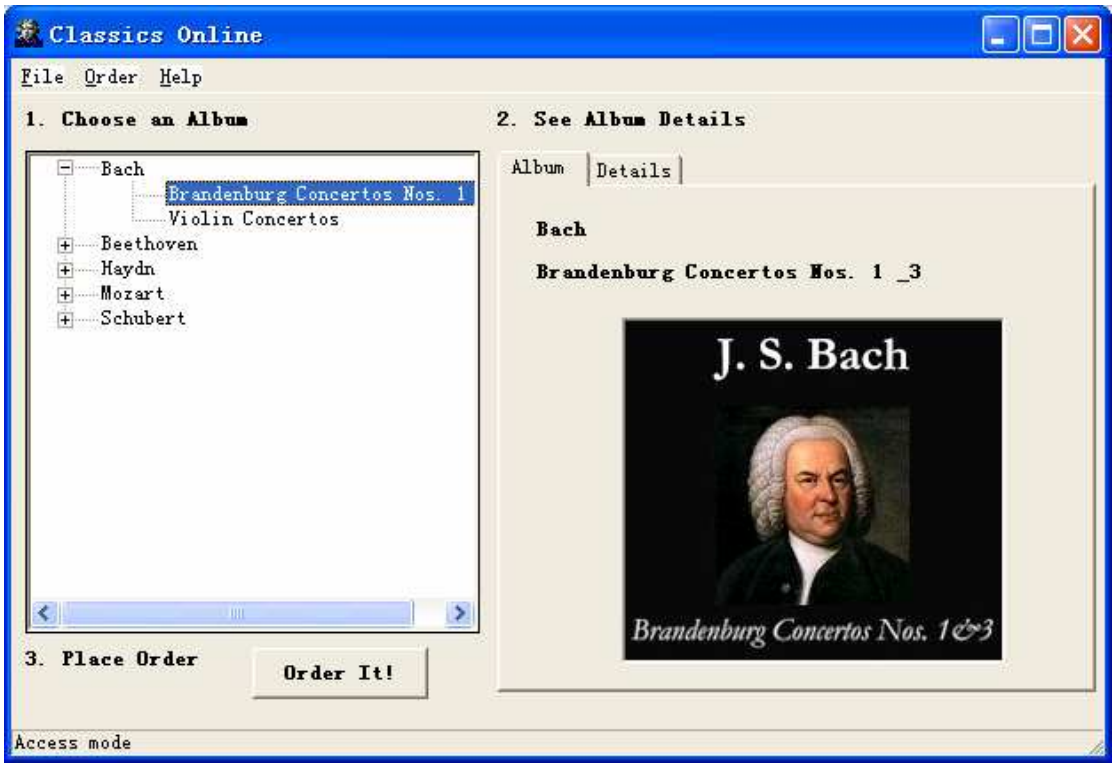
## 测试一个新的版本和评估测试结果

首先我们来比较一下这两个版本的不同，Classics A 和 Classics B 主要是主窗口的不同

Classics A


Classics Online
File Order Help
Bach
Brandenburg Concertos Nos. 1 & 3
Violin Concertos
Beethoven
Haydn
Mozart
Album
Details
Bach
Brandenburg Concertos Nos. 1 _3
press here to order
J. S. Bach
Brandenburg Concertos Nos. 1&3
Data Retrieved

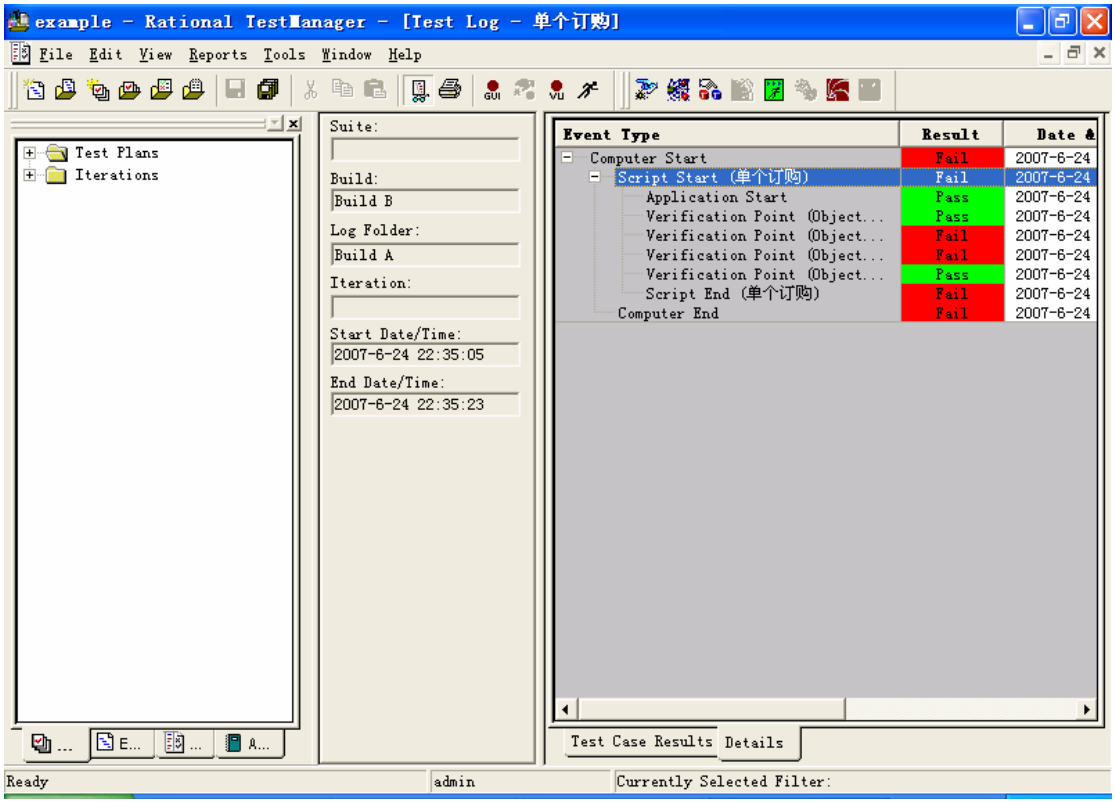
Classics B


Classics Online
File Order Help
1. Choose an Album
2. See Album Details
Bach
Brandenburg Concertos Nos. 1
Violin Concertos
Beethoven
Haydn
Mozart
Schubert
Album
Details
Bach
Brandenburg Concertos Nos. 1 _3
J. S. Bach
Brandenburg Concertos Nos. 1&3
3. Place Order
Order It!
Access mode

## 建立一个新的 Build 来回放测试脚本

1. 点击 File→Open Script
2. 选择“单个订购”，点击 OK
3. 在脚本中把测试路径改为 Classics B.exe
4. 点击 File→Save
5. 开始 playback
6. 选择“单个订购”，点击 OK
7. Robot 将编译脚本并回放一连串的行动和验证点。

在 LogViewer 中查看测试结果




你将看到有两个地方是 Fail。

## 在 Comparators 中分析测试结果

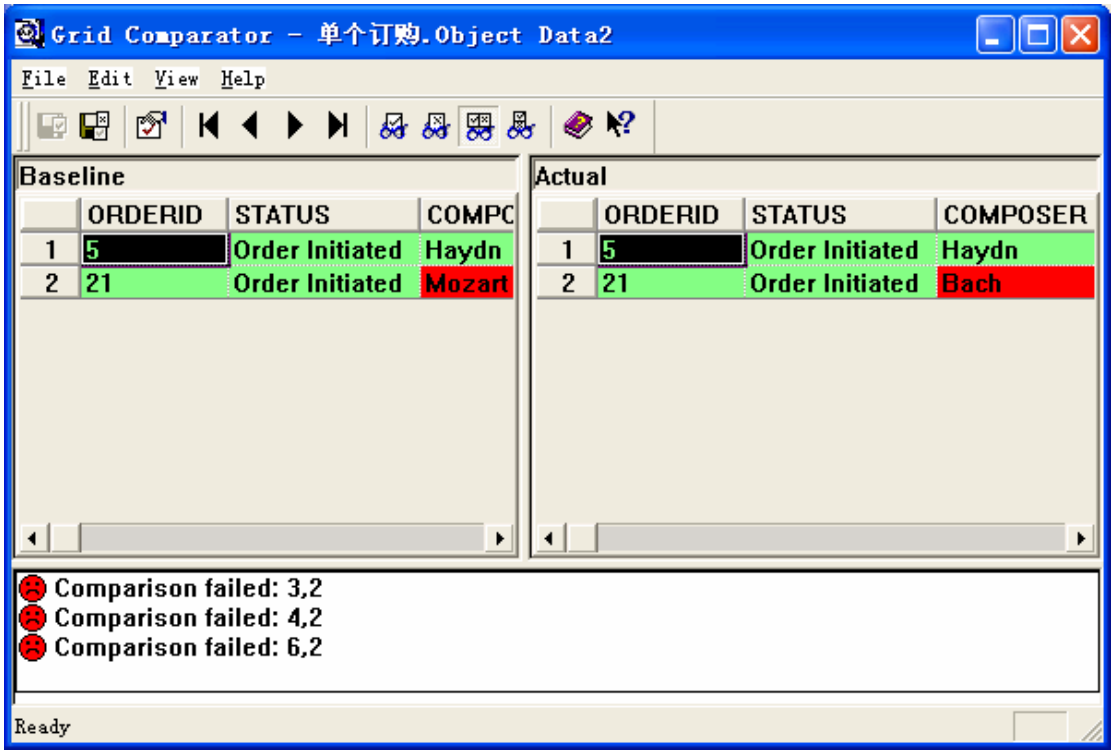
Comparators 提供了四种比较对象：

- 格子----------浏览输入文件如：文本和数据验证点
- 对象属性----------浏览对象属性用于捕获验证点属性
- 文本----------除了格子以外的浏览输入文件如：文本和

数据验证点

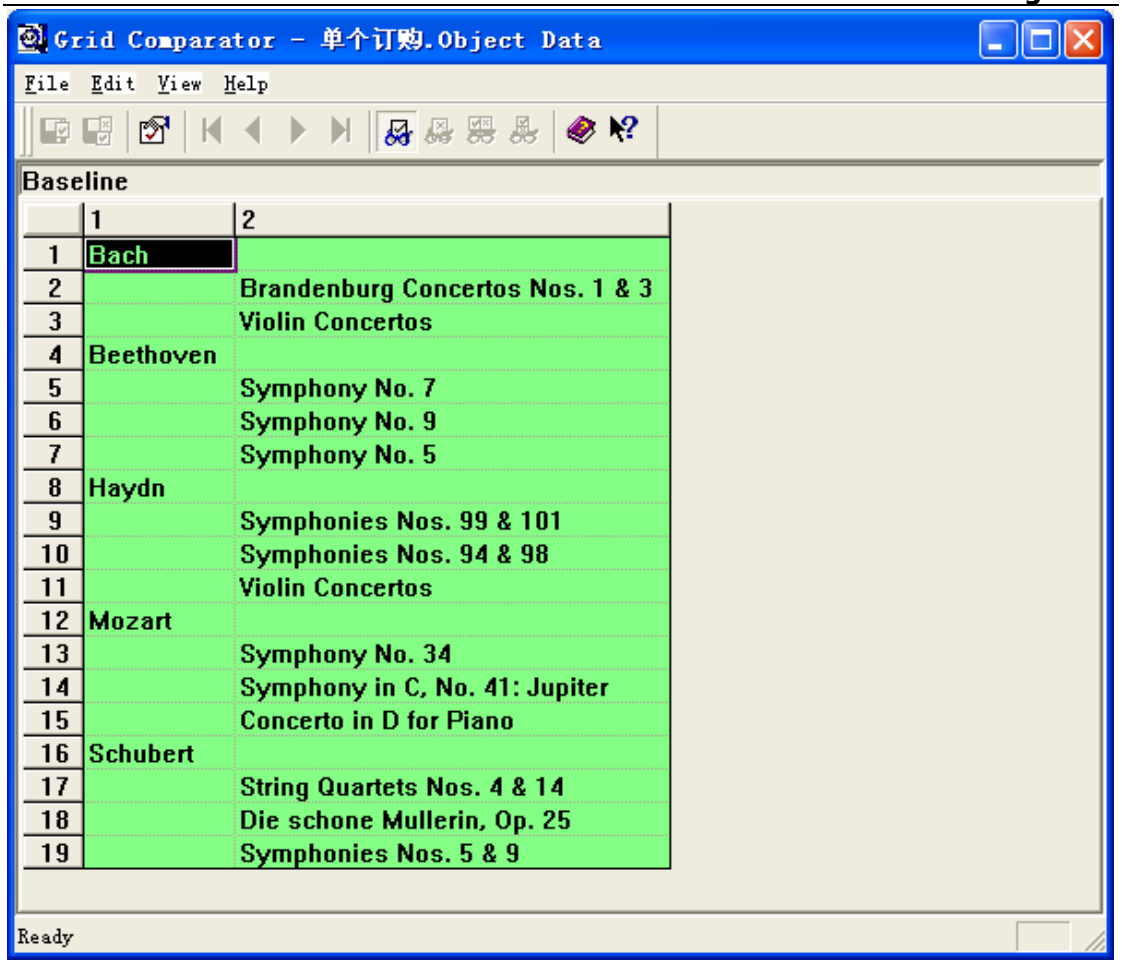
- 图形----------浏览图形文件如区域图形和验证点的窗口图形

如果没有“失败”，就只有 baseline file 来浏览记录的数据和图形。如果有“失败”，就会有一个 actual file 来浏览下个一 baseline file，通过这些文件的比较，你可以判定是由于 Build 的改变引起的，还是由于一个缺陷的产生。




浏览树型控制测试

1. 双击 Verification Point（Object Data）-Object Data，将浏览到验证点的通过信息



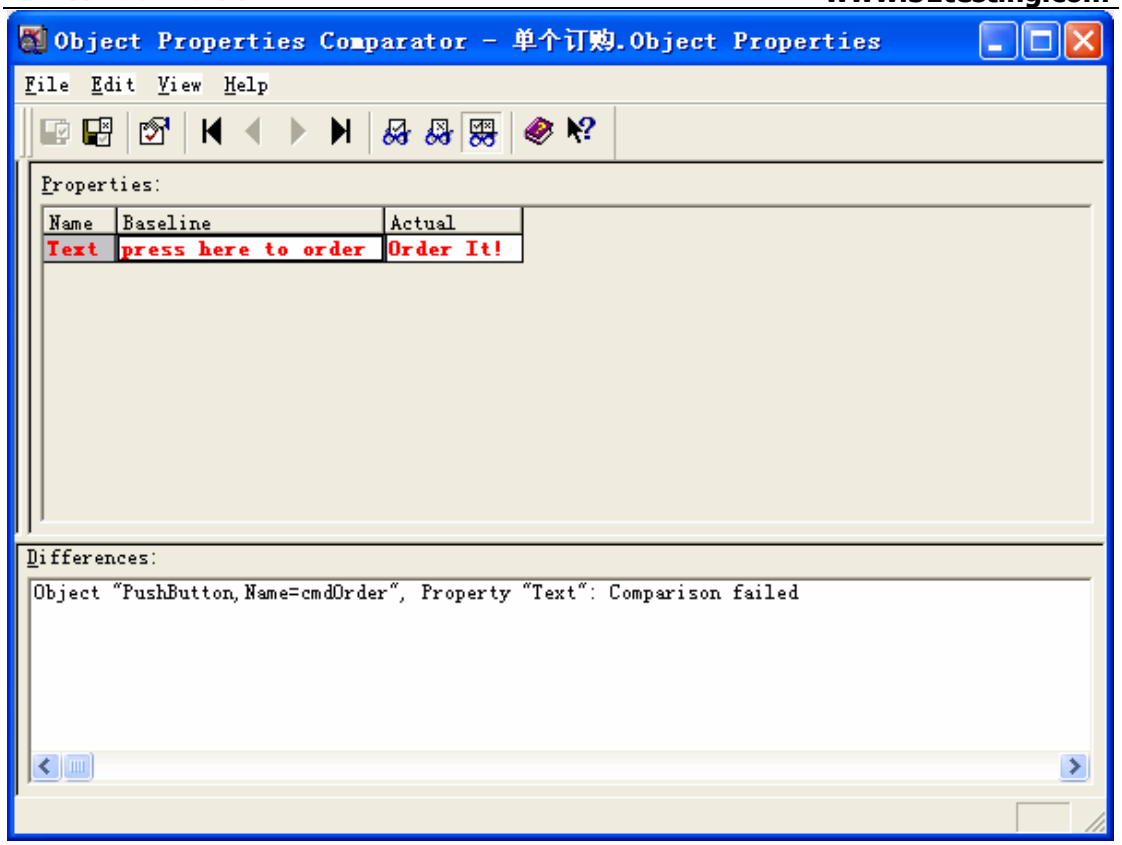

| | 1 | 2 |
|---|---|---|
| 1 | Bach | |
| 2 | | Brandenburg Concertos Nos. 1 & 3 |
| 3 | | Violin Concertos |
| 4 | Beethoven | |
| 5 | | Symphony No. 7 |
| 6 | | Symphony No. 9 |
| 7 | | Symphony No. 5 |
| 8 | Haydn | |
| 9 | | Symphonies Nos. 99 & 101 |
| 10 | | Symphonies Nos. 94 & 98 |
| 11 | | Violin Concertos |
| 12 | Mozart | |
| 13 | | Symphony No. 34 |
| 14 | | Symphony in C, No. 41: Jupiter |
| 15 | | Concerto in D for Piano |
| 16 | Schubert | |
| 17 | | String Quartets Nos. 4 & 14 |
| 18 | | Die schone Mullerin, Op. 25 |
| 19 | | Symphonies Nos. 5 & 9 |

