# 目录

# GUI 测试中的关注点

Steven Wang

Archonwang1981@msn.com

【摘要】本文列数了软件黑盒测试过程中，在被测试软件中可能存在的常见软件问题。本文不会详细讨论基本的软件测试思想与常用技术，仅针对在软件黑盒测试过程中若干的问题作描述并提供个人的参考测试意见与防范意见，希冀可以为初学者提供些许帮助。

【关键词】软件测试，黑盒测试

## 引言：不能不说的二个问题

### 1. 软件测试中的“二八”原则

80%左右的错误在进行用户测试之前已经被发现，而在剩余 20%左右的错误中，存在 80%左右的显性错误，剩余 20%左右的错误是较难发现的隐性错误。这条原则源自经济学的二八法则[1]。所以，我并不认为自己从事了一项伟大的工作，但是必须承认做好了这项工作对于整个软件开发体系在用户心目中的意义巨大。

### 2. 软件黑盒测试解决的问题

简单来说，黑盒测试所解决的问题主要在于以用户眼光验证软件的结果。白盒测试关注范围（控制结构），而黑盒测试更关注结果（即我们常说的所见即所得）。黑盒测试试图发现以下几类错误：

- 功能错误或遗漏
- 界面错误
- 数据结构或外部数据库访问错误
- 性能错误
- 初始化错误和终止错误

以下内容将会详细说明在软件黑盒测试中常见的各类错误。

## 正文：软件黑盒测试常见错误类型及说明

### 1. 用户界面错误

[1] 1897 年，意大利经济学家帕列托在对 19 世纪英国社会各阶层的财富和收益统计分析时发现：80%的社会财富集中在 20%的人手里，而 80%的人只拥有社会财富的 20%，这就是“二八法则”。“二八法则”反应了一种不平衡性，但它却在社会、经济及生活中无处不在（当然，也包括软件工程）。

软件是为了满足用户需求而诞生的产物[2]，无论是操作系统、游戏软件还是其他类型的应用软件。黑盒测试的很大一部分工作集中在用户界面上，不需要深入研究其内在结构，而是“表面化”[3]地使用软件，从输入输出的信息内容中寻找可能的错误和纰漏。总体上讲，用户对于软件的看法很大程度上依赖以下几点：

- 功能性（实现软件应具备的基本功能）
- 易用性（用户学习掌握该软件所耗费的时间及在具体业务流程上的简化）
- 执行速度（多数是启动速度，查询速度，刷新速度及响应时间等因素）
- 用户使用时产生错误的比率（在允许用户任意使用的情况下，越少越好）
- 用户满意度（这里指的是用户界面设计与功能设计的用户评价）

下面我们分开对该类型错误进行分析与描述。

## 1.1. 功能性

如果出现了以下情况之一，可以认为程序可能存在功能性错误：程序可以完成的某些事进行得非常困难，笨拙（繁琐），令人迷惑甚至难以忍受。主要表现为以下几个方面：

### 1.1.1. 过度功能性

将简单功能复杂化，这是设计上一个较常见的问题。尝试进行太多工作任务的系统将很难学习和掌握，而且容易忘记。它要求大量的文档（开发文档，帮助文档和屏幕）。如果功能模块间模块过于紧密，则发生关联错误的几率要提高不少。有时候，用户需要的只是简单功能，而不要让它过于膨胀成为一个“怪物”。

### 1.1.2. 夸大的功能性印象

用户手册和营销传单不能使程序功能实现得更多，尤其是营销传单。记住，在用户手册中哪怕宁愿对功能略微轻描淡写也不能夸大其词（当然，我们并不希望这样，我们总是要对此如实地进行编写——这是我们的责任）。

[2]请注意：这里想要说明的不是软件的定义。
[3]当然不是真的表面化。我们需要关注更多的东西，不光光是软件的外壳，而且需要了解其中输入输出信息之间的关联，那并不是一件容易事。

### 1.1.3. 对手头任务的不适当性

我们可以把它直观的理解为需求设计错误。对于任何一个项目，由于功能关键事项（就是常说的需求提炼）不存在、太有限（多数是因为没有完成）或者太慢（需要改进程序结构或是内部算法）而不能完成真正的工作。举例来说，查询一个有 8000 条记录的数据库需要 1 个小时（天哪，我想我连 10 分钟都等不了），虽然说具备了查询的功能，但是实在很令人怀疑此项功能是否会有人使用，更糟糕的情况是：由于用户使用了该功能而造成的恶劣印象难以在短时间内消除——虽然对于开发人员来说那可能只是一个参数拼写错误了而已。

### 1.1.4. 遗漏功能

功能没有实现，却出现在了用户手册中。或者是本来应该具备地特征性功能，在程序只能看到一个“影子”（有其名无其实）。多半情况下是由于需求变更时没有对手册进行检查和更新，也有可能是因为遗漏了需求说明中应包含的功能（如果是那样，需要好好检查以前的工作方式是否正确）。

### 1.1.5. 错误功能

一个本来应该完成查询工作的功能却干了排序的活儿。这种疏忽一般不是因为没有实现功能，而是在分配功能的时候出现了问题，当然这种情况的发现和排除应该不是一件困难的事。

### 1.1.6. 功能性必须由用户创建

最常见的情况之一就是要求用户自己配置软环境（如配置数据源，一般都可以在安装程序中自动完成；当然还包括程序用到的组件在系统中不存在，安装程序没有提供相应的支持，这对用户是不能接受的）。这类问题不完全一定都是错误，比如微软提供的 Office 宏的开发，是为了满足客户对于自身特色而设计的适合其专业工作的程序。

### 1.1.7. 不能做用户期望的工作

例如，极少有人会期望一个本来编写用来对姓名进行排序的程序却按照 ASCII 码的顺序排序。他们也不会指望用它来计算首位空格或区分大小写。当然用户名的排序还是要做的，问题是开发者需要重新构想一个新的排序规则来满足用户需求。

## 1.2. 人机交互

人机交互，程序与操作者之间的通信与交流。这不是早些年的科幻电影，我们也许每天都在做，在取款机前，在自动售卖机前——

### 1.2.1. 遗漏信息

你应该知道，所有的事都能从计算机屏幕上得到有效的消息。不要遗漏任何对于用户而言至关重要的信息，即使这些信息对你而言毫无用处。

- 没有任何屏幕指令

  如何找到程序的名称？[4]如何退出程序？你应该怎么样获取帮助？如果程序使用了某种命令语言，如何才能得到命令列表？程序可能仅仅只在它启动时显示这些内容。当然你也可以从帮助手册中获取这些信息，但并不是必要的。没有任何屏幕指令的程序可能会让人受不了，查询手册的话需要花费的时间可能会更长——这可能就会让用户觉得软件学习起来太复杂了。

- 假定打印出的文件随时可得

  丢了用户手册怎么办？有经验的用户不会非要依赖打印好的文档，提供一份电子版的吧。

- 无正式文字证明（说明）的功能特征

  如果大多数的功能特征或命令在屏幕上提供文字说明，那么所有的都应如此。仅略过几个功能特征将会导致UI形式上的混乱。同样，如果程序为很多命令描述其“特殊情况”下的行为，那么所有的命令都需要提供这类说明。这种情况在国人的软件开发过程中时有发生，并不是不能，而是不想……[5]

- 看起来不可能退出的状态

  如何取消一条命令或在一个深层菜单树中进行备份？程序应该允许你可以避免那些你不希望遇到的情况。比如，在软件安装时，要求插入磁盘，如果不插入正确磁盘就不能退出安装

[4]一般来说可以使用建立在桌面上的快捷方式。
[5]唉～开发不规范和懒惰的结晶，不幸的是，这种情况还是时有发生。

程序。没有告诉你如何避免就和发生灾难时，没有提供一条逃生路径一样糟糕。

- 没有光标[6]

大多数用户都依赖于光标。一个光标可以让用户觉得计算机仍然在正常运转（尽管有时候死机也是如此），每个交互程序都应该显示光标，当然，在关闭光标时别忘了提醒用户注意。

- 没有对输入做出响应

每个程序都应该对输入做出回应。如果没有，呵呵，保管80%以上的用户产生疑问：怎么没有响应？还要等多久？是不是我的电脑过时了？[7]

如果有以下几种情况，一般视为正常：

  - 选择一个菜单项时，如果你的按键操作没有回应，只要下一个屏幕立刻出现并且此屏幕上的标题与菜单项一样，就可以视为正常。
  - 如果程序忽视错误的命令或按键操作，当然可以不对其进行回应。
  - 如果你告诉程序不要对输入回应，它必须沉默，如果它进行了回应，应该立即告诉开发人员对其进行修改（可能是在忘记了继续处理另一种情况）。
  - 如果输入的是安全性代码（如密码等），那么程序决不应在屏幕上做出不恰当的回应（如显示你输入的密码明文）。

- 在长期延迟时没有表示其活动

给一段较长时间的程序延迟一个合适的响应，将是非常必要的举动。相信这样不需要再给用户做出过多的解释了。

- 当某个改变即将生效时没有给出建议

一个程序可能会比你预计的更早或更晚执行一个命令，例如：删除某些重要数据时，（而这个过程将持续一段时间），必要的提示是必须的。

[6]在所有GUI类型的系统中基本上都应遵循这个原则。
[7]我们也许应该庆幸还有用户会这样想，但是请不要把希望寄托在这上面。

- 没有对已经打开的文件进行检查

这个错误是非常常见的，尤其对于那些输入关键数据的程序而言。用户可能不会注意，但是在以后的工作中将发现略微的一丝更改就会引出大量的问题。你不能保证程序会对同一个文件在某个时刻做出不同修改所带来的后果。所以，决不允许同一文件同时被打开两次甚至更多，以确保程序输出的唯一性。

- 错误的、误导的或令人迷惑的信息

每个错误都会让你对程序显示的所有其他东西产生怀疑。使用户做出虚假归纳的细小错误，如遗漏条件或是不适宜的类比会比清楚的事实错误更让测试人员感到恼火——因为更难对它们进行改正。

  - 简单的事实错误

    在一个程序改变之后，更新屏幕显示是低优先级任务，结果则是：屏幕上大量的信息过时。记住：无论何时程序发生改变，都要仔细检查可能指示程序特征的每个消息，最简单的办法就是直接对更改后的屏幕进行刷新。

  - 拼写错误（错别字）

    我相信，这绝对不是设计上的问题，我也相信开发人员可能会不以为然。Oh，但是客户会在乎，会抱怨这些的——还是改正它们吧。

  - 不准确的简化

    在保持一个特征的描述尽可能简单的愿望中，一条消息的创作者可能只覆盖特征行为的最简单方面，而忽略了重要条件。举例来说，这种情况可能会引发歧义，比如说关闭（到底是关闭文件还是关闭程序？）。作为一个测试人员，需要你足够仔细的研究可能会出现问题的任何一个微不足道的细节。宁可错杀，不能放过！（虽然要极力避免错杀的情况。）

  - 无效的比喻（图标之类可以指示功能（特征）的事物）

    例如：回收站的图标可能不是一个好的比喻；对于文件一旦移除就永久消失的情况来说，碎纸机的图标可能来得更好一些。

  - 令人迷惑不解的特征（功能）名称

如果一个命令名称在计算机领域中或语言中有一个标准含义，就必须与其含义一致。别指望着胳膊能拧得过大腿，确定现行的标准是可靠的。

- 同一特征（功能）具有多个名称

相同的功能特征只要一个名称就够了——只要能表述清楚；用户可不一定有时间玩同义词的游戏。另外，这种情况对软件在用户面前显示的复杂度也有影响。

- 信息超载

不要让你的文档和帮助屏幕看起来太过专业——太多技术细节了。用户会不耐烦的，况且效果也不好。如果实在需要，可以把他们另外列出。尽量使用直白，用户能理解的话表述这些信息。另外，信息超载的另一个意义意味着烦琐冗长的语句，那是要极力避免的。

- 数据何时得到保存

假设你输入了一些程序需要保存的信息。当你进行切换或程序退出时，当你需要每隔一段时间进行保存时，它是否会把数据按照你想的方式进行保存呢？何时完成呢？如果你对答案感到困惑，那就意味着可能有问题出现了。曾经在同事的项目中发现过很多次这样的情况：每次修改后直接关闭程序，却不提示用户保存——我只知道，修改信息在关闭时也消失了。在对某个模块进行修改时，你应密切注意这个问题。

- 很差的外部模块性

外部模块性指的是程序从外部看起来产品的模块化程度（如同程序是可分割的实体）。你如何容易地理解模块组件？很差的外部模块会耗费大量的时间来学习产品，还会吓跑新用户——它看上去太复杂了！尽可能让信息独立展示出来，对任何特定任务而言，你需要知道的越少越好。

### 1.2.2. 帮助文本和错误信息

帮助文本和错误信息通常被认为是产品的次要部分。它们可能是由低级程序员编写的（比如我）或是作者编写，对其进行更新的工作可能被赋予低优先级。然而，作为产品而言，这又是必不可少的部分，

一份看上去赏心悦目的帮助文档可以“主观[8]”的降低软件的学习难度和用户的使用兴趣。

当你感到困惑或是有麻烦时，寻求帮助或倾向于使用错误处理程序将是一个明智的选择。你可能会觉得不爽，更多的时候是不耐烦。而如果其中有信息误导了你，那么无异于火上浇油。以下列出的是我以往在审核帮助文档及错误信息时碰到的一些常见问题。

- 不合适的阅读层次

在计算机终端上，人们不能很好的进行阅读。帮助文本和错误信息应该尽量措辞简单明了，多用主动语态，尽量少使用技术术语，即便用户中有计算机经验也是如此。

- 冗长[9]

避免你的帮助文档和错误信息变成裹脚布。大多数用户在需要更多信息时，会选择通过菜单获得进一步的信息。最好让他们自己选择所需的信息。

- 不合适的情绪语气

尽量不要使用感叹号，如“终止”、“崩溃”之类的带有激烈意味的词语都应如此。

- 事实错误

记得测试时需要测试那些更新过的功能，在旧的帮助上的方法可能在新软件中是行不通的。这个时候开发人员的代码更新日志就显得很必要了，你不必对每项功能进行检查，完全可以把这类回归测试交给自动化测试工具完成，而你只需要关注其新内容即可。

- 上下文错误

不要把上下文之间的关系搞错了，这会带来阅读时的不便。比较清晰的方法是首先列出上下文关联列表，并按照操作顺序的先后进行组织。[10]

[8]是的，这是主观的想法，不是客观现实——用户的主观感觉在这种时候会产生相当的作用，而不是说软件的学习难度降低了。

[9]有必要澄清这一点：冗长和所开发项目的大小没有直接关系。一般来说，项目越大，所要聚集的元素也越多，则产生的文档内容亦越复杂，但不是“冗长”，系统的规模不能简单用冗长形容，况且这样做也不科学。

[10]在这一点上，可以借鉴商业上提倡的头脑风暴（Braining Storm）：它要求你把你所设想到的所有问题列在

- 没有识别出错误来源

通常，一个好的错误信息不仅可以指出是什么情况，而且还要指出为什么有些东西出了错，以及如何处理此类错误的方法。在过往的项目中，常常有很多这样的问题：如“打印错误”、“保存错误”等等含糊的说明。如果用户在获得了错误信息后，还是一脸茫然，那就应该认真考虑一下错误说明的编写方式是否可以再改进一些了。

- 不能立刻重现的错误很可能是个大问题

- 没有说明原因就禁止一个资源[11]

如果程序尝试使用打印机、内存控件等资源，却做不到，错误信息应当包含的不仅仅是宣布失败，还需要说明原因。

- 报告说没有错误

首先，还是先承认这种情况是不太可能会出现的；错误信息只能由错误状态出发，如果大部分是通常情况的调试消息，或者是少部分并不一定由某个缺陷引起的事件报告，那么，你可能就会忽略所有的错误信息。

### 1.2.3. 显示上的（问题）缺陷

这是一个比较客观的问题，至少表面上看上去是这样的。任何可见的错误都会产生让人不快的感觉（尽管这些问题不一定很严重），用户就不一定会相信或者购买该产品。可能是因为此类错误大多都是属于低级错误，通常并不受到开发人员和项目经理的重视，但是我们必须重视它——它就是问题（Bug），它就是我们要找的东西之一。

另外提一点：总是拘泥于这类Bug可能会使测试失去意义——放过重要的功能需求，“吹毛求疵”[12]。它可能是造成开发人员和项目经理不重视测试的一个借口。尽管如此，我们还是要提出这类问题，但并不是说可以遗漏重要的功能需求。

- 数据写到了错误的屏幕位置

光标提示在正确的位置，但是数据却显示在屏幕错误的位

一个草稿上，随后对其进行整理，在编写帮助的时候也可以这样做。

[11]在业界有这样一种说法：不能提示用户在力所能及的范围内改正已知问题的错误信息不是一条好信息。

[12]这个说法可能是刻薄了一些，但是事实确实如此，在我还刚刚入门的时候也曾为了这个问题而挨过骂。

置（张冠李戴）。这类错误对开发人员而言，不应该是个很棘手的问题，但对用户来说，那是致命的。

- 未能清除部分屏幕

一条信息在屏幕上显示了几秒钟，接着却只有一部分被擦除了；或者你对前一问题的回应仍然留在屏幕上，我们固然可以以计算机整体性能为借口，但也不能排除技术因素。为了输入一些新东西，不得不在提示或不相关的回应上输入，这是令人头疼而且迷惑不解的。在以前测试的项目中，就曾经出现过由于屏幕未正确刷新导致的清屏不完整及无故弹出不相关提示的问题——这种问题比较普遍，需要多加注意。

- 未能突出显示部分屏幕

如果程序常常需要突出显示某个特定类别的项目，例如提示或者在激活窗口中的所有文本，那么它就必须一直这么做。[13]

- 未能清除突出显示

屏幕位置的属性与显示的文本分开储存时这是很普遍的。程序员删除了突出显示的文本，但是忘记了从屏幕的那一区域清除突出显示。这类问题一般都和数据刷新有关系，无论是界面上的处理还是系统底层的处理。

- 显示的字符串错误或不完整

显示的消息可能是毫无价值的文本，一个冗长的信息的一个片断或是一条应该显示在其他时间出现的完整信息。这其中任何一条都可能反映出程序逻辑上，用来发现消息文本的指针的值或者已储存的文本副本中的错误。

- 显示信息过长或者不够长

消息在屏幕上显示的时间应该足够长，至少应该保持到能让用户读到结束为止。如果对同一条消息有时显示时间长，有时显示时间短则需要注意，这可能预示着外部资源之间的竞争条件（比如对内存资源的争夺），往往这些条件是在我们考虑之外的，需要认真对待。

[13]一致性一直是计算机软件体系的一个组成部分，也是古典工程美学的理论之一。

### 1.2.4. 界面布局的显示

屏幕看上去应该是很有条理的，别让它看起来像是一个乱糟糟的房间。不同类别的对象应该在可预知的区域分开显示。你可以参考一些关于 UI 布局的文章，但归结起来说：显示布局应该很容易让你在屏幕上找到你要的东西。

- 从美学角度看屏幕布局很拙劣

  屏幕可能是不平衡的，大多数情况下是这样子，行或者列不对齐，或者只是看起来很“糟糕”。好好利用你的鉴赏力，如果没有信心，可以问问别人的意见，参考一些界面设计很合理的软件。如果对你而言它的布局的确看起来很糟，相信你的直觉，肯定有什么东西错了，尽管现在你还没有发现。

- 菜单布局错误

  这是最常见的问题之一了：我们有时候会发现在编辑菜单下突然冒出了一个文件关闭的选项，而一般它是放在文件一栏下的。在很多的参考文献中，已经对这方面的内容做了比较详实的说明，我想强调的是以下一些问题：

  - 相似的或从概念上相关的菜单选择应该分组，或者应该在屏幕上说明。
  - 选择一个菜单项通常应该独立。为了获得一个独立的结果，用户不应该必须在不同的菜单上做出两个或更多的选择（这可绝对“难”用）。
  - 通过键入其首字母来选择菜单项通常要比使用数字来得好。当然，你要留神不要给菜单项过于奇怪的名称；另外，还要注意不要在同一栏下面不要出项重复的字母。

- 对话框布局错误

  - 对话框应该一致。如：他们应该一致使用大小写，字体和文本对齐规则。对话框标题应当占据某个一致的位置，并与用来调用该对话框的命令名相符合。相同的快捷方式在不同对话框之间应该起相同作用——如<ESC>不应取消某些对话框，而在其他类似情况下完成其他的任务。

- 对话框中的控件布局必须合理安排。应使用必要的间隔把组隔开。
- 选择和录入区域应该垂直和水平排列，这样用户就可以以直线模式操作光标的运动（为了方便）。
- 留意对话框之间的相互依赖性[14]。如果某个对话框中的选择将最终决定另一个对话框的选项将是令人困惑的。如果程序不得不这样做，务必要求开发人员给出具体提示。

- 模糊不清的指示

你应该总是知道去哪里查找以找出下一步。如果屏幕排得很满，总是应该为命令和消息留出一块空间。使用气泡显示信息也是一个不错得选择。

- 闪烁的误用[15]

闪烁的图片或文本很引人注意，不过记得不要太多闪烁。太多的闪烁会让人觉得不舒服。你应该每次最多只让一个目标进行闪烁而且频率不能太高。

- 颜色的误用

不要太多颜色，它会让我们的眼睛很疲劳。颜色不应该使我们分散注意力，也不能使屏幕看上去杂乱无章，尽量使用统一风格的颜色，如果程序的颜色组合看上去很难看，抗议吧，没有人会愿意买毫无美感的产品的。

- 过于依赖颜色[16]

如果程序在项之间使用颜色为唯一分隔符，那么它将限制使用者的范围，对于一些特殊的产品，需要考虑到例如色盲之类对颜色不敏感的人群或是使用单色显示器的用户。

- 与整体风格不符合[17]

如果与计算机相关的风格提供了某种一致性和便利，尽量好好利用。也许对程序员来说可以使用更好的技术来代替，对

[14]一般来说，对话框之间的依赖性越低越好，特别是针对系统给出的对话框。
[15]我个人的意见是：尽量不要使用。如果一定要用，绝对不要使用过于刺眼的颜色设置和频率设置。
[16]这一条是比较特殊的，如果不是因为一次偶然的经历可能自己都没想到这个问题。
[17]这个风格实际上也包含了UI界面设计的风格意味，而不单单只是指使用环境。

于用户来说也未必不是不可接受的。例如：在习惯了鼠标和图标之后，恐怕很少有用户会习惯敲击键盘书写命令来完成计算机可以使用鼠标完成的工作。当大多数其他的程序以某种特定方式在屏幕的特定位置显示错误信息时，新程序也应该是这样的。

- 不能去掉屏幕上的信息

在屏幕上某个部分的可用命令选择菜单是很好的选择。一旦用户精通了程序，有些菜单就会成为屏幕空间资源的浪费。你应该可以提交一个命令能去掉和重新调用它。这点上，值得向微软的 Office 系列软件学习。

## 1.3. 命令结构和录入

这里只处理实现中的缺陷。（即假定程序员对风格的选择是合理的）

### 1.3.1. 不一致性[18]

增加永真规则的数量可以缩短学习时间，并减少文档，而且使程序看上去更专业。不一致性如此的普遍，是因为它需要规划并进行斗争来选择能一直遵循的操作规则。每个微小的不一致性都是不重要的，但是一旦达到了一定量，一个本来构想很好的产品有可能会变得难以使用，甚至变成废品。从开发人员本身来讲，也体现出其开发本身的严密性。一个好的测试实践要标注出所有发现的不一致性，无论多么微不足道都要如此。

### 1.3.2. “最优化”[19]

程序员有时候会有意引入不一致性来对程序进行优化。的确很吸引人，但是也要注意优化所带来的风险和一些优化的必要性：保存一两次按键操作是否与学习时间的增加或信任度的减少价值相当？未必。

- 不一致的缩写

如果没有很明确的缩写规则，缩写就不能容易地被记住。把 Delete 缩写为 Del，把 Grep 缩写为 Grep，是没有任何意义的。

[18]实际上，本文档前前后后关于一致性的话题就从来没有停止过。
[19]最优化，始终都是要代价的。我们应该尽量追求的是最标准化而不一定是“最优化”。

- 不一致的终止规则

  程序应该为多重键录入要求终结符。

- 不一致的命令选项

  如果一个选项对两个或者更多的命令有意义，那么它就应该对这些命令都可用（都不可用），它应该具有同样的名称，并且应该在两种情况下以同样的顺序被调用。

- 名称相似的命令

  如果两个命令名称相似，就很容易搞混。尽量不要使用相似的名称命名命令。这个问题在中文界面的软件中表现得尤为突出。

- 不一致的大写

  如果命令录入是区分大小写的，所有命令的第一个字符都应该使用大写（小写）。命令中嵌入单词的第一个字符应该一直大写（小写）。另外，如非必要，不要在一个命令中使用多国语言。[20]

- 不一致的菜单位置

  如果同一命令在不同子菜单中出现，那么要在不同菜单的同一位置保留同一命令是很困难的。这句话不是很好理解，不过把话说白了就好理解很多：要保持命令在同一子菜单中的位置，而不是让它东搬西迁在其他的子菜单中停留。

- 不一致的功能键用途与其说明

  功能键的意义在程序中应始终保持一致（颠倒或是功能冲突是不能接受的）。

- 不一致的错误处理规则

  当程序检测到一个错误之后，它可能会公布该错误，或者尝试更正错误。任何一个程序的行为都应该是可预测的。如果当提交错误数据时没有任何的提示或尝试更正错误的说明，那么用户就无法确认数据是否是干净的。

[20]很显然，虽然意思一样，但“复制Text”远没有“Copy Text”或“复制文本”更有接受面。

- 不一致的编辑规则

当你输入或稍后检查任何数据时，同样的键和命令应该可以用来对其进行修改。

- 不一致的数据保存规则

应该在每处都以同样的方式，在同样的时间和范围内保存数据。它不应该在每个区域输入数据时保存数据，而其他时间则在一个记录、一组记录的末尾保存数据，或恰好在退出前保存数据。

### 1.3.3. 浪费时间

看起来为了浪费时间而进行的设计会激怒每个人，应该把时间花在更有意义的事情上去。

- 曲折路径

如果为了到达想要的命令，你必须一个接一个做出选择。结果到最后发现，该命令不存在、不能实现或者要求你先完成某件事甚至几件事后才能使用——明显的欺诈行为！相信客户的不满和你（测试人员）的不满几乎没有任何区别。举个例子说：当用户辛辛苦苦填满了整整一页的数据，最后提交时发现：该页数据中的某项已经被使用了时，用户的焦躁可想而知。

- 不能采用的选择

事实上没有任何接口在一个不能建立的菜单中包含选择项。如果没有任何数据存在，你如何评审、保存和擦除数据？没有打印机，如何打印文档？有的命令不适合出现在某些条件下（虽然对使用没有什么影响），但是开发人员可能为了图方便而保留了此类命令（很遗憾地说：这太不专业了）；有时候，程序会提示寻求帮助，而当你真的去使用的时候，程序却告诉你“您没有使用帮助的权限”——面对可望而不可及的东西，很多人宁愿选择不去看见。这种情况很常见，至于常常被开发人员和测试人员共同忽略——但这是不应该存在的错误。

- 你真的，真的确定么？

严重毁坏数据的命令需要这样重复的确认。是的，这是必

须的，如：对一个写满数据的硬盘进行格式化的确需要多次确认。但是没有必要对每个细小的删除操作进行繁复的多次确认操作，用户会变得不耐烦，其结果有可能是：当用户真的在进行严重毁坏性命令时，无视屏幕提示，造成不可预计的后果。

- 模糊不清或者带有个人风格的命令

命令名应该明确指示该命令的作用。不要让用户一边使用软件，一边查询用户手册中关于该命令的解释。调查表明：大多数用户在使用软件产品的时候只对软件手册做大概的了解，甚至根本不阅读。别指望用户会很完整地阅读手册，那是我们的工作。没有任何理由在发布的程序中生成如：finger 等带有明显个人风格的命令，当然，如果在程序中出现了脏话，我希望（仅仅是希望）客户没有气到火冒三丈。

### 1.3.4. 菜单

菜单应该尽量简洁，当存在了太多风格不一，冗长和拙劣的图标或命令名，以及当选择隐藏在不明显的主题之下的子项时，理解它们将会变得非常复杂。一个菜单覆盖的命令越多，无论如何规划，都会变得复杂，如果没有良好的规划，这对用户的使用将是一大障碍。

- 过于复杂的菜单层次

如果必须在最后到达你想要的命令之前很吃力地通过一个又一个菜单，那么我想我会选择使用另外一个功能相近的程序。创建深层菜单树的程序员所引用的设计规则表明，没有哪个菜单应该具有七个以上的选项，这一点对新手来说可能时最好的。有经验的用户更倾向于每个菜单级别有更多选择，犯更少错误并更快速有效的做出反应，只要选项能合理组织，整齐排版，而且没有可笑的拥挤或拼写错误就行了。

- 不适当的菜单导航系统

即使在一个最适当的深层菜单系统中，你也必须能够返回到前一菜单，或者移到菜单结构的顶层，并能在任何时刻离开程序。

- 太多路径到达相同位置

如果许多命令在菜单中重复出现，那么程序就需要重新组织。让一个命令在不同位置重复可能会很便捷，但是存在一定的限制。如果感觉上你可以从程序的任何位置到达另一个任意位置——那就得重新考虑下程序的内部结构和可靠性。

- 相关的命令被归属到不相关的菜单

把一个复杂菜单中对命令或主题进行分组并不是一件容易的事情。人们都很容易忽视两项之间明显关系，并把它们分配到分开的菜单中去，当需要对此进行调整时，我们需要做的是：解释两项之间的关系，并建议两者都应属于哪个菜单。

- 不相关命令被放置到同一菜单下

有些命令被扔到了完全无关的菜单下，这样并不好，宁可重新选择一个更高级别的标题并重新组织这些命令也不要那么做——如果那样做导致的混乱比较严重的话。

### 1.3.5. 键盘不能正确使用

不能正确使用键盘无论在何时都是一个问题。

- 无法使用编辑键或功能键

如果一个程序从某些其他没有这些键的机器上移植过来，那没关系。相反可能就不行。要确保程序可以使用已有的编辑键和功能键。

- 光标和编辑键的非标准使用

这些键应该按照他们通常在该机器上工作的方式进行工作，而不是按照他们通常在其他某个机器上工作的方式来工作。

- 功能键的非标准使用

如果大多数程序用 F1 作为帮助键，那么如果在程序中将它定义为其他的功能，那将是不合适的。

- 不能过滤无效键

程序应该能挡住并抛弃无效字符[21]，比如进行数字相加时输

[21]这类控制可以避免大多数的输入验证错误，尤其是针对特定数据类型时。

入的字母。它不应该做出回应。与错误消息相比，这样做更快更有效。

- 未能指示键盘状态的改变。

  键盘上的灯或屏幕信息应该告诉你何时你的 Num Lock 键和其他类似的状态转换特征是开着的。

- 未能扫描功能键或快捷键

- 你应该能够告诉计算机从它正在进行的工作中退出；程序应该总是能辨别出任何其他系统指定的键——即那些本机上的程序通常可以很快识别出来的键。

## 1.4. 遗漏的命令

### 1.4.1. 状态转换

大多数程序从一个状态转到另一个状态，在你选择某个菜单项或者提交一个命令之前，程序处于某种状态。为了回应你的选择，程序回到另一个状态。程序员通常会对他们的代码进行足够的测试，以确保你能达到任何你应该可以达到的状态。

- 什么都不作就退出（状态返回）

  你应该能够告诉应用程序，你做出的最后一个选择有误，并返回到其前一个状态。

- 不能在程序中间退出[22]

  当使用一个程序但还没有对存储的数据造成不利影响时，你应该能够从中退出；如果你正在编辑的文件出现了预想不到的错误，在中止后应能回到先前保存过的状态。

- 不能在命令中间停止

  告诉程序停止一个命令很容易，而返回到起始点或选择一个其他的命令也应该不太难。如果其中任何一个方面出现了问题，就需要重新考量先前的设计是否真的合理了。

[22]可惜这个方式在操作数据库时较难应用，一般都是直接对数据库进行操作，一旦出现这类问题，很可能在数据库中会出现脏数据。

- 不能暂停

如果程序限定了你输入的时间，时间一到，状态就改变，那么当你离开时你就需要它暂停一会儿。这中类型的情况在游戏软件中见得较多[23]，比如说暂停游戏。

### 1.4.2. 危机预防[24]

系统故障和用户错误发生了，程序应具备将其后果降到最低的能力。

- 没有备份工具

对开发人员而言，为了一个文件做一个额外备份应该不是一件困难的事。如果你正在修改一个文件，计算机应当保留原始版本的一个副本，因此如果你的更改有错误，还能返回到一个已知的好的版本。

- 不能撤销

撤销一个你已经发出的编辑命令，至少是一步。恢复被删除的文件是一种受限制的撤销，它能让你恢复错误删除的数据。撤销是可取的，恢复被删除文件也应是必须的。

- 没有“你确定吗？”的提示

提交一个大量数据清除的工作，或者提出一个清除少量数据但是会影响其他作业的命令或者很容易错误提交的命令，都需要程序在用户操作时进行确认，不声不响地进行将会带来安全方面的隐患（尤其是在后台进行暗箱操作时，这种危险性更高）。

- 没有增量保存[25]

当输入大量文本或数据时，你应该能告诉程序相隔一定时间对你的工作进行保存，至少应该提供用户此类选项。对于突发的掉电和硬件损坏情况这样做将是非常有好处的。

### 1.4.3. 由用户进行的错误处理

[23]虽然自己很喜欢玩游戏，可惜一直都没有机会参与游戏的测试：（
[24]危机预防机制是一种确保用户数据安全的方式。
[25]这类设计不一定都是最好的，但它的确给用户带来了实惠。

人们可以捕获自己的错误，而经验告诉我们，他们还容易犯其他的错误。他们应能自理修复错误，并建立自己的错误检查机制。

- 没有用户能指定的过滤器

  当设计数据录入表格和电子表格模板时，你应该能够让每个区域指定什么样的数据类型有效、程序应忽略或拒绝什么。例如，你可以让程序拒绝数字、字母、不在某个特定范围内的数值，一个有效日期或者与磁盘上匹配的日期等等。

- 难用的错误更正

  修改一个错误应该很容易。不应该因为犯了错误的数据录入而让整个系统重新启动。在输入一串数据时，你应该能在不重新输入剩余部分的情况下，更正错误的数据。

- 不能包括注释[26]

  当设计数据录入表格，电子模板，专家系统时，你应该能够为未来参考和调试输入注释信息，这是很必要的。

- 不能显示变量之间的关系

  录入表格、电子模板中有些变量是相互关联的，应该能很容易的检查任意变量对其他变量的依赖性。这在设计时应该被周密考虑到。在大多数的财务管理系统中，这类应用是较普遍的。

### 1.4.4. 其他问题

- 隐私和安全性[27]

  对于某些特定的程序，需要考虑数据的充分安全性，保护公司，集体或个人的机密和隐私。在多用户系统上，你应该能很好的隐藏你的文件（一般都是通过加密手段实现的），并且能很好的锁定你的文件不被篡改或阅读。

- 安全性的困扰

  一个程序的安全性控制应尽可能谨慎考虑与实施。

[26]严格来说，这不是一个传统意义上的BUG，更像是一个可参考的意见。
[27]近几年的发展方向。人们对于自身隐私和安全性的考虑已提升了许多。

- 隐藏菜单[28]

很多程序在顶端、底部或屏幕边缘显示一个菜单（包括我以前测试的大多数应用程序）。它们使用屏幕的剩余部分作为数据录入和处理的区域。菜单只是记忆的辅助物。一旦用户了解了他所需要的所有命令后，就应该给予用户充分的自由，让其选择是否需要保留菜单的权力。

- 不支持标准 O/S 特征

例如：如果系统使用了子目录，那么程序就应该能引用其他子目录。如果操作系统提供了通配符（比如*），那么程序也应该能使用。

- 不允许长名称

应当允许使用长名称（只要不是太离谱），毕竟，内存不足和编译器反应迟钝的时代已经成为过去了。别让自己的程序也成了文物。

### 1.5.程序僵化

程序有灵活与固定之分。灵活的程序更显个性化一些，而固定僵化的程序一般都是由于业务流程的关系而不得不如此。如借还书的过程，必先借了才能还。别给用户太多的自由，否则程序看起来会让人觉得松散，也不要把程序定得死死的，那看上去给人太多压力了。

### 1.5.2.用户可调整性

- 不能关掉噪音

犯错误时，许多程序都会给出“咄”的警告声。而如果每次接触键盘都发声，这简直是不能容忍的——特别是在公共场所。必须关闭没有必要的噪音，至少程序要提供这类控制选项。

- 不能关闭大小写区分

一个能区分大小写的系统应该允许你告诉它忽略大小写的问题。

[28]很多软件在这一点上做得足够好了。用户个性化的菜单定制也是关怀用户的一种有效方式。

- 不能配合手边的硬件

有些程序被锁定针对了特定的输入输出程序。升级或更换了设备的用户可能就无法使用这些功能了。这是令人遗憾的。同时，也会让用户觉得是否是一种商业捆绑的模式而拒绝使用此产品。尽量让开发人员编写通用的硬件接口代码以适应大部分通用的硬件设备。

- 不能改变设备初始化

一个应用程序应该能够发送用户定义的初始状态，或者至少应该让它保持现状。每次启动都需要重新配置将会是很烦人的工作。假定你想要向一个打印机发送控制代码，以转换到压缩字符，如果打印数据的程序不让你初始化打印机，你就不得不从设备来改变打印机的模式和状态，然后重新运行程序。如果程序阻挠了你的打印机设置，那就是一个设计错误[29]。

- 不能关闭自动保存

自动保存是件好事，不过无事生非也是一件糟糕的事情。过于频繁的自动保存可能会让用户觉得程序不可靠。所以还是老老实实加上关闭自动保存的选项吧。

- 不能改变滚动速度

严格来说，这不是一个很严重的问题。目前很多的设备驱动中已经提供了此类选项。

- 不能重复上次的操作

这样的例子比如 Word 软件中的 Redo。

- 无法找到你上次完成的内容[30]

此类问题对数据编辑特别是文本编辑类的程序较为常见。应当提供保存的文件列表，除非用户禁止了它。

- 无法执行一个定制的命令

在程序选项中，一旦对其更改，应该立即生效，不需要再

[29]发现设计错误最好的解决办法就是立即修改原先的设计，时间拖得越久，代价就越高昂。
[30]这也是一个贴心用户的设计，虽然没有达到这种要求还称不上是非常严重的问题，但是用户会更乐意使用具备该设计的程序。

次重新启动程序进行加载——特殊情况除外（如果你无法避免）。

- 无法保存定制的命令[31]

你不应该只告诉程序此次运行关闭了警告声，而是应该让程序永远可以关闭这些——只要设置没有发生变化。

- 特征更改的副作用

这种情况较常见，修改了某一个特征，相关的特征也会改变。如果确实存在副作用，应当在手册和屏幕中提供详实的文档证明和提示。

- 无限可调整性

开发人员可以改变某些程序的方方面面。但是在开篇时说过，提供了太多灵活性的程序并非一直都是一件好事。好好斟酌再做决定要比草率地修改来得更好。

### 1.5.2. 控制方式

有些程序很“霸道”。它们的错误信息和帮助消息自认为高人一等，它们的风格不可原谅的不便——你甚至无法放弃命令或者在输入数据后进行更改。程序应该使你觉得能更容易，更惬意地完成一个任务。至少，它不应该浪费你的时间。

- 一个概念风格的不必要和不合理要求

有些程序要求你以某种顺序输入数据，或要求你在进行下一步之前先完成某项任务，再要么就要求你再考虑它们的潜在后果之前做出决定。例如：

当设计一种数据录入格式时，为何你必须再屏幕显示之前确定一个数据录入域的名称，类型，宽度或者计算顺序？当你察觉不同的域放在一起看起来有神么不妥的时候，你是否会更改某些域，把它们的位置换来换去，甚至去掉少数域？你可能不得不在使用该格式之前输入域的规格说明，但是在该约束条件下，你应该决定何时填充细节。

当向一个项目管理系统描述任务时，为何你必须首先列出

[31]有些程序在启动时从外部文件初始化设置数据，个人认为：这是个值得借鉴的设计方式。

所有的任务，然后列出说有可用的人员，接着在为下一项工作输入任何数据之前就把分配给某个人的工作完全对应？既然你可能试着决定什么工作分配给什么人，那么你不想看到结果后再更改这些数据么？

限制的数量如此多，是因为有些开发人员认为，人们应该以某种方式组织它们的工作。但是他们所想的最佳途径未必都是最佳的。我们应该更清醒地意识到，除了业务流程上的禁锢，不需要再对用户在风格上多加任何的限制——当然，如果用户需要的话。

- 对新手友好，但是对老手并不一定友好[32]

为初学者设计的进行过优化的过程可能为他们掌握系统会有帮助，但是同时会令一些有经验的老手带来烦恼。他们更希望能自由地使用软件；其中一个解决办法就是提供两条以上地路径实现对不同层次用户的需求。

- 人工智能与自动化

有些更智能和更便利的程序会猜测你的下一步动作，并枉加执行这些猜测；自动纠错的程序的确很好，除非它“纠正”了正确的数据。而有时候，用户并不一定希望这样做。提供可用的选项可以缓冲一下这方面的矛盾。

- 过剩或多余的必需信息

过剩和多余的必需信息就像我们身上的赘肉一样讨厌。有些程序会请求他们从来不会使用或者只会用来在屏幕上显示一次的信息，或者会要求你重新输入你已经输入过的数据——并非是为了验证，仅仅只是重新获取一次数据，这对时间的浪费是非常惊人的。

- 不必要的步骤重复

如果你在输入一个较长的命令步骤或数据时犯了一个错误，有些程序就不管青红皂白让你重新输入；有的程序则强迫你重新输入或确定可能有错误的任何命令。为了某些任务，你甚至可能需要确认每个步骤。相信我：不必要的重复或确认都

[32]实际上，这个问题还是很难界定的。

是浪费时间。

- 不必要的限制

为什么要把一个数据库限定为如此多的字段或记录，或限制一个电子表格仅使用数字，把一个项目经理限制在如此多的任务中，一个字处理程序限制在如此多的字符呢？对性能或可靠性而言并非必要的限制不应该成为约束条件。

## 1.6. 性能

许多有经验的用户认为性能是最重要的可用性因素：使用一个快速程序，他们感到更能集中精神工作，而且更多的东西处在掌握之中。在极少出现异常的情况下，程序越快越好。

性能有很多定义，大致来说主要有如下的感性认识：

- 程序速度：执行标准任务的速度有多快？
- 用户吞吐量：你能使用程序执行标准任务的速度有多快？
- 感觉到的性能：在用户看来，该程序速度有多块？它是否令你满意？

无论如何定义，程序速度总是一个关键因素。但是一个界面设计拙劣的快速程序无论怎么看都要比它实际的运行和处理速度要慢得多。

### 1.6.1. 降低程序速度

很多设计和代码错误会降低一个程序的执行错误。程序可能会执行很多不必要的工作，如对一个在读前会被重写的内存区域进行初始化；也可能会对工作进行不必要的重复，如在一个在循环中执行的任务可以在循环外完成；设计的决策也会影响到程序的速度，而且通常要比明显错误导致的慢速情况更加严重。[33]

### 1.6.2. 缓慢回应

程序应立即对输入做出响应，如果在你输入一个字母的时刻和你看到这个字母的时刻有延迟，显然，程序就太慢了。快速反馈对任何输入事件都必须是有效而且是必要的，它包括：鼠标，键盘，轨迹球，

[33]实际上，在程序设计阶段进行的工作将有可能直接影响到程序的性能表现。

键盘等等。

### 1.6.3. 如何减少用户吞吐量

一个闪电般快的程序执行任务时可能比蜗牛还要慢。这包括：

- 任何可能使用户错误更可能发生的事情。（培训不周，用户习惯，程序风格等等）
- 缓慢的错误恢复。如：在输入一长串字符后发现错误却必须要重新输入。
- 任何使你感到迷惑，却得不到帮助文档或手册提供资料的事情。
- 输入很多，却做得很少的程序——这不是一个好程序。如：把一个简单任务划分成很小的子任务，要求对所有事情进行确认等等。

在测试时，使用比较性测试是个有效的方法：即把开发中的产品与竞争对手的产品进行比较，如果人们使用你的产品花费的时间要更长，那么发现这个问题就是有意义的。

### 1.6.4. 反应拙劣

我曾经测试过一个产品，在输入第一条数据后，居然花了将近一分钟才从数据库中将数据取回。这不能不说是一个反应很慢的程序。一份表单做下来将近 300 条数据，按此速度计算，我将花上至少半天才能完成输入。这种情况是不能允许的。迅速地对输入做出回应是一个程序最最基本的功能。一个反应迅速的程序不会强迫你在提交下一个命令之前让你持续等待，而是让你继续做其他事。

### 1.6.5. 没有提前输入

一个允许提前输入的程序会让你在它从事其他工作的时候仍然可以键入，它会记住输入内容，加以显示并在稍后进行处理。你不应该等着输入下一个命令。

### 1.6.6. 没有给出某个操作会花很长时间的警告

如果程度需要超过几秒钟的时间来进行某事，程序应该告知用户。对于较长时间的任务，它应该给你一个大概的时间印象而不是让

你干等着它结束。给出大致需要完成的时间或是进度条是处理此类问题一般的方法。

### 1.6.7. 程序太多提示和询问[34]

提示，警告以及询问可能很有用，不过如果它们出现得太频繁，就会让人很窝火。

### 1.6.8. 尽量使用简单命令和提示

在慢速终端上，帮助文本，长菜单以及漂亮的图片常常会令人不耐烦。你应该使用简要的语言取而代之。不要使用诸如“你真的想以500k/s 的速度传送此邮件到某邮箱”么之类罗嗦的语句。

## 1.7. 输出

程序的输出应如输入一样完整。它要求更精确，尽量快速和能实现多路径及对输出内容更有效的管理，这四类标准几乎决定了输出功能的主要表现特性。

### 1.7.1. 不能输出某种数据

你应该能打印出你输入的任何信息，打印不出输入的内容对任何程序而言都是致命伤。

### 1.7.2. 不能重定向输出

你应该可以重定向输出。特别是，你应该能向磁盘发送一个很长的“打印输出”标记，并稍后打印该磁盘文件。程序不应该阻止你把数据输出发送到预料之外的设备，如绘图仪，磁带，打印机等。

### 1.7.3. 与一个后续过程不兼容的格式

如果一个程序声明能够以第二个程序可以理解的格式保存数据，那么就应该测试它是否可以真正做到。这意味着购买或借用第二个程序的副本。使用第一个程序保存数据，用第二个程序读数据，同时看看第二个程序得到了什么，这是对此进行测试的最简单方法。

### 1.7.4. 必须输出的很少或很多

你应该可以修改报告，从而呈现你需要的信息。不得不在仅包含少量有用信息行的打印输出的大量页中找出所需信息，几乎和没有得

[34]题外话。“礼多人不怪”，实际的情况是，礼多人不怪，只不过心里的感觉不自在。

到信息一样糟糕。

### 1.7.5. 不能控制输出布局

你应该可以改变字体，对输出信息增加特殊标记来强调信息。你应该可以修改信息之间的间距，最低限度来说，程序应该可以以一种由合适文字处理进行修饰的格式把报告输出到磁盘文件。

### 1.7.6. 荒谬的精度输出级别[35]

要是说 4.2 加上 3.1 等于 7.3000000 或者说 3.1111+2.11 等于 5.22110102 都是很愚蠢的。在最终输出的结果中，程序应该按照规定的格式和精度输出最后的数据。

### 1.7.7. 不能控制表或图的标记

你应当能够改变字型，措辞及任何说明，包括标题，表格，图形或是图表中文本的位置。

### 1.7.8. 不能控制图形的缩放比例

绘图程序应该提供默认的垂直和水平比例，不要告诉我你最后输出打印报表中的图形超出了整个页面或是只占据了整个页面的一角。

## 2. 错误处理

在处理错误时发生的错误通常是最常见的缺陷。错误处理产生的错误包括：未预料到错误发生的可能性并防止其发生，没有注意错误状态，以及较严重的：程序可能与错误数据一起工作并最终产生错误结果的情况。[36]

### 2.1. 错误预防

程序应具备这种能力：它能保护自己不受到系统其他部分的影响（包括有害输入和有害处理）。如果程序可能与错误数据共同工作，确保其在发生严重可怕的影响之前（如程序崩溃，数据丢失与错误，系统崩溃等），检查并消除这些问题。

### 2.1.1. 不充分的初始状态验证

[35]有一些精度输出是必需的，比如说：金融业。
[36]这类情况比较难发现，一般都不会提示已经发生了错误，或是发生了错误，数据已经在不经意的时候被程序更改，而测试人员无法对其进行校验而遗漏。

如果内存的某个区域必须以其中所有位都是 0 开始，那么程序应该可以运行一个抽样检查，而不是假定已经存在 0 值。这会导致程序初始化时发生内存错，甚至不能启动。

### 2.1.2. 不充分的用户输入检查

此类问题非常常见，开发人员可能会在编写程序时遗漏掉大量这方面的问题。告诉人们输入 1 位到 3 位数是不够的，有些人可能会输入 5 位甚至更多，也有人会输入特殊字符或是运算符，还有些人会按下功能键一次或多次，如果程序允许输入，那么程序就应能顺利应付，而不是一打非专业人士不能明白的提示甚至更糟的情况。

### 2.1.3. 对受损数据不能充分预防

没有人能保证磁盘上的数据是好的。可能是有人已经编辑过或者根本是有硬件问题。即使开发人员认定在保存前的文件是有效的，那么他也应该检查（校验）下次打开的是否是同一个文件。

### 2.1.4. 不充分的参数传递测试[37]

一个子程序不应该假定得到了正确的调用（事实上，采用相反的想法可能会让测试进行得更加顺利一些）。它应该确保传递给它的数据在其可控制的范围之内。

### 2.1.5. 针对操作系统的预防不充分

操作系统存在缺陷——不光是过去，现在甚至将来也是，应用程序可能会触发其中存在的问题。如：如果开发人员知道，他把数据送到磁盘后很快又把数据送到打印机，会引起一些操作系统的崩溃，那么他就应该确保程序在任何情况下都不会这样做。

### 2.1.6. 不适当的版本控制[38]

如果可执行代码不止存在于一个文件中，那么有人会尝试把某一文件的新版本和老版本一起使用，客户对其软件升级使得这类问题时常发生，但是又不能明白出了什么问题。新版本应确保所有的代码都是最新的，当然也要对老版本有完整的备份。

### 2.1.7. 针对恶意使用的不充分预防

[37]这类问题使用白盒测试的方式要更容易检测，使用黑盒测试方法检测需要花费的时间和精力可能要更多一些。

[38]该问题属于配置管理应关注的范畴。严格的配置管理下是不容许出现这样的问题的。

人们有时会有意无意的提供程序有害输入，或者尝试触发错误状况（特别是做测试的家伙们）。不要假定“任何有理智的人都不会这么做”，相信我：程序不完全是为了“有理智”的人开发的。

### 2.2. 错误检测

程序通常有足够的可用信息来检测数据中或其操作中的错误。这部分内容将指导一部分常见的错误检测方式并对其进行归类。

#### 2.2.1. 忽视溢出

一个数值计算结果对于程序来说太大以至于无法处理时，就会产生溢出。溢出由较大数字相加和相乘或者被 0 除或是由于过小的分数除而引起。在有既定规则的情况下，溢出是很容易检测到的。

#### 2.2.2. 忽视不可能的值

在计算机当中，几乎没有什么不可能发生的事。程序应该检查其变量，以确保它们在合理的界限之内，它应可以捕获并拒绝如 2 月 31 日这样的日期值。如果当变量为 0 时，程序完成某动作，变量为 1 时完成另一工作，并假设其他所有值都是不可能，那么就必须保证变量只能为 0 或者是 1，对其他所有值进行额外的处理。在一个项目中，经过数年维护编程后，旧的假定就不一定安全了。[39]

#### 2.2.3. 忽视看上去不真实的值

有些人可能会从自己帐户里提取 100，000，000，000 的钱，那么就算他有这么多钱，也应该在通过交易之前向几个不同的人进行确认。这类看似荒唐的用例往往包藏着错误的祸心，应该小心应付。

#### 2.2.4. 忽视错误标志

程序调用了一个子程序，结果操作不成功，它在一个被称为错误标志的特殊变量中报告了该失败。与通常一样，程序能检查或忽略它，并把从例程返回的误用数据当作真实的进行处理。

#### 2.2.5. 忽视硬件缺陷或错误情况

程序应该假定它能连接的设备是失败的，许多设备能够发送警告某件事情出错的返回信息。如果有设备这样做了，停止尝试与其交互，

[39]这个问题的根源源自于人的麻痹和先入为主的思维方式：认为以前没有出现的错误现在也不可能出现。这对测试人员来说是需要杜绝的问题之一。

并向某人或更高级别的控制程序报告该事件。不能忽视该类情况以避免造成不必要的困扰。

### 2.2.6. 数据比较

结算银行存折时，你有一个你自己认为的余额数值，银行也会提供一个余额数值。在考虑了服务费，银行利息，最近帐单等等数据之后，两个数据不吻合，那么就要好好查一查了。在互相检查两个数据集或计算结果集时，也会产生类似的情况。

## 2.3. 错误恢复

程序中存在错误，程序已经检测到了错误，而且现在正设法对其进行处理。许多错误恢复代码只是稍微进行了测试，或者根本没有进行测试。错误恢复例程中的缺陷可能比原始问题更严重。

### 2.3.1. 自动错误更正

通过检查其他数据或规则集，有时程序不仅能检测错误，而且还能纠正错误，而用不着麻烦任何人。这样的程序是令人满意的，但仅当这种“纠正”是正确的情况时才是如此。

### 2.3.2. 未能报告一个错误

程序应该报告任何检测到的内部错误，即使它能自动纠正错误产生的后果也一样。在稍微不同的环境下，它可能检测不到相同的错误。程序可以向用户报告这一错误，也可以向一个多用户系统的操作员，向磁盘上的日志文件，或者是这些对象的任一组合报告错误。总之，不管怎么样，只要发生了，它就必须报告。[40]

### 2.3.3. 未能设置一个错误标志

某子程序被调用，但是结果失败，假定它在失败时设置了一个错误标志，它把控制返回给调用程序，却没有对这个标志进行设置，那么调用程序就会把无用数据当作有效数据传回去，这是应坚决避免的状况。这可能会造成数据冗余，脏数据或是直接导致当前操作失效，严重的则会引起崩溃。

### 2.3.4. 程序要走向何方？[41]

[40]没有理由隐瞒错误的发生，无论是对开发人员而言还是你或是用户。
[41]这不是一个哲学命题，值得庆幸。但是这个问题仍需要我们进行思考。

一部分代码失效了。它记录了错误，并设置了一个错误标志，接下来干嘛呢？尤其是经过了几个跳转的情况下，它如何才能得知程序中的什么地方返回了控制？

### 2.3.5. 中止错误

停止了程序，或者当它在检测到错误时自动停止了，那么它是否关闭了任何打开的输出文件呢？它是否在关闭时会记录退出的原因呢？在最普通的条件下，在即将结束之前它是否进行了整理或者它只是结束但可能留下一团混乱呢？这都是开发人员和测试人员需要考虑的问题之一。

### 2.3.6. 从硬件问题中恢复

程序应该适度地处理硬件故障。如果磁盘或其目录已满，你应该能够放入一张新的磁盘，而不只是关闭了你所有的数据。如果一个设备很长时间还没有准备好接收输入，你就没有应该假定它已经断线或断开连接而进行处理。程序决不能够让我们永远在那里坐等。

### 2.3.7. 不能从遗失磁盘中退出

假定你的程序要求你插入一张具有它需要的文件的磁盘，如果插入的磁盘不正确，它会再次提醒你，直到插入了正确的磁盘为止。然而，如果没有正确的磁盘，你就没有任何办法可以退出，除非你重启系统。不，这种做法是极端蛮横的，决不能允许出现这样的问题。

## 2.4. 边界相关的错误[42]

一个边界描述了程序的一个改变点，假定程序在边界的一边以某种方式做所有事，而在边界的另一边，它以不同的方式完成所有事。

边界相对立的两边的典型“东西”就是数据值。存在三种标准边界缺陷：

- 边界情况的处理不当

  如果一个程序把任何小于 100 的两个数相加，不接收任何大于 100 的数，那么当你恰恰输入 100 时它会做何反应？它又该怎么做？

[42]在软件测试领域里，有一种方法称之为边界值法，使用的思想和下面所列内容有共通之处。

- 错误边界

规格说明表示，程序应该把任意两个小于 100 的数相加，同时不接收大于 95 的数。

- 边界外情况的错误处理

边界某一边的值是不可能，不可信，不能接收，或是预料之外的，没有为其编写任何处理代码。程序是否成功拒绝了大于 100 的值？或者是否当它获取了一个大于 100 的值时就会崩溃？

我们把边界的概念看得更广泛，边界描述了考虑一个程序以及它在其极限周围得行为得方式。存在很多种的极限：最大，最小，最新，最旧，最近，第一个等等。相同类型的缺陷可以伴随其中任何一种极限而产生，我们可以用相同或类似的观点考虑它们。

### 2.4.1. 不同边界错误的考虑方式

不同类型的边界，其考虑方式也是不同的，但是其思想基本上都相近：无外乎上溢出与下溢出。

### 2.4.2. 数值边界

有些数值边界是任意的，如大于 100；而有些则要描述自然极限，如三角形的特征和子母的 ASCII 码等。

### 2.4.3. 与一个边界相等

在一个列表中的所有元素可能相同，也可能不同。如果你试着对任一列表进行排序，会发生什么？如果列表由数字组成，当你尝试计算平均值，标准偏差，对称系数时又会发生什么？（以上是概要统计概念，按算法，对称系数可能会计算为 0 或引起被 0 除的错误。）

### 2.4.4. 多种多样的边界

一个输入串可以长达 80 个字符么？如果你输入 79、80 或 81 个字符会如何？程序是否在每种情况下都接收你的输入？一个列表可以只是一个元素么？没有元素可以么？仅含一个数值的标准偏差又是什么呢？