2. 选择 File→Exit 关闭 Comparator

浏览两个失败的验证点：

浏览按扭的测试：

1. 双击 Verification Point（Object Properties）-Object Properties




2. 点击 File→Replace Baseline with Actual

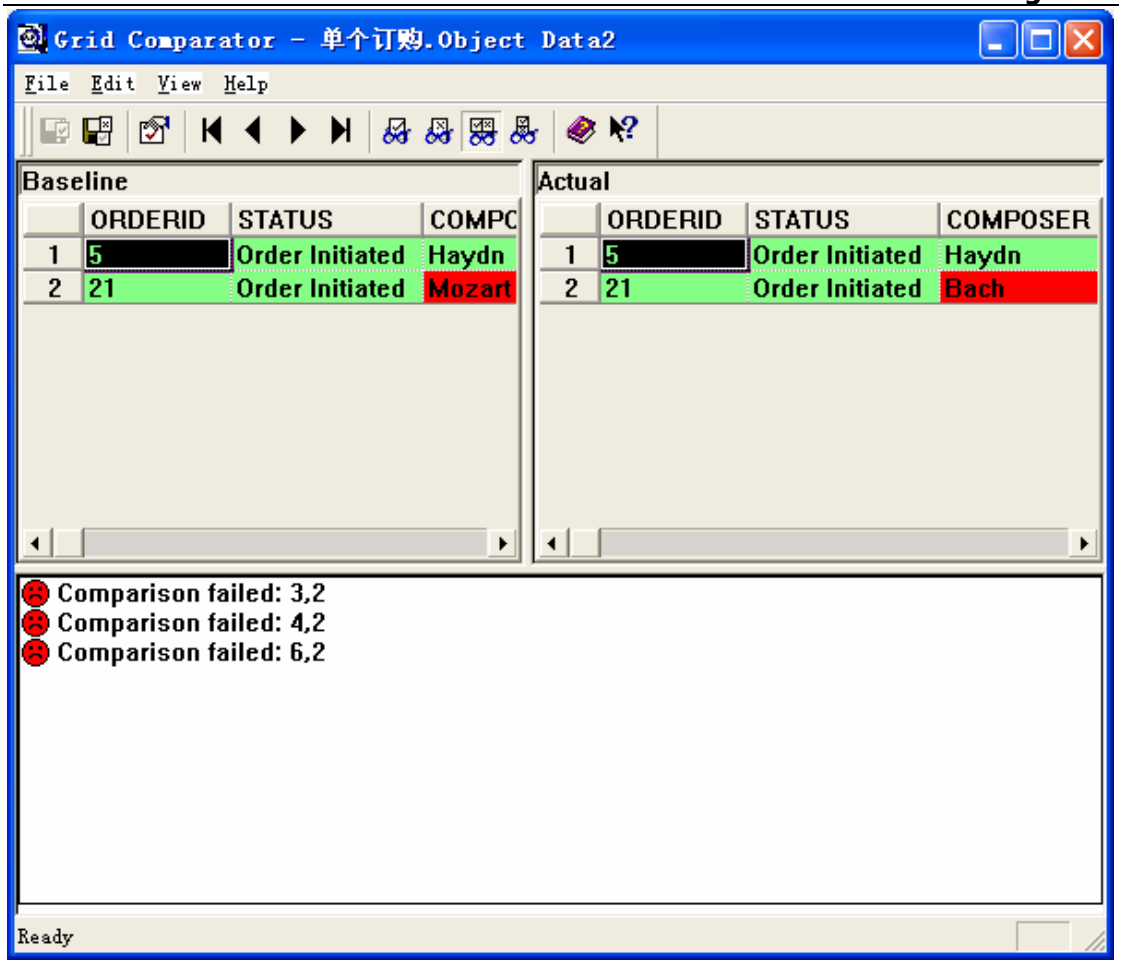
3. 点击 Yes

4. 下一次回放单个订购，该验证点就会被通过

5. 选择 File→Exit 回到 LogViewer

浏览订购数据库的测试数据

1. Verification Point（Object Data2）-Object Data




2. 选择 File→Exit 回到 LogViewer

总结：

你已经找到怎么使用 LogViewer 和 Comparators 来分析从 build 到 build 的改变，你也会发现当一个应用程序的改变是如此的简单。

- 浏览了“单个订购”的脚本结果
- 分析了每个确定的失败
- 当发生改变时，升级了 baseline fileEXAMPLE 5

## 分析和管理缺陷

该部分由 ClearQuest 来进行，公司已使用，不做过多描述

EXAMPLE 6

## 自动生成测试报告

由于没有使用 Crystal Report，所有暂时不考虑使用 EXAMPLE 7

## 加强测试脚本

**目的：**

定义“单个订购”脚本通过增加头文件和库文件，在回放脚本前检查订购信息输入数据库。

**情节：**

当你录制“单个订购”脚本时，你已经插入了四个验证点。他们中的一个已经被订购信息管理库中的新订购所代替。

如果在回放脚本前，一个新的订购没有被删除，相同的订购也将再次进入数据库，那么脚本回放将失败。为了避免这个订购副本，相同的应用程序将会在你关闭应用程序时，删除新的订购。但是你需要测试确定这些发生了。

为了完成这个，你需要加强你检查订购信息数据库资产而创造的的“单个订购”脚本。你将在“单个订购”脚本中输入一个头文件来回放脚本。

**录制和手工定义脚本：**

1、 在 Robot 中点击 File→Record GUI

2、 输入名字：暂时脚本，点击 OK

3、 点击工具栏按扭：Display GUI Insert 

4、 点击 Start Application 按扭 

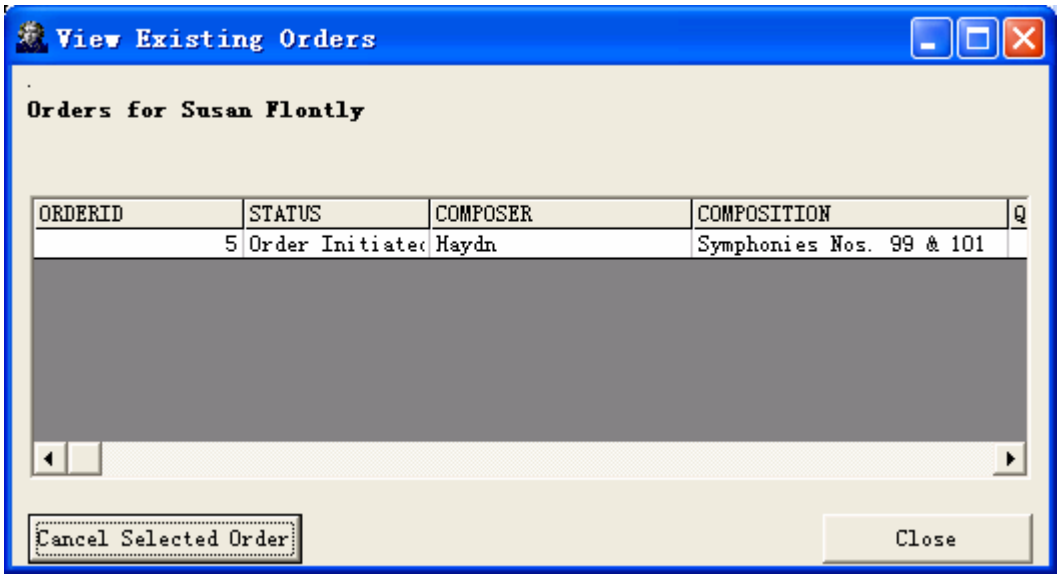
5、 Type 和 browse 选择默认的例子程序：

C：\Program File\Rational\Rational Test\Sample Application\Classic Online\ClassicA.exe

6、 点击 Open，后点击 OK

7、 以 Susan Flontly 登陆，后点击 OK

8、 点击 Order→View Existing Order Status

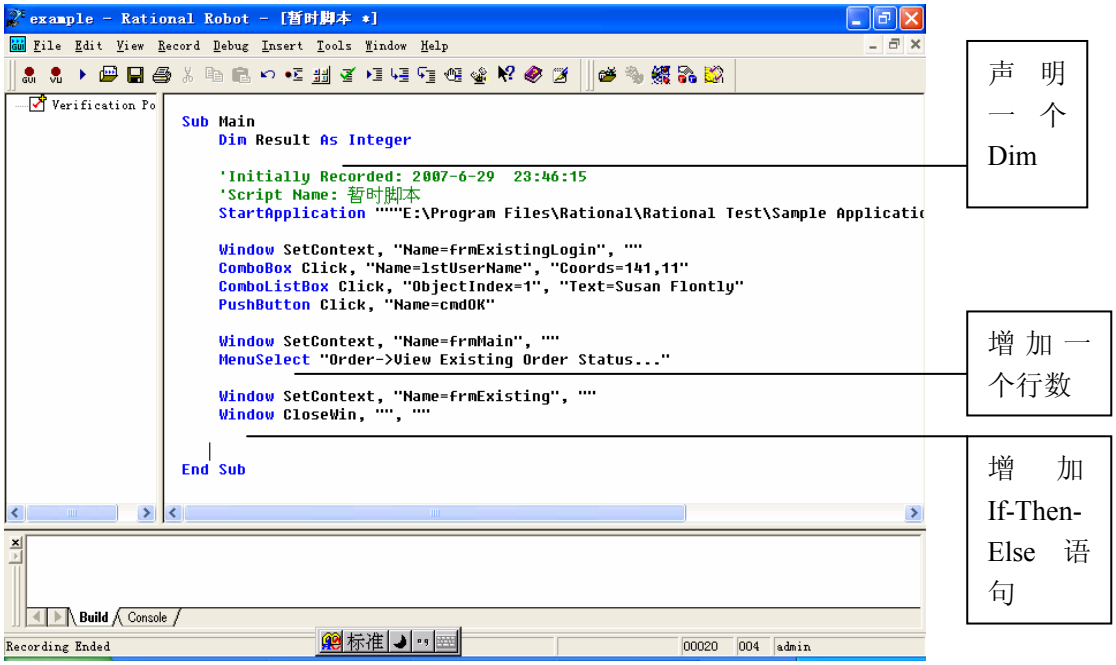



9、 关闭对话框和例子程序

10、 点击 Stop Recording

**手工定义脚本：**

1、 确定“暂时脚本”打开




2、 脚本顶部之后的第一个 Dim 申明后，增加：

Dim RowCount As Variant

3、 在 MenuSelect 命令之后，增加一个 SQABasic 命令。这个命令从 FlexigridOrders 控制中重新得到 Rows 的属性。这是属性包含了格子中的行数------这个行数包括了头行和记录行-------把

这些数放到行数的变量中：

```
Result = SQAGetProperty ( " \ ; Name=frmExisting;\;Name=Flexigridorders " , " Rows " , RowCount)
```

注意：保证该命令出现在脚本时为一行。

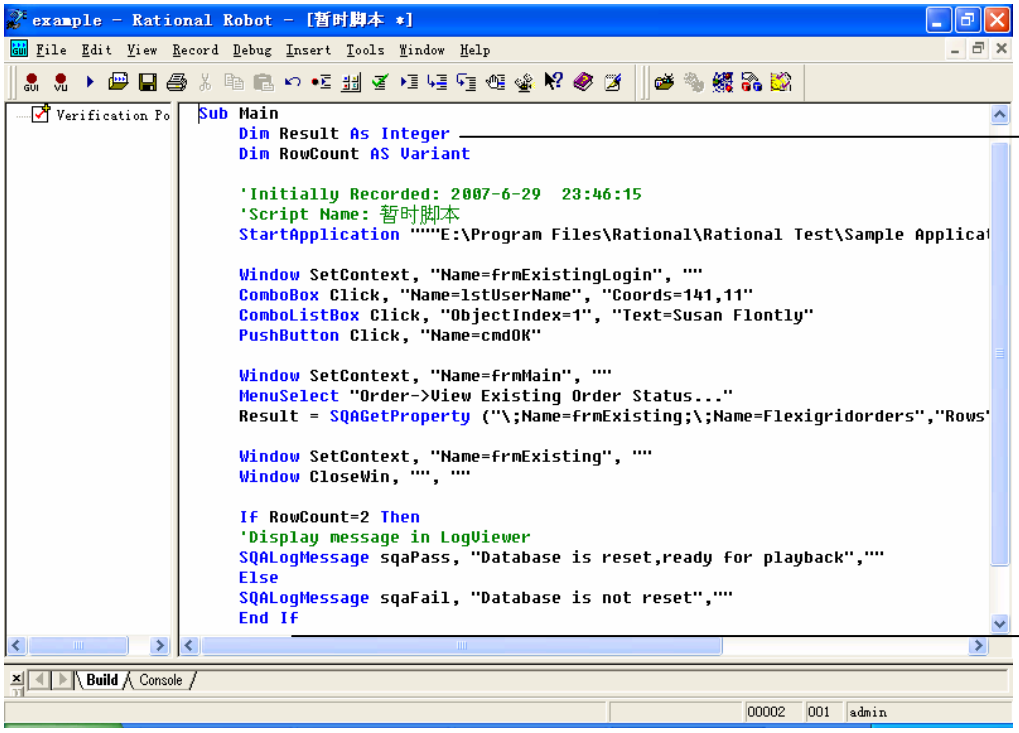
4、 在脚本结尾之前的 End Sub，加入一个“If-Then-Else”语句。如果清除成功，它将送有一个信息给 LogViewer 来阐述数据库的正确性：

```
If RowCount=2 Then
′ Display message in LogViewer
SQALogMessage sqaPass, " Database is reset,ready for playback " , " "
Else
SQALogMessage sqaFail, " Database is not reset " , " "
End If
```

5、 点击 File→Save 来保存增加的暂时脚本。

**创造一个库文件：**

1、 确定“暂时脚本”的打开




2、 选择 Dim Result As Integer 到 End Sub 的部分

3、 点击 Edit→Copy

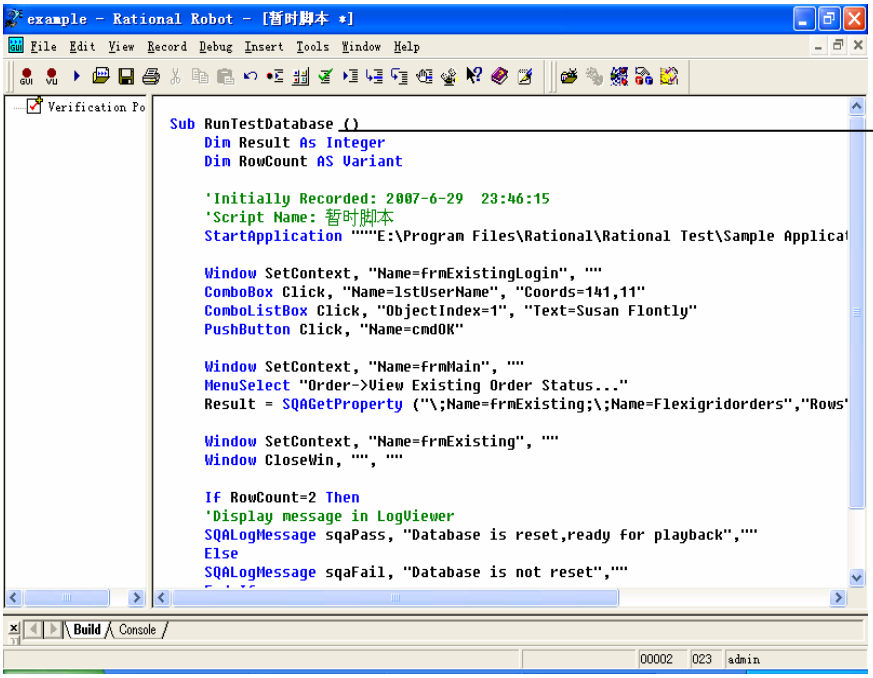
4、 开始创建库文件：File→New→SQABasic File

5、 点击 Library Source File 点击 OK

6、 点击 Edit→Paste

7、 在库文件的顶部，定义一行你通过文件程序的名字：

Sub RunTestDatabase ( )




8、 点击 File→Save

9、 定义名字 CheckDBLib，点击 Save

10、 点击 File→Compile 来编译库文件

**创建头文件：**

1、 在 Robot 中点击 File→New→SQABasicFile

2、 点击 Header File 点击 OK

3、 在头文件后面的一行声明库文件，该库文件包含了一个自过程 RunTestDatabase：

Declare sub RunTestDatabase BasicLib " CheckDBLib " ( )

4、 点击 File→Save

5、 保存为 CheckDBHeader 作为.sbh 头文件格式

6、 点击 Save

7、 点击 File→Close

**增加头文件和库文件到脚本中**

**1、** 点击 File→Open Script

**2、** 选择“单个订购”，点击 OK

**3、** 在脚本的顶部之上 Sub Main，输入：

′ $Include " CheckDBHeader.sbh " .