### 2.4.5. 空间中的边界

例如，如果一个绘图程序绘制了一个图形，并在其周围绘制了一

个方框，那么该如何处理一个应当在方框外正确显示的点？

### 2.4.6.时间的边界

假定程序显示了一个提示，停留 60 秒等待你回应。然后，如果你没有输入内容就显示菜单，如果正当它开始显示菜单时，你开始输入内容，会发生什么？

假定你在计算机仍然在从磁盘中装入程序时按下空格键，发生了什么事？空格键是被发送给操作系统，为正在装载的程序进行了保存，还是仅仅因为预料之外而导致计算机崩溃？

### 2.4.7.硬件相关边界

如果一台主机可以连接 100 台终端，那么当你加入 99，100，101 台时会发生什么？如果你让 100 台终端同时登陆会怎样？

如果磁盘已经塞满了会如何？如果一个目录能保存 1000 个文件，当你尝试保存第 999、1000、1001 个的时候会发生什么？如果打印机有较大的输入缓冲区，当你的数据填满缓冲区，但是却还没有更多数据要传递时，会发生什么？当打印机缺纸或颜料用完了又会发生什么？

## 2.5.计算错误

程序计算一个数据得到了一个错误的结果。发生计算错误通常因为下面三种类型的原因：

### 2.5.1.很差的逻辑：

可能存在一个录入错误，开发人员可能会在编制程序时无意中错误简化了复杂关系地表达式，或者由于拼写错误或笔误导致。另外一种糟糕的情况则是设计错误，开发人员关于代码如何做的概念可能一开始就错了。

### 2.5.2.很差的算法：

如果 1＋1＝1，可以理解为一种特殊逻辑，但是如果把它作为数值运算，恐怕是没有人会同意的。无论何时，一个错误的算法总不会得出正确的结果。

### 2.5.3.不精确计算：

由于使用了舍入的计算方法，很可能在计算时丢失精度。

## 3. 小结

这篇文章最早成形于 2004 年 10 月，当时正接手一个项目的黑盒测试工作。然而，在对软件项目实施黑盒测试的过程中，的确也看到了很多值得思考的地方。作为一名刚进入测试行业的学步小儿，也谈不上有什么丰富的经验累积，唯希冀本文能给刚进入测试领域的同仁提供了一些有价值的参考。

# 漫谈人机界面测试

中国系统分析员协会高级会员，51Testing顾问 张华

【摘要】俗话说“人靠衣裳马靠鞍”，良好的外观往往能够吸引眼球，激发顾客（用户）的购买欲望，最终达成商业利益的实现。软件的设计亦如此，Window XP 在商业上的巨大成功很大一方面来自于它一改往日呆板，以突出“应用”的灰色界面，从“用户体验”角度来设计界面，使界面具有较大的亲和力。就目前的软件设计的发展趋势来说，良好的人机界面设计越来越受到系统分析、设计人员的重视。但是如何对设计的人机界面（包括帮助等）进行测试，给出客观、公正的评价，却鲜见于报端。本文试从共性分析和个性分析的角度，给出一些测试意见和原则，简单且易于上手。起到一个抛砖引玉的目的、以飨读者。

我们知道：“不立规矩无以成方圆”。在软件界面设计强调张扬个性的同时，我们不能忘记软件界面的设计先要讲求规矩－简洁、一致、易用，这是一切软件界面设计和测试的必循之道，是软件人机界面在突出自我时的群体定位。美观、规整的软件人机界面破除新用户对软件的生疏感，使老用户更易于上手、充分重用已有使用经验，并尽量少犯错误。由此我们在对软件人机界面进行测试时（设计评审阶段和系统测试阶段结合进行），不妨从下列一些角度测试软件的人机界面。

## 1. 一致性测试

一致性使软件人机界面的一个基本要求。目的是使用户在使用时，很快熟悉软件的操作环境，同时避免对相关软件操作发生理解歧义。这要求我们在进行测试时，需要判断软件的人机界面是否可以作为一个整体而存在。下面是进行一致性测试的一些参考意见：

- 提示的格式是否一致
- 菜单的格式是否一致
- 帮助的格式是否一致
- 提示、菜单、帮助中的术语是否一致
- 各个控件之间的对齐方式是否一致
- 输入界面和输出界面在外观、布局、交互方式上是否一致
- 命令语言的语法是否一致

- 功能类似的相关界面是否在在外观、布局、交互方式上是否一致（比如商品代码检索和商品名称检索）
- 存在同一产品族的时候，是否与其他产品在外观、布局、交互方式上是否一致（例：Office 产品族）
- 同一层次的文字在同一种提示场合（一般情况、突显、警告等）在文字大小、字体、颜色、对齐方式方面是否一致
- 多个连续界面依次出现的情况下，界面的外观、操作方式是否一致（当然可能会有例外，比如操作结束的界面）

**2. 信息反馈测试**

假设系统的使用者是一个初出茅庐的生手，你能指望她（他）在进行操作不出错吗？但这还不是问题的所在，问题的所在在于我们都会犯错误，我们都有自己不了解的东西。如何避免，这要求我们的人机界面有足够的输入检查和错误提示功能。通过信息反馈，用户得到出错提示或是任务完成的赞许之语。但有些不幸的是，我们很多系统都在此方面做的不尽人意。下面是这类测试的一些参考意见：

- 系统是否接受客户的正确输入并做出提示（例：鼠标焦点跳转）；
- 系统是否拒绝客户的错误输入并做出提示（例：弹出警告框，声响）；
- 系统显示用户的错误输入的提示是否正确，浅显易懂（例：“ERR004”这样的提示让人不知所云）；
- 系统是否在用户输入前给出用户具体输入方式的提示（例：网站注册程序）；
- 系统提示所用的图标或图形是否具有代表性和警示性；
- 系统提示用语是否按警告级别和完成程度进行分级（若非某些破坏性操作，请对用户温和一些）；
- 系统在界面（主要是菜单、工具条）上是否提供突显功能（比如鼠标移动到控件时，控件图标变大或颜色变化至与背景有较大反差，当移动开后恢复原状）；
- 系统是否在用户完成操作时给出操作成功的提示（很多系统都缺少这一步，使用户毫无成就感）。

**3. 界面简洁性测试**

你的人机界面像你的脸一样对称、干净吗？我们往往看到的使很多系统在人机界面设计上就像长了天花的病人。因此我们不得不对其进行美容前的检查，下面是一些供检查的建议条款。

- 用户界面是否存在空白空间（没有空白空间的界面是杂乱无章的，易用性极差）；
- 各个控件之间的间隔是否一致；
- 各个控件在垂直和水平方向上是否对齐；
- 菜单深度是否在三层以内（建议不要超出三层，大家可以参考微软的例子）；
- 界面控件分布是否按照功能分组（菜单、工具栏、单选框组、复选框组、Frame 等）；
- 界面控件本身是否需要通过滑动条的滑动来显示数据（建议采用分页显示并提供数据排序显示功能）；

实际上，一个处理该类测试的原则性的东西就是：干掉多余的东西，尽可能分组。

## 4. 界面美观度测试

你的界面美观吗？试想一个服装模特穿一身不得体的衣服其展示效果会如何？我至今还记得在学习美学时老师讲过的一句话：美是对比的产物。在软件界面的美观度测试上，我们不得不注意下面的一些建议。

- 前景与背景色搭配是否反差过大；
- 前景与背景色是否采用较为清淡的色调而不是深色（比如用天蓝色而不用深蓝色和墨绿色）；
- 系统界面是否采用了超过三种的基本色（一般情况下不要超过三种）；
- 字体大小是否与界面的大小比例协调（一般中文采用宋体 9-12，英文采用 Arial 或 Times New Roman，日文采用 SimSun 或明朝）；
- 按钮较多的界面是否禁止缩放（一般情况下不宜缩放，最好禁止最大、最小化按钮）；
- 系统是否提供用户界面风格自定义功能，满足用户个人偏好；

## 5. 用户动作性测试

“科学是懒人的哲学”，这是我大学专业老师的一个观点。我们的计算机系统也不例外。我们的系统能让用户尽可能地偷懒吗（少动手肘，少记命令等），从这个角度出发，相信你会对用户动作性测试的本质有较深的体会。我相信没有一个测试员愿意做的多而收获的少。此外用户从某种角度上是心怀不测的挑衅者和肇事者。他们很少有太多的耐心来对待他们寄以很大期望的系统。下面是一些判断用户是否能够“偷懒”和“发泄防止”的测试建议。

- 是否存在用户频繁操作的快捷键；
- 是否允许动作的可逆性（Undo，Redo）；
- 界面是否有对用户的记忆要求；
- 系统的反应速度是否符合用户的期望值；
- 是否存在更便捷、直观的方式来取代当前的界面的显示方式；（比如用菜单界面代替命令语言界面）
- 用户在使用时任何时候是否能开启帮助文档（F1）；
- 系统是否提供模糊查询机制和关键字提示机制减少用户的记忆负担（比如清华紫光输入法的模糊音设定）；
- 是否对可能造成长时间等待的操作提供操作取消功能；
- 是否支持对错误操作进行可逆性处理，返回原有状态；
- 是否采用相关控件（如：日历，计算器等）替代用户手工键盘输入；
- 选项过多的情况下是否采用下拉列表或者关键字检索的方式共用户选择；
- 系统出错是是否存在恢复机制使用户返回出错前状态（如：Office XP 的文件恢复）；
- 在用户输入数据之前，用户输入数据后才能执行的操作是否被禁止（如特定的按钮变灰）；
- 系统是否提供“所见即所得（WYIWG）”或“下一步提示”的功能（比如预览）；

## 6. 行业标准测试

每个行业都有自己的一套标识体系。请尽可能不要与其“撞车”。这就需要我们的人机界面测试人员对软件行业的符号体系有所了解，否则将很难担此大任。

- 界面使用的图符、声音是否符合软件所面向领域的行业符号

体系标准；

- 界面说使用的术语是否符合软件所面向领域的行业命名标准；
- 界面的颜色是否与行业代表色彩较为相近；
- 界面的背景是否能够反映行业相关主题（比如：反映环保的背景一般采用自然风光作为背景）；
- 界面的设计是否反映行业最新的理念和大众趋势；

当然、每一个软件也应当具有自己的一些个性，这些个性是体现软件开发商和所面向的用户领域的特定需要的。比如微软的启动界面和苹果的启动界面就完全是两码事。一个不失个性的软件，其本身就是软件制作商的“广告代言人”。既要突出制作商，又不能喧宾夺主。下面我们给出一些常见的软件个性测试原则。

- 软件的安装界面是否有单位介绍或产品介绍，并拥有自己的图标；
- 软件的安装界面是否在界面上不同于通用的安装工具生成的界面（比如：金山快译的安装界面就比较有特色）；
- 主界面的图标是否为制作商的图标；
- 系统启动需要长时间等待时，是否存在 Splash 界面，它是否包含或反映制作者信息；
- 软件是否有版本查看机制，版本说明上是否有制作者或是用户的标识；
- 软件的界面的色彩、背景、布置是否与同类产品有不同之处，如果有，是否更为简洁、美观；
- 软件界面操作与同类产品相比，是否能够减少用户输入的频繁度；
- 软件界面操作与同类产品相比，是否在出错预防机制和提示上更为直观、醒目；
- 软件界面是否为特殊群体或是特殊的应用提供相应的操作机制（比如 Windows 的放大镜）；

【小结】总而言之，软件人机界面的测试需要一个立足“共性”但又要强调“个性”的测试思路，软件人机界面的测试与其他类型测试不同，更加强调从用户的角度、审美观去看待待测软件。既不能过于“大俗”，又不能过于“大雅”。很多时候，需要在强调规整和强调个性间进行权衡。这迫切需要我们的界面测试人员用大脑去思考，用

心去体会。这对人机界面测试人员在审美观上也是一个极大的挑战。

# 软件质量保证、测试及配置管理面面观

作者：海林[43]

【摘要】如何才能成为 SQA？SQA 的具体工作是什么？TESTER 的具体工作是什么？SQA 与 TESTER 的区别是什么？配置管理做些什么？SQA 的工作与配置管理的工作有什么区别？……常在论坛上看到有同行问及此类问题，为了让大家完完整整地了解到底什么是 SQA？什么是 TESTER？什么是配置管理？以及三者之间到底存在什么区别？以下将带着这些问题进行展开说明。

【关键字】SQA、TESTER、CM

【正文】软件质量保证（Software Quality Assurance，简称 SQA），即参照一定的质量标准、目标及各项软件流程、规范来监督，管理公司产品的质量；在许多质量体系还不是很成熟的公司，维护和发展这些质量标准、流程规范等也是由质量保证人员进行。行内有个这样的说法：“软件质量保证并不能够保证软件的质量”，事实也是如此，软件质量的好坏不是一个人，一个部门能够决定的。但是，我们可以把提高软件的质量作为我们从事软件质量保证工作的目标。

在 ANSI/IEEE 中提到以下了六个品质要素：

- 正确性（correctness）：实现的功能达到设计规范，并满足用户需求的程度
- 可靠性（reliability）：在规定的时间和条件下，仍能维持其性能水准的程度
- 易用性（usability）：用户掌握软件操作所要付出的时间及努力程度
- 效率（efficiency）：软件执行某项功能所需电脑资源（含时间）的有效程度
- 可维护性（maintainability）：当环境改变或软件发生错误时，执行修改或恢复所做努力的程度
- 可移植性（portability）：从一个电脑系统或环境移到另一电脑系统或环境的容易程度

软件测试（Software Test），尽早、尽可能多地发现软件系统中存在的缺陷及问题。但是，好的软件并不是“测”出来的，而是“做”

[43]作者简介：海林（笔名）曾做过测试、品质管理、配置管理工作，目前在一家知名软件公司担任软件项目测试主管。至今积累了丰富的软件实践经验。

出来的，所以，每一个测试人员都应该清楚这样一点：任何人都不可能找出软件中隐含的所有缺陷和问题，正如世界上没有人是十全十美的。

软件配置管理（Software Configuration Management，简称 CM），简单来讲，配置管理即是对软件工作中间件及工作成果的各种管理。其中包括：配置项、配置库、配置标识、基线（详见下文）。

## 1. 软件质量保证

### 1.1. 工作内容

- 建立软件质量保证活动的实体
- 制订软件质量保证计划
- 坚持各阶段的评审和审计，跟踪其结果，并作合适处理
- 监控软件产品的质量
- 采集软件质量保证活动的数据
- 度量软件质量保证活动

### 1.2. 手段

“请介绍几种有效的质量保证工具？……”，论坛中经常有朋友这么问。我想，他可能是把质量保证手段统称为工具了，在我看来，称作“手段”可能更贴切一点。

下面就来介绍一种有效的质量保证手段——同行评审（Peer Review），以此可以提前发现许多软件各阶段隐含的问题。比如：

- 在需求阶段的评审：发现需求是否已被很好的定义，其中是否存在技术上的风险等；
- 在设计阶段的评审：检查架构、数据库等设计的是否合理；
- 在编码阶段的评审：检查 Source code，发现代码中不符合编码规范部分；
- 在测试阶段的评审：检查测试计划安排的是否合理，测试用例设计的覆盖率等；
- 在验收阶段的评审：检查项目文档是否都已存在于配置管理库中；总结项目开发过程中的一些经验教训并作记录，存入公司的资源库等。

### 1.3. Peer Review 过程中需注意的细节

- 做评审计划；
- 人员不要太多，保证会议的效率；
- 经常性地举行；
- 尽量提前点时间将会议上要审核的资料发送给参加会议的人，并请参会人员对发现有疑问的地方及时做出标识，以提高会上的效率；
- 千万注意不要将评审会议变成聊天或过分争论会议；
- 评审团队最好是由经过同行评审培训，并有产品经验的相关人员组成；
- 对评审人员提出的问题，在会后，应该由文档作者修改，并最终通过评审团队成员的一致确认。

### 1.4. 常见误解

- 误解一：认为发布出去的软件发现了问题，就是 QA 和 TESTER 的责任；还是那句话，“软件质量保证”并不能保证软件的质量；
- 误解二:QA 就是公司的文档管理员,常有 QA 称自己是“打杂工”。实际上，成为一名出色的 QA 并不是一件容易的事情，工作年限、工作经验、工作成果等都是对其衡量的标准；
- 误解三：QA 的工作有 TESTER 兼职进行即可。这是目前多数软件公司采取的措施。个人认为，最好将二者分开，TESTER 的工作也是有质量可言，同样需要 QA 进行监督、检查，而 QA 的直接上司即公司的高层领导；

## 2. 软件测试

### 2.1. 工作内容

- 编写测试计划
- 编写测试用例
- 对软件开发过程的重要文档进行测试，如：需求文档、设计文档等
- 测试执行
- 测试结果总结、分析
- 测试数据的度量（更高要求）

以上工作基本上涵盖了测试的所有工作，在分工比较明细的公司，通常由不同人员来进行，如：

- 测试设计员：制定和维护测试计划；设计测试用例及测试过程；评估测试，生成测试分析报告
- 测试员：执行集成测试和系统测试；
- 设计员：设计测试需要的驱动程序和稳定桩；
- 编码员：编写测试驱动成员和稳定桩；执行单元测试

### 2.2.手段

- 手工测试

目前多数测试人员都采取的一种测试工具，虽存在一定的弊端，但也绝对不可以省略。

- 自动化测试工具

想了解并学习自动化测试工具的同仁不妨去www.51testing.com看看，里面有介绍自动化工具的专栏，很不错的。

### 2.3.BUG 管理工具

通常叫 BMS（BUG MANAGEMENT SYSTEM)。如果公司不愿意投入成本购买大公司的 BUG 管理工具，不妨根据自己的情况开发一个，便宜又好用，并且一劳永逸！

## 3. 软件配置管理

### 3.1.术语解释：

- 配置项：凡是纳入配置管理范畴的工作成果都叫做配置项。
- 配置库：所有配置项的集合
- 配置标识：为了方便管理，而对各配置项作的标识
- 基线：由一组配置项组成，这些配置项构成了一个相对稳定的逻辑实体。基线中的配置项被“冻结”了就不能再被任何人随意修改，直接进入变更阶段。通常将交付给用户的基线称为一个“Release”，为内部开发用的基线则称为一个“Build”。

### 3.2.工作内容

- 创建配置库，并至少创建配置库的所有一级目录
- 为每个项目组成员分配权限
- 根据配置管理计划中的基线计划维护基线，“冻结”配置项，控

制变更

定期清楚配置库里的“垃圾”文件（“垃圾”文件即在项目中产生，但最终不用存档的一些文档）

- 配置状态发布
- 定期备份配置库

### 3.3.手段

常用配置管理工具如下：VSS、CVS、CLEARCASE 等，具体内容我已在论坛中发布过，大家可去查看。

### 3.4.SQA 与 TESTER 的区别

通过上面各自含义及工作内容的描述，我想 SQA 与 TESTER 的区别应该比较清晰了吧，现在总结为以下两点：

- SQA 重点是对软件开发过程进行监督、管理、控制；
- TESTER 重点是对软件开发的成果进行检查、控制。

### 3.5.SQA 与 CM 的区别

SQA 重在监督项目的进度，督促项目按开发流程及规范进行，控制并记录好各项变更，并通知配置管理员对配置项做好相应的变更；同时 SQA 还可以作为 SEPG（Software engineering process group）的成员之一，参与公司的软件过程改进工作。

配置管理工作的主要目的是为了保证项目资料得到清晰、有效地管理，加快工作效率的同时，也避免一些意想不到的风险，如：代码丢失或恶意修改等。同时，有效地配置管理工作，还可作为公司建立复用资源库的主要来源之一。建立复用资源库对公司来讲势在必行，频繁的人员流动是再充分不过的理由。

【小结】

本篇文章可能更适合于新人阅读，希望能有所帮助；对于描述不当的地方，还请谅解。

软件质量保证、测试和配置管理都刚被各软件公司重视起来，但它们的春天还未真正到来，否则就不会经常听到同行们的牢骚和抱怨。不管怎么说，既然你选择了这一行，就一定要坚定自己的选择，并一如既往地走下去。一定要养成主动学习的习惯，不要等着别人去

教你怎么做，或直接从别人那里取来就用，而应该尽量自己创造。当前大部分软件公司在这些方面还处于起步阶段，技术及资料还不是很成熟，这些对新人来讲不能不说是学习的好机会，同时也是展示自我的好机会。坚信，我们的明天会更好！

# 剖析层出不穷的 BUG

作者：楚才

【摘要】本文重点针对测试中的周期性特点，阐述 BUG 层出不穷的几个原因，越来越多的企业领导关注并重视测试，他们往往专注于软件测试中的 BUG 现状，对层出不穷的 BUG 产生居多的疑问，本文可以给出部分的参考。

【关键字】周期性，黑盒测试，自顶向下

## 1. 引言

由于软件企业对软件质量的重视程度越来越高，软件测试在软件研发中的地位越来越重要。越来越多的企业领导也将注意力更多的投入到软件测试方面来，确实测试很需要得到领导的重视与理解并且毫无疑问的支持，如果你所处的团队目前已经很好的得到了领导的重视和支持，那真是一件幸事。可不幸的是，目前很多国内软件相关人士对软件测职业岗位还出于不理解状态，这其中当然包括领导一层，可能大家在日常工作中有时候会碰到领导过问到测试状况或者 BUG 的事，特别是那些规模小，流程体系还不够完善，处于人治状态的公司也许几率更高一些，通常来说，领导见到那些层出不穷的 BUG，直觉反应是测试工作做的不够理想而导致 BUG 的遗漏，当然这样的假设不一定成立，到底为什么会有层出不穷的 BUG 呢？

## 2. 层出不穷的 BUG

大家可能都有这样一种感觉，软件几乎天天在修改，审核，验证再测试，可相当长一段时间的测试过程中会发现居多这样那样的缺陷，层出不穷的发现 BUG，到底是什么原因？

笔者总结以下几种常见原因

### 2.1. 测试遗漏

测试的设计主要体现在测试用例的设计，以及通过测试策略将这些测试用例同测试计划，测试执行，还有测试结果数据的收集整理结合在一起执行，由于测试人员水平的高低，测试工具使用的熟练程度，以及对所测试对象的理解深度等原因，测试设计很难完善，主要表现

在测试用例设计的不全面，存在遗漏，或者测试方案的不周密，以及可能的测试人员执行时产生的偏差等等，这些测试方面的遗漏和偏差都可能导致软件问题没有及时发现，造成测试的遗漏。

### 2.2. 设计及修改原因

软件需求或者设计方案经常被更改，特别是变更没有导入一套成熟的变更管理体系的情况下，每次变更无疑于埋下大量的地雷，这些都为 BUG 提供滋生的环境。另外后期修改维护中对 BUG 未做准确的分析定位，修改方案未审核，或者修改过程中程序员出现“头痛治头，脚痛治脚”,“补了东边漏了西边”等不良修改过程中引发出新的问题，也是导致 BUG 被扩大的原因。

### 2.3. BUG 的概率性及偶然性

有些 BUG 的出现呈现概率的特性，它需要反常数量，频率，或者资源的方式下执行系统才能被查找，即通常所说的压力测试。

### 2.4. BUG 的潜伏性及阶段性

有时候，BUG 实际存在但由于触发它的条件不满足从而呈现潜伏状态只能在某个阶段才能被发现，单元测试，集成测试，系统测试等阶段测试重点关注的对象就不同，如集成测试可以发现单元测试通过后的模块之间接口上的错误。特别象嵌入式系统中多进程以及多任务处理问题、系统容错性问题、内存问题等等，这些情况下表现出的潜伏性更加复杂多变，导致发现这些 BUG 需要一个特别长的周期或者需要某一特定测试环境能被有效搭建的情况下才能查找出。

### 2.5. BUG 的隐蔽性和周期性

该 BUG 实际存在但由于其他 BUG 的存在导致它所在的代码没有得到执行，因而无法暴露该 BUG，这种情况在以黑盒测试为主的测试中表现尤为突出，只有通过周期性的 BUG 修复及测试才能发现该类 BUG。

## 3. 测试的周期性

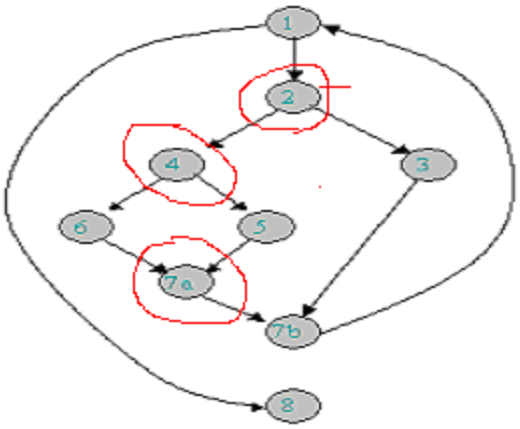
上文提到 BUG 的隐蔽性和周期性决定了测试必须是一个周期性的工作，这个周期性不是表现为简单的重复。下面针对黑盒测试的特点来详细阐述这一特性。

黑盒测试的对象大多针对图形用户界面（GUI），它以窗口，菜单

以及按键的表现形式，针对它们的测试，通常通过模拟用户的操作来完成，这种特性决定它只能通过自顶往下的测试方法，即只能通过菜单一级一级往下的方式测试，从程序的角度来说必须先有父窗口再到子窗口的过程。当然如果采用白盒测试的方法，可以通过驱动的方式来激发，这里就不阐述！

以手机这类嵌入式系统测试举例，手机软件的测试对象通常是窗口结构、信息流、控制流，以及逻辑控制等都具有很强的层次性，一旦某一窗口出现 BUG，通过黑盒测试来引导程序运行的测试很可能就因该 BUG 的存在而就此中止了运行，从而无法执行剩余的测试用例来遍历其他的路径，这样就无法查找到那些未遍历的代码中可能存在的缺陷。

假设下图所示意的测试模块中红色框圈中的为 BUG 所处的节点，该模块假设共在 3 个节点处存在错误。（为了方便大家的理解不防把各“节点“看成一窗口菜单，从而下图就看成一菜单树结构）。




（图一）

**测试周期 1：**

上图中，如果节点“2“处出现了 BUG，程序在此中断，黑盒测试无法遍历从节点”2“到节点”4“，以及节点”2“到节点”3“的边，测试就无法发现在节点”4“及节点”7a“处的的 BUG，这就是因为节点”2“处的 BUG 存在将其它 BUG 隐蔽起来了，剩余遍历其它路径的测试用例无法再执行，测试在此中断，等待修复节点“2“处的 BUG。

**测试周期 2：**

新版本解决完节点“2“处得 BUG，测试可以执行从节点”2“到节点”4“的边，从而又暴露出节点”4“处的 BUG，同样的道理，测试继续被中断，无法遍历从节点”4“—”6“—”7a“，以及节点”

4“—“5—“7a””这两条路径，无法发现节点”7a“处的的 BUG，这就是因为节点”4“处的 BUG 存在将其它剩余的 BUG 隐蔽起来，等待修复节点“4“处的 BUG。

**测试周期 3：**

解决完节点“4”的 BUG 后，前一版本无法遍历的路径在此可以被执行，从而发现节点“7a”处的 BUG，从上所叙的黑盒测试过程，需要通过整整 3 个周期的测试，才能执行完所有测试用例，达到遍历上图所有路径。

上面只是一个简单的例子，实际的测试情况比这要复杂得多，目的在于阐述 BUG 的隐蔽性决定了发现它们需要一个周期，从而说明查找 BUG 不能象大家想象得那样，通过在某一个版本上投入大量的人力物力做尽量多的测试就可以暴露出所有的缺陷，同样的道理，如果希望在某一个版本的测试报告中去评估一个系统的稳定性或者质量状况，也是不科学的，上图是一个很好的例子，一个节点处的错误足可以屏蔽数倍于它的 BUG，有的甚至是致命的错误。

## 4. 结论

笔者整理此文曾经是因为领导的一个疑问“为什么会有层出不穷的 BUG？”，很庆幸遇到一位不耻下问又虚心学习的领导，通过对此文的解读，让他更深的理解了测试的周期性特征，测试很需要得到领导的重视与理解并且毫无疑问的支持，如果大家对测试理解更深入一点，就会对测试这行业更热爱些，领导的注意力和投入也会更多一些，希望此文能给大家一些可以。

# QTP 中的 descriptive programming

姓名：周坚

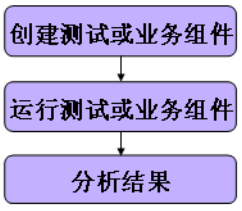
【摘要】自动化功能测试是一种企业级的用于检验应用程序是否如期运行的功能性测试工具。通过自动捕获，检测，和重复用户交互的操作，能够辨认缺陷并且确保那些跨越多个应用程序和数据库的业务流程在初次发布就能避免出现故障，并且保持长期可靠运行。在市场上用的比较多的主要包括 Mercury 公司的 WinRunner，QuickTest Professional 和 IBM 的 Rational Robot。笔者对于 QuickTest Professional 相对较为熟悉，希望有机会向大家逐步介绍 QuickTest Professional 中的一些要点及技巧。在本文里主要介绍了 QuickTest Professional 中的一项核心内容 Descriptive Programming，希望对大家有所借鉴和帮助。在文中，为了方便起见，将 QuickTest Professional 简称 QTP。(本文是基于 Quick Test Professional V8.0 而写)。

【关键词】

| | |
|---|---|
| 描述性编程 | Descriptive Programming |
| 功能测试 | Functional Test |
| 专家视图 | Expert Vie w |
| 关键字视图 | Keyword View |
| 对象模型 | Object Model |
| 运行时对象 | Run-Time Object |
| 测试对象 | Test Object |