4、 在脚本的结尾之上 End Sub 中加入一行：

Call RunTestDatabase( )

5、 回放你的脚本看发生了什么改变

**总结**

- 录制脚本
- 捕获验证点
- 增加 SQABasic 命令行到脚本
- 创建一个头文件
- 创建一个库文件

# VS2005 Team Suite 轻松搞定白盒测试

作者：俞戴龙

**摘 要：**本文将介绍 VSTS 强大的单元测试创建功能和具体实例，结合黑盒测试与白盒测试的测试用例设计方法，有效完成单元测试，在软件研发前期就发现缺陷，从而有力保证软件质量的具体操作方法。

**关键词：**VSTS；单元测试；缺陷；软件质量

## 一、 引言

在开始本文之前，我们有必要先探讨几个问题：为什么需要做单元测试？单元测试应该让谁来完成？做单元测试的工具有哪些？

笔者就自己对于单元测试的认识，简单做如下回答：

我们熟知的“软件测试的 10 大原则”中第一条就是“测试是一个持续进行的过程，而不是一个阶段”，就是说软件测试需要尽早的参与研发投入，V&V 模型就很好的为我们描述了这个观点。根据缺陷放大理论，在白盒测试阶段发现 BUG 是投入成本最低也是最高效的。

51testing 的论坛上有一个贴，讨论过这个问题，笔者发现大多数的公司根本不进行白盒测试，或者进行白盒测试的也是开发人员完成的。更为奇怪的是，我在几年前参加的程序员考试竟然也认为应该把单元测试的任务移交给开发人员。

笔者认为，测试人员之所以选择测试，就是因为我们比开发人员更加具有对于缺陷的敏感性，并且对缺陷的认识程度远高于开发人员；另外，开发人员对于自己开发的程序具有很显著的思维定势，单元测试的职责理所应当让测试人员来完成。

单元测试的工具主流的是 XUnit，还有 C++Test、Insure++、CompuwareDevParterner、Purify、PC-Lint、LogiScope、TestBed 等

也是很强大的单元测试工具，另外基于 Java 也有自己的工具：Agitator。此外，笔者曾用高级语言调用 TCL 编译器，在 TCL 脚本语言环境创建单元测试用例也是个不错的选择。

## 二、 VSTS 单元测试简介

Team 版的 VS2005 里面包含了完整的 Test 功能，具体有：Unit Test，WebTest 和 LoadTest。这一整套的测试基本涵盖了软件开发会使用到的测试功能。

今天要介绍的是 VSTS 自带的单元测试工具（Unit Test），它最大的好处就是与开发的代码无缝连接，并且能自动创建单元测试环境，能够让不懂编程的测试人员也轻松进行单元测试。

## 三、 测试实战

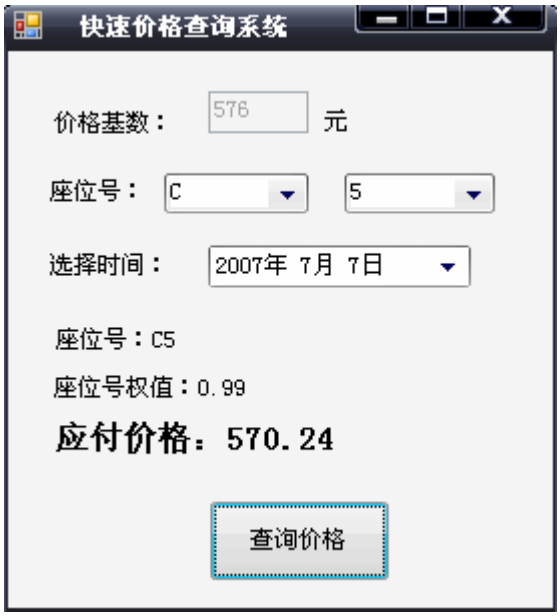
下面我们就通过一个实例来具体体验 VSTS 单元测试。

在测试过程中，将按照如下顺序进行：

被测系统简介->被测试函数分析->测试用例的编写（黑盒/白盒测试方法）->进行单元测试

### 1. 被测系统简介

被测系统是“快速价格查询系统”，通过选定座位号的两个值，系统自动算出价格（界面如下图）。“座位号”下拉框的可选范围分别为“A”～“F”和“0”～“9”




如上图所示，“应付价格”＝“价格基数”×“座位号权值”

## 2. 被测试函数分析

我们需要测试的就是权值计算的函数，开发人员在函数申明中是这样描述的：

```
        //权值计算
        /// <summary>
        /// 如果X是A、F，则靠窗，纵向权值＝1.8
        /// 如果X是B、E，则靠中间，纵向权值＝1.2
        /// 如果X是C、D，则靠走廊，纵向权值＝0.9
        /// 如果Y是0~2，则靠头部，横向权值＝1.6
        /// 如果Y是7~9，则靠尾部，横向权值＝0.7
        /// 如果Y是3~6，则靠中间，横向权值＝1.1
        /// 权值＝纵向权值*横向权值
        /// </summary>
        /// <param name="x">纵列值</param>
    /// <param name="y">横列值</param>
    static double position(char x, int y)
    {
```

……

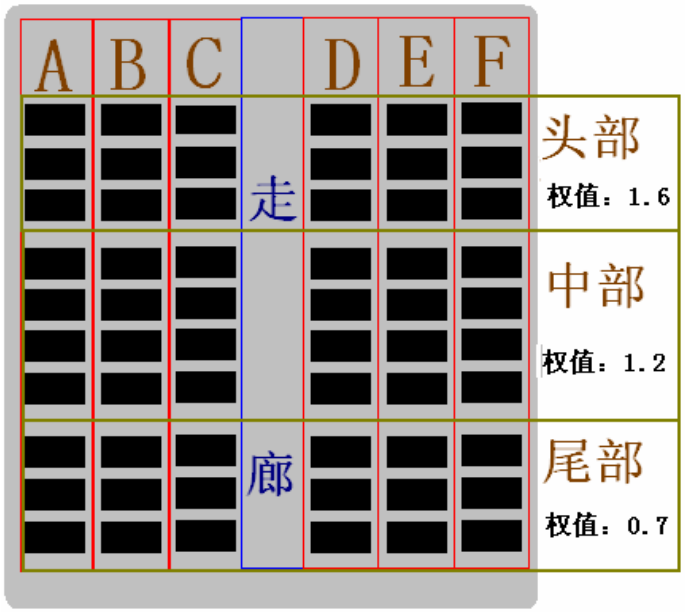
}

### 3. 测试用例的编写（黑盒测试方法）

也许有人会产生疑问：单元测试怎么可以用黑盒测试方法设计测试用例呢？黑盒测试不是做系统测试的吗？

其实我觉得测试方法都是一家，我们在设计用例的时候要秉承一个原则，那就是：测试是无穷尽的，我们要以尽量少（20%）的测试用例发现尽量多（80%）的缺陷。在设计用例的时候大可以集成天下，只要是为高效发现缺陷的用例设计思路都可以采纳，不必拘泥于特定的条条框框。我们只要把函数具体的实现方式抽象成一个黑盒，即可用黑盒设计思路。

根据开发人员提供的描述，我脑海中即刻浮现出一幅图：




| | 靠窗 | 靠中间 | 靠走廊 |
|---|---|---|---|
| 权值 | 1.8 | 1.2 | 0.9 |

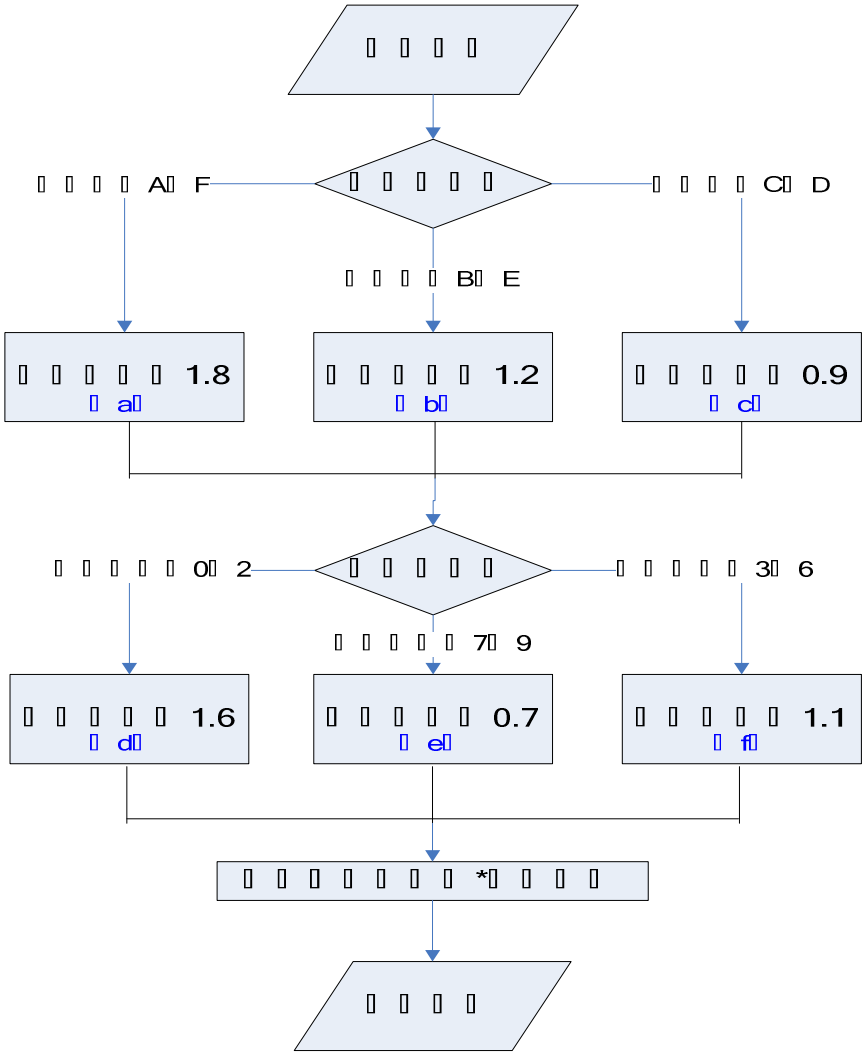
限于篇幅有限，且重点不在于用例设计上，笔者在这里象征性的用划分等价类的方法设计了 5 个测试用例：

| 序列号 | 方法 | 输入X | 输入Y | 预期输出 |
|---|---|---|---|---|
| 1 | 有效等价类 | A | 0 | 2.88 |
| 2 | 有效等价类 | B | 3 | 1.32 |
| 3 | 有效等价类 | C | 7 | 0.63 |
| 4 | 无效等价类 | 不输入 | 4 | 出错提示 |
| 5 | 无效等价类 | B | 不输入 | 出错提示 |

注：实际测试过程中可以结合边界值、正交法、错误猜测、因果图等方法设计出更多测试用例进行测试

## 4. 测试用例的编写（白盒测试方法）

白盒测试用例设计方法少不了函数流程图：




限于篇幅有限，且重点不在于用例设计上，笔者在这里象征性的用语句覆盖的方法设计了 3 个测试用例：

| 序列号 | 覆盖语句 | 输入X | 输入Y | 预期输出 |
|---|---|---|---|---|
| 1 | a、d | A | 0 | 2.88 |
| 2 | b、e | B | 7 | 0.84 |
| 3 | c、f | C | 3 | 0.99 |

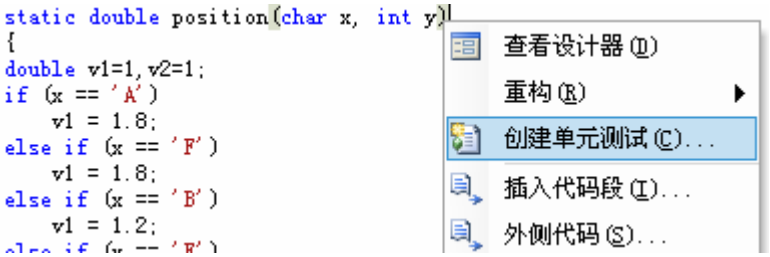
注：实际测试过程中可以结条件覆盖、判定覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖等方法设计出更多测试用例进行测试

## 5. 进行单元测试

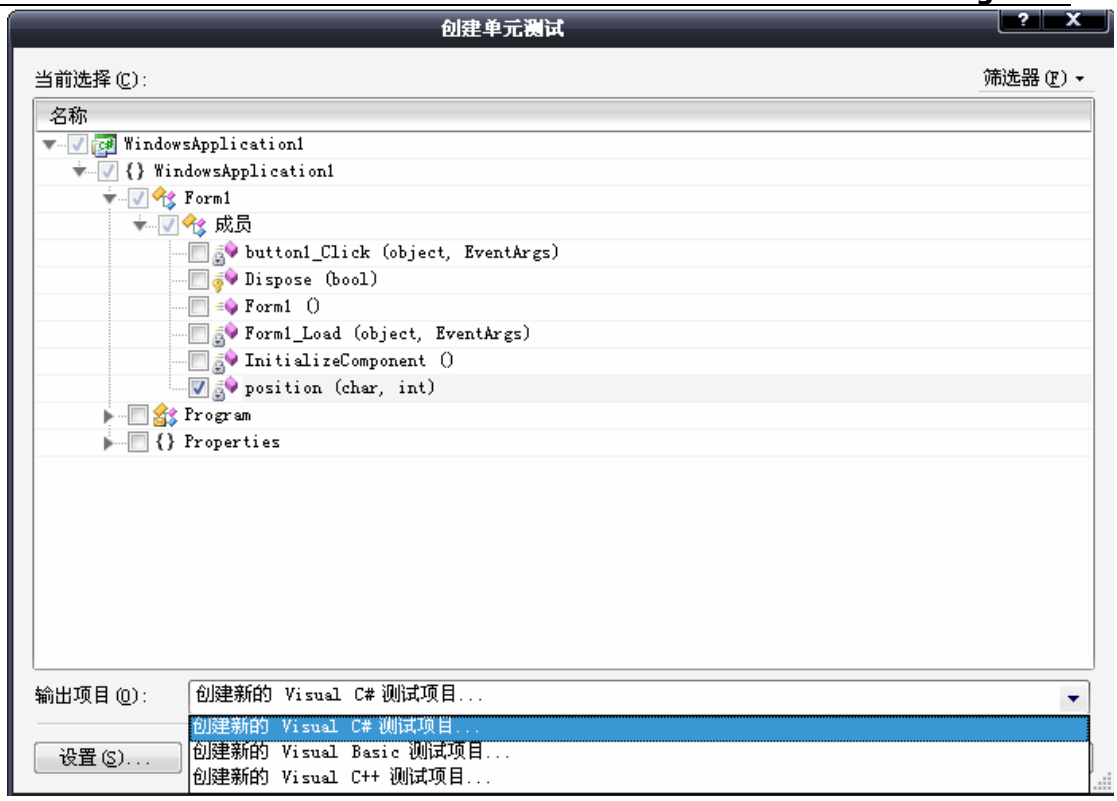
### 创建单元测试

至此，我们简单设计了 8 个测试用例，接下来就要在 VSTS 里创建单元测试了

1. 首先，鼠标留在被测函数 static double position(char x, int y)上，点击右键，弹出如下图所示菜单：



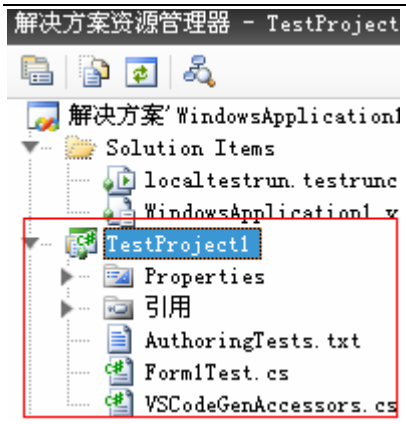

2. 选择“创建单元测试”，弹出如下对话框：




在这里我们可以看到，无论开发的代码是用何种语言写的，只要是用 VS2005 创建的，我们测试的时候，可以选择 C#、VB、C++都可以实现，这正是 VS2005 强大之处之一。