## 1. QTP 功能测试基本方法

我们简单介绍一下有关功能测试的基本方法，这实际上对于所有自动化功能测试产品来说都是一样的。一般情况下，用 QTP 来进行功能测试的基本方法主要包括三个主要阶段：



### 1.1.创建测试或组建

首先可以通过在应用程序或网站上录制会话，或者建立对象库并使用关键字驱动功能向关键字视图中手动添加步骤来创建测试或组件。在 QTP 里面我们可以通过两种方式添加步骤来创建测试或组件：

- 在应用程序或网站上录制会话。
- 建立对象库并使用这些对象在关键字视图或专家视图中手动添加步骤

然后在在测试或组件中插入检查点，检查页面、对象或文本字符串中的特定值或特征，通过它可以标识网站或应用程序是否正常运行。或是通过用参数替换固定值扩展测试或组件的范围。提供数据表中的数据，定义环境变量和值，定义测试、组件或操作参数和值，或者使用 QTP 生成随机数字或当前用户和测试数据等。

最后如果需要的话使用 QTP 中众多的功能测试功能来增强测试或组件或添加编写语句来实现更复杂的测试目标。

### 1.2.运行测试和组建

控制运行会话，帮助标识和消除测试或组件中的缺陷。使用“单步执行”、“单步跳过”和“单步退出”命令逐步运行测试或组件，或设置断点使测试或组件在预定点暂停。每当测试或组件在断点处停止时，可以在“调试查看器”中查看其变量的值。

### 1.3.分析结果

在运行测试或组件之后，通过两种方式可以查看其结果：在“结果”窗口中查看结果；自动报告在运行会话过程中检测到的缺陷，可能的话并上报到其他缺陷管理产品中。

## 2. 试图与对象模型

在介绍 QTP 中的 Descriptive Programming 前，我们有必要先介绍一下 ExpertView 及在 ExpertView 里进行编码的一些基本知识。

在 QTP 里面提供了两种视图，第一种我们把它称为 KeywordView（关键字视图，在早期的版本中称为 TreeView），第二种把它成为 ExpertView（专家视图），这两种视图分别是针对两种类型的人进行使用的。

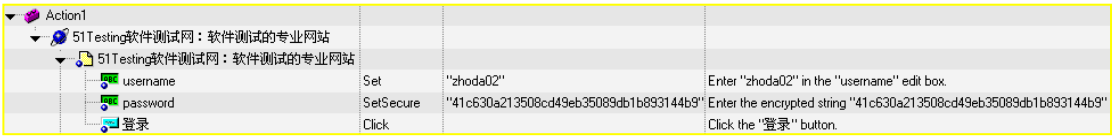
## 2.1.KeywordView（关键字视图）

通过关键字视图，QTP 提供了一种模块化的表格格式创建和查看测试或组件的步骤。每个步骤在关键字视图中都是一行，这样用户可以轻松的修改任何一部分组成。

在录制会话过程中，用户在应用程序上执行的每个步骤在关键字视图中记录为一行。例如，在 51testing 的页面上执行的下列三个步骤：

- 在“用户名”编辑框中输入 zhoda02。
- 在“密码”编辑框中输入加密字符串 41c630a213508cd49eb35089db1b893144b9。
- 单击“登录”按钮。

那么，关键字视图将包含下列行：




很显然，关键字视图非常直观有效，使用的人可以很清晰的看到被录制对象的录制层次及运行步骤，比较适合那些对于业务操作流程熟悉的人员使用。但是，如果需要一些增强型的操作，那就需要切换到专家视图里进行了。

## 2.2.ExpertView（专家视图）

QTP 在关键字视图中的每个节点在专家视图中对应一行脚本。上面例子对应的脚本如下：（删除后一句是因为前后重复，一句可以说明问题）

```
Browser("51Testing 软件测试网：软件测试的专业网站").Page("51Testing 软件测试网：软件测试的专业网站").WebEdit("username").Set "zhoda02"
Browser("51Testing 软件测试网：软件测试的专业网站").Page("51Testing 软件测试网：软件测试的专业网站").WebEdit("password").SetSecure "41c630a213508cd49eb35089db1b893144b9"
Browser("51Testing 软件测试网：软件测试的专业网站
```

").Page("51Testing 软件测试网：软件测试的专业网站").WebButton("登录").Click

对于 QTP 来说，其核心编码语言是 Visual Basic Script，因此，如果用户熟悉 VBScript，可以运用自如的添加和更新语句，并通过编程方式增强测试和脚本，而这一切必须在专家视图中完成。

更为重要的是，有些操作是必须在专家视图中才可以完成的，例如：要处理动态对象、客户化报告、获取对象运行时的属性值（Run-time Value）等等，这些都必须通过专家视图中的 VBScript 编码完成。

然而，QTP 里所有的操作都是基于对象进行的，所以我们必须对对象模型有一个基本了解，才可以在专家视图内进行 Descriptive Programming。

## 2.3.测试对象模型

测试对象模型是一大组对象类型或类，QTP 用这些对象类型或类来表示应用程序中的对象。每个测试对象类都有一个可以唯一标识属于该类的对象的属性列表，以及一组 QTP 可以对其进行录制的方法。它包括测试对象（Test Object）和运行时对象（RunTime Object）。

测试对象是 QTP 在测试或组件中创建的用于表示应用程序中的实际对象的对象。QTP 存储有关该对象的信息，这些信息有助于它在运行会话期间标识和检查该对象。

运行时对象是网站或应用程序中的实际对象，在运行会话期间执行针对该对象的方法。

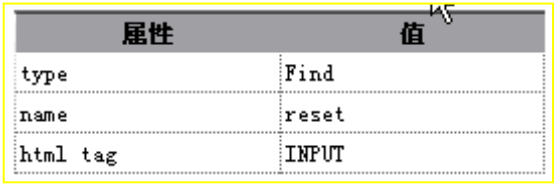
如果录制时执行应用程序的相应操作，则一般情况下 QTP 将完成以下操作：

- 标识 QTP 测试对象类（表示执行了操作的对象），并创建相应的测试对象
- 读取应用程序中对象属性的当前值，然后将属性和属性值列表与测试对象一起存储。
- 选择该对象的唯一名称，一般使用该对象某个重要属性的值。
- 使用适当的 QPT 测试对象方法录制对对象执行的操作。

例如，假定使用以下 HTML 源代码单击“查找”按钮：

<INPUT TYPE="submit" NAME="Find" VALUE="Find">

QTP1 将单击的对象标识为 WebButton 测试对象。它将创建一个名为 Find 的 WebButton 对象，然后为该 Find WebButton 对象录制下列属性和属性值，同时还会录制对 WebButtion 的 Click 方法。



| 属性 | 值 |
|---|---|
| type | Find |
| name | reset |
| html tag | INPUT |

在关键字视图及专家视图中显示内容分别为：

| 项 | 操作 | 文档 |
|---|---|---|
| Action1 | | |
| Mercury - Business Tech... | | |
| Mercury - Business ... | | |
| search_right | Click | 单击 "search_right" image。 |

Browser("Mercury Interactive").Page("Mercury Interactive").WebButton("Find").Click

运行测试或组件时，QTP 通过其测试对象类及其描述（一组用于唯一标识该对象的测试对象属性和属性值）来标识应用程序中的每个对象。测试对象及其属性和属性值的列表存储在对象库中。例如在上例中，QTP 将在运行会话期间在对象库中搜索 WebButton 对象，通过名称 Find 来查找其描述。QTP 根据找到的描述，在应用程序中查找 WebButton 对象，该对象带有 HTML 标记 INPUT、类型为 submit、值为 Find。找到对象后，它将对其执行 Click 方法。

在这样一组对象模型的基础上，QTP 为各类应用对象都提供了一组方法和属性，例如 Web Objects，Windows Objects，SAPGUI Objects，ActiveX，Java 等。下面是一些 Web Objects 的方法和示例：

| 对象 | 方法 |
|---|---|
| Browser | Check |
| Frame | Click |
| Image | Exist |
| Link | GetCellData |
| Page | GetProperty |
| WebArea | GetROProperty |
| WebButtion | Mouseover |
| WebCheckBox | RowCount |

| | |
|---|---|
| WebEdit | Select |
| WebList | Set |
| WebRadioGroup | SetProperty |
| WebTable | Submit |

例1：获取单元格中的值

```
thisText = Browser(…).Page(…).Frame.(…).WebTable("sample").GetCellData(2,1)
```

例2：获取图片的名称

```
ObjectName = Browser(…).Page(…).Image("Find").GetProperty("Name")
```

例3：检查某个对象是否存在，如果存在弹出对话框说明对象存在。

```
If
    Browser("Browser").Page("Page").Applet("login.html")
    .JavaEdit("username").Exist
Then
   MsgBox("The object exists.")
End if
```

## 3. 描述性编程（descriptive programming）

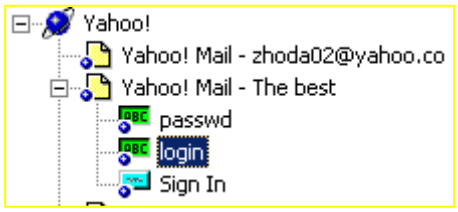
### 3.1.descriptive programming 概述

通常情况下，当在录制一个操作时，QTP 会将被操作对象加入到对象库里（Object Repository）。一旦对象存在于对象库里，我们就可以在专家视图里通过添加相关的对象方法来对该对象进行操作。我们可以通过引用层次型对象库里的对象描述（Object Description）来添加相应的方法。

因为 QTP 对象库中的每个对象都具有唯一名称，所以在引用时对象名是必须需要指定的。然后在测试运行期间，QTP 在对象库中根据这个对象的名称和父对象来查找对象，并使用为这个测试对象存储的测试对象描述，在网站或应用程序中标识该对象。

例如我们用 QTP 录制 Yahoo Mail 登录情况时我们需要输入用

户名，于是在录制时我们就会录下一个 WebEdit 对象，它的缺省逻辑名为“login”，该编辑字段位于名为 “Yahoo! Mail - The best” 的页面上，并且该页面在浏览器中使用名称 Yahoo!进行录制。如图所示，即为录制时的对象库的内容：




那么如果我们想要应用该对象，就可以在专家视图输入以下信息：

Browser("Yahoo!").Page("Yahoo! Mail - The best").WebEdit("login").Set “xxx”

或者我们也可以调用一些方法，获取改对象在运行时的对象名，如：

Browser("Yahoo!").Page("Yahoo! Mail - The best").WebEdit("login").GetROProperty(“name”)

然而，我们可以发觉到，上面的例子在处理对象时，对象已经存在于对象库里，因此我们可以应用这个对象的逻辑名。实际使用中，情况往往并非如此简单，我们经常会遇到很多在页面上动态产生的对象，换而言之，对象库里没有这些对象，我们也无从引用。因此我们必须采用其他的技术来完成这类操作，这也就是我们需要讲解的 Descriptive Programming。

为了满足上面提到的动态对象的处理问题，QTP 允许用户通过将对象属性编码到测试脚本里来动态识别对象，这就是我们通常意义下称为的 Descriptive Programming。通过这种方式，我们可以指示 QTP 不通过引用对象库和对象名来对实际对象进行操作。具体操作中，我们只需要为 QTP 提供对象的一组属性和值，这样 QTP 就可以来识别相应的对象并对其进行操作。这相当于，告诉 QTP 要识别对象的一些关键特征，根据这些特征 QTP 就可以一一匹配然后识别出来这个对象。

而且，更为重要的是，通过这种 Descriptive Programming 的方式，还可以让 QTP 识别具有某些相同属性的对象。我们先来举个例子来看一下：我们假设当前的 Windows 系统中打开了若干的 Yahoo 主页

面（多于一个），现在我们要关闭所有的正在浏览 Yahoo 主页面的浏览器。

对于上面那个例子来说，我们先看一个简单一点的情况，假设只有且仅有一个 Yahoo 主页面：那么我们可以用下面的方法来

```
Window("Text:=Yahoo! - Microsoft Internet Explorer").Close
```

我们可以看到语句里我们要查找的对象是 Window 窗口标题为“Yahoo! - Microsoft Internet Explorer”，然后把它关闭，具体的语法说明我们稍后为解释。但是上面的语句仅仅适合前面提到的条件“只有且仅有一个 Yahoo 主页面”，如果有多个同样的窗口就会出错，原因是通过语句可以匹配到多个对象，而 QTP 不知道应该对哪个对象进行关闭动作。我们需要进一步的缩小匹配范围：

```
Dim i
i = 0
while (Window("Text:=Yahoo! - Microsoft Internet Explorer", "index:="&i).exist)
   Window("Text:=Yahoo! - Microsoft Internet Explorer", "index:="&i).close
   i = i +1
wend
```

这里我们可以看到，对于具有相同属性的对象，我们可以通过 index 参数来对其进行区别，第一个对象为 index=0，第二个为 index=1 等等，依次类推。当然我们还可以通过 CreationTime 和 Location 参数来定位对象，这里就不详细叙述了。

通过上面的例子，我们对 Descriptive Programming 有一个基本了解了，下面我们详细讲解一下 Descriptive Programming：在具体实现中，我们有两种类型的 Descriptive Programming 方法。可以列出直接在测试语句中描述对象的属性和值的集合；或者向 Description 对象中添加属性和值的集合，然后在语句中输入 Description 对象的名称。下面我们分别举例介绍。

### 3.2.直接在语句中输入编程描述

通过多个指定描述对象的 *property:=value* 对，可以直接在语句中描述对象，这是最直接有效的方法。

常规语法为：

TestObject("PropertyName1:=PropertyValue1", "..." , "PropertyNameX:=PropertyValueX"}

TestObject - 测试对象的类。

PropertyName:=PropertyValue - 测试对象的属性及其值。各个 property:=value 对之间应用逗号和引号分开。

例如：以下语句指定 Mercury Tours 页面中名为 author 且索引值为 3 的 WebEdit 测试对象。当测试运行时，QTP 将查找具有匹配属性值的 WebEdit 对象，并输入文本 jojo。

Browser("Mercury Tours").Page("Mercury Tours").WebEdit("Name:=Author", "Index:=3").Set "Mark Twain"

我们也可以从从描述中的特定位置（从 Page 对象描述开始）开始使用 Descriptive Programming。

Browser("Mercury Tours").Page("Title:=Mercury Tours").WebEdit("Name:=Author", "Index:=3").Set "jojo"

此外，如果我们希望在在一个测试或组件中多次使用相同的 Descriptive Programming，则可以将创建的对象赋值给变量，这样使用会方便很多。

例如：我们需要完成下面一系列操作

Window("Text:=HyperSna").WinButton("Caption:=日 期").Click

Window("Text:=HyperSna").WinButton("Caption:=时 间").Click

Window("Text:=HyperSna").WinButton("Caption:=确 定").Click

那么，为了方便其见，我们可以将 Window("Text:=HyperSna") 赋值到一个变量，然后再使用，参见下面的代码：

```
Set WinHyper = Window("Text:=HyperSna")
WinHyper.WinButton("Caption:=日期").Click
WinHyper.WinButton("Caption:=时间").Click
WinHyper.WinButton("Caption:=确定").Click
```

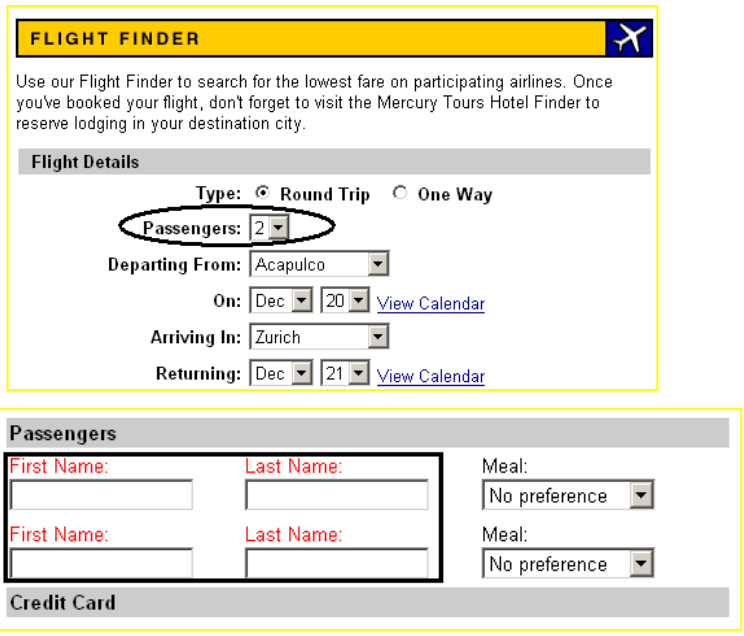
如果使用了 VBScript 里面的 With 语句，还可以简化为以下代码：

```
With Window("Text:=HyperSna")
.WinButton("Caption:=日期").Click
```

```
.WinButton("Caption:=时间").Click
.WinButton("Caption:=确定").Click
End With
```

下面我们来看一个更为详细的例子，在 QTP 产品缺省安装里面自带了一个网上订机票的示例称为 Mercury Tour，我们看一下在订票过程中何时需要用 Descriptive Programming。

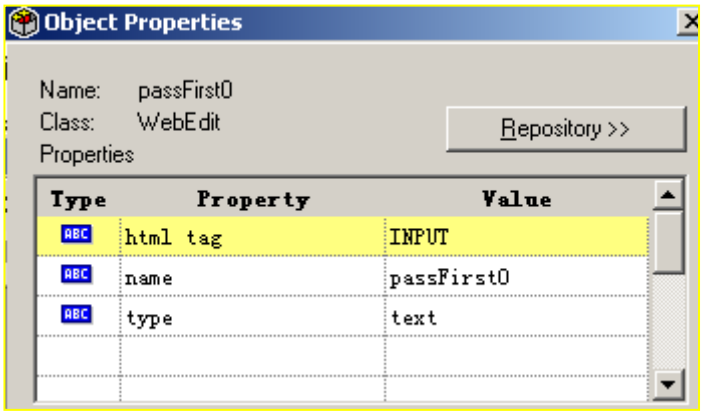
首先登入系统后，如果需要订票，就要先搜索航班，此时系统要求输入订票乘客的数量，假设我们在第一次录制脚本时选择了 1 个 Passenger，并成功完成订票。然后，我们需要参数化乘客数量来测试订票系统，我们会发现回放会失败。原因在于，不同的乘客的数量导致在订票时需要输入每个乘客的姓名，而录制时，只输入了一个乘客的姓名。而乘客姓名的输入框是随着乘客数量的变化而动态生成的，我们不可能从对象库里得到没有录制的对象，因此必须使用 Descriptive Programming。




在录制单个乘客时，我们得到的录制语句是：

```
Browser("Welcome: Mercury Tours").Page("Book a Flight: Mercury").WebEdit("passFirst0").Set "Michael"
Browser("Welcome: Mercury Tours").Page("Book a Flight: Mercury").WebEdit("passLast0").Set "Wang"
```

显然 WebEdit("passFirst0")和 WebEdit("passLast0")是录制时产生的对象并存放到对象库里的。通过对象库，我们可以看到对象的属性如下：




系统对于发生多个 FirstName 时，命名规则是 passFirst0，passFirst1…依次类推。因此只要通过简单的 Descriptive Programming 就可以完成动态 FirstName 与 LastName 的识别工作。这里我们假设参数化的乘客数已经赋值给 intPassNum，下面是脚本中的关键语句：

```
counter = 0
For i = 0 to (intPassNum)
Browser("Find a Flight:").Page("Book a Flight:").WebEdit("name:=passFirst"&i).Set "Michael"
Browser("Find a Flight:").Page("Book a Flight:").WebEdit("name:=passLast"&i).Set "Wang"
counter = counter + 1
Next
```

### 3.3.使用 description 对象

使用 Description 对象可以返回包含一组 Property 对象的 Properties 集合对象。Property 对象由属性名和值组成。然后，可以在语句中指定用返回的 Properties 集合代替对象名。（每个 property 对象都包含一个属性名和值）。

要创建 Properties 集合，可以使用以下语法输入 Description.Create 语句：

```
Set MyDescription = Description.Create()
```

创建 Properties 对象（例如，以上示例中的 MyDescription）后，就可以输入语句，以便在运行会话期间在 Properties 对象中添加、编辑、删除或检索属性和值。这样，就可以在运行会话期间，使用动态方法确定哪个属性以及多少个属性应包含在对象描述中。

在 Properties 集合中填充一组 Property 对象（属性和值）

后，可以在测试语句中指定用 Properties 对象代替对象名。

例如，假设我们需要完成以下一个操作：

```
Window("Error").WinButton("text:=OK", "index:=1").Click
```

我们可以通过 Description 对象来实现同样的功能，参加下面的代码：

```
Set MyDescription = Description.Create()
MyDescription("text").Value = "OK"
MyDescription("index").Value = 1
Window("Error").WinButton(MyDescription).Click
Set MyDescription = Nothing
```

【小结】以上是对QTP中有关处理动态对象中的Descriptive Programming的简单介绍，希望对大家能够有所帮助，就总体而言，如果能够熟练掌握Descriptive Programming，那么有很多实际中的问题就可以迎刃而解。当然Descriptive Programming只是QTP中的一个功能，QTP在实际功能测试中还有很多强大的功能，作为QTP学习的一个系列有机会我会一一介绍。

# TestDirector 项目数据迁移

## ——让您的 TD 项目活起来

作者：单吉勇 林万枝[44]

【摘要】TestDirector，（以下简称 TD）它是 Mercury Interactive 公司推出的基于 WEB 浏览器环境下的测试管理工具。通过 TD 的流程控制可以规范软件企业的测试流程、改善测试质量、减轻测试人员的负担、提高工作效率。在接触 TD 过程中仍然存在着很多未知领域等待着我们这些从事软件测试工作的同行去研究去拓展，如何更有效的使用 TD 提高我们的测试管理，将是我们继续研究关注的方向。本文总结了我们在移植 TD 项目方面的一些经验和技巧，希望对大家有所帮助。

【关键词】项目移植 集成工作环境 分布式工作环境

这里我们先将以上的几个名词解释一下：

项目移植：这里说的项目移植是指将已经建立的 TD 项目整体文件在保证数据安全和完整的前提下移植到其他服务器的过程，这个过程包括以下几个方面（1）数据库的移植（2）项目文件的移植（3）项目配置文件的移植。经过移植后的 TD 项目可以实现双机备份的功能。

集成工作环境：我们把 TD 服务程序和 TD 使用的数据库存放在同一台计算机上的这种工作环境称为集成工作环境。这种工作环境节省成本，维护较复杂，不利于数据的安全性。

分布式工作环境：我们把 TD 服务器程序和 TD 后台使用的数据库存放在不同机器上，也就是使用单独的一台计算机作为 TD 项目的数据库服务器，TD 服务程序通过网络访问数据库服务器，这种工作环境称为分步式工作环境。这种环境的成本较高，但是利用维护，数据的安全性较高。对一些专业性的企业尤其是需要将 TD 开放到 Internet 上，我们建议使用这种工作环境。

### 1. 移植说明

基于 IIS WEB 服务下的 TD 服务程序支持的数据库有 ACCESS,SQL SERVER、SYBASE、ORACLE。由于 ACCESS 数据库的迁移比较容易本文

[44]单吉勇：shanjiyong@shinetechnology.com；林万枝：linwanzhi@shinetechnology.com；我们可以共同探讨测试流程管理的相关知识。

就不介绍，本文主要讨论 SQL SERVER 数据库的移植。掌握 SQL SERVER 数据库类型的 TD 项目移植，对 ORACLE、SYBASE 类型数据库的项目移植工作也能做到触类旁通。下面大家就跟我们一切进入 TD 项目移植的具体工作吧！

下面的移植工作，我们选用了一个名为：E-CIS 的项目进行实例移植，同时我们将其他计算机上的 TD 项目（SAAA、SIMS、E-ICID）与 E-CIS 进行了合并。

## 2. 集成工作环境的数据迁移

### 2.1. 拟分析产生的原因和解决方法

产生的原因：

由于我们配置的 TD 服务器与数据库服务器是同一台计算机，一旦计算机发生故障，TD 项目中的数据既便是备份出来也很难恢复，其主要原因是：

- 使用 TD 建立项目时，TD 会在数据库中自动建立一个名为 TD 的用户，我们运行的 E-CIS 项目中所有的表都是由 TD 这个用户建立的，无法删除和添加这个用户。
- 在 SQL 查询分析器中可以看到所有相关的表都是 td.*的格式，这样会造成如果 SQL 查询分析器是以其他用户登入，执行 select * from all list 出现错误。出现一定要加前缀后才能查询出来数据问题，这个就是造成恢复数据库后即使其他项目都配置正确，但 TD 依然不能正常使用的一个重要原因。
- 在安装 TD 后会在安装盘符：\Program Files\Common Files\Mercury Interactive\Domsinfo 目录下，有一个用来存放系统信息的 ACCESS 数据库 Doms.mdb，这个数据库由 TD 创建，并且是经过加密的，它存放着 TD 所有的配置信息。这个数据库是是否能够成功恢复 TD 项目的一个重要因素。

### 2.2. 移植前的备份工作

在正式移植之前，我们首先要做好数据的备份工作，这也是我们移植工作的一部分，虽然我们现在的移植技术已经相对较成熟，但是我们还是要建议大家移植之前的备份是很有必要的。备份主要备份以下的数据：

#### 2.2.1. 备份 DomsInfo 目录；

默认在 C:\Program Files\Common Files\Mercury Interactive 下。这个目录包含了 Doms.mdb 文件（用户信息和工程列表）、connection strings（连接字符串）、parameters（参数）、global style sheets（全局风格表）、the database template（Empty_DB.mdb）（一些临时数据）

### 2.2.2.备份 TD 项目安装文件；

TD_Dir 包含每个工程的自动测试、附件、设置、风格列表。默认在 C:\TD_Dir\Default 目录下,打开这个目录可以看到TD的工程文件,我们 需要将这些工程文件备份下来。

### 2.2.3.备份项目数据库；

备份项目用的数据库文件。

## 2.3.移植工作

移植工作分为四个部分：

- 数据库的迁移；
- 建立访问数据表文件的 TD 用户；
- 修改 TD 的项目配置数据库(Access)；
- 修改 Project 中的 INI 文件。

### 2.3.1.数据库的迁移：

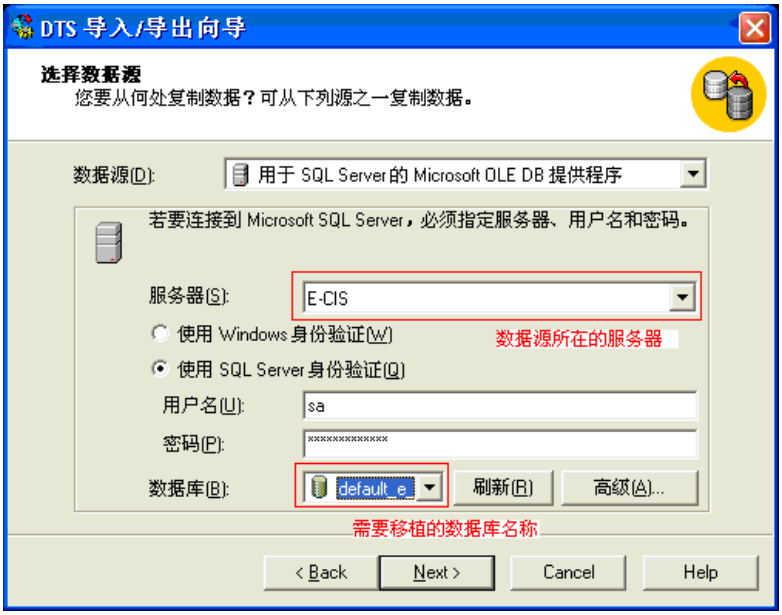
数据库的移植目的是：将项目数据库从原来的计算机移植到新的计算机。

具体思路：通过 SQL Server 集成的导入和导出数据功能将原数据库文件导入新的 SQL Server 服务器中。在这个过程，原表中由 TD 用户创建的表也会更改为 DBO 用户创建。例如：数据库中的表 ALL_LISTS 原来由 td 用户创建，移植后 ALL_LISTS 表的用户更改为 dbo 这样问题也就解决了。

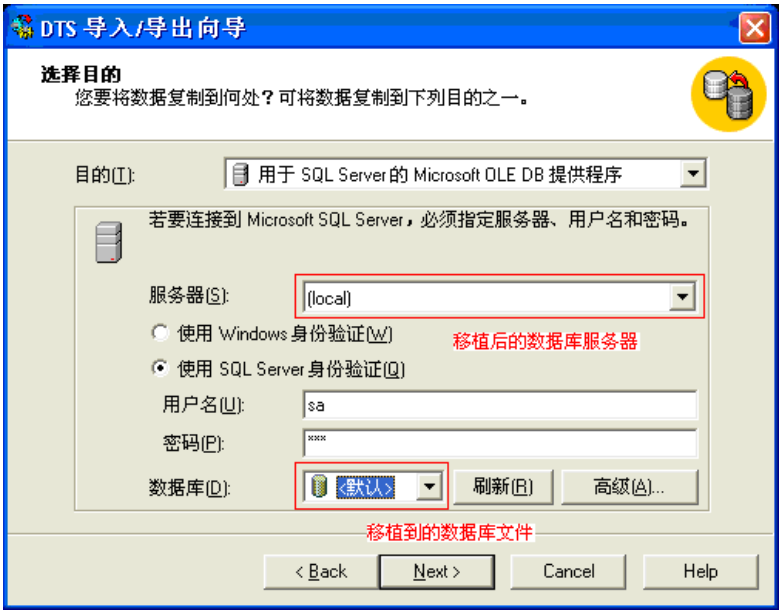
我们现在开始图解数据库的操作：

从开始菜单中打开“导入和导出数据”如图：


单击 “Next”

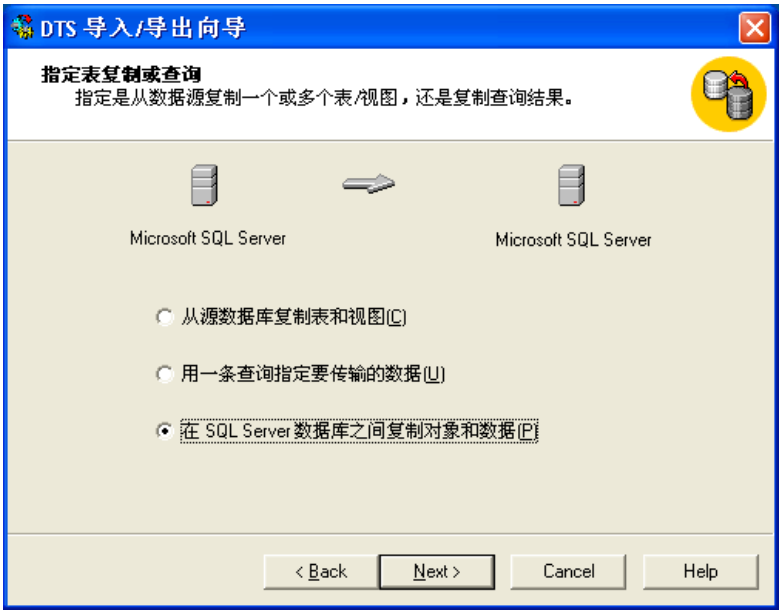



选择要恢复的数据源后单击 “Next”

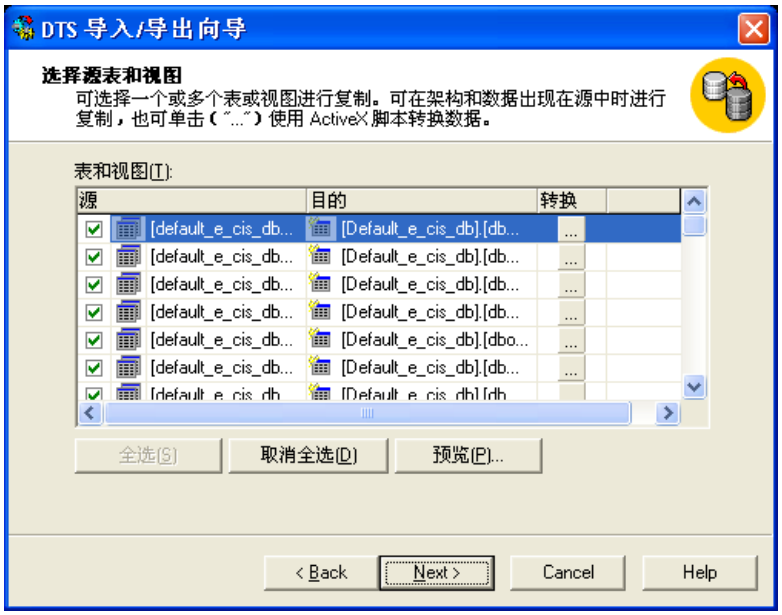



设置数据库恢复到的位置，注意：需要在数据库 数据库(D): 这里设定数据库被恢复的名称，点击 <默认> 的下拉箭头，单击新建按钮，出现：

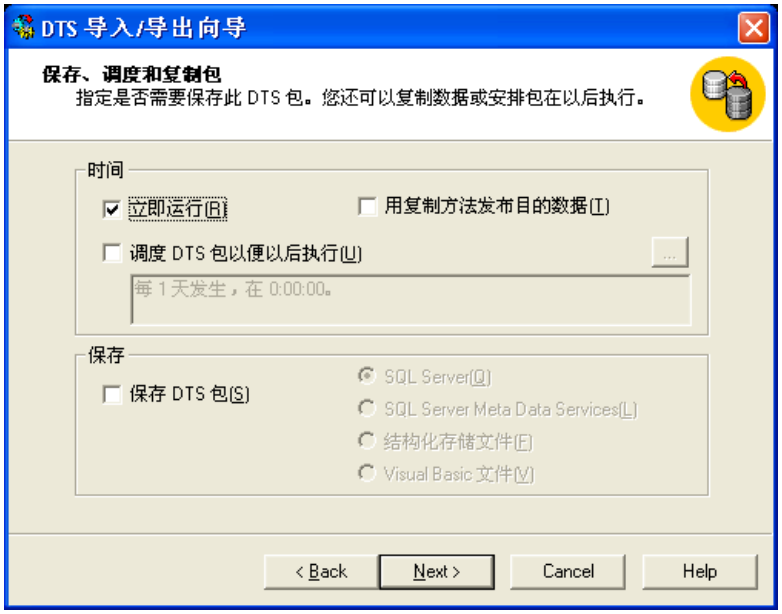

建立数据库名称后，单击确定后，再“Next”按钮：




再“Next”（这个选项选择第三项更具有完整性）

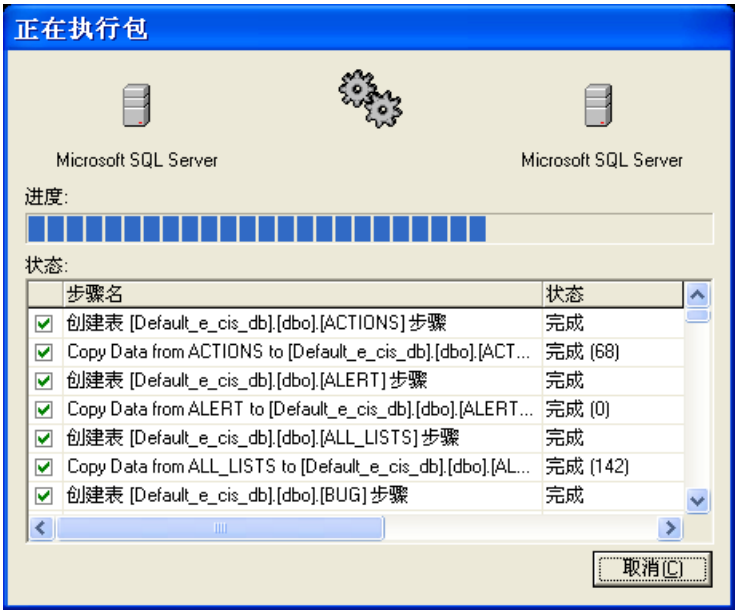



选择需要恢复的数据表后（在这里选择全部表），再”Next”：




准备开始恢复，单击“Next”：


单击”Finish”按钮，开始恢复：




恢复成功后出现提示：

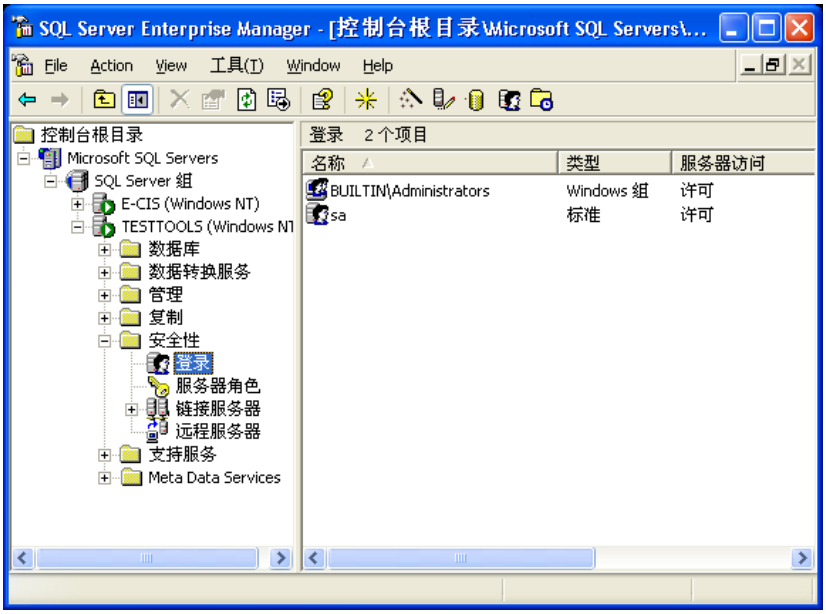

数据库恢复完成。

### 2.3.2. 建立访问数据表文件的 TD 用户：

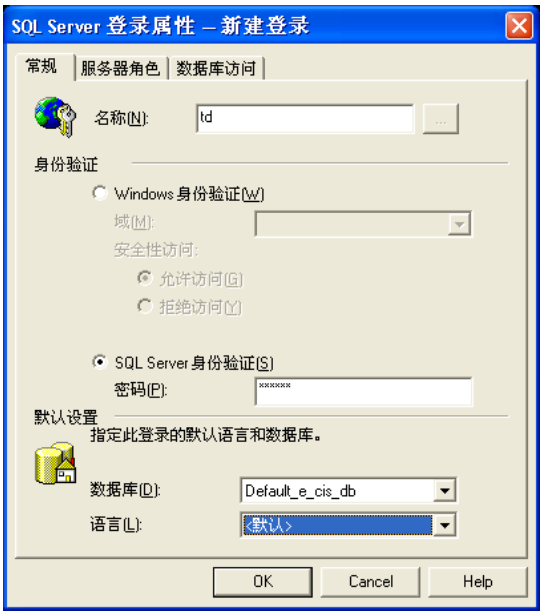
建立 td 用户的目的是：使 td 用户具有访问新恢复项目数据库的权限。

具体思路：建立一个名为 td 的用户，使这个用户具有访问 TD 项目数据库的控制权限，TD 通过这个用户对项目数据库进行增、删、改的操作。

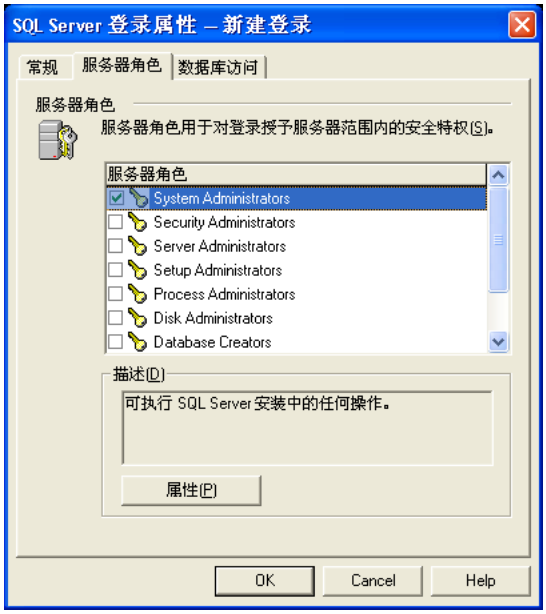
打开企业管理器，在目标数据库计算机中建立 TD 用户，并且设定 TD 用户访问的数据库。




在登录项中新建用户，在弹出的对话框中设定 td 用户相关的属性，如：




设定的密码可以根据用户确定，这里设定为：tdtdtd；在“服务器角色”中设定 td 用户的权限，我们给出建议设定为“System Administrators”：




完成“服务器角色”的设定，为 td 用户设定数据库访问的权限：


设定 td 用户能够访问 TD 项目的数据库和 Master 数据库。设定完成后，确认 td 用户默认数据库是否为 Default_e-cis_db

设定完成。

### 2.3.3. 修改 TD 的配置项目文件(Access)：

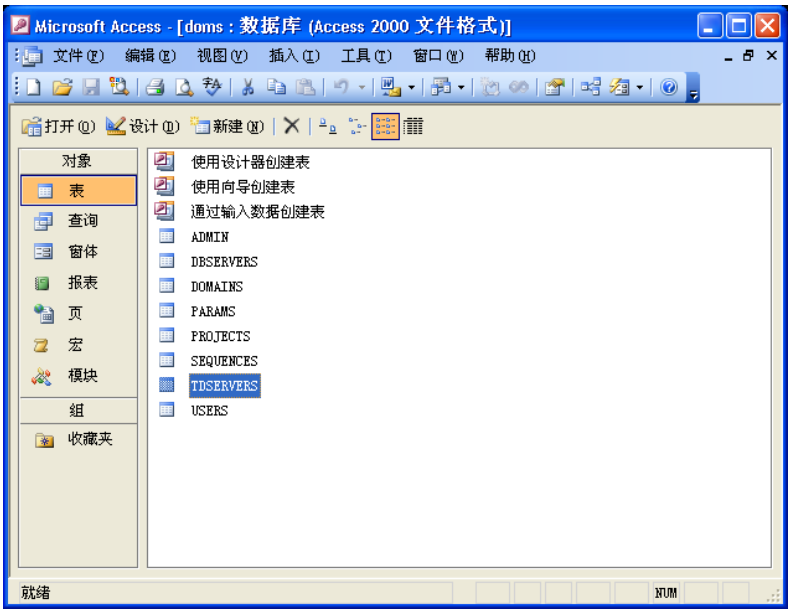
修改配置文件目的是：将项目文件（TDDIR 目录下存在的项目文件）和项目数据库进行关联。

具体思路：修改 ACCESS 数据库中每一个配置参数，使 TD 项目文件和项目数据库关联，通过这个操作我们还可以实现多个 TD 项目合并删除的功能。（在这里我们就不详细介绍，有兴趣的朋友可以研究

一下）

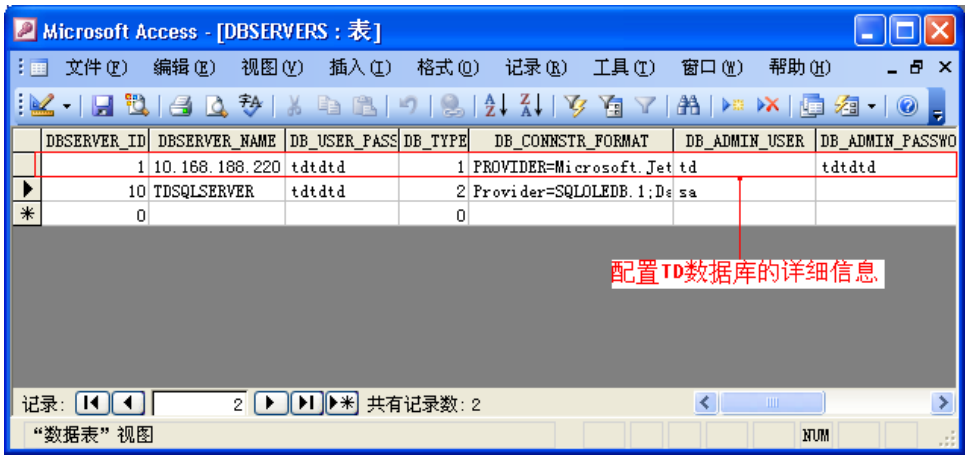
在 TD 安装的计算机中的 C:\Program Files\Common Files\Mercury Interactive\DomsInfo 目录下打开 doms.mdb 文件，此文件是经过加密处理的，其密码为：tdtdtd

打开后可以看到相应的表：



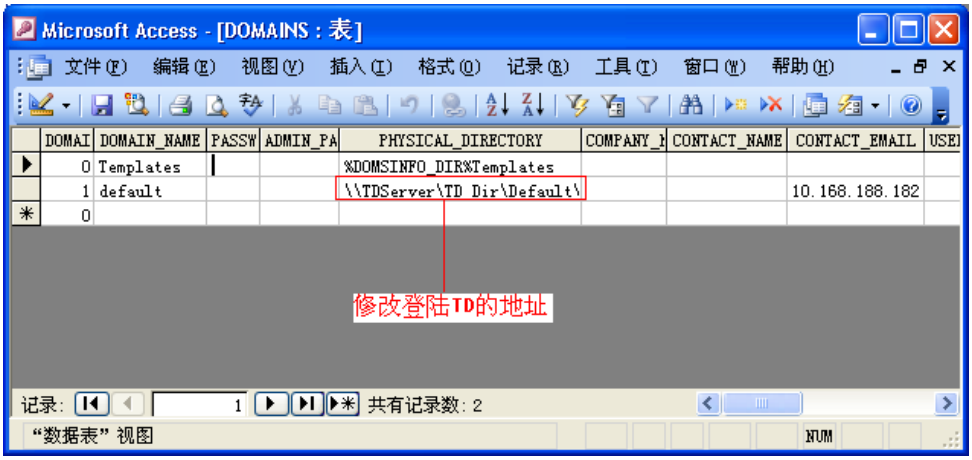

这些表，存放着 TD 所有的配置信息，是很重要的，我们根据自己的工作环境实际配置这个表，其中需要修改的表是：DBSERVER、DOMAINS、PARAMS、PROJECT 和 TDSERVERS 这五个表。

a.DBSERVER 表中主要修改 DBSERVER_NAME（目标数据库服务器的 IP 如本例为 1 0 .168.188.151）、DB_USER_PASS（用户密码如 tdtdtd）、DB_ADMIN_USER（管理用户名如 td）和 DB_ADMIN_PASSWORD（管理用户密码如 tdtdtd）字段的值，如下图：

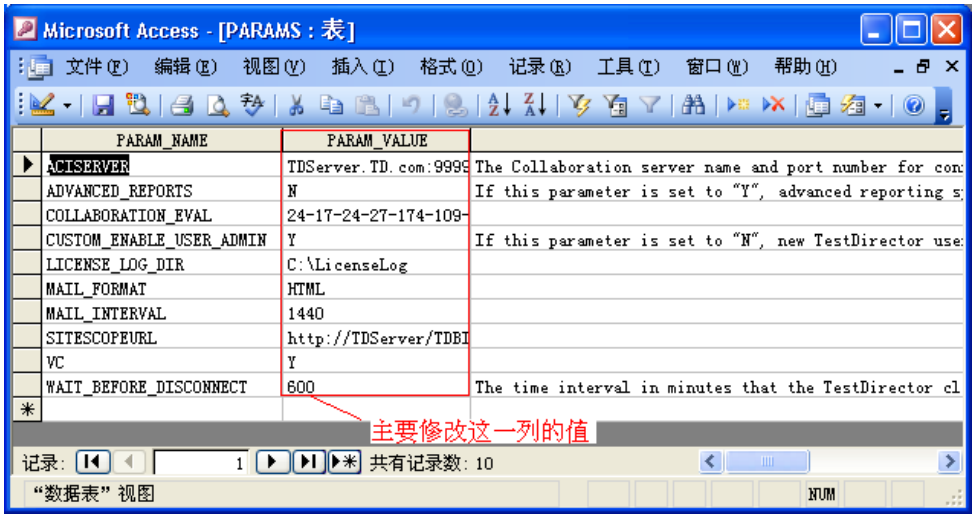



b.DOMAINS 表中主要修改 PHYSICAL_DIRECTORY 字段的值。（基本上不

要修改）

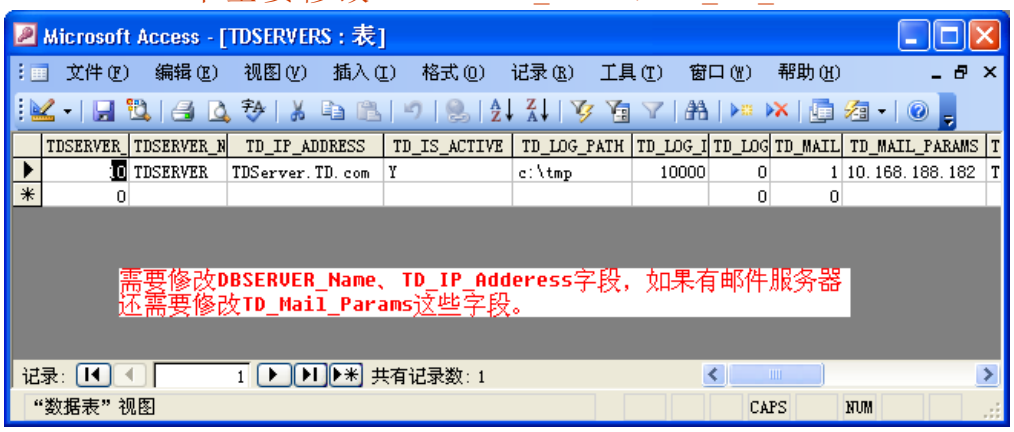



c.PARAMS 表中主要修改 PARAM_VALUE 字段的值。（基本上不要修改）

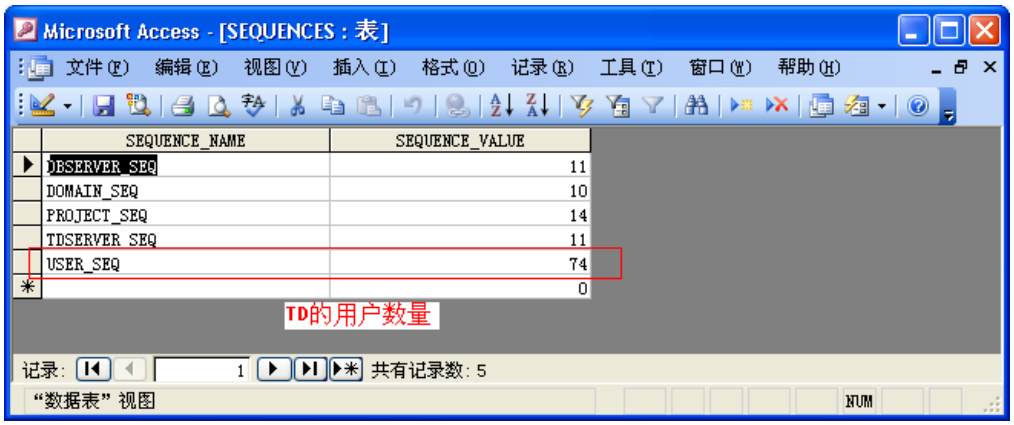



d.PROJECT 表中主要增加或修改一个记录就行了，其关键字段表示的意思是 PROJECT_ID(项目 ID 号不能重复),PROJECT_NAME(项目名称),DB_NAME（项目对应的数据库名称如本例的 default_e_cis_db）,DB_TYPE(数据库类型 1 为 ACCESS，2 为 SQL SERVER),PHYSICAL_DIRECTORY(TD 服务器中的共享目录如本例为\\10.168.188.152\td_dir\default\e_cis、DB_USER_PASS(TD 用户密码如本例为 tdtdtd)、DBSERVER_NAME(数据库的 IP 地址)。


e. TDSERVERS 中主要修改 DBSERVER_NAME、TD_IP_ADDRESS




**说明：**以上修改是针对移植工作修改的表，如果我们需要将分布在多台计算机上的 TD 项目合并移植到一台服务器上，还需要修改 ACCESS 数据库中的 SEQUENCES 表中的 USER_SEQ 字段的数值，这个字段是用来控制用户数量，如果数值太小 TD 部分用户就出现无法登录的现象。




**总结：**修改的内容还需要根据自己的实际情况进行修改，我这里只列出常修改的一些字段。

### 2.3.4. 修改 Project 中的 INI 文件：

修改 INI 文件目的是：ini 文件保存数据库信息，TD 程序信息，项目名称以及相关的内容。

具体思路：修改 ini 文件配置参数，使 TD 项目文件和项目数据库关联。

将 C：\TD_Dir\Default 目录下面的 E-CIS 项目文件，拷贝到备份的计算机中，然后修改 E-CIS 文件夹内的 Dbid.ini 文件：

Dbid.ini 内容：

```
[General]
Database_Type=MSSQL
Created_Date=08/11/04 15：44：39
Created_By=td
AliasName=E-CIS
Database Name=default_e_cis_db
Database Server=10.168.188.229
Domain Name=DEFAULT
SendAllQualified=Y
Has_VCS_DB=Y
```

主要修改的内容是：

```
Database Name=default_e_cis_db
Database Server=10.168.188.229
```

按照实际情况填写这些信息，就 ok 了。

**总结：**通过以上的介绍，我们集成工作环境的移植工作就已经完成了。经过这样的移植，我们可以实现 TD 项目的双机备份，双机备份可以让我们的数据更安全、更有保障。

### 3. 分布式工作环境的备份和恢复方法

分布式工作环境的项目移植相对集成工作环境移植更简单，在这里我们不详细介绍，相信大家看过集成工作环境的移植对分布式移植不在话下了。

我们对分布式工作环境移植的思路：

- 强调首先备份项目的数据。
- 移植 TDDir 目录下的项目文件到新的服务器上。
- 复制 DomsInfo 目录到新服务器 C:\Program Files\Common Files\Mercury Interactive 下。
- 修改这个目录下 Doms.mdb 文件的参数。（修改的详细方法见集成工作环境的移植）

总结：相对集成工作环境移植来说，分布式结构移植少了数据库的移植，可以继续使用原有的数据库，在移植过程中数据库的移植是移植成功与否的关键因素之一，少了这个过程相信大家移植更为方便。

【小结】以上我们介绍了两种结构的移植工作，经过上述修改项

后，迁移数据的工作就已经成功，td 服务器可以访问迁移后的数据库。通过移植工作，同时可以解决 TD 项目的双机相互备份的问题，保障数据的安全性。

再次备份 TD 项目，只需要将原来 TD 后台数据库表中的数据导入另一台计算机中的数据库中，就可以实现。

【作者的话】针对项目移植的操作方法和注意事项介绍到此结束。由于我们文笔有限，可能有所疏漏之处，希望大家包涵。如果在以后的工作中，有关 TD 方面的问题也可以和我们联系。

# 游戏测试过程

作者：落叶夏日（陈卫俊）

## 游戏测试起因

近几年来，网络游戏成了网络最新的弄潮儿，从盛大之传奇般的掘起，吸引了无数公司的眼球。但由于随着玩家的品位的升高，代理费用的上升，单一的代理国外游戏的模式已经很难在国内立足，而有中国传统文化特色的网络游戏则在国内大受欢迎，比如剑侠情缘，大话西游等一些国内的精典之作已经进入了一流网游的阵营。与此同时随着大家对网游稳定性，可玩性要求的升高，网络游戏测试开始成为大家关注的话题。

## 游戏测试与软件测试的区别

游戏测试作为软件测试的一部分，它具备了软件测试所有的一切共同的特性：

- 测试的目的是发现软件中存在的缺陷。
- 测试都是需要测试人员按照产品行为描述来实施。产品行为描述可以是书面的规格说明书，需求文档，产品文件，或是用户手册，源代码，或是工作的可执行程序。
- 每一种测试都需要产品运行于真实的或是模拟环境之下。
- 每一种测试都要求以系统方法展示产品功能，以证明测试结果是否有效，以及发现其中出错的原因，从而让程序人员进行改进。

总而言之，测试就是发现问题并进行改进，从而提升软件产品的质量。游戏测试也具备了以上的所有特性，不过由于游戏的特殊性，所以游戏测试则主要分为两部分组成，一是传统的软件测试，二游戏本身的测试，由于游戏特别是网络游戏，它相当于网上的虚拟世界，是人类社会的另一种方式的体现，所以也包含了人类社会的一部分特性，同时它又是游戏所以还涉及到娱乐性，可玩性等独有特性，所以测试的面相当的广。 我们称之为游戏世界测试，主要有以下几个特性：

- 游戏情节的测试，主要指游戏世界中的任务系统的组成，有人也称为游戏世界的事件驱动，我喜欢称为游戏情感世界的

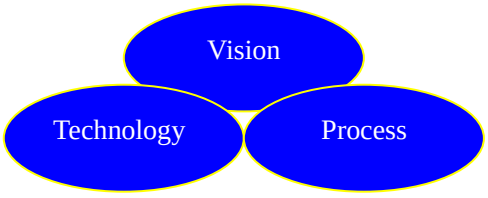
测试。

- 游戏世界的平衡测试，主要表现在经济平衡，能力平衡（包含技能，属性等等），保证游戏世界竞争公平。
- 游戏文化的测试，比如整个游戏世界的风格，是中国文化主导，还是日韩风格等等，大到游戏整体，小到 NPC（游戏世界人物）对话，比如一个书生，他的对话就必需斯文，不可以用江湖语言☺。

## 游戏测试概述

很多人有这样一个观点:“就是在软件开发完毕后,再进行测试。”殊不知，这种关点是有悖于软件开发的生命周期的，软件缺陷的发现必须是越早越好，这样才可以有效的规避风险，而在“最后进行测试”的测试观念的指导下测试工作必将会产生很多问题，这种观念的错误在于：生命周期中的“测试阶段”表明在该阶段测试工作是主要的工作，而不是说，测试工作只发生在“测试阶段”。通常，到了测试阶段，测试的主要任务是运行测试，形成测试报告。而想要提高游戏的质量，则必需要做到测试的早期介入，诸如测试计划，测试用例的确定以及测试代码的编写等等都是要在更早的阶段进行。如果你把测试完全放在最后阶段，就错过了发现构架设计和游戏逻辑设计中存在严重问题的最好时机，到那时，要修复这些缺陷将很不方便，因为缺陷已经扩散到系统中去了，所以这样的错误将很难寻找与修复，代价更高。

要了解如何测试游戏必需了解如何做游戏，了解它的开发过程，才能真正的测好游戏。游戏要成功，其基本的必要条件有三。分别为 Vision(设计)、technology(技术)和 Process(过程)。三个条件，缺一不可如图所示：