3. 勾选上我们需要测试的函数，点击“确定”，弹出如下对话框：


4. 输入测试项目名后，点击“创建”，在“解决方案资源管理器”中，我们可以看到创建成功了一个以我们刚才键入的名字命名的测试方案（如下图红框所示）




**代码分析**

在自动生成的 form1Test.cs 里，有这么一段代码：

```
        /// <summary>
        ///position (char, int) 的测试
        ///</summary>
        [DeploymentItem("WindowsApplication1.exe")]
        [TestMethod()]
        public void positionTest()
        {
            char x = '\0'; // TODO: 初始化为适当的值
            int y = 0; // TODO: 初始化为适当的值
            double expected = 0;
            double actual;
            actual =
TestProject3.WindowsApplication1_Form1Accessor.position(x,
y);
            Assert.AreEqual(expected, actual,
"WindowsApplication1.Form1.position 未返回所需的值。");
            Assert.Inconclusive("验证此测试方法的正确性。
");
        }
```

这就是单元测试的测试代码的主体部分，我们来详细分析一下：[TestMethod()]：说明了以下代码是一个测试用例，在form1Test.cs中的“#region 附加测试属性”里还有很多带有中括号的方法我们在后文再讲述：

```
char x = '\0'; // TODO: 初始化为适当的值
int y = 0; // TODO: 初始化为适当的值
```

这两句是被测函数的输入参数，需要我们去修改它的值，说白了，也就是我们输入测试用例的地方。

在这里我们可以看到：VSTS自动为x赋了初值'\0'，为y赋了初值0，我们要做的，只是更改它的值就可以了，VSTS自动为char类型加上了单引号，显示出非常人性化和智能化：

```
double expected = 0;
double actual;
```

这两句话浅显易懂，前一句话是定义了期望值和对它进行初始化，后一句话是定义了实际值。对expected进行了赋值=0实际上是在提醒我们：这里是需要测试人员在进行测试时需要更改的地方——多么的人性化！

可能有人要问了：那么实际值actual改怎么去赋值？下面的这句语句就是在对它进行赋值：

```
actual =
TestProject3.WindowsApplication1_Form1Accessor.position(x,
y);
```

太棒了，现在期望值和实际值都已经获取到了，该对他们进行比对了吧？没错，下面的这句语句就实现了这个功能：

```
Assert.AreEqual(expected, actual,
"WindowsApplication1.Form1.position 未返回所需的值。");
```

Assert在这里可以理解成断言：在VSTS里做单元测试是基于断言的测试。在MSDN网站上，有这样的描述：“可定义为“实或您相信为事实的内容”。从逻辑角度看，请考虑该语句“when I do {x}, I

expect {y} as a result”。”查阅帮助文档，可以看到有以下这些断言：

| 断言类 | StringAssert 类 | CollectionAssert 类 |
| --- | --- | --- |
| AreEqual()<br>AreNotEqual()<br>AreNotSame()<br>AreSame()<br>EqualsTests()<br>Fail()<br>GetHashCodeTests()<br>Inconclusive()<br>IsFalse()<br>IsInstanceOfType()<br>IsNotInstanceOfType()<br>IsNotNull()<br>IsNull()<br>IsTrue() | Contains()<br>DoesNotMatch()<br>EndsWith()<br>Matches()<br>StartsWith() | AllItemsAreInstancesOfType()<br>AllItemsAreNotNull()<br>AllItemsAreUnique()<br>AreEqual()<br>AreEquivalent()<br>AreNotEqual()<br>AreNotEquivalent()<br>Contains()<br>DoesNotContain()<br>IsNotSubsetOf()<br>IsSubsetOf() |

乍看起来有点摸不着头脑，举个简单的例子，你就会明白：比如说，我们在刚才的测试用例1中，输入的是(A,0)，预期输出是2.88，那么现在expected=2.88，实际值是actual，我们认为expected=actual测试才算通过，如果不相等的话，把“测试用例不通过”这句话纪录进测试结果，那么我们就把这句语句改为：

```
Assert.AreEqual(expected, actual, "测试用例不通过");
```

简单的说，断言就是我们去定义为测试通过的准则

帮助文档里有各个断言的描述：

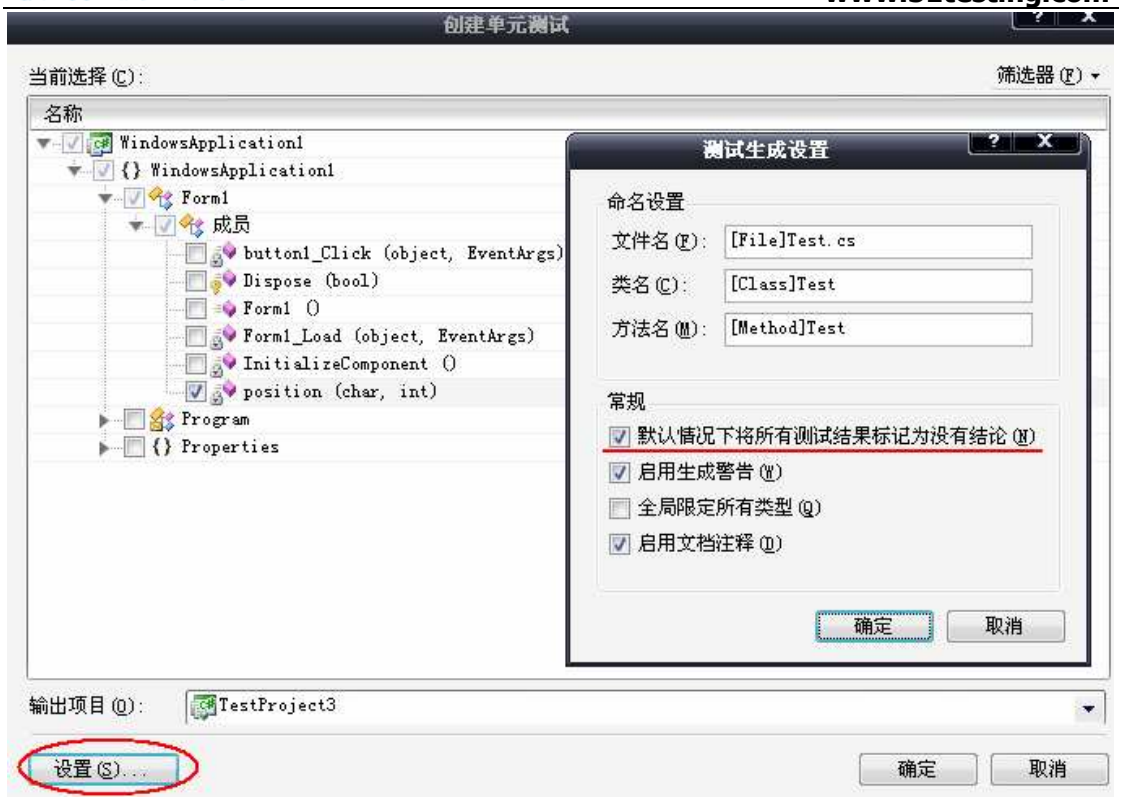
| 名称 | 说明 |
| --- | --- |
| AreEqual | 已重载。 测试指定的值是否相等；如果两个值不相等，则测试失败。 |
| AreNotEqual | 已重载。 测试指定的值是否不相等；如果两个值相等，则测试失败。 |
| AreNotSame | 已重载。 测试指定的对象是否引用不同的对象；如果两个输入内容引用相同的对象，则测试失败。 |
| AreSame | 已重载。 测试指定的对象是否都引用相同的对象；如果两个输入内容引用不同的对象，则测试失败。 |
| Fail | 已重载。 断言失败。 |
| Inconclusive | 已重载。 表示无法证明为 true 或 false 的测试结果。 |
| IsFalse | 已重载。 测试指定的条件是否为 false；如果该条件为 true，则测试失败。 |
| IsInstanceOfType | 已重载。 测试指定的对象是否为所需类型的实例；如果所需的实例不在该对象的继承层次结构中，则测试失败。 |
| IsNotInstanceOfType | 已重载。 测试指定的对象是否不是错误类型的实例；如果指定的类型在该对象的继承层次结构中，则测试失败。 |
| IsNotNull | 已重载。 测试指定的对象是否为非空；如果它为 空引用（在 Visual Basic 中为 Nothing），则测试失败。 |
| IsNull | 已重载。 测试指定的对象是否为 空引用（在 Visual Basic 中为 Nothing）；如果它不为空，则测试失败。 |

现在再来看最后一句话：

```
Assert.Inconclusive("验证此测试方法的正确性。");
```

对照上表就很容易看懂了，是指“表示无法证明为 true 或 false 的测试结果。”

如果在创建单元测试之初点击“设置”，把“默认情况下把所有测试结果标记为没有结论”这个钩去掉（见下图）




那么创建出来的单元测试就没有Assert.Inconclusive("验证此测试方法的正确性。");这句话。没关系，我们手动把它删除

**进行测试初体验**

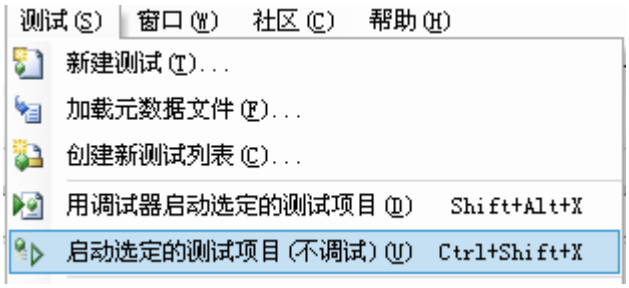
根据刚才的语句分析，我们现在就更改测试代码，输入第一个测试用例：

| 方法 | 输入X | 输入Y | 预期输出 |
|---|---|---|---|
| 有效等价类 | B | 7 | 0.84 |

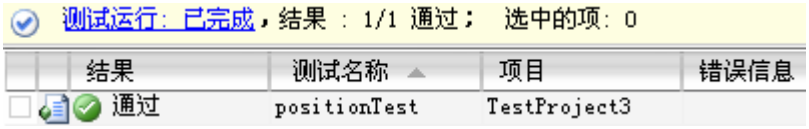
```
[DeploymentItem("WindowsApplication1.exe")]
[TestMethod()]
public void positionTest()
{
    char x = 'B'; // TODO: 初始化为适当的值
    int y = 7; // TODO: 初始化为适当的值
    double expected = 0.84;
    double actual;
    actual =
```

```
TestProject3.WindowsApplication1_Form1Accessor.position(x,
y);
                Assert.AreEqual(expected, actual, "实际值：
"+actual+" "+"期望值："+expected);
        }
```

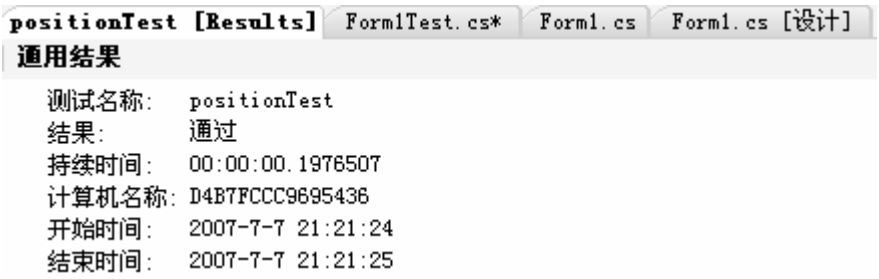
用例输完了，我们先试一下，看看结果：点击菜单栏“测试”->“启动选定的测试项目（不调试）”（如下图）




然后我们在最下面的“测试结果栏”可以看到如下图所示的测试结果：

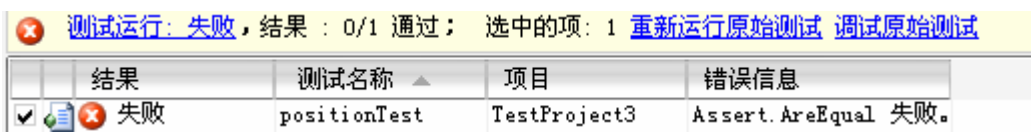



双击它可以看到更详细的测试结果（如下图）：




测试成功通过了，耶！真是激动人心！

不相信自己？好的，那我们改一下代码，把double expected = 0.84;改成double expected = 8.88;，然后再执行一遍测试，看看是什么结果：




双击它：

positionTest [Results] | Form1Test.cs | Form1.cs | Form1.cs [设计]

**通用结果**

测试名称: positionTest
结果: 失败
持续时间: 00:00:00.7443047
计算机名称: D4B7FCCC9695436
开始时间: 2007-7-7 21:34:00
结束时间: 2007-7-7 21:34:02

**错误信息**

Assert.AreEqual 失败。应为: <8.88>，实际为: <0.84>。实际值：0.84 期望值：8.88

**错误堆栈跟踪**

在 TestProject3.Form1Test.positionTest() 位置 D:\WindowsApplication1\TestProject3\Form1Test.cs:行号 86

哈哈，VSTS 明察秋毫，这回相信自己了吧！

### 输入测试用例和进行测试

现在我们要把刚才的 8 个测试用例全部输入测试用例：

| 序列号 | 方法 | 输入X | 输入Y | 预期输出 |
|---|---|---|---|---|
| 1 | 有效等价类 | A | 0 | 2.88 |
| 2 | 有效等价类 | B | 3 | 1.32 |
| 3 | 有效等价类 | C | 7 | 0.63 |
| 4 | 无效等价类 | 不输入 | 4 | 出错提示 |
| 5 | 无效等价类 | B | 不输入 | 出错提示 |

| 序列号 | 覆盖语句 | 输入X | 输入Y | 预期输出 |
|---|---|---|---|---|
| 1 | a、d | A | 0 | 2.88 |
| 2 | b、e | B | 7 | 0.84 |
| 3 | c、f | C | 3 | 0.99 |

怎么输呢？很简单，只要把刚才那段测试代码复制、粘贴一下，更改输入值和预期值就可以了。需要注意的是测试函数名不能相同。我这里写两个测试用例来举个例子：

| 方法 | 输入X | 输入Y | 预期输出 |
|---|---|---|---|
| 有效等价类 | A | 0 | 2.88 |
| 有效等价类 | B | 3 | 1.32 |

测试代码：

```
[TestMethod()]
public void positionTest1()
{
    char x = 'A'; // TODO: 初始化为适当的值
    int y = 0; // TODO: 初始化为适当的值
    double expected = 2.88;
```

```
            double actual;
            actual =
TestProject3.WindowsApplication1_Form1Accessor.position(x,
y);
            Assert.AreEqual(expected, actual, "实际值：
"+actual+" "+"期望值："+expected);
        }

        [TestMethod()]
        public void positionTest2()
        {
            char x = 'B'; // TODO: 初始化为适当的值
            int y = 3; // TODO: 初始化为适当的值
            double expected = 1.32;
            double actual;
            actual =
TestProject3.WindowsApplication1_Form1Accessor.position(x,
y);
            Assert.AreEqual(expected, actual, "实际值：" +
actual + " " + "期望值：" + expected);
        }
```