图：游戏开发三大基石

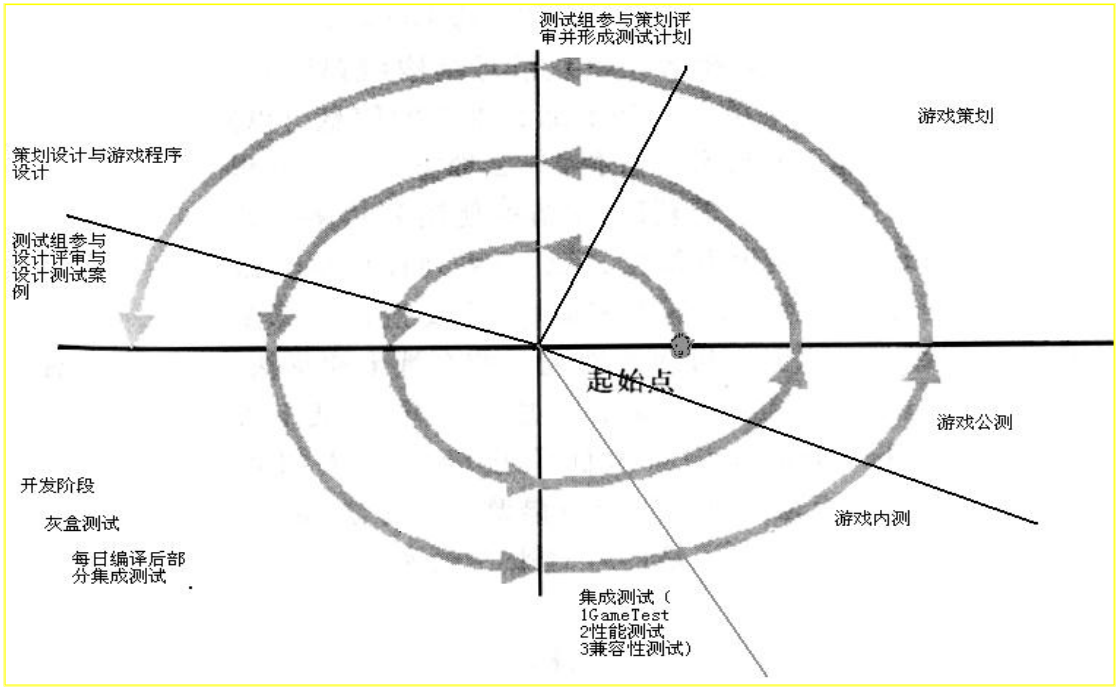
- Vision 则是对游戏还没有实现的总体上的把握，前瞻性的理

解与策略的考量。

- Technology：有了 vision，如果没有技术的话，则各种美妙的想法只能停留在虚无缥缈的阶段，通过技术来实现 Vision。
- Process：有了 Vision 作为指导，有了技术作为保证，也不一定能够把好的想法转换成高质量的游戏。要创造高品质的游戏，尚缺重要的一环，即过程，制造游戏是一个非常是一个长时间的动态过程。游戏产品的质量则是要靠动态过程的动态质量来进行保证。过程由很多复杂的相互牵制的环节与部件组成，如果任意的环节或者是部件出了问题都会对最终的产品形成质量上的影响。因此对这个动态的过程，一定要有规划与控制，以保证按步就班，按质按时完成工作。

## 游戏测试与开发过程的关系

CMM（Software Capability Maturity Model）软件成熟模型，大家都比较熟悉了，但在实施的过程中却存在这样那样的问题，对于游戏开发就更没有一个固定的路可以讲了，我们的团队是一个长期的游戏开发团队，对游戏开发有着很深的认识，我们认为游戏的 Process(过程)实际上也是软件过程，不过是特殊的游戏软件开发过程，各个生命周期还是相通的。所以我们总结一套以测试作为质量驱动的、属于自己的开发过程。下图是游戏的迭代式开发过：




图：游戏迭代式开发与测试

由于网络游戏的生命周期也是 3、4 年，所以采用迭代式的开发过程，既可以适应网络游戏本身这种长周期的开发，又可以利用 RUP 的迭代式开发的优点与 CMM 的里程碑控制，从而达到对游戏产品的全生命周期的保证。

在游戏开发过程中，通用软件的需求分析阶段被策划所代替，但起的作用是一样的，明确游戏的设计目标（包括风格，游戏玩家群），游戏世界的组成，为后期的程序设计，美工设计，测试提出的明确的要求。由于开发是一个阶段的过程，所以测试与开发的结合就比较容易，从图上我们可以看到测试的工作与游戏的开发是同步进行的，每一个开发阶段中测试都进行了参与，能够深入的了解到系统的整体与大部分的技术细节，从而从很大程度上提高了测试人员对错误问题判断的准确性,并且可以有效的保证重要游戏系统的稳定。

**游戏策划与测试计划**

测试过程不可能在真空中进行。如果测试人员不了解游戏是由那几个部分组成的，那么执行测试就非常的困难，同时测试计划可以明确测试的目标，需要什么资源，进度的安排，通过测试计划，既可以让测试人员了解此次游戏测试中那些是测试重点，又可以与产品开发小组进行交流。在企业开发中，测试计划书来源于需求说明文档，同样在游戏开发过程中，测试计划的来源则是策划书。策划书包含了游戏定位，风格，故事情节，要求的配制等等。在策划评审中我们的高级测试人员可以参与进来，得到详细的游戏策划书，从里面了解到游戏的组成，可玩性，平衡（经济与能力），与形式（单机版还是网络游戏），而我们测试在这一阶段主要的事情就是通过策划书来制定详细的测试计划，主要分两个方面一是游戏程序本身的测试计划，比如任务系统，聊天，组队，地图等等由程序来实现的功能测试计划，二是游戏可玩性有测试计划，比如经济平衡标准是否达到要求，各个门派技能平衡测试参数与方法，游戏风格的测试，三是关于性能测试的计划，比如客户端的要求，网络版的对服务器的性能要求。同时测试计划书中还写明了基本的测试方法，要设计的自动化工具的需求，为后期的测试打下良好的基础。同时由于测试人员参与到策划评审，资深的游戏测试人员与产品经理由于对游戏也有很深入的了解，会对策划提出自己的看法，包含可玩性，用户群，性能要求等等并形成对产品的风险评估分析报告，但这份报告不同于策划部门自己的风险分析报告，主要从旁观者的角度对游戏本身的品质作充分的论证，从而更

有效的对策划起到控制的作用。

**游戏设计与测试**

设计阶段是做测试案例设计的最好时机。很多组织要么根本不做测试计划和测试设计，要么在即将开始执行测试之前才飞快地完成测试计划和设计。在这种情况下，测试只是验证了程序的正确性，而不是验证整个系统本该实现的东西。而我们的测试则会很明确，因为我们的测试计划已经写的很明确，需要测试那些游戏系统，但是我们还需要了解系统的组成，而设计阶段则是设计系统的过程，所有的重要系统均是用UML状态图进行了详细的描述，比如用户登陆情况。如图 2：

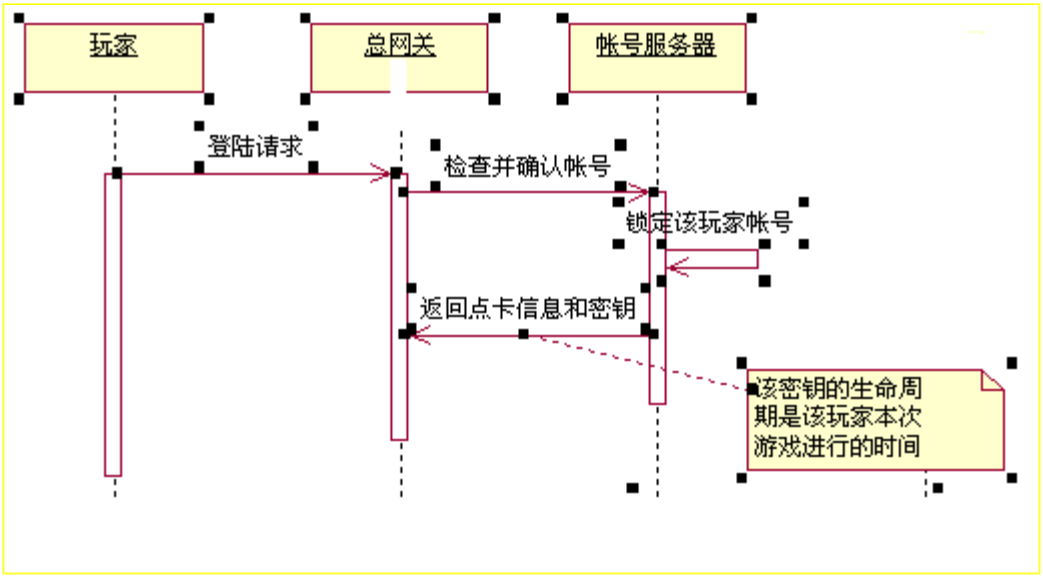



图 2 用户登陆情况

在我们的团队中资深的测试人员要具备的一项基本的素质就是可以针对 UML 的用例图，时序图，状态图来设计出重要系统的测试案例，只有重要系统的质量得到充分的测试，游戏程序的质量才可以得到充分的保证。比如上图中就是一个用户登陆游戏系统的时序图。从这里我们可以很明确的了解玩家是如何验证并登陆系统的，在这个过程中要与那些对象进行交互，比如这里我们就是三个系统之间的交互，客户端（玩家部分），网关，账号服务之间的一个时序变化关系，为了能够完整的对这个流程进行测试，我们必需设计出可以覆盖整个流程的测试案例，并考虑其中可能的非法情况，因为这个时序图只是考虑了用户正常登陆成功的情况，并没有考虑密码错误，通信失败等许多可能存有的情况，并形成完整的测试案例库，从而对登陆系统的

系统化测试做了充分的准备。同时通过这张图，性能分析人员还可以分析出可能存的性能瓶颈，比如这里可能有的瓶颈如下，总网关是否可以达到多少用户的并发，是如果达不到，是否可以采用分布式部署或是支持负载平衡，三者之间的网络带宽的比例分配，账号服务器是否可以承载多个网关的连接请求，最大连接请求可以达到多少等等，同时会针对这些风险做性能测试的设计，并提出自动化测试的需求，比如模拟玩家登陆的压力工具等等。

同时在设计评审时，测试人员的介入可以充分的对当前的系统构架发表自己的意见，由于测试人员的眼光是最苛刻的，并且有多年的测试经验，可以比较早的发现曾经出现的设计上的问题，比如在玩家转换服务器时是否作了事务的支持与数据的校验，在过去设计中由于没有事务支持与数据的校验从而导致玩家数据丢失，而这些风险可以在早期就规避掉。上面所说的是对游戏程序本身的测试设计，对于游戏情节的测试则可以从策划获得，由于前期的策划阶段只是对游戏情节大方向上的描述，并没有针对某一个具体的情节进行设计，进入设计阶段时，某个游戏情节逻辑已经完整的形成了，策划可以给出情节的详细设计说明书，称为任务说明书，通过任务说明书我们可以设计出任务测试案例，比如某一个门派的任务由那些组成，我们可以设计出完整的任务测试案例，从而保证测试可能最大化的覆盖到所有的任务逻辑，如果是简单任务，还可以提出自动化需求，采用机器人自动完成。

**游戏测试与开发**

开发与测试一直有人认为是不可以平行进行的，必需要先开发后测试，但是软件的开发过程又要求测试必须早期介入，但在这里这种矛盾得到了很好的解决。我们采用了每日编译，将测试执行和开发结合在一起，并在开发阶段以编码--测试--编码--测试的方式来体现。也就是说，程序片段一旦编写完成，就会立即进行测试。普通情况下，先进行的测试是单元测试，但是一个程序片段也需要相关的集成测试，甚至有时还需要一些特殊测试。特别是关于接口测试，像游戏程序与任务角本、图片的结合，大家都认为需要提前测试，通过每日编你可以把已经写好的程序片段接合起来，形成部分的集成测试，从而有效的体现的接口优先测试的原则。同时由于软件测试与开发是并行进行的，并且实行的是软件缺陷优先修改的策略，所以很少会出现缺陷后期无法修改的情况，并且由于前期的测试案例的设计与自动化工具的准备，我们不需要投入太多的人力就可以极高的保证游戏软件的

产品质量，特别是重要系统的质量。由于我们的游戏程序是每日不断的完善，所以集成测试也在同步的进行之中，当开发进入最后阶段时，集成测试也同步的完成了。这里有一个原则，也就是我前面所说的，测试的主体方法和结构应在游戏设计阶段完成，并在开发阶段进行补充（比如在游戏开发中会有相应的变动，或是某个转移变地址的变化，这就需要实时的更新）。这种方法会对基于代码的测试（开发阶段与集成阶段）产生很重要的影响，但是不管在那个阶段，如果在执行前多做一点计划和设计，都会大幅度的提高测试效率，改善测试结果，同时还有利于测试案例的重用与测试数据的分析，所以我们的测试计划是在策划时就形成了，为后继的测试形成了良好的基础。

**集成测试阶段**

集成测试是对整个系统的测试。由于前期测试与开发的并行，集成测试已经基本完成，这时只需要对前期在设计阶段中设计的系统测试案例运行一下就 OK 了。我们主要的重心在集成测试中的兼容性测试，由于游戏测试的特殊性，对兼容性的要求特别高，所以我们采用了外部与内部同部进行的方式，内部我们有自己的平台试验室，搭建主流的硬软件测试环境，同时我们还通过一些专业的兼容性测试机构对我们的游戏软件做兼容性分析，让我们的游戏软件可以跑在更多的机器上。

**游戏可玩性测试**

游戏可玩性测试也是非常重要的一块，主要包含四个方面：

- 游戏世界的搭建，包含聊天功能，交易系统，组队等可以让玩家在游戏世界交互的平台。
- 游戏世界事件的驱动，主要指任务。
- 游戏世界的竞争与平衡。
- 游戏世界文化蕴涵，游戏的风格与体现。

这种测试主要体现在游戏可玩性方面，虽然策划时我们对可玩性作了一定的评估，但这是总体上的，但一些具体的涉及到某个数据的分析，比如 PK 参数的调整，技能的增加等一些增强可玩性的测试则需要职业玩家对它进行分析，这里我们主要通过三种方式来进行：

- 内部的测试人员，他们都是精选的职业玩家分析人员，对游戏有很深的认识，在内部测试时，对上面的四点进行分析。
- 利用外部游戏媒体专业人员对游戏作分析与介绍，既可以达到

宣传的效果，又可以达到测试的目的，通常这种方式是比较好的。

● 利用外部一定数量的玩家，对外围系统的测试，他们是普通的玩家，但却是我们最主要的目标，主要的来源是大中院校的学生等等，主要测试游戏的可玩性与易用性，发现一些外围的 Bug。

游戏进入到最后阶段时，还要做内测，公测，有点像应用软件的beta版的测试，让更多的人参与测试,测试大量玩家下的运行情况。

可玩性测试是游戏重要的一块，只有玩家的认同，我们才可能成功。

**性能测试与优化**

最后要单独提一下的是性能优化，在单机版的时代，性能的要求并不是很高，但是在网络版的时代，则是两个完全不同的概念，主要包含了以下几个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，三方面有效、合理的结合，可以达到对系统性能全面的分析和瓶颈的预测。不过在测试过程中有这样一个原则，就是由于测试是在集成测试完成或接近完成时进行，要求测试的功能点能够走通，这时你首先要进行优化的是数据库或是网络本身的配制，只有这样才可以规避改动程序的风险。同时性能的测试与优化是一个逐步完善的过程，需要前期的很多的工作，比如性能需求，测试工具等等，不过由于前期工作的完善，这些都在前期完成了。这里我只做原则性的描述。

数据库的优化的原则主要是这样的，首先是索引进行优化，由于索引的优化不需要对表结构进行任何改动，是最简单的一种，又不需要改动程序就可能提升性能若干倍，不过要注意的是索引不是万能的，若是无限的增加会对增删改造成很大的影响。其次是对表，视图，存储过程的优化。不过在分析之前需要知道优化的目标，客户行为中那些SQL是执行的最多的，所以我们必需借助些SQL的跟踪分析工具，例如SQLProfile,SQLExpert,等工具，这样会迅速的定位问题。

关于网络的优化，这里我所说的并不是针对网络本身的优化，而是对游戏本身的网络通信的优化，所以它是与程序的优化是结合在一起的，首先也是发现问题，通过Monitor与Sniff先定位是什么应用占用了较多的网络流量，由于网络游戏的用户巨大，所以这也是一个重在的问题。对于程序的性能优化，最主要的是找到运行时间最长的函数，只有优化它，性能才有大幅度的提升，具体的方法我就不做详细的描述了。

## 总述

游戏测试是一个新的领域，它既有通用测试的特点，又有自己的特点，有许多未知的路要走，每天都在总结，希望给大家带来一些帮助，同时在这里也谢谢所有支持我的同事。

# 软件测试的基本常识

作者：张华（51testing 顾问）

软件开发和使用的历史已经留给了我们很多由于软件缺陷而导致的巨大财力、物力损失的经验教训。这些经验教训迫使我们这些测试工程师们必须采取强有力的检测措施来检测未发现的隐藏的软件缺陷。

我们生产软件的最终目的是为了满足客户需求，我们以客户需求作为评判软件质量的标准，认为软件缺陷（Software Bug）的具体含义包括下面几个因素：

- 软件未达到客户需求的功能和性能；
- 软件超出客户需求的范围；
- 软件出现客户需求不能容忍的错误；
- 软件的使用未能符合客户的习惯和工作环境。

考虑到设计等方面的因素，我们还可以认为软件缺陷还可以包括软件设计不符合规范，未能在特定的条件（资金、范围等）达到最佳等。可惜的是，我们中的很多人更倾向于把软件缺陷看成运行时出现问题上来，认为软件测试仅限于程序提交之后。

在目前的国内环境下，我们几乎看不到完整准确的客户需求说明书，加以客户的需求时时在变，追求完美的测试变得不太可能。因此作为一个优异的测试人员，追求软件质量的完美固然是我们的宗旨，但是明确现实与理想的差距，在软件测试中学会取舍和让步，对软件测试是有百益而无一弊的。

下面是一些软件测试的常识，这些常识对我们进行软件测试时能够更好的把握软件测试的尺度。

## 1. 测试是不完全的（测试不完全）

很显然，由于软件中逻辑路径的组合性，输入数据的大量性及结果多样性等因素，哪怕是一个极其简单的程序，要想穷尽所有逻辑路径，所有输入数据和验证所有结果是非常困难的一件事情。举一个例子，求最大公约数。其输入为两个正整数。但是将整个整数域的数字进行一番测试，从其数目的无限性我们便可证明是这样的测试在实际

生活中是行不通的，即便某一天我们能够穷尽该程序，只怕我们都早已作古了。为此作为软件测试，我们一般都采用等价类和边界值分析等措施来进行实际的软件测试，寻找最小用例集合成为我们精简测试复杂性的一条必经之道。

### 2. 测试具有免疫性（软件缺陷免疫性）

软件缺陷与病毒一样具有可怕的“免疫性”，测试人员对其采用的测试越多，其免疫能力就越强，寻找更多软件缺陷就更加困难。由数学上的概率论我们可以推出这一结论。假设一个 50000 行的程序中有 500 个软件缺陷并且这些软件错误分布时均匀的，则每 100 行可以找到一个软件缺陷。我们假设测试人员用某种方法花在查找软件缺陷的精力为 X 小时 /100 行。照此推算，软件存在 500 个缺陷时，我们查找一个软件缺陷需要 X 小时，当软件只存在 5 个错误时，我们每查找一个软件缺陷需要 100X 小时。实践证明，实际的测试过程比上面的假设更为苛刻，为此我们必须更换不同的测试方式和测试数据。该例子还说明了在软件测试中采用单一的方法不能高效和完全的针对所有软件缺陷，因此软件测试应该尽可能的多采用多种途径进行测试。

### 3. 测试是“泛型概念”（全程测试）

我一直反对软件测试仅存在于程序完成之后。如果单纯的只将程序设计阶段后的阶段称之为软件测试的话，需求阶段和设计阶段的缺陷产生的放大效应会加大。这非常不利于保证软件质量。需求缺陷、设计缺陷也是软件缺陷，记住“软件缺陷具有生育能力”。软件测试应该跨越整个软件开发流程。需求验证（自检）和设计验证（自检）也可以算作软件测试（建议称为：需求测试和设计测试）的一种。软件测试应该是一个泛型概念，涵盖整个软件生命周期，这样才能确保周期的每个阶段禁得起考验。同时测试本身也需要有第三者进行评估（信息系统审计和软件工程监理），即测试本身也应当被测试，从而确保测试自身的可靠性和高效性。否则自身不正，何以服人。

### 4. 软件缺陷具有空间聚集性（80-20 原则）

80%的软件缺陷常常生存在软件 20%的空间里。这个原则告诉我们，如果您想使软件测试有效地话，记住常常光临其高危多发“地段”。在那里发现软件缺陷的可能性会大的多。这一原则对于软件测试人员

提高测试效率及缺陷发现率有着重大的意义。聪明的测试人员会根据这个原则很快找出较多的缺陷而愚蠢的测试人员却仍在漫无目的地到处搜寻。

### 5. 为效益而测试

为什么我们要实施软件测试，是为了提高项目的质量效益最终以提高项目的总体效益。为此我们不难得出我们在实施软件测试应该掌握的度。软件测试应该在软件测试成本和软件质量效益两者间找到一个平衡点。这个平衡点就是我们在实施软件测试时应该遵守的度。单方面的追求都必然损害软件测试存在的价值和意义。一般说来，在软件测试中我们应该尽量地保持软件测试简单性，切勿将软件测试过度复杂化，拿物理学家爱因斯坦的话说就是：Keep it simple but not too simple。

### 6. 缺陷的必然性

软件测试中，由于错误的关联性，并不是所有的软件缺陷都能够得以修复。某些软件缺陷虽然能够得以修复但在修复的过程中我们会难免引入新的软件缺陷。很多软件缺陷之间是相互矛盾的，一个矛盾的消失必然会引发另外一个矛盾的产生。比如我们在解决通用性的缺陷后往往会带来执行效率上的缺陷。更何况在缺陷的修复过程中，我们常常还会受时间、成本等方面的限制因此无法有效完整地修复所有的软件缺陷。因此评估软件缺陷的重要度、影响范围，选择一个折中的方案或是从非软件的因素（比如提升硬件性能）考虑软件缺陷成为我们在面对软件缺陷时一个必须直面的事实。

### 7. 软件测试必须有预期结果

没有预期结果的测试是不可理喻的。软件缺陷是经过对比而得出来的。这正如没有标准无法进行度量一样。如果我们事先不知道或是无法肯定预期的结果，我们必然无法了解测试正确性。这很容易然人感觉如盲人摸象一般，不少测试人员常常凭借自身的感觉去评判软件缺陷的发生，其结果往往是把似是而非的东西作为正确的结果来判断，因此常常出现误测的现象。

### 8. 软件测试的意义-事后分析

软件测试的目的单单是发现缺陷这么简单吗？如果是“是”的话，

我敢保证，类似的软件缺陷在下一次新项目的软件测试中还会发生。古语说得好，“不知道历史的人必然会重蹈覆辙”。没有对软件测试结果进行认真的分析，我们就无法了解缺陷发生的原因和应对措施，结果是我们不得不耗费的大量的人力和物力来再次查找软件缺陷。很可惜，目前大多测试团队都没有意识到这一点，测试报告中缺乏测试结果分析这一环节。

软件测试是一个需要“自觉”的过程，作为一个测试人员，遇事沉着，把持尺度，从根本上应对软件测试有着正确的认识，希望本文对读者对软件测试的认识有所帮助。

# 浅谈软件开发中的注意事项

作者：金兵

本人从事了四年左右的软件开发工作，曾参与开发过多个大型项目，对于整个项目开发中我觉得有很多地方都应该应当值得注意，特写如下观点和各位看客共同探讨！

## 1. 项目设计

项目设计的主导思想，我觉得可以理解为两种，一种是完全设计，一个是简单设计。

完全设计是指在具体编写代码之前对软件的各种方面都调查好，做好详细的需求分析、编写好全部的开发文档，设计出程序全部流程后再开始写代码。 换句话说，就是全部的计划好了，能看到最终的样子，再开战。这好像也是很多“软件工程”书里要求的那样。开始的时候，我觉得这种方法不错也。什么都计划好了，照着做就是了。不过这里有个明显的问题，就是谁来做这个完美的计划？估计只有及其 BT 的人了，但是大部分人的想要完全设计，并且没有错误，或者已经有几种后备的容错方案，并能准确无误的推行。以达到最终目标。这样的境界，没有很多年的工作经历是不可能的。我也没有这样的本事，所以我也就放弃了这种想法。

简单设计：简单设计一种概念，一种可以接受的简单的设计，最起码数据库已经定下来，基本流程已经确定的方案，来作为程序设计的开始，并随时根据实际情况的进展来修正具体的功能设计，但这种功能修改不能是修改数据库结构。也就是说数据库结构是在编程之前经过反复论证的。这种方法减少了前期设计的时间，把代码编写工作和部分设计工作放在了一起，实际缩短了项目开发的时间。如果说完全设计方法要求有很厉害的前期设计人员，那么简单设计要求有很有设计头脑的编程人员。编程人员不仅仅是 K 代码的人而且要负责程序架构的设计。所以对程序员的要求就很高了。 简单设计的成功的一个基点是编程人员设计的逻辑结构简单并能根据需要来调整其逻辑结构，就是代码结构灵活，简单设计带来的另外一个变化就是会议会比较多，编程人员之间的交流就变的很重要。现在一般的中小型软件公司基本上都是采用简单设计的，除非那些很大型的软件公司。

总结，简单设计考验的是开发人员的能力。完全设计考验的是前期设计人员和整个项目组完整能力。（各种文档的编写，开发人员一定会要写一部分的。）

## 2. 设计变化和需求变化

开发人员最怕的是什么呢？设计变化，还是需求变化？我觉得需求变化是最最致命的。当你的一个项目数据库都定下来后，而且已经开发了若干个工作日，突然接到甲方公司提出，某个功能要改变，原先的需求分析要重新改，如果这个修改是涉及的数据库的表结构更改的话，那真是最致命的。这就意味着项目的某些部分得重新推倒重来，如果这个部分跟已完成的多个部分有牵连的话，那就后果更可怕了。所以当碰到这种情况发生，作为项目经理的你就应该考虑先查责任人，究竟是自己的需求分析做的不够好，还是客户在认同了需求分析后做出的修改，如果是后者的话，你完全可以要求客户对他的这个修改负责任！那么，呵呵，客户先生，对不起了，本次新增加的需求将归入另外一个版本。如果是改变前面某个需求的定义，那么说不定就要推倒重来了，不过这个时候到不用太在意，毕竟错的是客户。（项目正式开始前没有没有说清楚其需求）。所以，各位看客，在需求分析做好后，在开工之前一定要叫客户认可签字，并且在合同上要注明，当由客户原因引起的需求改变而造成开发成本的增加，客户要为此买单地。

如果在需求不变的情况之下，设计发生了变化，这个仅仅是我们内部之间的矛盾，商量一下就能解决。在简单设计中，因为前期的设计是不完整的，那么当进入任何一个新的模块进行开发时，都有可能引起设计的变化。开发人员的水平的高低就基本上决定了软件的好坏。

## 3. 代码编写

当需求定下来数据库也定下来后， 其实我们就可以进行实质性的编码了，按照我的看法，一个人单独编程最好，能随时偷懒。（上网，和 MM 聊聊），但是现在的软件项目越来越大，工期也越来越紧，事实上我们一个小组里面，一般有 3-5 程序员，所以我们要强调团队合作性。那么你写的代码使得别人要能够看懂，我们必须在实际的编写代码过程中要有详细的编码规范，编码规范在很多书籍里面都提到

过。但最起码以下的一些规范是我们必须要遵守的：

### 3.1.源程序文件结构：

每个程序文件应由标题、内容和附加说明三部分组成。

（1）标题：文件最前面的注释说明，其内容主要包括：程序名，作者，版权信息，简要说明 等，必要时应有更详尽的说明（将以此部分以空行隔开单独注释）。

（2）内容控件注册等函数应放在内容部分的最后，类 的定义按 private 、 protected 、 pubilic 、 __pubished 的顺序，并尽量保持每一部分只有一个，各部分中按数据、函数、属性、事件的顺序。

（3）附加说明：文件末尾的补充说明，如参考资料等，若内容不多也可放在标题部分的最后。

### 3.2.界面设计风格的一致性：

由于采用可视化编程，所有的界面均与 Win32 方式类似，相应采用的控件等也大都为 Windows 操作系统下的标准控件，而且参考了其他一些市面上相关的企业内部管理的应用软件。

基于简单易操作的原则，贴近用户考虑，用户界面采用 Windows 风格的标准界面，操作方式亦同 Windows 风格，这样在实施过程，可以降低对客户的培训，也可以使用户容易上手，简单易学。

### 3.3.编辑风格：

（1）缩进：缩进以 Tab 为单位，一个 Tab 为四个空格大小。全局数据、函数 原型、标题、附加说明、函数说明、标号等均顶格书写。

（2）空格：数据和函数在其类型，修饰（如 __fastcall 等）名称之间适当空格并据情况对 齐。关键字原则上空一格，不论是否有括号，对语句行后加的注释应用适当空格与语句隔开并尽可能对齐。