就是这样，把所有的测试用例全部输入后，进行测试，根据测试结果，填写《测试报告》。

大功告成，让我们举杯庆祝一下吧！

## 四、 “附加测试属性”的实践

在 Form1Test.cs 中有 

这么一个功能，这又是 VSTS 单元测试之强大的地方。

我们可以点开它，点开后我们看到如下的代码：

```
        #region 附加测试属性
        //编写测试时，可使用以下附加属性:
        //使用 ClassInitialize 在运行类中的第一个测试前先运行代码
        //[ClassInitialize()]
        //public static void MyClassInitialize(TestContext testContext)
        //{
        //}
        //使用 ClassCleanup 在运行完类中的所有测试后再运行代码
        //[ClassCleanup()]
        //public static void MyClassCleanup()
        //{
        //}
        //使用 TestInitialize 在运行每个测试前先运行代码
        //[TestInitialize()]
        //public void MyTestInitialize()
        //{
        //}
        //使用 TestCleanup 在运行完每个测试后运行代码
        //[TestCleanup()]
        //public void MyTestCleanup()
        //{
        //}
        #endregion
```

这是干什么的呢？呵呵，其实它已经说的很明白了，这是“附加测试属性”。默认都是被注释掉的，只要我们取消注释就可以使用了。具体的属性有：[ClassInitialize()]、[ClassCleanup()]、

[TestInitialize()]、[TestCleanup()]。他们的作用在它们上面的注释里都已经讲的很明白了，笔者在这里就不展开叙述了。

微软之所以加进来这么个功能，以我之所见，是为了加大测试的灵活性。

我举一例，大家就会看得很明白：现在我们打算把测试报告输出到外部文档去，这样方便日后的缺陷跟踪。可以修改代码如下：

```
#region 附加测试属性
//编写测试时，可使用以下附加属性:
//使用 ClassInitialize 在运行类中的第一个测试前先运行代码
[ClassInitialize()]
public static void MyClassInitialize(TestContext testContext)
{
    StreamWriter m_streamWriter = new StreamWriter(@"C:\log.txt", true);
    m_streamWriter.Write("Test Time,");
    m_streamWriter.WriteLine(DateTime.Now);
    m_streamWriter.Flush();
    m_streamWriter.Close();
}
//使用 ClassCleanup 在运行完类中的所有测试后再运行代码
[ClassCleanup()]
public static void MyClassCleanup()
{
    StreamWriter m_streamWriter = new StreamWriter(@"C:\log.txt", true);
    m_streamWriter.WriteLine("");
```

```
                m_streamWriter.Write("Total Passed:,");

m_streamWriter.WriteLine(Convert.ToString(Passed) + ",");
                m_streamWriter.Write("Total NotPassed:,");

m_streamWriter.WriteLine(Convert.ToString(NotPassed) +
",");
                m_streamWriter.Flush();
                m_streamWriter.Close();
            }
            //使用 TestInitialize 在运行每个测试前先运行代码
            [TestInitialize()]
            public void MyTestInitialize()
            {
                StreamWriter m_streamWriter = new
StreamWriter(@"C:\log.txt", true);
                if (actual == expected)
                {
                    m_streamWriter.WriteLine("Test Passed");
                    Passed++;
                }
                else
                {
                    m_streamWriter.Write("Test NotPassed,");
                    m_streamWriter.Write("Input X:,");
                    m_streamWriter.Write(Convert.ToString(x)
+ ",");
                    m_streamWriter.Write("Input Y:,");
                    m_streamWriter.Write(Convert.ToString(y)
```

```
+ ",");
                    m_streamWriter.Write("Expected:,");

m_streamWriter.Write(Convert.ToString(expected) + ",");
                    m_streamWriter.Write("Actual:,");

m_streamWriter.Write(Convert.ToString(actual));
                    m_streamWriter.WriteLine("");
                    NotPassed++;
                }
                m_streamWriter.Flush();
                m_streamWriter.Close();
            }
            //使用 TestCleanup 在运行完每个测试后运行代码
            //[TestCleanup()]
            //public void MyTestCleanup()
            //{
            //}
               #endregion
```

同时引用：`using System.IO;`

申明变量：

```
static int Passed=0;
static int NotPassed=0;
static double expected;
static double actual;
static char x;
static int y;
```

并且把测试用例里的对于 x、y、actual、expected 的申明删除

这时候我们再点击进行测试后，就会在 c:\下面生成 log.txt，内容如下：

Test Time,2007-7-7 23:29:57

Test Passed

Test NotPassed,Input X:,B,Input Y:,3,Expected:,1.31,Actual:,1.32

Test Passed

Test NotPassed,Input X:,A,Input Y:,0,Expected:,8.88,Actual:,2.88

Test Passed

Total Passed:,3,

Total NotPassed:,2,

可能大家在以上文本或者代码中感到奇怪：为什么要加进去那么多的逗号？

原因在此：把 log.txt 更名为 log.csv，那么，就可以用 excel 来打开它（如下表）：

| | | | | | | |
|---|---|---|---|---|---|---|
| Test Time | 2007-7-7 23:29 | | | | | |
| Test Passed | | | | | | |
| Test NotPassed | Input X: | B | Input Y: | 3 | Expected: | 1.31 |
| Test Passed | | | | | | |
| Test NotPassed | Input X: | A | Input Y: | 0 | Expected: | 8.88 |
| Test Passed | | | | | | |

| | |
|---|---|
| Total Passed: | 3 |
| Total NotPassed: | 2 |

这样清晰多了吧？而且更加易于维护

笔者在这里只是拿外部文件的输出作为一个例子，“附加测试属性”的其它有意义的用途欢迎大家共同探讨

## 结束语

VSTS 的单元测试的功能之强大由此可见一斑，本人更希望以此文能召唤起更多的国内公司能更加重视软件测试，更加规范软件测试，把我国的软件测试行业做强、做精！

**参考文献：**

①郑人杰.计算机软件测试技术.清化大学出版社，1990.

②古乐.软件测试技术概论.清化大学出版社，2004.

# 软件测试新手的感受

作者:牛牛_82

**摘　要：**测试人员的工作不只是要找出 BUG,而且是更大程度上要提出更多的解决 BUG 的方法和建议。在不能提出方法和建议的时候，要尽量详细和清楚的描述 BUG 产生的前后所做的操作。能够重现 BUG,也是测试人员的基本职责。做为一名测试人员应该注重知识的全面的学习和积累，能够很好的和开发人员、实施人员进行沟通，把好软件的质量关。

**关健词：**测试人员；BUG

“软件测试”这一职业最近几年在 IT 行业越来越引起大家的重视。从 05 年下半年即将毕业开始，我就特别关注软件测试这一行业，一直希望可以找到一份软件测试的工作。

从 06 年五月份开始参加工作，第一份工作并不是软件测试而是“软件实施”或者称作“技术支持”，然后才转向了软件测试。具体的说从事软件测试这一行业才刚刚半年之久，　所以对自己的东西不能冠以软件测试行业的一些名词，只能说是一个刚入行的测试人员的一些感受和体会。

## 一、 测试人员应该注重业务知识的学习

测试的时候应该能够从用户的角度出发，能够分权限，分角色按照业务流程来进行测试。

现在的软件开发大多都是面向对像基于 WEB 结构的开发，所以在测试的时候就是要分角色、分权限，根据业务知识和业务流程按照不同的用户、不同的岗位进行测试。只有这样才能够测试出来系统在权限和角色的划分中的问题。

由于本人从事过技术支持所以更想能够有利于从用户的角度出发来对软件进行测试。其实 IT 行业的很多工作都是相通的。

二、 与开发人员的沟通

测试人员的工作不只是要找出 BUG,而且是更大程度上要提出更多的解决 BUG 的方法和建议。在不能提出方法和建议的时候要尽量详细和清楚的描述 BUG 产生的前后所做的操作。能够重现 BUG,也是测试人员的基本职责。开发人员经常由于工作比较紧，所以更希望测试人员能够简洁、明了的来描述 BUG。如果能够给出他们合理的修改 BUG 的建议和方法，他们也会对测试人员刮目相看，从而建立更好的合作关系。

三、 应该注重数据库知识和简单代码的学习

如果能够了解到更多的数据在数据库中的存储情况，将有利于查找测试原因更有利于和开发人员的沟通。例如，在一次的测试过程中发现数据字典，增加功能无效，就是增加过的记录在页面上看不到，通过对数据库的查询发现，记录已经正确存储在数据库中，只是页面上没有表现出来。当把这一情况告诉开发人员时，开发人员很快就分析出了错误的原因，并能够在最短的时间内对程序进行修改。

代码的学习，最基本的是可以看懂一些简单的代码。进行软件的性能测试时，需要从代码的合理性、是否造成内存的浪费、是否存在不合理的竞争等多个方面来考虑软件的性能问题。如果以后想要往白盒测试方向发展的话，那么代码的学习就是必须的了，而且还要学习大量的复杂的代码。

四、 需要学会区分 BUG 的严重程度和可修复程度

有时由于缺少对开发知识的了解，测试出一个 BUG 之后，不知道修复的难易程度和产生错误的原因。开发人员会说出这个不是一个 BUG,就会对这个 BUG 进行关闭，而以后用户会认为它确实是存在

的一个 BUG .盲目的听从开发人员我想也是初学测试者的一个误区。做为一名测试人员其实有时需要从经济学的角度考虑 BUG 的严重程度和可修复程度对整个项目所带来的影响。

五、 测试方法和测试流程的规范

我所在的公司项目比较多，项目比较紧。很多项目都是项目初期的时候还写测试用例进行测试方法的讨论，后来由于一些实际情况把这些用例和方法都丢弃了。这样导致在测试的过程中都是随意的进行测试。程序的有些地方都测试过很多遍，而有一些地方却没有测试到，造成了大量的重复劳动和低效率的劳动。

六、 测试工具的学习

参加测试半年了，测试工具的学习不是很多，一般的功能方面的测试，还是通过简单的编写测试用例来进行测试。只有在性能测试时偶尔会用到 LOUNDERRUNER。

以上是本人做测试几个月以来的一些感受，也许称不上文章但只有这样写下来我才知道，自己干测试这几个月以来学到了哪些东西。我会努力用自己的能力来赢得别人的尊重。

# Bug 不能重现的原因分析及其对策

作者：刘洪刚

**摘　要：**本文简要分析了无法重现的 Bug 的可能产生原因，包括环境不一致、缺少最准确的描述和浏览器的不当设置。针对这些原因，本文给出了相应的对策。通过这些措施，可以重现许多以前认为不可重现的 Bug。

**关键词：**重现；Bug；环境

在测试人员提交 bug 后，最不希望看到的结果是它们被标记为 INVALID，尽管你坚信这一定是 Bug。开发人员查看了 bug 的 Description 后，最不希望的结果是你无法重现它们，尽管他使用了所有可能的方法去重现它。一旦出现这样的情况，测试人员会很伤心，开发人员也会对测试人员有意见。这就使得关系本来就不怎么融洽的测试人员和开发人员之间的关系更加紧张。这对于关系紧张的测试人员和开发人员来说，无异于是火上浇油。

为了减少这种情况的出现，非常有必要分析一下 Bug 不能重现的原因。根据我的测试经验，Bug 不能重现的原因有：

一、 环境不一致

这是 bug 不能重新的最主要的原因。在开发人员认为这是无效的 bug 里面，估计至少有 80%的 Bug 是因为环境不一致的原因造成的。这既包括开发环境和测试环境的不一致，也包括开发环境和系统的实际运行环境不一致。

Bug 产生是有一定原因的，它的重现也需要一定的环境。如果没有相应的环境，那么你可能很难这个 Bug。

从广义上来说，保证或影响系软件的任何因素都是环境。例如，硬件的配置、软件的设置、网络的带宽、网速等。通常，环境中的软件因素有：系统的 Build、Application Server 的类型和 Version、

Operation System 的语言和 Version、浏览器的语言和 Version 等。

下面是我的一些有趣的经历：某个 Bug 只出现在 WebSphere 6.0.2.15 上，按照开发人员的要求，把 WebSphere 升级到 6.0.2.17 后，此 Bug 就自动消失了。因此，此 Bug 是因为 WebSphere 的版本不一致引起地。

几个图片在某个 Build 上莫名其妙地消失了，刚开始怀疑是开发人员修改别的 Bug 而引起的错误。后来经过仔细认真地测试才发现，原来是操作系统的语言搞的“鬼”：测试人员使用的机器的操作系统语言是简体中文，开发人员使用的是繁体中文。

Bug 的 Description 里面缺少重现 bug 的最准确的操作,下图是测试系统在增加一个 Role 时的页面：

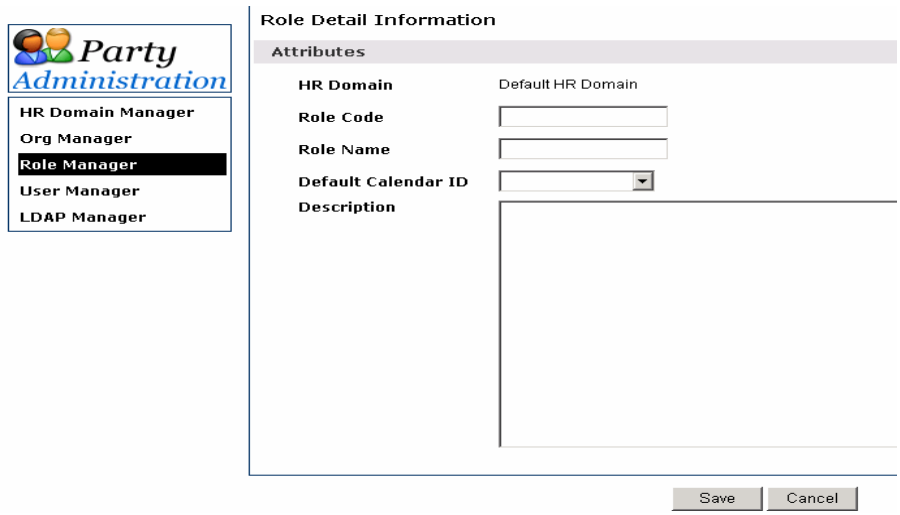



图 1：增加 Role 的页面

测试人员在输入某些数据、然后点击 Save 后，“一不小心”就出现了 java.lang.NullPointerException 的错误。说一不小心，是因为这是测试人员在无意中发现了，并且出现这个错误后，你再也无法增加任何 Role 了（当然也无法重现此 Bug 了）。最糟糕的是，别的与 Role 有关的许多功能点也无法验证。尽管不知道为什么发生了这个错误，也无法重现此 Bug，但考虑到此 Bug 的严重性，测试人员还是把此 Bug 提交到 Bug 库里去了（事后证明这是非常明智的举动）。

在测试下一个 Build 的时候，我要求测试人员重点关注此 Bug。后来在 DBA 的帮助下找到了产生此 Bug 的真正原因：输入 Role Code 后，如果 Role Name 为空，页面没有进行检查（前台没有检查）；但点击 Save 后，数据需要保存到数据库时，Role 所在的 Table 要求 Role Code 和 Role Name 都不能为空。因此就出现了 java.lang.NullPointerException 这个错误信息。

找到此 Bug 发生的原因后，我建议开发人员把 Role Code 当作 Role Name，如果用户输入 Role Code 而没有输入 Role Name。经过测试，此 Bug 就再也不会出现了。

下面是此 bug 的完整历史记录：

[Step]

1. Add one Role on Role Manager;
2. Click on "Save" button to save it.

[Result & Memo]

One error message appears: java.lang.NullPointerException. As a result, I could not do any operation on “Role Manager” tab. This bug appears even after I restart the application server.

------- *Additional Comment #1 From Jim 2006-11-25 14:08* -------

I cannot reproduce this bug. Please describe testing scenario in details + attach error logs.

------- *Additional Comment #2 From Mike 2006-12-15 10:28* ------

[Step]

1. On ‘Role Manager’ tab, Click on ‘Add’ button;
2. Enter one valid role code and click on ‘Save’ button. Note: you are forbidden to enter any value for ‘Role Name’ in order to reproduce this bug.

Now there is one error message on the page when clicking on 'Role Manager' tab.

[Memo]

According to the schema TBCN_WF_ROLE, the 'Name' column is forbidden to be null. But there is no code to check it when such occasion happens. For detail information, please view the attached file.

To avoid this bug happening, I suggest you set the value of "Role Code" as "Role Name" if such occasion happens.