（3）对齐：原则上关系密切的行应对齐，对齐包括类型、修饰、名称、参数等各部分对齐。

另，每一行的长度不应超过屏幕太多，必要时适当换行。

（4）空行：程 序文件结构各部分之间空两行，若不必要也可只空一行，各函数实现之间一般空两行。

（5）注释：对注释有以下三点要求：

A、必须是有意义；

B、必须正确的描述了程序；

C、必须是最新的。

注释必不可少，但也不应过多，以下是四种必要的注释：

- 标题、附加说明；
- 函数说明：对几乎每个函数都应有适当的说明，通常加在函数实现之前，在没有函数实现部分的情况下则加在函数原型前，其内容主要是函数的功能、目的、算法等说明，参数说明、返回 值说明等，必要时还要有一些如特别的软硬件要求等说明；
- 在代码不明晰或不可移植处应有少量说明；
- 及少量的其它注释。

### 3.4. 命名规范：

坚持采用匈牙利变量命名惯例，所有标识符一律用英文或英文缩写，杜绝采用拼音，标识符中每个单词首字母大写，缩写词汇一般全部大写，只在必要时加“_”间隔词汇。

## 4. BUG 修补

程序出现了 BUG 谁来修补呢，嘿嘿嘿……

最好的办法是谁编写谁修补，谁改坏谁修补。一个人改坏的代码一人去修。两个人一起改坏的代码两人一起修。

## 5. 开发人员的测试

开发人员的测试是保证代码能正常运行，在开发时候发现的错误往往比较容易修正。（另外一个好处就是没有人来骂你。因为只有你自己知道）。但是一旦软件到了测试小组那里出了问题，那么就多了很多时间来修正 BUG，如果到了客户哪里才发现的 BUG，那么时间就更长了，开发人员本身受到的压力也是到了最大话了。客户->公司->测试小组->开发人员。 这个完全是倒金字塔型的，承受能力差的一环很容易出事情的。

另外开发人员的测试除了保证代码能正常运行以外，还有一个很重要的方面就是要保证上次能正常运行的代码，这次还是能正常运

行。如果做不到这点，那么 BUG 就不断的会出现，很多 BUG 也会反复出现。于是软件看上去就有修补不完的 BUG 了。如果出现这种情况，那么开发人员有必要再教育。一般公司教育的方式有四种。第一种，扣工资，第二种，加班，反复加班+精神攻击。 第三种，开除。第四种，调动人员来帮助那个出了麻烦的家伙。 但愿看这个文章的人不要受到前面三种教育。

# 加强测试用例在测试过程中的地位

## ——强化测试用例管理过程

Sincky Zhang[45]

sinckyzhang@hotmail.com

常闻测试人员如此抱怨：

测试用例在实际中没有起多大作用；

在实际测试时根本没有按用例执行；

测试执行后没有把新的用例补充到用例库中……

为什么会有这样的抱怨？

先说说当前我们的软件企业为何测试流程不规范：

从事物的发展规律看，软件测试行业在我国还是新兴行业，目前还处于起步和探索期，虽然国外的同行业发展到了一定阶段，但事实上他们也在不断否定自我并摸索着更直接有效的方法；而国内的测试行业发展期不足 10 年，所谓的测试管理流程不规范，也就情有可原了。

从企业个体角度讲，测试部门的整顿和加强，按照企业自身发展的优先层次，还没有被纳入优先解决的程度，开拓市场/签售定单才是首要问题，也是维系企业生存发展的命脉。当然国内很多优秀的大中型软件公司的测试部门相对完善，如神州数码/用友/金蝶等，他们和大型跨国软件公司的合作，也从中汲取了宝贵的管理经验。

还有一个普遍存在的问题，近几年国内软件企业为了加强企业的竞争优势和名气提升，通常大搞特搞 ISO/CMM 认证；笔者并不反对这么做，但通过这些认证后的企业有多少真正按照那些规定/设计的标准在后续的测试或软件开发管理工作中着手开展下去呢？社会上流传着这样的话：任何认证到中国，最后都免不了砸牌了！笔者读书时很多高校搞的 MCSE 认证，有培训机构明目张胆声称“百分百通过率”！当年也有专门媒体报道此事。听到这样的话，我们都会寒心，这里真

[45]更多测试资料，欢迎访问Sincky的软件测试专业blog：天行健－君子以自强不息！（http://blog.51testing.com/index.php?blogId=19）

心希望我们的软件企业通过 ISO/CMM 后真正为企业的内部软件开发流程带来一点曙光。

最后一个原因，我想是企业内部测试管理人员和技术人员技能的不足，还有自身工作态度的不够端正。有了再好的规范标准，没人遵守不行！没人实施不行！应该说，很多中小软件企业的高层都或多或少的逐渐意识到软件测试的重要性和必要性，以及它的标准化/流程化改革的紧迫性，但也有很多的工程师/技术人员并不理会这套，常常在实际工作中投机取巧；也有很多测试管理人员经验不足/技能不够，对公司测试管理工作考虑不到位，和开发工程师交流不充分，和上层领导反映不及时等等。

总之，任何问题的出现都不是单方面的原因，从宏观的社会形势到微观的企业个人，都有无可推卸的因素；正因为如此，解决问题也要对症下药，如何完善软件测试流程，就要从小处出发；本文不可能将软件测试流程改进的话题阐述的面面俱到，因此只谈测试用例的管理流程改进。

测试用例在实际中没有起多大作用，在实际测试时根本没有按用例执行，测试执行后没有把新的用例补充到用例库中…为何如此？我们分析认为，根本原因是测试流程不完善，针对测试用例的管理流程更不完善，从三个方面具体来说：

- 测试用例是软件测试工作执行环节的依据，如果这个依据都没用了，那是不是说这个依据不明确，依据设计的不合理呢？答案是肯定的！
- 制定了完备有效的测试用例，为什么不按测试用例执行测试呢？首先是因为企业没有严格和良好的机制促使测试执行者这样做，其实是个别测试人员投机取巧心理的表现。
- 测试用例设计得不可能天衣无缝，测试执行过程里肯定会发现有些测试路径或数据在用例里没有体现；那么事后该将其补充到用例库里，以方便他人和后续版本的测试；如果没有这样做，那么测试部门负责人和每个测试员都难辞其疚，是该重新坐下来思考一下公司的测试用例管理规范或流程了。

那么究竟如何做，才能尽量避免上述问题呢？我们不妨从需求阶段就把这些问题考虑进去，以便从初始就力争将问题缩到最小，以防

后期出现问题是互相推卸责任或干脆束手无策！

### 1. 软件需求分析阶段

我从来认为测试人员从软件生命周期的需求阶段就开始介入，因为测试人员的工作通常开展在该周期的末尾，如果前期不涉入，如何通晓整个系统的架构而对其测试呢？虽然该观点被大多数同行所认可，但我知道依然有很多公司为了节省费用，不让测试人员参与前期调研或制定需求，经常的做法是等到系统开发完毕或将近完成，跟测试经理说一声“这边有个项目，你找几个人来测一下吧！”经验表明，这样的做法实不可取。

测试人员在需求阶段的任务有：

- 全面了解系统需求，从客户角度考虑软件测试需要达到的验证值。
- 参与软件需求调研，以测试角度分析需求的可测性；对不可测问题与客户或项目经理协调解决。

如果企业采用类似与 rational requistepro 的需求管理工具，这个工作也需要测试人员的配合实施。

### 2. 软件分析设计阶段

测试人员除制定测试计划书等本职工作外，我认为还有一个必不可少的任务，那就是将系统的可测性落实到书面文档，以备将来编写测试用例。

之所以要这么做，是因为考虑到很多企业编写测试用例直接参考需求规格说明书或者分析流程图，这样跨度大，难度也大，是造成测试用例不完备、覆盖范围小的重要原因。

如果公司采用类似于 rational rose 的建模分析工具，这个工作更好执行；如果没有，那么测试人员更有必要编写一份《软件功能点测试描述书》，它是软件的详细测试分析文档，其主旨是将系统分析人员的分析文档加工成站在测试角度的功能点分析文档，重要的是描述对系统分解后每个功能点逐一的校验描述，包括何种方法测试 何种数据测试 期望测试结果等，这些信息都是描述性的，无须具体数据；它的作用是据此编写测试用例，以及测试执行时的参考依据，它直接来源于需求，覆盖最全，也最贴近客户。

当然该文档不是非要不可，我只是提倡一种原则，如果有类似的替代文档或有自动工具可实现此功能，则会倍加受推崇！

### 3. 软件开发阶段

编写测试用例。我不想从技术角度探讨到底如何编写功能强大质量优异的测试用例（可参见我在“天若有情”转载的这篇文章- 如何设计编写软件测试用例 ），这里只想从管理角度和大家谈谈如何有效控制测试用例的流程。应该遵守的原则是：

首先，从覆盖率来说，测试用例库的用例要达到最大覆盖软件系统的功能点。按照我上述所言的测试工程师从前期阶段顺次下来，编写测试用例时，基本就是将《软件功能点测试描述书》中的每个功能点进行操作上的细化：一是将从步骤上描述到达校验点的方式，二是从内容上以何种数据校验功能点。

其次，从数量来讲，我觉得很多公司的测试用例太少，甚至远远不能覆盖系统需求，这也是很多测试人员测试开展前期按照用例执行，渐渐凭“意念”去测试的原因。应该说测试用例的数量很难用数学模型来模拟，更没办法衡量，但凭借个人经验来说，一个多于半年开发周期（指从编码开始直到提交客户的时间段）的软件系统，它的用例数量不要低于 4000 个，甚至更多！也许有人惊讶这一数字，不过了解 IBM Microsoft 的人士会认为 4000 还是很少的。试想，对于一个中型软件系统，如果设计出 5000 个用例，那它的测试覆盖率还怕不高么！

再次，如此众多的测试用例的管理。是的，需要管理工具软件！本人从来都反对以 word 或 excel 来编写测试用例，那样不仅在格式上难于编写——尤其对于一些共性内容：测试目标、测试环境、参考说明等，每次都要拷贝；而且难于管理——几千个文档文件放在一个共享文件夹，你的眼睛要必将经过缭乱的洗礼！更是难于执行——莫非真要针对几千个用例都是打开一个 word、执行测试、输入测试结果、关闭 word？而且，根本没办法追踪测试结果——输入完本轮回归测试的结果，下一论输入哪里？输入了这些测试结果什么用？用它追踪什么？追踪得到吗？

使用word等软件编写测试用例的种种不便不多说，但换个思路思考一下使用集成工具的种种优势就一见分晓。测试同业者们都了解的

测试用例管理工具便是rational testmanager（点击http://www-900.ibm.com/cn/software/rational/products/testmanager/index.shtml了解其基本功能），其实还有很多国内外中小型工具，如微创的testcase manager（点击http://www.wicresoft.com/products/devmgmt/tcm/docs/whitepaper.pdf 下载工具白皮书）等，他们是专业的测试用例管理工具，其设计出发点就已经考虑到了我们上述的种种困境，因此给予了良好的解决方案。

而且，个人觉得测试行业的快速发展，必将带来从每个环节都逐渐向自动化和标准化方向迈进，尽早适应这一趋势，不仅提高了工作效率，也提高了企业的信誉和名誉。

最后，说一下测试用例格式上一般说明外的几个要点：

- 一是在测试管理工具中制定适合本公司的测试用例模版
- 二是模版有关键字索引，以方便按关键字分类查找，如测试方法（分手工/自动两种）
- 三是测试用例要有状态跟踪，如执行失败要链接到缺陷报告
- 四是测试用例的修改及运行都有日志记录

## 4. 软件测试阶段

测试负责人划分不同的测试阶段(如集成测试 系统测试 回归测试 性能测试等)，再划分不同的子测试周期（如前两个星期做冒烟测试，可手工，也可自动；接着做第一模块功能测试，顺次…），再为项目测试人员分配测试用例，测试人员则按照详细的用例文档去执行测试，这里要遵循的几个原则是：

A.有健全且严格的体制保证测试执行者严格按照测试用例执行测试。这并不妨碍测试者创造力的发挥，因为前期用例设计和编写就是项目全体测试人员智慧的结晶！我们一直追问众多测试工程师加班加点辛苦工作的原因，其实大都发生这一阶段，如此实施，即便没有解决根本问题，也会大大提高测试执行效率。

B.如有对用例认识模糊或遗漏的地方，必须经测试负责人或项目其他管理人员同意方可更新用例库。

C.测试负责人每日负责跟踪本测试子周期或阶段的测试用例执行情况，以及每日提交的缺陷报告，根据执行进展状态以及缺陷数量或严重等级和项目高层或其他人员交流，商议解决途径，并确定或调整未来时间的测试任务。

D.测试执行者负责执行自己区域的测试用例，也要负责跟踪该区域软件缺陷的修改进展，根据其状态不断验证软件功能点。

E.应该提及的是，大多数软件公司都采用集成的缺陷管理工具来管理软件缺陷，如 rational clearquest、mantis、bugzilla、TestTrack Pro 等，这是好事情，因为这些集成工具都提供了清晰的报告模版及优秀的追踪功能，测试团队的每一成员按照自己的角色和权限要不断追踪缺陷的状态。

F.对于自动测试（包括性能/压力测试），有一些特殊要点。本人的原则是自动化测试无须编写测试用例，故而到此阶段才提及自动化测试；只要在编写时将用例库里适合或需要自动测试的用例的测试方法（不同工具可能名称不同）设为自动即可。自动化测试的实施方案有所不同，每款测试工具的使用和流程也不同，但都是从在这一阶段编写测试脚本，并运行自动测试。针对自动化测试原则，可参阅我的自动化测试要点 ，这里要提的几个基本原则是：

- 一是选择恰当的测试工具品牌，并要求提供培训
- 二是项目测试成员有专人负责此事的进行，他们可不参与日常测试
- 三是确定自动化测试成员在项目中的角色，一般自动化测试成员隶属于项目测试负责人，负责人同样跟踪其工作状态
- 四是选择最简单 最重用的测试用例使用自动测试方法
- 五是使用工具厂商提供的测试框架编写脚本，千万别单纯的录制/加校验点/回放，以开发出健壮的且重用性强的测试脚本
- 六是有专人更新脚本，也有专人跟踪自动测试结果
- 七是一般选择的测试工具品牌和缺陷管理工具品牌是同一厂商，以方便不同类型缺陷的集中管理

G.由于不同公司开发产品的特殊性，也许需要特殊类型的测试，如安全测试，甚至代码级单元测试等，这些需要酌情考虑测试用例的编写，以及测试的执行。

**5. 软件验收阶段，除了提交软件测试评估报告**

（各种类型测试结果的评估都有报告）这些传统工作外，对于测试用例，此时要集中时间更新，更新整个测试周期中一切需要更新的内容，以方便未来新版本的测试，即便是项目软件——提交客户后没有新版本，那也需要后期维护，维护阶段需要重新测试某功能点，然而用例不准确，碰巧又是个新员工，那就死翘翘了！

退一步说，如果您公司的测试部门经历一次这样重大的洗礼，有一个项目真正按照此原则实施一次，也必将对未来取得事半功倍的效果。

**总结：**

综上所述，我们得出结论：

测试用例在测试中没起到应有的作用，是因为测试用例编写质量不高，覆盖不够，执行不利；

测试执行时不遵循测试用例，执行后不更新用例库，是测试部门的整体工作流程不健全不规范；

测试行业仍处在群雄逐鹿、百家争鸣的时期，芸芸纷说，不如从自身出发，确立最适合自我的解决方案，整顿自身的工作流程，那才是金玉良言的上上策！

# TestDirector 用户手册

作者：江永刚

【摘要】TestDirector 是 Mercury Interactive 公司推出的基于 WEB 的测试管理工具。它能够指导进行测试需求定义、测试计划、测试执行和缺陷跟踪，即整个测试过程的各个阶段。通过整合所有任务到软件测试中，来使整个测试管理工作更有效，并确保客户收到更高质量的产品。

【关键词】需求定义 测试计划 测试执行 缺陷跟踪 需求树 测试计划树 测试集

## 1. 欢迎使用 TestDirector

欢迎您使 Mercury Interactive 公司推出的基于 WEB 的测试管理工具——TestDirector。它能够帮助你组织和管理软件测试过程的每一个阶段，包括测试需求定义、测试计划、测试执行和缺陷跟踪。

### 1.1. 如何使用本手册

本手册描述了如何使用 TestDirector 来管理整个软件的测试过程。它包括如下七个部分：

**Part I　　TestDirector 概貌**

提供关于 TestDirector 系统特征和使用方法的总体描述。

**Part II　　需求定义**

描述如何通过构造需求树来定义测试需求。

**Part III　测试计划**

描述如何计划从构建测试计划树到创建测试的整个测试过程。

**Part IV　　测试执行**

描述如何创建测试集，执行手动测试和自动测试并且检查测试结果。

**Part V　　缺陷跟踪**

描述如何报告软件缺陷到 TestDirector 工程中并且跟踪缺陷的修复过程直到这个缺陷被解决。

Part VI　TestDirector 分析

描述如何通过创建报告、图表和工程文档来监控测试和缺陷跟踪过程。

Part VII　附录

描述如何使用 VAPI-XP 测试工具。

### 1.2. TestDirector 文档套件

除了本手册之外，TestDirector 还附带了如下已印刷的文档：

**TestDirector 安装手册**

描述如何去安装 TestDirector 和需要连接到 TestDirector 工程数据库的客户端数据库软件。

**TestDirector 指南**

一步一步的教你如何使用 TestDirector 来管理软件测试过程。

**TestDirector 管理员使用手册**

描述如何在工程自定义窗口中自定义工程和如何使用工程管理站点来创建和维护工程。

**TestDirector 开放测试架构手册**

描述如何使用 TestDirector 的开放测试架构来整合你自己的配置管理、缺陷跟踪和一些自主研发的测试工具。它还包括对 TestDirector 中新增基于 COM 的 API 的完整说明。

### 1.3. 在线资源

TestDirector 包括如下在线资源：

**自述**

提供了关于 TestDirector 的最新新闻和信息。

**TestDirector 新增内容**

描述了在 TestDirector 最新版本中所拥有的一些新功能。

**在线书籍**

展示了所有PDF格式的文档套件。所有的在线书籍能够通过 Adobe Acrobat Reader 5.0 来阅读或打印。Adobe Acrobat Reader 5.0 可以从 Adobe 的官方网站下载（http://www.adobe.com/products/acrobat/readstep2.html）

### 在线帮助

提供了在你使用TestDirector时碰到问题的快速响应，它们将以菜单和对话框的形式出现，并且向你展示如何完成TestDirector任务。察看 Mercury Interactive 公司的 Customer Support Web 网页（http://support.mercuryinteractive.com）来更新TestDirector的帮助目录。

### 在线技术支持

使用默认的浏览器登陆到Mercury Interactive公司的Customer Support Web网页（http://support.mercuryinteractive.com）。在这个网页能够使你经过授权的来浏览相关的知识点和增加你自己的文章，发布并且寻找用户讨论会议，提交需要帮助支持的信息，下载补丁，更新文档甚至更多的东西。

### 支持信息

介绍 Mercury Interactive 公司的网址和为用户提供支持的网址，Email 地址和其他的一些有用的信息，并且还列出了 Mercury Interactive 公司在全球范围内的所有的办公地点的所在地。

### Mercury Interactive 网站

你可以使用默认的浏览器来访问Mercury Interactive公司的主页，在这里提供了非常多的且经常被更新的有关Mercury Interactive的信息和介绍，这包括了新发布的软件信息，研究会和商业展示，用户支持，教育服务和其他一些更多的东西。Mercury Interactive公司的网址是http://www.mercuryinteractive.com。

# Part I TestDirector 概貌

## 2. 总体介绍

欢迎使用 TestDirector，它是 Mercury Interactive 公司推出的基于 WEB 的测试管理工具，无论是通过 Internet 还是 Intranet，你都可以以基于 Web 的方式来访问 TestDirector。

应用程序测试是非常复杂的，它需要开发和执行数以千计的测试用例。通常情况下，测试需要多样式的硬件平台、多重的配置（计算机，操作系统，浏览器）和多种的应用程序版本。管理整个测试过程中的各个部分是非常耗时和困难的。

TestDirector 能够让你系统地控制整个测试过程，并创建整个测试工作流的框架和基础，使整个测试管理过程变得更为简单和有组

织。

TestDirector 能够帮助你维护一个测试工程数据库，并且能够覆盖你的应用程序功能性的各个方面。在你的工程中的每一个测试点都对应着一个指定的测试需求。To meet the various goals of a project, you organize the tests in your project into unique groups. TestDirector 还为你提供了直观和有效的方式来计划和执行测试集、收集测试结果并分析数据。

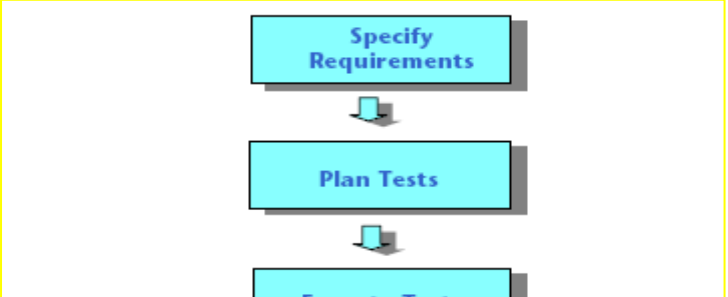
TestDirector 还专门提供了一个完善的缺陷跟踪系统，它能够让你跟踪缺陷从产生到最终解决的全过程。TestDirector 通过与你的邮件系统相关联，缺陷跟踪的相关信息就可以被整个应用开发组，QA，客户支持，负责信息系统的人员所共享。

TestDirector 提供了与 Mercury Interactive 公司的测试工具（WinRunner, LoadRunner, QuickTest Professional, Astra QuickTest, QuickTest Professional for MySAP.com Windows Client, Astra LoadTest, XRunner, Visual API and Visual API-XP）、第三方或者自主开发的测试工具、需求和配置管理工具、建模工具的整合功能。TestDirector 能够与这些测试工具很好的无缝链接，为你提供的全套解决方案选择来进行全部自动化的应用测试。

TestDirector 会指导你进行需求定义、测试计划、测试执行和缺陷跟踪，即整个测试过程的各个阶段。通过整合所有的任务到应用程序测试中来确保你的客户收到更高质量的产品。

## 2.1. 测试管理过程

TestDirector 的测试管理包括如下四个阶段：



**需求定义（Specify Requirements）：**

分析应用程序并确定测试需求。

**测试计划（Plan Tests）：**

基于测试需求，建立测试计划。

**测试执行（Execute Tests）：**

创建测试集（Test Set）并执行测试。

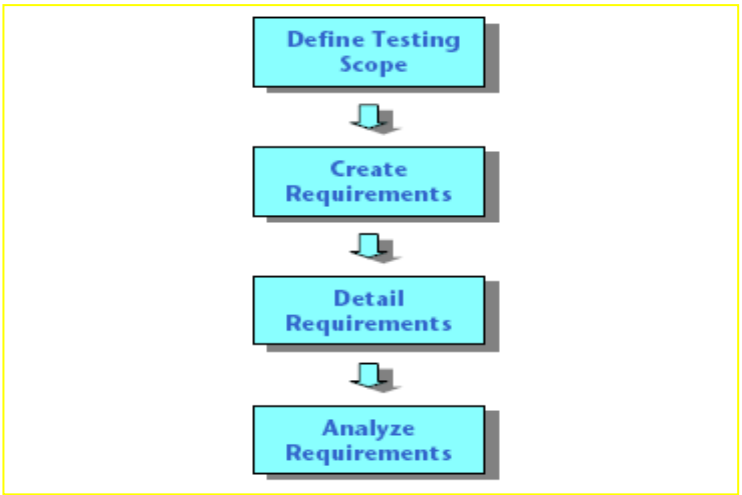
**缺陷跟踪（Track Defects）：**

报告程序中产生的缺陷并跟踪缺陷修复的全过程。

贯穿测试的每一个阶段，你能够通过产生详细的报告和图标对数据进行分析。

## 2.2. 需求定义

分析应用程序并确定测试需求。




**定义测试范围（Define Testing Scope）：**

检查应用程序文档，并确定测试范围——测试目的、目标和策略。

**创建需求（Create Requirements）：**

创建需求树（Requirements Tree），并确定它涵盖所有的测试需求。

**描述需求（Detail Requirements）：**

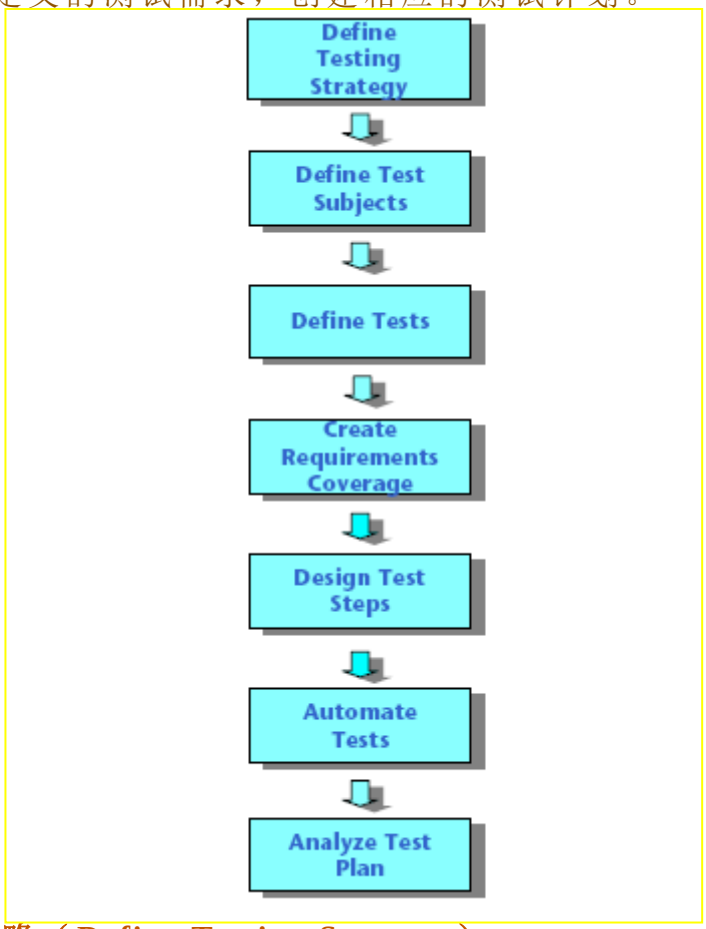
为“需求树”中的每一个需求主题建立了一个详细的目录，并描述每一个需求，给它分配一个优先级，如有必要的话还可以加上附件。

**分析需求（Analyze Requirements）：**

产生报告和图表来帮助你分析测试需求，并检查需求以确保它们在你的测试范围内。

## 2.3. 测试计划

基于已定义的测试需求，创建相应的测试计划。




**定义测试策略（Define Testing Strategy）：**

检查应用程序、系统环境和测试资源，并确认测试目标。

**定义测试主题（Define Test Subject）：**

将应用程序基于模块和功能进行划分，并对应到各个测试单元或主题，构建测试计划树（Test Plan Tree）。

**定义测试（Define Tests）：**

定义每个模块的测试类型，并为每一个测试添加基本的说明。

**创建需求覆盖（Create Requirements Coverage）：**

将每一个测试与测试需求进行连接。

**设计测试步骤（Design Test Steps）：**

对于每一个测试，先决定其要进行的测试类型（手动测试和自动测试），若准备进行手动测试，需要为其在测试计划树上添加相应的测试步骤（Test Steps）。测试步骤描述测试的详细操作、检查点和每个测试的预期结果。

**自动测试（Automate Tests）：**

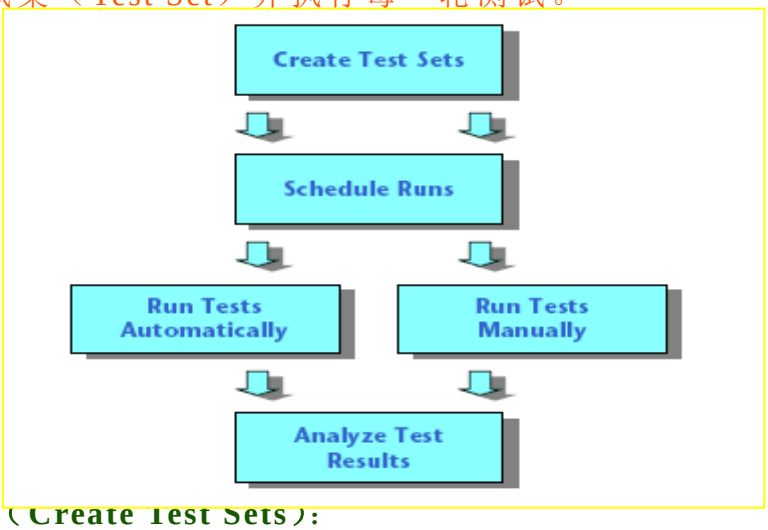
对于要进行自动测试的部分，应该利用 MI、自己或第三方的测试工具来创建测试脚本。

**分析测试计划（Analyze Test Plan）：**

产生报告和图表来帮助你分析测试计划数据，并检查所有测试以确保它们满足你的测试目标。

## 2.4. 测试执行

创建测试集（Test Set）并执行每一轮测试。




**创建测试集（Create Test Sets）：**

在你的工程中定义不同的测试组来达到各种不同的测试目标，他们可能包括，举个例子，在一个应用程序中测试一个新的应用版本或是一个特殊的功能。并确定每个测试集都包括了哪些测试。