------- *Additional Comment #3 From Jim 2006-12-17 18:06* -------

Fixed in build # >= 484

二、 浏览器的不当设置

对于 Web 测试来说，IE 的设置又是对重现 Bug 起着重要的作用。下面的几个图片说明了浏览器的几个关键设置：

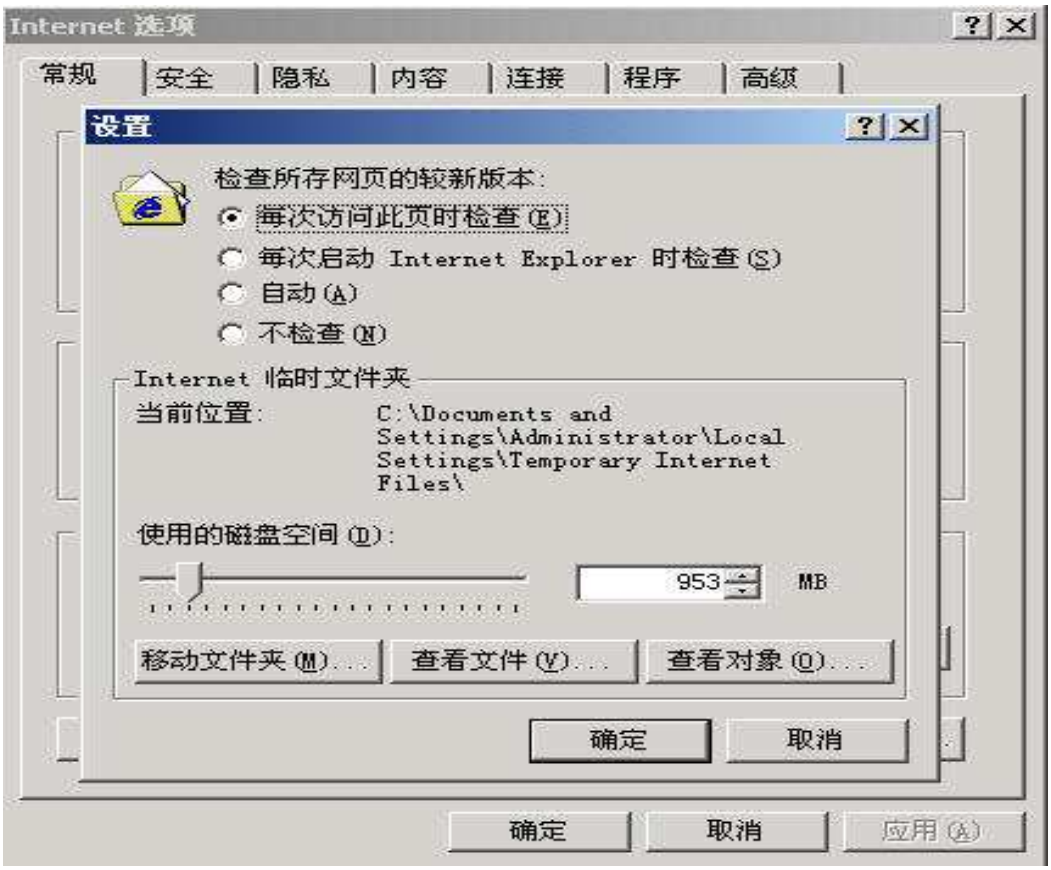



图 2：Internet 临时文件的设置

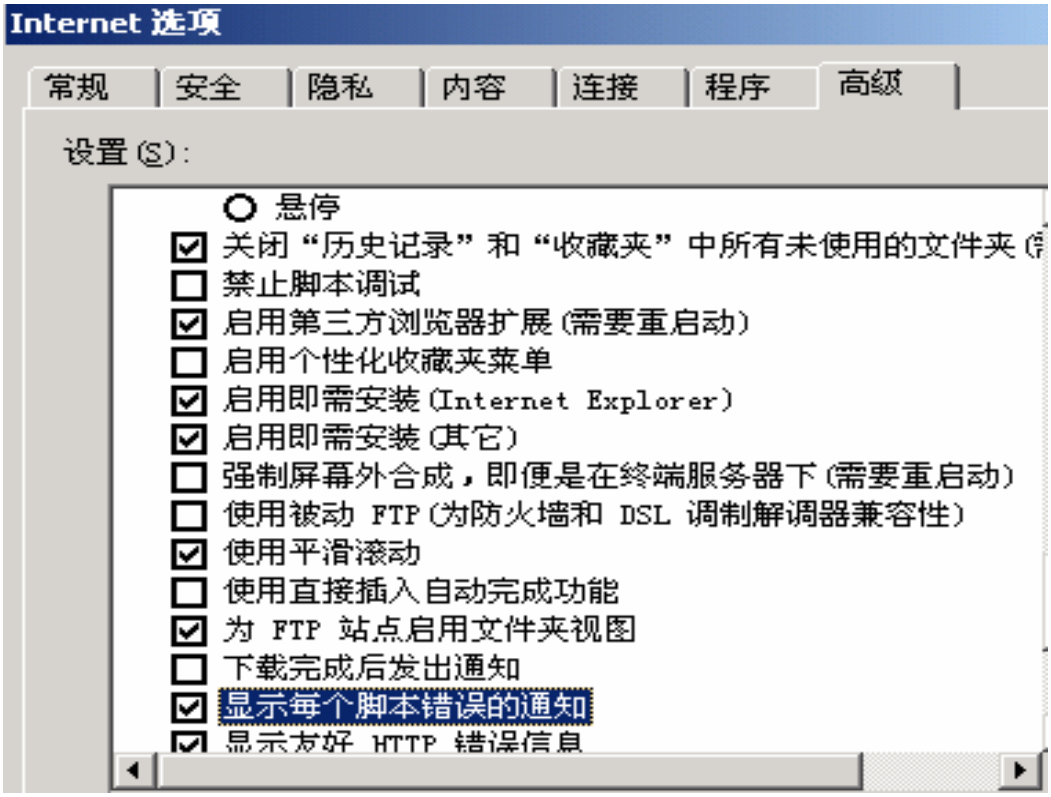


图 3：“高级”选项的设置

说明：在“高级”选项中，对测试起重要作用的是“禁止脚步调试”（不选择，即前面不要有勾）和“显示每个脚本错误的通知”（选择它，即前面要出现勾）。

这些设置对重现有关脚本错误的 Bug 起着重要的作用。根据这些脚本错误的通知，开发人员可以快速发现代码中的错误，并及时修复它。

三、 内存泄露

某些系统长期运行后出现运行速度慢、反应迟钝等现象，其中一个主要的原因就是内存泄露。有的开发人员没有养成及时回收内存的习惯，结果系统在不断地申请内存空间，却没有一点内存释放。这类错误在短期内不会出现，但当系统长期运行时就会出现，并且由此会引发一系列的问题。此类问题只有经过长时间的运行测试，才可能会被发现。

四、 函数接口类型不匹配

此类错误难以重现，并且难以发现它的真正原因，一般只有在查看源代码后才能发现。某些类型的数据会被系统自动转换，一般也不会出现错误；但有些数据被截断或被强制转换成另外一种数据类型时，会出现一些潜在的错误。在集成测试时此类错误最有可能发生。

既然分析出了这些 Bug 不能重现的原因，我们就可以对症下药了：

1. 测试人员要有重视测试环境的意思，并在 Bug Report 里面增加对测试环境的准确描述，特别是影响重现此 Bug 的那些环境因素。

2. Bug 的 Step 要准确说明操作步骤。为了重现一个 Bug，测试人员可能需要对几个 Build 进行连续跟踪、测试和定位产生这个 Bug 的最根本原因。

3. 正确设置浏览器的选项。

4. 对性能有要求的软件或系统一定要进行长期负荷测试(Loading Testing)，以发现内存泄露等需要长期运行才能出现的问题。根据微软的测试经验，如果软件能通过 72 小时的强力测试，则该软件 72 小时后出现问题的可能几率微乎其微。因此只需对软件进行 72 小时的强力测试即可。

5. 集成测试时一定要注意函数接口类型是否匹配。

6. 测试人员要与开发人员、DBA 等保持良好的关系。遇到问题要及时、主动与他们沟通，听取他们的意见。在他们的帮助下，你可以更容易地找到问题的关键所在之处。

根据上面的这些建议，我相信大多数不能重现的 Bug 都可以重现了。当然由于测试的系统的开发语言、开发平台等因素的不同，恕笔者不能一一列举出无法重现的 Bug 发生所有原因。如果还是遇到某些严重的、却又无法重现的 Bug，那么也不必惊慌，你可以按照下面的操作去查找产生 Bug 的原因：

1. 积极回忆 Bug 的症状和所有的环境因素，一丝一毫的细节都不要错过。

2. 与开发人员、DBA、系统设计人员、项目经理等一起分析那些环境因素，根据以往的经验分析影响此 Bug 重现的重要因素，并在相同的环境上安装同样的系统进行测试，以验证所做的猜测。

另外，对于某些无法重现、但严重程度不是很高的 Bug，可以暂时只作记录、而不必花费大量的人力和物力去分析。如果下次又出现了，那么根据发生的频率再去分析是否需要跟踪此 Bug。如果需要跟踪它，那么在它又出现后一定要立刻对当时的环境进行截图，如错误信息、界面、日志等。这样也利于开发人员定位、分析它，从而准确、快速地修复它。如果条件允许，测试人员应立即保护现有环境，并邀请相关的开发人员和系统分析人员一起研讨产生此问题的原因和解决方法。

**参考文献：**

①Lydia Ash、李昂. Web 测试指南.机械工业出版社，2004 年.

②陈宏刚.微软开发的科学与艺术.电子工业出版社，2002 年.

# CUnit Framewor 介绍

作者：灵儿

**摘　要：**继 Junit CppUnit 的成功后，C 语言(C/C++)环境下也出现了开发源码的白盒测试用例 CUnit。CUnit 以静态库的形式提供给用户使用，用户编写程序的时候直接链接此静态库就可以了。它提供了一个简单的单元测试框架，并且为常用的数据类型提供了丰富的断言语句支持。本文通过对 Cunit 的实际使用，写下一点心得。

**关键字：**CUnit；单元测试；TestCase

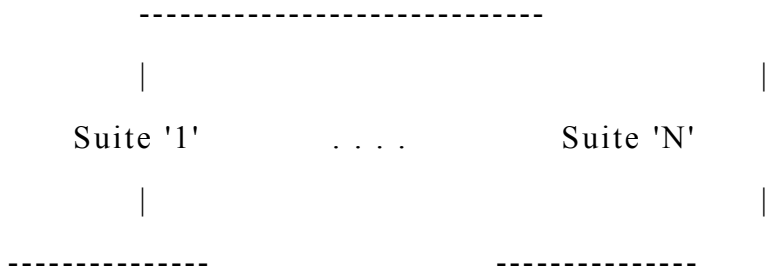
## 1. CUnit Framework 介绍

继 Junit CppUnit 的成功后， c 语言(C/C++)环境下也出现了开发源码的白盒测试用例 CUnit。 CUnit 以静态库的形式提供给用户使用，用户编写程序的时候直接链接此静态库就可以了。它提供了一个简单的单元测试框架，并且为常用的数据类型提供了丰富的断言语句支持。下面介绍一下 CUnit 结构框架和具体使用：

### 1.1 结构框架

在 CUnit 的主页上可以看到对他结构简单描述

Test Registry



| | | |

Test '11' ... Test '1M' Test 'N1' ... Test 'NM'

CUnit 的测试是单线程启动，只能注册一个测试用例 Test Registry， 一次测试（Test Registry）可以运行多个测试包（Test Suite），而每个测试包可以包括多个测试用例（Test Case），每个测试用例又包含一个或者多个断言类的语句。具体到程序的结构上，一次测试下辖多个 Test Suite，它对应于程序中各个独立模块；一个 Suite 管理多个 Test Case，它对应于模块内部函数实现。每个 Suite 可以含有 setup 和 teardown 函数，分别在执行 suite 的前后调用。

注册一个测试用例(如果已经注册了你可以 cleanup 掉然后重新注册使用)然后 CU_add_suite 增加你的模块然后 CU_add_test 再在你的模块下挂载你的模块内的测试函数。所有的挂载完毕后，调用你想使用的界面进行测试。

### 1.2 测试模式

下面是四种测试模式：

1. Automated Output to xml file Non-interactive
2. Basic Flexible programming interface Non-interactive
3. Console Console interface (ansi C) Interactive
4. Curses Graphical interface (Unix) Interact

注意 1，2 是没有交互功能的，4 的 Unix 下的我是 windows 用户， 选择第 3 个介绍 console 而且 console 是可以人机交互的。

### 1.3 测试基本流程

使用 CUnit 进行测试的基本流程如下所示：

1) 书写代测试的函数（如果必要，需要写 suite 的 init/cleanup 函数）
2) 初始化 Test Registry - CU_initialize_registry()
3) 把测试包（Test Suites）加入到 Test Registry - CU_add_suite()
4) 加入测试用例（Test Case）到测试包当中 - CU_add_test()
5) 使用适当的接口来运行测试测试程序，例如

CU_console_run_tests()

6) 清除 Test Registry - CU_cleanup_registry()

### 1.4 TestCase 的构成

一个 C Unit TestCase 的构成有以下文件：test.c、testcase.c、Main.c 与 Makefile 构成。

即一个测试范例由①被测函数，②测试函数（定义策识用例和测试包），③运行测试函数及④Makefile 四部分构成。

## 2. CUnit Framework 的安装

### 2.1 CUnit Framework 的下载

Cunit Framework 的当前版本为：CUnit-2.1-0-src.tar.gz。

### 2.2 CUnit Framework 的安装

CUnit 的 Framework 的安装环境为 Fedora 5。安装命令如下：

1) 解包

#tar xzvf CUnit-2.1-0-src.tar.gz

2) 编译与安装

#cd CUnit-2.1-0

#aclocal (if necessary)

#autoconf (if necessary)

#automake (if necessary)

#chmod u+x configure (if necessary)

#./configure --prefix <Your choice of directory for installation>

#make

#make install

3) 加载 CUnit 的库(only one time)

#cd /usr/local/lib

#ldconfig

# 3. CUnit TestCase 构成

## 4.1 CUnit TestCase 的构成

一个 C Unit TestCase 的构成有以下文件：test.c、testcase.c、Main.c 与 Makefile 构成。

## 4.1 CUnit TestCase 主要构成函数说明

以下以一个简单的 testcase 为例说明。

我要测试的是整数求最大值的函数 maxi，我使用如下文件组织结构：

1. test.c：被测函数（定义 maxi()函数）
2. testcase.c：测试函数（定义测试用例和测试包）
3. Main.c：运行测试函数（调用 CUnit 的 Automated 接口运行测试）
4. Makefile ：生成测试程序。

1) 被测函数 test.c

```
/**
 *file：  test.c
 **/
int maxi(int i，int j)
{
   return i>j?i：j;
}
```

2) 测试函数（定义策识用例和测试包）testcase.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <CUnit/CUnit.h>
#include <CUnit/Automated.h>
#include <CUnit/TestDB.h>
/**//*---- functions to be tested ------*/
extern int maxi(int i，  int j);
```

```
/**//*---- test cases -----------------*/
void testIQJ()
{
        CU_ASSERT_EQUAL(maxi(1，1)，1);
        CU_ASSERT_EQUAL(maxi(0，-0)，0);
}

void testIGJ()
{
        CU_ASSERT_EQUAL(maxi(2，1)，2);
        CU_ASSERT_EQUAL(maxi(0，-1)，0);
        CU_ASSERT_EQUAL(maxi(-1，-2)，-1);
}

void testILJ()
{
        CU_ASSERT_EQUAL(maxi(1，2)，2);
        CU_ASSERT_EQUAL(maxi(-1，0)，0);
        CU_ASSERT_EQUAL(maxi(-2，-1)，-1);
}

CU_TestInfo testcases[] = {
        {"Testing i equals j：",  testIQJ},
        {"Testing i greater than j：",  testIGJ},
        {"Testing i less than j：",  testILJ},
        CU_TEST_INFO_NULL
};
```

```
/**//*---- test suites ------------------*/
int suite_success_init(void)
{ return 0; }

int suite_success_clean(void)
{ return 0; }

CU_SuiteInfo suites[] = {
        {"Testing the function maxi: ", suite_success_init,
suite_success_clean, testcases},
        CU_SUITE_INFO_NULL
};


/**//*---- setting enviroment -----------*/

void AddTests(void)
{
        assert(NULL != CU_get_registry());
        assert(!CU_is_test_running());
        /**//* shortcut regitry */

        if(CUE_SUCCESS != CU_register_suites(suites)){
                fprintf(stderr, "Register suites failed - %s ",
CU_get_error_msg());
                exit(EXIT_FAILURE);
        }
}
```

3) 运行测试函数 Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "Basic.h"

int main(int argc, char* argv[])
{
   CU_BasicRunMode mode = CU_BRM_VERBOSE;
   CU_ErrorAction error_action = CUEA_IGNORE;
   int i;

   setvbuf(stdout, NULL, _IONBF, 0);

   for (i=1 ; i<argc ; i++) {
      if (!strcmp("-i", argv[i])) {
         error_action = CUEA_IGNORE;
      }
      else if (!strcmp("-f", argv[i])) {
         error_action = CUEA_FAIL;
      }
      else if (!strcmp("-A", argv[i])) {
         error_action = CUEA_ABORT;
      }
      else if (!strcmp("-s", argv[i])) {
         mode = CU_BRM_SILENT;
      }
```

```
    else if (!strcmp("-n",  argv[i])) {
        mode = CU_BRM_NORMAL;
    }
    else if (!strcmp("-v",  argv[i])) {
        mode = CU_BRM_VERBOSE;
    }
    else if (!strcmp("-e",  argv[i])) {
        return 0;
    }
    else {
        printf("\nUsage:  BasicTest [options]\n\n"
               "Options:  -i  ignore framework errors [default].\n"
               "          -f  fail on framework error.\n"
               "          -A  abort on framework error.\n\n"
               "          -s  silent mode - no output to screen.\n"
               "          -n  normal mode - standard output to screen.\n"
               "          -v  verbose mode - max output to screen [default].\n\n"
               "          -e  print expected test results and exit.\n"
               "          -h  print this message and exit.\n\n");
        return 0;
    }
}
```

```
    if (CU_initialize_registry()) {
        printf("\nInitialization of Test Registry failed.");
    }
    else {
        AddTests();
        CU_basic_set_mode(mode);
        CU_set_error_action(error_action);
        printf("\nTests completed with return value %d.\n" ,
CU_basic_run_tests());
        CU_cleanup_registry();
    }

    return 0;
}
```

4) Makefile

```
INC=-I/usr/local/include/CUnit
LIB=-L/usr/local/lib/
all：  func.c test_func.c run_test.c
#gcc -o test $(INC) $(LIB) -lcunit $^
    gcc -o test $(INC) $(LIB) -lcunit    $^
clean：
    -rm -rf *.o test
```

### 4.1 测试报告

```
CUnit - A Unit testing framework for C - Version 2.1-0
http：//cunit.sourceforge.net/

Suite：  suite_success_both
```

```
Test:  testSuccess1 ... passed
Test:  testSuccess2 ... passed
Test:  testSuccess3 ... passed
Suite:  suite_success_init
Test:  testSuccess1 ... passed
Test:  testSuccess2 ... passed
Test:  testSuccess3 ... passed
Suite:  suite_success_clean
Test:  testSuccess1 ... passed
Test:  testSuccess2 ... passed
Test:  testSuccess3 ... passed
Suite:  test_failure
Test:  testFailure1 ... FAILED
1. ExampleTests.c: 52  - 0
Test:  testFailure2 ... FAILED
1. ExampleTests.c: 53  - 0
Test:  testFailure3 ... FAILED
1. ExampleTests.c: 54  - 0
WARNING - Suite initialization failed for suite_failure_both.
WARNING - Suite initialization failed for suite_failure_init.
Suite:  suite_success_but_failure_clean
Test:  testSuiteFailure1 ... FAILED
1. ExampleTests.c: 49  - 0
Test:  testSuiteFailure2 ... passed
WARNING - Suite cleanup failed for suite_success_but_failure_clean.
Suite:  TestSimpleAssert
Test:  testSimpleAssert ... FAILED
1. ExampleTests.c: 63  - 0
2. ExampleTests.c: 64  - !1
```

3. ExampleTests.c：65 - 0

Test： testFail ... FAILED

a) ExampleTests.c：70 - CU_FAIL("This is a failure.")

b) ExampleTests.c：71 - CU_FAIL("This is another failure.")

Suite： TestBooleanAssert

Test： testAssertTrue ... FAILED

a) ExampleTests.c：79 - CU_ASSERT_TRUE(!CU_TRUE)

b) ExampleTests.c：80 - CU_ASSERT_TRUE(CU_FALSE)

Test： testAssertFalse ... FAILED

1. ExampleTests.c：88 - CU_ASSERT_FALSE(!CU_FALSE)
2. ExampleTests.c：89 - CU_ASSERT_FALSE(CU_TRUE)

Suite： TestEqualityAssert

Test： testAssertEqual ... FAILED

1. ExampleTests.c：99 - CU_ASSERT_EQUAL(10，11)
2. ExampleTests.c：100 - CU_ASSERT_EQUAL(0，1)
3. ExampleTests.c：101 - CU_ASSERT_EQUAL(0，-1)
4. ExampleTests.c：102 - CU_ASSERT_EQUAL(-12，12)

Test： testAssertNotEqual ... FAILED

1. ExampleTests.c：111 - CU_ASSERT_NOT_EQUAL(10，10)
2. ExampleTests.c：112 - CU_ASSERT_NOT_EQUAL(0，-0)
3. ExampleTests.c：113 - CU_ASSERT_NOT_EQUAL(0，0)
4. ExampleTests.c：114 - CU_ASSERT_NOT_EQUAL(-12，-12)

Suite： TestPointerAssert

Test： testAssertPtrEqual ... FAILED

a) ExampleTests.c：121 - CU_ASSERT_PTR_EQUAL((void*)0x100，(void*)0x101)

Test： testAssertPtrNotEqual ... FAILED

4. ExampleTests.c : 128 - CU_ASSERT_PTR_NOT_EQUAL((void*)0x100 , (void*)0x100)

Suite: TestNullnessAssert

Test: testAssertPtrNull ... FAILED

1. ExampleTests.c: 136 - CU_ASSERT_PTR_NULL((void*)0x23)

Test: testAssertPtrNotNull ... FAILED

1. ExampleTests.c: 143 - CU_ASSERT_PTR_NOT_NULL(NULL)

2. ExampleTests.c : 144 - CU_ASSERT_PTR_NOT_NULL((void*)0x0)

Suite: TestStringAssert

Test: testAssertStringEqual ... FAILED

1. ExampleTests.c: 155 - CU_ASSERT_STRING_EQUAL(str1, str3)

2. ExampleTests.c: 156 - CU_ASSERT_STRING_EQUAL(str3, str2)

Test: testAssertStringNotEqual ... FAILED

1. ExampleTests.c : 168 - CU_ASSERT_STRING_NOT_EQUAL(str1, str2)

Suite: TestNStringAssert

Test: testAssertNStringEqual ... FAILED

1. ExampleTests.c: 181 - CU_ASSERT_NSTRING_EQUAL(str2, str3, 4)

2. ExampleTests.c: 182 - CU_ASSERT_NSTRING_EQUAL(str1, str3, strlen(str1))

Test: testAssertNStringNotEqual ... FAILED

1. ExampleTests.c : 194 - CU_ASSERT_NSTRING_NOT_EQUAL(str1, str2, 2)

2. ExampleTests.c : 195 - CU_ASSERT_NSTRING_NOT_EQUAL(str2, str3, 2)

Suite： TestDoubleAssert

Test： testAssertDoubleEqual ... FAILED

1. ExampleTests.c：205 - CU_ASSERT_DOUBLE_EQUAL(10，10.0001，0.00001)

2. ExampleTests.c：206 - CU_ASSERT_DOUBLE_EQUAL(10，10.0001，-0.00001)

3. ExampleTests.c：207 - CU_ASSERT_DOUBLE_EQUAL(-10，-10.0001，0.00001)

4. ExampleTests.c：208 - CU_ASSERT_DOUBLE_EQUAL(-10，-10.0001，-0.00001)

Test： testAssertDoubleNotEqual ... FAILED

1. ExampleTests.c : 218 - CU_ASSERT_DOUBLE_NOT_EQUAL(10，10.001，0.01)

2. ExampleTests.c : 219 - CU_ASSERT_DOUBLE_NOT_EQUAL(10，10.001，-0.01)

3. ExampleTests.c : 220 - CU_ASSERT_DOUBLE_NOT_EQUAL(-10，-10.001，0.01)

4. ExampleTests.c : 221 - CU_ASSERT_DOUBLE_NOT_EQUAL(-10，-10.001，-0.01)

Suite： TestFatal

Test： testFatal ... FAILED

1. ExampleTests.c：227 - CU_FALSE

--Run Summary： Type Total Ran Passed Failed

suites 16 14 n/a 3

tests 35 31 10 21

asserts 89 89 47 42

Tests completed with return value 22.

# 4. 项目中 CUnit TestCase 的作成

## 4.1 项目中 TestCase 作成的指导思想

CUnit 的 TestCase 作成的指导思想：TestCase 和项目代码分开，作成的 TestCase 独立地放在其它目录中。

# 测试职业经历随谈

作者：董杰

**关键字：**测试；Bug；测试误解；创造力；坚定信念

## 1. 开场白

首先，作为一个一直在网络和电信设备测试领域工作了几年的老油条，我想把我这几年的工作经历以及在我身边看到的故事与大家分享。希望能帮助一些刚进入测试的朋友能更好的从学生角色转换为一个职业人，少一些刚毕业时的迷茫。

我在大学时代学的是计算机，在毕业前对网络的认识也就是学校开设的两门课计算机网络以及计算机网络与通信课程，让我印象最深的是学了一大堆与高数有关的数学公式，例如什么香农公式这类。毕业前连 IP 子网都算不清，也就大概知道 TCP,UDP 的区别。我更多的时间和精力都放在了程序的开发上,曾用 Delphi 独立开发了一个全省系统的 MIS 系统，用 Linux+Gcc+Gdb＋UDP 开发了一个模拟 QQ 原理的即时聊天软件。

一晃到了大四找工作，我当时找工作的想法是：一定要进大公司，工资要高。我至今仍清晰的记得那句从此影响了我职业选择的话：“我们开发都要研究生，对测试是否有兴趣”。当时想了想测试没做过，可以先试试。再加上这家公司当年在成都地区非常火，在成都给本科生开得工资是最高的。所以，就抱着先试试的感觉应聘了测试。也许是我大学时代动手实习的经验较多，在没有笔试的情况下，成功应聘进了这家 M 公司。也是在大四的第二学期，我参加了 M 公司的第三期企业文化培训，这期参加培训的上百名研究院的新同事中本科生一共 10 人左右,而且来自成都高校的本科生才 4 人。说实话，我很庆幸自己的幸运。

2. **“阴差阳错”**

在随后 2 个月的网络知识培训中，我第一次见到路由器、交换机、VOIP 网关、电信设备，在稀里糊涂的状态下进行着配置练习。我们那时真是魔鬼式培训，从早上 8 点半开始，一直持续到晚上 11 点。有一次还持续到了晚上 12 点。每天上午第一件事就是考试，将头天的学习内容进行笔试考试，基本上 TCP/IP 详解中每一个协议的报文结构都默写过。笔试成绩每天公布分数和排名。下午从 1 点开始一直到晚上 11 点就是上机操作。做完一个练习就做下一个练习，反正就是一直要做到晚上 11 点才下班回家。就这样一直维持了 2 个月的培训，每天都是新的理念知识，新的上机练习。压力非常大也非常累，进步也非常快。

七月分配产品线，我来到了最不愿意去的电信产品线。因为在培训期间，电信产品线的理念最难理解，也最难完成上机练习，远比 TCP/IP 详解难的多，所以几乎没有一个人愿意来到电信产品线。我和另 3 名同事都来到了电信产品线，没办法再难也只有蒙着头冲了。

八月我再一次被迫接受了不愿意的工作分配，电信产品线的测试经理将 FrameRelay 协议（因为这是一个老协议，所以我一直最不愿意测试这个协议）给我负责，并要求在半年内必须独挡一面，意思是以后公司遇到任何关于 FrameRelay 协议的问题都将由我来解决，就是公司这个协议的专家。对于一个刚毕业 2 个月的学生，面对这样一个要求，既感到压力也感到了一种挑战。在必须成为公司内该协议的专家且没有任何退路的情况下。

随后半年里，对于该协议无论理论还是任何实际问题我都以几乎十全十美的态度来学习或解决。不但对该协议的每一个细节，每一个 bit 的组合意义烂熟于心，更对该协议在产品中从 FPGA 到 HDLC 到驱动的所有实现原理进行了深入学习。再通过对实验室里和市场上各种问题的分析，最终在半年内渐渐做到了可以独挡一面，可以独立面对和解决任何与 FrameRelay 协议有关的市场技术问题。

### 3. “坚定信念，一定还有 bug”

也就在这半年时间里，我对测试的看法发生了两次大的变化。第一次变化，在进行手工测试 3 个月后，我心中还是认为开发比测试有意义，有挑战。而测试则是简单，无聊的工作，让我做测试就是大材小用。在这时，我内心开始讨厌测试工作，因为自己从心里看不起测试。有一次我测试了 5 天 FrameRelay 协议，没有发现一个 bug，之前我已发现了这个协议上百个 bug。正当我沮丧时，一位老员工的一句话改变了我对测试的认识“想想办法，肯定还有 bug”。于是，我怀着半信半疑的态度，利用周末梳理了自己的思路。从自己已发现 bug 的方式和现象，到 FrameRelay 协议的所有实现原理，最后自己又提出了 10 个新的测试方法。周一上班，只用了 4 个新方法就发现了一个一级 bug，一个二级 bug。从此以后，在我心中对测试的看法发生了根本性的改变，不再认为测试只能由能力差的人去做。测试同样具有很强的挑战性和成就感，测试是一个必须需要创造性的工作，而且是每天都会用到创造性能力的工作。

当你发现不了 bug 的时候，请相信这句话：“一定还有 bug”。只要你坚定这个信念，一定会发现 bug 的。“坚定信念”成为了我以后鼓励新员工时最爱用的一句话。对于测试人员，每天的工作都是对自己潜能的一次挑战。昨天，自己还认为没有 bug 了，今天通过自己坚定的信念和创造性的劳动，又发现了新的 bug。这不证明好的测试态度能让测试成为一份非常有挑战、创造性和成就感的工作吗？

在其后 1 年的工作中，因为我同时具有开发和测试的能力，公司希望让我做测试工具开发工作，但我拒绝了这个调岗，同期也拒绝了华为（不是慧通）邀请我从事测试工具开发工作的 offer。我选择了测试。测试不但让我每天都能感受到自我挑战带来的成就感，进行创造性思考所带来的快乐。更能让我比普通的开发者用更短的时间了解到更多产品或功能实现的细节，能对系统了解的更广更多，让自己不但看到树枝，更能看到整个森林。这也是我只用短短 3 年

多时间就对很多网络和电信协议的应用及设备原理有了比较系统的了解。结合我自己的特点，我喜欢在适度深度的基础上，希望能对网络尽可能宽广的了解和掌握。而不喜欢一个点或是几个点一路走下去，掌握每一个细节。所以，我喜欢测试比开发更多一点。

4. **“测试误解”**

那么测试人员的水平怎么体现出差异呢？

我的看法是：不是你发现的 bug 数量多就很厉害，也不是你会写自动化脚本就比做手工测试的厉害，更不是你会用的测试工具多，你就厉害。好的测试人员应该首先具有严密的逻辑思路，同时还要有很强的发散性思维，能经常创造性的提出新的测试方案，并且具有非常强的分析问题、定位问题的能力。只要具备了很强的分析问题定位问题的能力和创造力，即使你所负责的模块是一个小模块，你也能发现很多严重的 bug。

在测试中有些同事常有这样的误解：

**情形 1：**有的同事所负责的模块是个老模块，小模块，所以发现新 bug 的数量很少。这时很多人常常就容易松懈，认为这个模块没 bug 了，再认真测试是浪费时间。

我的看法是：1、任何模块永远都有 bug，只是还没找到触发条件。 2、一般人的常规思路用完了，这时正是你挖掘自身创造力的好机会。如果你这时能在自己思路严密分析的基础上提出一系列新测试方案，发现了新 bug。你的领导一定会对你刮目相看，而自己的自信心一定会得到大大的提高。

**情形 2：**有的同事所负责的模块是个新模块，大模块，随便乱搞也能发现许多 bug。同样，这时测试者也很容易松懈，而不会进行有意识的分析和创造性的思考。

我的看法是：1、虽然你发现了很多 bug，但是是否你能通过严密的分析，使得找 出的一级、二级 bug 数能占到每轮测试所发现 bug 数的大部分。不要总是在产品快要 release 市场发布前，忽然发现许多一级、二级 bug。2、 只有通过自己严密的分析和创造力

发现的 bug，才比乱测试所发现的 bug 更具价值。对于产品的质量才更有真正的保障，对于测试者的水平提高才有意义。否则，测试者自身能力得不到提高，公司产品更是留下了许多隐患的 bug。

5. **总结赠言**

最后送大家几句我关于测试的看法：

1、 坚定信念，一定还有 bug。

2、 测试是一份依赖严密思路分析和创造力的工作。

3、 测试能让你每天挑战自己。

4、 测试能让你的自信心不断得到提高。

5、 测试能锻炼你的创造力。

6、 测试能锤炼力你的性格，让你成为一个不易轻易放弃的人。

7、 测试能让你比普通开发者更快的了解到系统。

# 浅谈软件测试工具在工作中的作用

作者：张磊

经常有 51testing 上的同行加我 QQ,和我一起交流测试心得以及工作经验，我发现他们都会问我一个同样的问题“你用什么测试工具”，是啊，自动化测试的介绍以及不断推广，很多用人单位在应聘广告上都登出要求会使用一到两种测试工具。但是我对测试工具却有另一种看法。

首先我们需要了解测试工具，目前测试工具的种类很多，我想大致可以分成 2 种，一种为自动化或者辅助测试工具，主要有大家熟悉的 winrunner，loadrunner 等，另一种就是测试管理工具，主要是对测试用例的管理以及 bug 的追踪的工具。

其次测试工具与软件测试工作的关系，在讨论这个问题之前，我们先看看什么是测试软件测试，软件测试就是为了发现程序中的错误而执行程序的过程，所以测试的目的是证明程序的错误。为了达到此目的，在软件测试工作中测试人员使用了各种测试方法，而测试工具因此产生。目前国内大部分软件企业还是以黑盒测试为主，黑盒测试的局限性在于需要花大量的时间和人力，进行重复性的操作，无法保证对程序的完全覆盖，同时黑盒测试无法保证测试人员能在测试中发现每个细小的 bug，以及很难对偶发的问题进行重现、追踪。而测试工具的使用就克服了这些黑盒测试的缺陷，可见测试工具对软件测试起到了重要作用。

测试工具固然对软件测试工作有重要作用，但是不代表学会了使用测试工具就能成为一个好的软件测试工程师。既然是一种软件工具，那学会使用它就不是一件困难的事，因为软件工具也是软件，软件的特点就是用最简单的方式让使用者能很快上手，所以一般工具有丰富的快捷按键设置。就像 word，我相信很多人自己摸索，只

看了些初级教程就能应用自如，同时在使用中不断发现其更加实用好用的功能。软件测试工具也是这样，当你工作中需要使用到它，那么你学会掌握它其实是一件很容易的事情。而如果没有使用环境，只是去看工具的使用手册，一般总是云里雾里，觉得似乎很高深。所以我觉得没有必要因为目前工作不涉及到软件测试工具，而盲目去学习软件测试工具，并要求自己达到熟练掌握的程度，我想只需要了解每个测试工具对测试工作的作用、帮助，用于什么方面的测试即可。

目前测试行业，大部分公司还是黑盒测试，使用测试工具的公司一般分为 2 类，一类是小公司，使用盗版的测试工具软件，这类公司多数会在招聘信息上要求熟练掌握某个测试工具，希望求职人员能以最快速度投入测试工作，另一类是上市公司，公司产品符合 Mercury 或者 Rational 等大型测试工具公司的产品进行自动化测试，使用正版测试工具，有良好的测试工具培训机制，对招聘时更注重求职人员的测试能力。而在测试行业中还有大部分企业不使用具有版权的测试工具，而自行研发测试工具，来帮助提高测试效率，这部分企业一般规模较大，大型测试工具对其开发产品测试不能起到重要作用。

测试工具的范围很广，优秀的测试工程师应该会根据项目，制定测试计划，使用需要的测试工具。正如之前所说测试工具是为了提高测试的效率，那所有用于实现此目的的与测试相关的辅助工具都可以称之为测试工具。我曾经在测试 STK 功能的时候，自己写了一个测试工具，此工具可以按照我的需要生成不同的 STK 主动式命令脚本，来模拟 SIM 发出的各个 STK 命令，这样的方法可以使得测试不需要因为 sim 卡上 STK 功能的局限而无法全面测试。所以测试人员拿到产品开发书后应该思考如何完整测试，哪些测试工具的使用可以提高测试效率，甚至是如何开发一个适合自己公司项目测试工作的测试工具来帮助完成测试。

在软件测试业逐步走向成熟的今天，测试工具的使用将对于企业保证产品品质，提高测试水平起到决定性的作用。作为一位测试人，我们应该时刻思考如何将测试工具在工作中更好的运用。

# 软件企业中测试的组织与实施

作者：迷惘的白羊座

**摘　要：**本文分析国内软件企业软件测试的组织架构、人才培养、测试自动化以及测试流程的现状，指出一些可能存在的问题以及讨论相应的解决措施。

**关键字：**软件测试；组织；实施

## 引言

软件测试是一个带着寻找错误的目的执行程序的过程[1]，它作为软件开发过程中不可缺少的一个质量保证环节，日益受到国内软件企业的重视。但是目前软件测试在国内还处于起步阶段，测试人员的水平参差不齐、测试过程缺乏必要的控制和管理等问题都困扰着大多数的软件企业。本文尝试着从组织架构、人才培养、测试自动化、测试流程制定等方面对软件测试在软件企业中的组织实施展开探讨。

## 组织架构

测试团队的能力在很大程度上影响着软件测试的成败，一个有效的测试团队包括了技术专家和领域专家的结合。除此之外，测试团队的结构必须是合理的，各种角色和职责分工明确，测试人员各司其职，很少有职能上的交叠，也不会发生关于哪个团队成员应该完成哪项职责方面的任何不确定性[2]。测试团队的结构在各个组织内是不同的，它的深度和构成也依赖于待测试的对象和测试团队的任务[3]。在软件测试的组织架构上，目前较成规模或者比较规范的软件企业，大都采用独立设置测试部门的方式，测试部门设一名测试经理，负责测试人员的组织和管理工作；测试人员则根据项目的

需要，分别安排跟进若干个开发项目，一名测试人员可能只参与一个项目的测试工作，也可能同时参与几个项目的测试工作。在参与同一个项目的测试人员中，可以设一名测试主管，该测试主管负责该项目测试工作的统筹安排，可以由测试经理兼任，也可选择经验较丰富并且有较强组织协调能力的测试人员担任。

在此本文着重讨论一下测试人员的构成。测试人员的来源目前主要有三种：一类是原先的编程人员，由于企业或项目的安排，或者自身的选择成为了测试人员；一类是从计算机专业毕业后一直从事软件测试工作；还有一类是从其他专业转行到计算机行业从事软件测试的工作。而测试工作的构成主要可以分为三类：测试用例设计、自动化测试脚本开发和测试用例执行。上面提及的第一类人员有相对丰富的编程经验，适合从事自动化测试脚本的开发工作，而第二第三类人员有比较丰富的测试经验，适合进行测试用例的设计，测试用例的执行可由一些初级测试人员担任。当然这不是绝对的，在实际的开发过程中，除了根据测试人员的专长，还应结合其自身的兴趣进行工作的分配。因为单纯的测试执行工作相对来说还是比较机械比较枯燥的，无论从培养测试人员能力还是从留住人才方面考虑，都不应该让一个测试人员长期从事单一的测试工作，每一个测试人员应该有一个自己比较精通的方向，比如界面测试、接口测试或者安全测试，但是同时也应兼顾其他方向技术能力的培养。

## 人才培养

一个理想的测试工程师的技能要求是非常高的，这些技能要求随测试种类和使用的测试工具而有所不同。针对 GUI 测试，测试工程师可能需要熟悉 Visual Basic、MS Acess、SQL Server 以及 Windows 操作系统；针对服务器端测试，测试工程师可能需要能够使用 C、C++、SQL、UNIX 以及 UNIX 脚本等等。由于软件测试作为一个行业在国内也还处于起步阶段，目前整个行业内真正具有成熟完备的能力的测试工程师还比较少，每个企业内部可能会有一两个或者若

干个这样的测试工程师，但是仍属于少数，多数的测试工程师刚从学校毕业不久或仅有一到两年软件开发经验，那么知识的传递和技能的培训在软件测试组织中就显得尤为重要。测试人员的技能提升，有助于测试所起的作用在整个开发团队中得到平等的待遇和尊重[4]。有计划地进行知识的传递和技能的培训，有助于测试团队整体水平的提高，同时也能避免初级测试人员因为长期得不到技能上的提高而选择退出。在有些测试团队中，测试技能的传递职责由 SMT（Systems Methodology and Test）团队承担。这个小组可以被看作测试团队内部的咨询人，负责提升理论和标准的知识传递、为项目引入测试工具、负责评估和培训自动化测试工具。不管采用的形式是什么，基于软件测试的复杂性或是基于测试组织稳定性的考虑，都应当重视知识传递和技能培训。

**自动测试还是手工测试**

从测试执行的手段来看，软件测试可以简单地分为自动测试和手工测试。自动化测试工具能够增强测试力度——前提条件是期望是现实可行的、工具能够被正确理解、并且选择了一个与系统工程环境相兼容并且适合当前任务的测试工具[2]。采用自动化测试还是手工测试并不是一个简单的选择，它需要综合项目的性质、项目开发的流程、测试的侧重点等多方面的因素来考量。性能测试、负载测试通常采用测试工具自动地执行，而就功能测试而言，自动化测试很适合在进行产品开发的软件企业中实施，因为软件产品的开发周期相对比较长，在开发过程中能够有计划地留出比较充裕的时间进行软件测试，测试用例重复使用的次数较多；而对于进行项目开发的软件企业以及第三方的软件评测机构，则要视具体项目的周期而定，如果测试本身只留出较短的时间，在回归测试的次数不多、测试用例的复用度比较低的情况下，简单地采用手工测试会是一个比较理想的方案。需要明确的是，自动化测试仅仅是一部分的解决方案，它们并不是针对测试问题的银弹，自动化测试工具永远不能

替代指导测试的分析技巧，也不能替代手工测试，它只能被视作对手工测试的一个辅助。

**测试工具的选用**

自动化测试依赖的是测试工具，提起测试工具，首先想到的就是那些昂贵的商业化的通用测试工具，但实际上还有很多开源的测试工具可以使用，除此之外有的企业也有测试开发工程师编写的针对具体应用的测试工具。下面简单分析一下每种工具的应用情况。

先说商业化测试工具。商业化的测试工具以其简单易用性受到测试工程师的欢迎，尤其是基于录制/回放的功能性能测试工具，大大地简化了测试工程师的工作；而一些复杂的测试，例如静态分析、内存检测，使用测试工具更是起到事半功倍的作用。但是大多数中小规模的软件企业没有足够的预算来购买昂贵的商业工具，另一方面测试工程师还没有全面掌握使用商业工具的技能。业界比较知名的几个测试工具厂商如 MI、Sugue、Compuware 公司提供的测试工具，动辄数十万、上百万，不是一般软件企业能够承受的，即便能够承受，往往也找不到适当的人选来使用，真正灵活地使用商业化工具软件，需要较强的综合能力，包括数据库的使用、脚本编程等等；除此之外，由于是通用的商业软件，不一定百分百地切合具体应用的测试需求，我们常常能在论坛上看到各式各样求助的帖子，这些关于工具具体使用的问题，即便求助于工具厂商的技术支持，也不一定能够轻松地解决，因为它需要综合工具的解决方案和具体应用的背景；另一方面，企业级的应用往往庞大而复杂，尤其是在分布式环境下，仅靠单一工具难以完成全部验证的任务。有的时候经过对工具的研究和评估，得出的结论是没有现成的工具完全地适合项目的需要，或者有现有的商用工具适合当前的应用，但是工具提供了许多超出实际需要的功能，大大增加了花费，这些情况下可以考虑使用开源工具。

以测试 Java 应用为例，最典型的开源测试工具是 JUnit，它提

供了一个进行回归测试的框架。作为一个测试框架，JUnit 并不局限于应用在单元测试中，同样也可以应用于集成测试。比如对 web 应用进行集成测试的 HttpUnit 等测试工具，同样是基于 JUnit 框架。基于 JUnit，利用开源工具提供的高层 API，我们可以构造出各种随需应变的测试，而回避底层细节。除了一些为测试而设计开发的工具，另外有一些小工具，虽然不是特别为测试而开发的，但在一定场合下也可以被使用，以简化测试过程，加快测试执行的速度。开源工具的好处不言而喻，因为是免费的，可以省去购买商业工具的昂贵预算，但实际上开源工具的优势不局限于此。相对成熟的开源工具在互联网上都有相应的技术社区，一些技术支持可以较快地在社区内得到响应，比较理想的情况下增加新功能或者是修复缺陷的请求也都能够比较快地得到响应，由于源代码公开，如果不满意工具现有的功能而不想等到下一个版本的发布，还可以自行修改以适应自身的需求。但是开源工具的劣势也同样存在，例如和商业工具相比而言还不够成熟，工具演化的前景不明，加上使用起来对测试人员技术上的要求也更高。

如果以上的方案都不能适应当前应用的需要，那么可以考虑开发一个独立的解决方案，使用脚本或定制的工具，或者仅仅依靠手工测试。当然，开发测试工具必须经过仔细的分析以确保从长远来看不会比购买工具花费更大。有些情况下必须使用独立开发的测试工具，尤其是当软件或者软件运行的环境非常特殊，无法使用现有的工具进行测试的时候。这些情况包括：操作系统不兼容、应用程序不兼容（例如因包含第三方插件导致的被测软件与捕获/回放工具不兼容）、特殊的测试需求。

**测试流程的制定**

在传统的瀑布模型的过程模型下，软件的测试是在分析、设计、编码工作全部完成后才着手进行的，瀑布模型由于许多原因已经不能适应当前大规模企业应用的开发，包括它引入测试的时机太晚，

造成修复在测试中发现的问题的代价太大。当前更具有普适性的过程模型，无论是无论是 UP 还是 XP，都强调尽早测试甚至是测试先行。从整个工业界来看，总体的趋势是在整个过程中让测试人员尽早地介入，并且更多地与业务分析和开发团队协作[4]。目前国内已经有越来越多的软件企业开始意识到软件测试的重要性，从过去那种完全没有测试或者仅由开发人员在项目收尾阶段简单验证一下大体的功能的做法，转变到成立独立的测试组或测试部门来执行测试的任务。但是在多数中小规模的软件企业中，测试组的工作仍然比较被动，“测试先行”不能得到很好的贯彻，需求分析、设计阶段都没有严格的评审，代码的可测试性也常常被忽略。

对于测试流程的建立，测试团队常常无奈地表示管理层认为流程会浪费时间，或者当前任务繁多，还没有精力去建立一个测试流程。面对这种情况，测试人员首先应当明确自己需要的流程究竟是什么样的，如果自己都只有一个很模糊的概念，是很难说服管理层去启动这个计划的。

一个规范化的软件测试过程通常包括：拟定软件测试计划、编制软件测试大纲、确定软件测试环境、设计和生成测试用例、实施测试以及生成软件测试报告。实际上，软件测试过程与整个软件开发过程基本上是平行进行的，从需求阶段开始测试就应当介入，需求应当首先被验证，同时一旦完成需求分析，就可以着手测试过程的设计。对整个测试过程应当进行有效的管理，明确定义测试执行周期的起始和结束、将测试环境和开发环境相隔离、实施缺陷跟踪流程、对整个测试过程进行跟踪[1]。除此之外还有许多细节上的规范需要根据企业各自具体情况制定。

## 小结

以上是笔者结合自身工作中的体会对软件测试在企业中的组织和实施的一些分析和体会。软件测试在国内还处于起步阶段，但在国外有相对比较成熟的理论和实践，值得我们借鉴。相信在不久的

将来，国内的软件测试能有一个飞跃的发展。

**参考文献：**

①Art of Software Testing, Second Edition Glenford J. Myers

②Effective Software Testing: 50 Specific Ways to Improve Your Testing, Elfriede Dustin

③Automated Software Testing: Introduction, Management and Performance, Elfriede Dustin,Jeff Rashka, John Paul

④Q&A with Industry Experts: How Are e-Business Trends Impacting Testers and Testing Teams? Jack Wilber, Writer/Interviewer:The Rational