**确定进度表（Schedule Runs）：**

为测试执行制定时间表，并为测试员分配任务。

**运行测试（Run Tests）：**

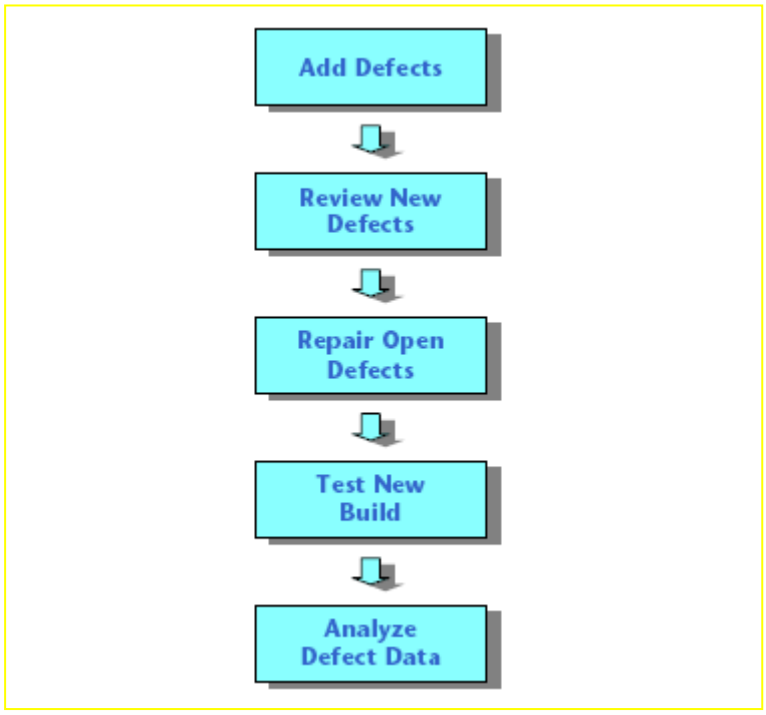
自动或手动执行每一个测试集。

**分析测试结果（Analyze Test Results）：**

查看测试结果并确保应用程序缺陷已经被发现。生成的报告和图表可以帮助你分析这些结果。

## 2.5.缺陷跟踪

报告程序中产生的缺陷并跟踪缺陷修复的全过程。




**添加缺陷（Add Defects）：**

报告程序测试中发现的新的缺陷。在测试过程中的任何阶段，质量保证人员、开发者、项目经理和最终用户都能添加缺陷。

**检查新缺陷（Review New Defects）：**

检查新的缺陷，并确定哪些缺陷应该被修复。

**修复打开的缺陷（Repair Open Defects）：**

修复那些你决定要修复的缺陷。

**测试新构建（Test New Build）：**

测试应用程序的新构建，重复上面的过程，直到缺陷被修复。

**分析缺陷数据（Analyze Defect Data）：**

产生报告和图表来帮助你分析缺陷修复过程，并帮助你决定什么

时候发布该产品。

## 2.6. 使用工程数据库

当你创建一个 TestDirector 工程后，你需要存储和管理 TestDirector 自身产生和连接的数据库。每一个工程都支持通过数据库来存储工程信息。

TestDirector 是一个知识库，它存储着需求、测试、测试集、测试个案（Test Run）、工程文档和定制信息。为了应用程序测试工程能够正常工作，TestDirector 需要持续不断地访问这些数据。

可以使用下面的数据库应用软件来存储和管理 TestDirector 信息：

- Microsoft Access
- Sybase （仅适用于TestDirector企业版）
- Microsoft SQL （仅适用于TestDirector企业版）
- Oracle（仅适用于TestDirector企业版）

关于创建和管理 TestDirector 的更进一步信息，请参考《TestDirector 管理员手册》。

## 2.7. 用户权限

TestDirector 能够让你对用户访问工程的权限进行管理。通过创建一个授权的用户列表，为每个用户分配一个密码，并将其分配到相应的用户组中，从而控制每个用户对工程访问的权限。在 TestDirector 中用户所拥有的权利是由该用户所在的用户组决定的。TestDirector 具有特定的权限和许可机制，能够让你为工程中指定的字段创建访问规则。

关于 TestDirector 中的用户组、口令分配和权限的更详细的信息，请参考《TestDirector 管理员手册》。

## 2.8. 从 Word 中导入文档信息

你能够将已存在的 Word 格式的需求或测试文档中的内容，直接导入到需求树或测试计划树中。若想使用 Word 导入功能，必须先下载 Microsoft Word 插件。

**下载 Microsoft Word 插件：**

1）在TestDirector的窗口选项中，点击**Add-ins Page**链接。“TestDirector Add-ins”页被打开。

2）点击**More** TestDirector **Add-ins**链接，“More TestDirector Add-ins”页被打开。

3）在**Microsoft Add-ins**下，点击**Microsoft Word**链接，“Microsoft Word Add-in”页被打开。

4）点击Microsoft **Word Add-in Readme**链接，查看Microsoft Word 插件使用说明。

5）点击Download **Add-in** 链接，开始安装Microsoft Word插件。

关于 TestDirector 插件的更进一步信息，请参考《TestDirector安装手册》。

## 2.9.从 Excel 中导入文档信息

你能够将已存在的 Excel 格式的需求或测试文档中的内容，直接导入到需求树或测试计划树中。若想使用 Excel 导入功能，必须先下载 Microsoft Excel 插件。

**下载 Microsoft Excel 插件：**

1）在TestDirector的窗口选项中，点击Add-ins Page链接。“TestDirector Add-ins”页被打开。

2）点击More TestDirector Add-ins链接，“More TestDirector Add-ins”页被打开。

3）在Microsoft Add-ins下，点击Microsoft Excel链接，“Microsoft Excel Add-in”页被打开。

4）点击Microsoft Excel Add-in Readme链接，查看Microsoft Excel插件使用说明。

5）点击Download Add-in 链接，开始安装Microsoft Word插件。

关于 TestDirector 插件的更进一步信息，请参考《TestDirector安装手册》。

# 3. 开始使用

本章对 TestDirector 进行粗略介绍，并解释它是如何开始工作的。包括如下几个部分内容：

- 启动TestDirector（Starting TestDirector）

- TestDirector窗口（The TestDirector Window）
- TestDirector工具条（The TestDirector Toolbar）
- 修改密码（Changing Passwords）
- 修改用户属性（Changing User Properties）
- 清除历史记录（Clearing History）

## 4.1. 启动TestDirector

你可以通过你工作站上 WEB 浏览器启动 TestDirector。

1）打开 Web 浏览器

并输入 TestDirector 所在的 URL（http://[Server name]/[virtual Directory name]/default.htm），TestDirector 的首页将被打开。若不知道正确的路径，请与系统管理员联系。

TestDirector 选项窗口被打开。


**注意：**如果你不能启动你的 TestDirector，请联系系统管理员来确定 TestDirector 是否已经被安装到了公司的 Web 服务器上。更进一步信息，请参考《TestDirector 安装手册》。

2）点击 TestDirector 链接。

在你第一次运行 TestDirector 时候，TestDirector 组建将会被下载到你的计算机上，随后 TestDirector 会自动进行版本检查，若发现存在新的版本，它将会帮你下载新的版本。一旦 TestDirector 进行完版本检查和更新（假如需要的话），TestDirector 的登陆页面将被显示。

**注意：**关于运行 TestDirector 时下载组件到计算机的更进一步信息，请参考《TestDirector 安装手册》。

3）在 Domain 列表中选择你准备进入的域。

你可以选择名为 DEFAULT 的默认域。若不知道具体应该选择哪个域，请与 TestDirector 管理员联系。

**注意：DEFAULT** 域仅在 TestDirector 的标准版中才有效。

4）在工程列表中选择一个工程。

假如工程列表是空的，请查阅 TestDirector 的知识库（http://support.mercuryinteractive.com）并搜索关键字“empty project list”。

若 TestDirector 的示例工程已经被安装在 TestDirector 的服务端，你则可以选择名为 TestDirector_Demo 的工程（确信你在 Domain 列表中已经选择了 DEFAULT 域）。

此工程会为你介绍 TestDirector，包括需求、测试、测试集、Test Runs 以及缺陷。更进一步信息，请参考《TestDirector 指南》。

5）在 User ID 框中，选择或输入你的用户名称。

若不清楚你的用户名，请与系统管理员联系。

注意：User ID 列表信息是与客户端本身所在的机器有关的，故你在（某台机器上）第一次登陆 TestDirector 时，应该输入你的用户名。

6）在 Password 框中，输入管理员指派给你的密码。

（若是第一次以 Admin 的身份登陆，你不需要输入密码，此时密码为空）,若需要对密码进行修改，请查看第 19 页的“修改密码”。

7）点击 Login 按钮。

TestDirector 会打开在你上一次运行 TestDirector 任务时所用过的那个模块（需求、测试计划、测试实验室和缺陷）。若想定制模块名称，请查看《TestDirector 安装手册》。

8）对于退出和返回到 TestDirector 登陆窗口，请点击在右上角的 LOGOUT 按钮。

## 4.2. TestDirector窗口

当你打开一个工程时，TestDirector 的主窗口会打开你上次工作时使用过的模块。在标题栏，TestDirector 会显示工程名称和你的用户名。


TestDirector 包含如下几个模块：

| | |
|---|---|
| 需　　求（Requirements） | 定义测试需求。<br><br>包括定义你正在测试的内容、定义需求的主题和条目并分析这些需求。 |
| 测试计划<br><br>（Test Plan） | 开发一个测试计划。<br><br>包括定义测试目标和策略、将测试计划分为不同的类别、对测试进行定义和开发、定义哪些需要自动化测试、将测试与需求进行连接和分析测试计划。 |
| 测试实验室<br><br>（Test Lab） | 运行测试并分析测试结果。 |
| 缺陷（Defects） | 增加新缺陷、确定缺陷修复属性、修复打开的缺陷和分析缺陷数据。 |

**技巧：**你可以在两个模块间利用快捷键进行切换。用 **Ctrl＋Shift**

＋1 来访问需求模块，用 Ctrl＋Shift＋2 来访问测试计划模块，如此类推。

所有的 TestDirector 模块都包括如下内容：

| | |
|---|---|
| TestDirector 工具栏（TestDirector Toolbar） | 位于 TestDirector 工程名的紧上面。<br><br>假如此工具栏不可见，请点击 Show Toolbar 按钮。关于 TestDirector 工具栏的更多信息，请查看第 18 页的“TestDirector 工具栏”。 |
| 菜单栏（Menu Bar） | 位于 TestDirector 工程名的紧下面。<br><br>菜单名称随你选择的模块名称不同而改变。 |
| 模块工具栏（Module Toolbar） | 位于菜单栏下面。<br><br>包括当前所使用 TestDirector 模块中经常使用到的命令。 |
| TOOLS 工具按钮（**Tools** Button） | 位于窗口的右上角。<br><br>能够让你改变用户密码和另外的一些用户属性、change the language direction for a user in a project from left to right or right to left、清楚历史数据、查看每一个 TestDirector 客户端组件的版本信息或打开文档引擎。<br><br>关于文档引擎的更进一步信息，请查看第 28 章“产生工程文档”（Generating Project Documents）。<br><br>若想定制工具菜单，请查看《TestDirector 安装手册》。 |
| HELP 帮助按钮（Help Button） | 位于窗口的右上角。<br><br>能够通过它访问 TestDirector 的在线资源。<br><br>若想定制帮助菜单，请查看《TestDirector 安装手册》。 |

## 4.3.TestDirector工具栏

公用的 TestDirector 工具栏对所有的 TestDirector 模块都是适用的。包含如下的一些按钮：

| | | |
|---|---|---|
| 导航按钮 |  返回（Back） | 返回到先前 TestDirector 所在的位置。 |
| |  前进（Forward） | 假如你已经使用了返回的导航按钮，你可以使用前进按钮返回回来。 |
| |  首页（Home） | 登出并且进入 TestDirector 登陆窗口。 |
| 拼写按钮 |  拼写检查（Check Spelling ） | 为所选中的单词或文本框作拼写检查。<br><br>假如不存在错误，一个确认的消息将被弹出。假如错误被发现，将会弹出对话框显示相应的提示信息。 |
| |  拼写选项（Spelling Options） | 打开拼写选项对话框，并能够让你对 TestDirector 的拼写检查执行方式进行配置。 |
| |  辞典（Thesaurus） | 打开辞典对话框，并显示所选中单词的同义、近义或反义词。你能够替换掉所选择的词或查找新的词。 |
| 缺陷按钮 |  添加缺陷（Add Defect） | 打开添加缺陷对话框，并能够让你添加一个新的缺陷。<br><br>关于更进一步的信息，请查看第 25 章“添加和跟踪缺陷”（Adding and Tracking Defects）。 |
| 帮助按钮 |  帮助按钮（Help Button） | 打开在线帮助并为当前的内容显示帮助主题。 |

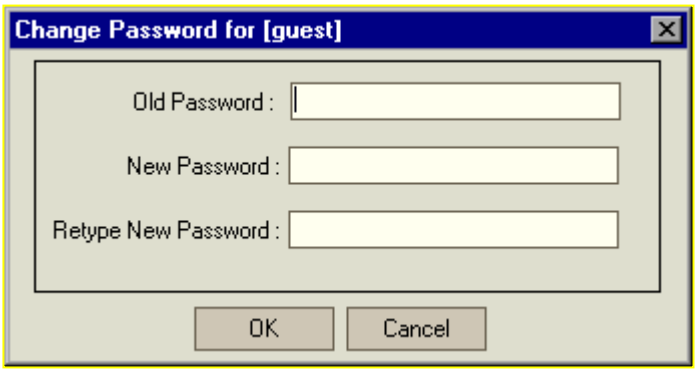
## 4.4. 修改密码

你能够修改访问 TestDirector 工程的密码。

**注意：**管理员能够改变并覆盖用户密码。更进一步信息，请查看《TestDirector 管理员手册》。

1) 在窗口右上角，点击 **Tools** 按钮并选择 **Change Password** 菜

单项。或者在工程定制窗口点击 **Change Password** 链接。修改用户密码的对话框将被弹出。



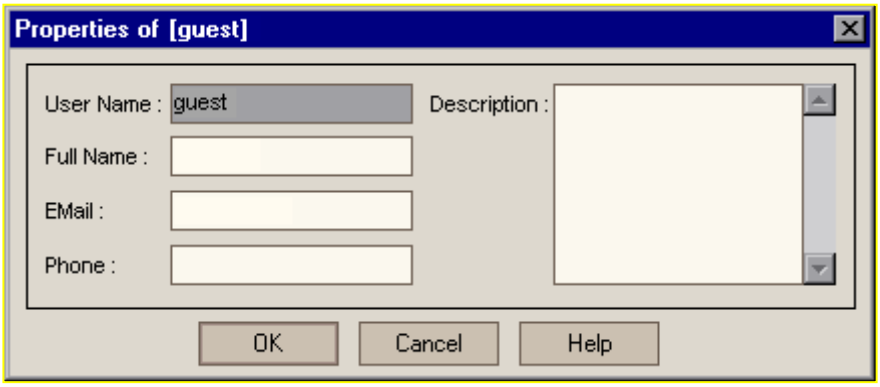

2) 在 **Old Password** 框中输入你的旧密码。
3) 在 **New Password** 框中输入你的新密码。
4) 在 **Retype New** Password 框中重新输入你的新密码。
5) 点击 **OK**，关闭修改密码对话框。

### 4.5. 修改用户属性

你能够修改你的用户属性，包括全名、Email 地址、电话号码和描述信息。注意，Email 地址信息是非常重要的，因为能够直接通过你的邮箱，让你接收到缺陷、需求和测试集的信息。

**注意：**管理员能够改变并覆盖用户属性信息。更进一步信息，请查看《TestDirector 管理员手册》。

1) 在窗口右上角，点击 **Tools** 按钮并选择 **Change User Properties** 菜单项。或者在工程定制窗口点击 **Change User Properties** 链接。用户属性对话框将被弹出。




2) 编辑如下的用户属性：**Full Name**、**Email**、**Phone**、**Description**。
3) 点击 **OK** 按钮，保存你的修改。

## 4.6. 清除历史记录

在自定义 TestDirector 工程时，你可以要求 TestDirector 来保存系统中的日志信息，以及在需求、测试和缺陷实体中的用户字段。产生的历史记录数据会被显示在需求、测试计划和缺陷模块的历史记录属性页上面。对于更多关于为 TestDirector 域设置历史记录的信息，请查看《TestDirector 管理员手册》（《TestDirector Administrator's Guide》）。

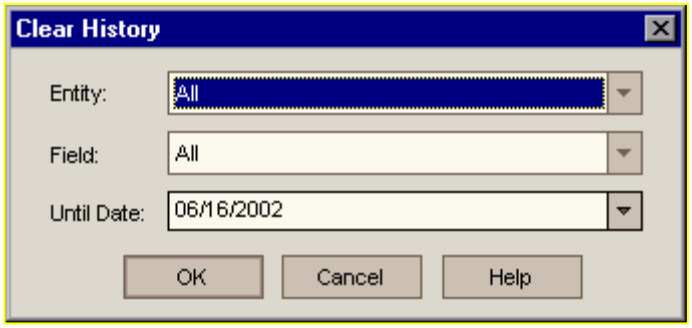
一旦你不想存储历史数据，TestDirector 允许你将这些历史数据从 TestDirector 工程中删除。举个例子，假如你已经成功地运行了你创建的测试集，你可能想从 TestDirector 工程中清除这些历史记录。

你能够清除所有的历史记录，或指定实体或域的历史记录。另外，你能够让 TestDirector 仅删除直到某一天（包括这一天）的历史记录。TestDirector 所清除的历史记录显示在各自模块的 History 属性页下。

**注意：**默认状态下，只要具有管理员权限的用户才能够清除历史记录。用户权限是能够被定制的。对于关于权限的更进一步信息，请查看《TestDirector 管理员手册》。

**清除历史记录：**

1) 在窗口右上角，点击 **Tools** 按钮并选择 **Clear History** 菜单项。清除历史记录对话框将被弹出。




2) 在 **Entity** 框中，选择你准备删除历史记录所属的实体。若你准备删除需求、测试和缺陷实体的历史记录，请选择 **All**。
3) 在 **Field** 框中，选择你准备删除的历史记录所在的字段，若想删除历史记录的所有字段，请选择 **All**。
4) 在 **Until Date** 框中，选择一个日期。TestDirector 所删除直到

所选择日期的历史记录（包括所选择日期当天）。

5) 点击 **OK**。

## 4. 使用TestDirector数据

利用 TestDirector 网格和树，你能够查看和修改你工程中的数据。本章描述如下几个部分内容：

- 组织列（Arranging Columns）
- 过滤记录（Filtering Records）
- 高级/交叉过滤记录（Advanced/Cross Filtering Records）
- 记录分类（Sorting Records）
- 刷新并清除设置（Refreshing and Clearing Setting）
- 将数据保存到文件中（Saving Data to a File）

### 4.1. 关于使用TestDirector数据

TestDirector 利用网格或树组织并显示数据。

| 数/网格 | 描述 |
|---|---|
| 需求树（Requirements Tree） | 适用于需求模块。为 TestDirector 工程显示测试需求。<br><br>更进一步信息，请看第八章“开发需求树”（Developing the Requirement Tree）。 |
| 测试计划树（Test Plan Tree） | 适用于测试计划模块。在 TestDirector 工程中显示测试和对应的组。<br><br>更进一步信息，请看第十一章“开发测试计划树”（Developing the Test Plan Tree）。 |
| 测试网格（Test Grid） | 适用于测试计划模块，选择 **View > Test Grid** 时。在 TestDirector 工程中显示所有的测试。<br><br>更进一步信息，请看第十章“测试计划模块一览”（The Test Plan Module at a Glance）。 |
| 设计步骤网格（Design Steps Grid） | 适用于测试计划模块。显示测试的步骤。<br><br>更进一步信息，请看第 127 页的“构造测试”（Building Test）。 |

| | |
|---|---|
| 测试集树（Test Sets Tree） | 适用于测试实验室模块。在 TestDirector 工程中显示测试集——一组测试，运行它们能够达到指定的测试目标。<br><br>更进一步信息，请看第十八章“创建测试集”（Creating Test Sets）。 |
| 执行网格（Execution Grid） | 适用于测试实验室模块。显示测试集中的所有测试。<br><br>更进一步信息，请看第十七章“测试实验室模块一览”（The Test Lab Module at a Glance）。 |
| 缺陷网格（Defects Grid） | 适用于缺陷模块。在 TestDirector 工程中显示测缺陷。<br><br>更进一步信息，请看第二十五章“添加并跟踪缺陷”（Adding and Tracking Defects）。 |

当你利用 TestDirector 网格和树进行工作时，你能够对列进行排列、根据条件过滤记录、设置分类属性、刷新清除过滤和分类设置、保存数据到文件。注意：当 TestDirector 网格和树所显示内容的类型发生变化时，本章中的描述不总是完全适用。

注意：你能够按照你自己的喜好保存你的网格设置，比如分类和过滤。更进一步信息，请看第五章“使用喜好视图”。

## 4.2. 组织列

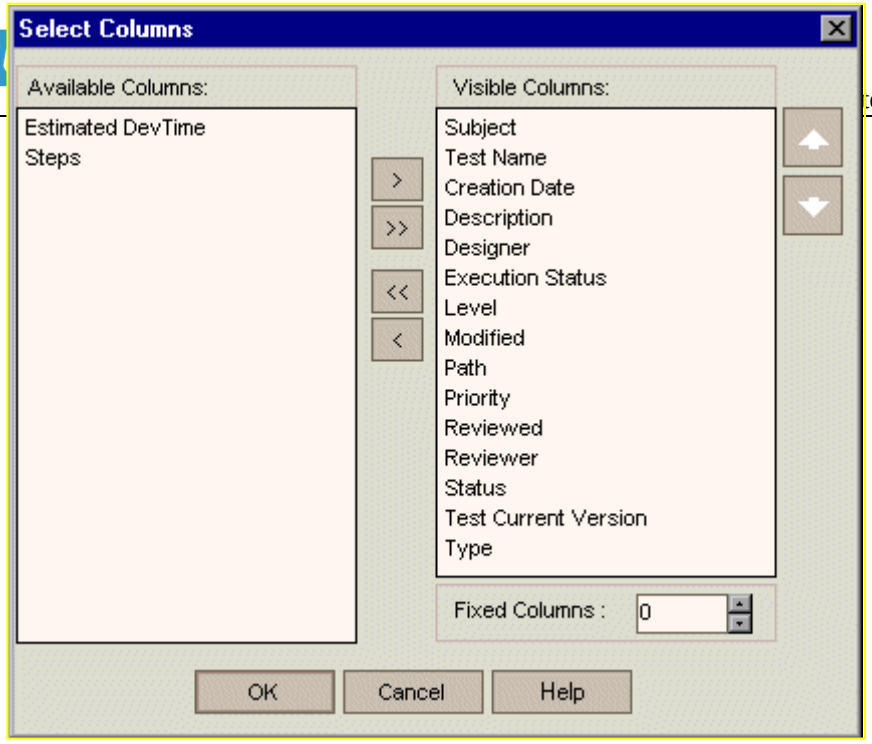
你能够自定义每一列显示内容的顺序并且可以对每一列的长度进行调整。对列设置的修改将会在下次启动时仍然有效。

### 设置列顺序（Setting Column Order）

使用选择列对话框，你能够决定哪些列显示在 TestDirector 中，并决定所显示列的顺序。比如说，在 Test Grid 中你可以选择 Subject 作为第一列。

**决定列的显示和顺序：**

1) 点击 **Select Columns** 按钮  ，Select Columns 对话框将被弹出。




Available Columns 列表框中显示当前没有被显示的列。

Visible Columns 列表框中显示了当前正在显示的列。

2) 选择列名称并点击箭头按钮（<和>），将它们在 Available Columns 和 Visible Columns 列表框间移动。点击双向箭头按钮（<<和>>），将所有的列从一个列表框移动到另一个列表框。注意，你也可以点击列表名在两个列表框间进行拖动。

3) 在 **Visible Columns** 列表框中，你可以通过 Up 和 Down 箭头  ，挑战列显示的顺序。注意，你也可以通过上下拖动列名称来调整它们的顺序。

4) 设置非滚动列（Non-scrolling Columns）。通过在 **Fixed Columns** 框中设置你想要的非滚动列的数量，可以从最左边开始的这些数量的列设置为非滚动列。当你水平拖动滚动滑块时，非滚动列的位置是保持不变的，并且以阴影显示。（注意，此功能在需求模块中是无效的）

5) 点击 **OK** 按钮，关闭对话框并应用新的列顺序。

**调整列宽度：**

你能够用鼠标调整每一列的尺寸。点击在列表头的右边界，通过拖动去调整列的宽度。注意，你仅仅只能够调整没有固定的列，即没有设置为非滚动列的列。

## 4.3. 过滤记录

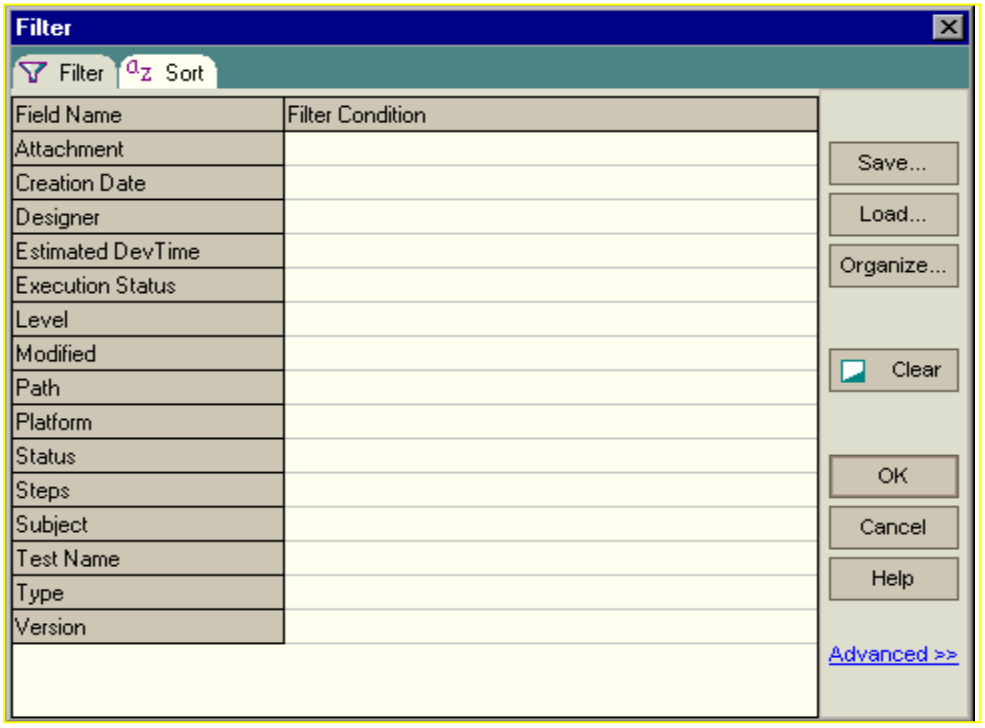
你能够过滤 TestDirector 数据，仅仅只按你定义的标准进行显

示。你能为过滤条件指派一个简单的项（比如“Failed”），或一个合理的表达（比如“Passed Or Failed”）。只有当记录满足所有的过滤标准时，才会显示在 TestDirector 网格或树中。

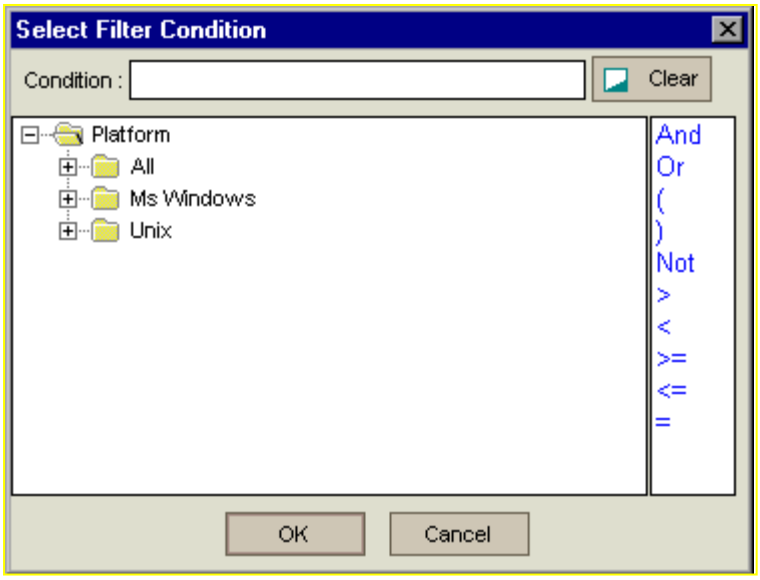
你也能够指定多个过滤条件。举个例子，你能够指定 Status 的过滤条件为“Failed”，为 Tester 指定过滤条件为“David Or Mark”。TestDirector 将仅仅只显示由 David 或 Mark 操作的，并且失败的测试。

**定义一个过滤：**

1) 点击 Set Filter/Sort 按钮  。过滤对话框将被弹出，并显示 Filter 属性页。



2) 点击相应的 **Filter Condition** 输入框，为指定的列设置过滤条件。点击 **Browse** 按钮，**Select Filter Condition** 对话框将被弹出。




3) 定义过滤条件。假如列表可用的话，从列表中选择项。你也能够增加一些操作，从而创建一个合理的表达式。

**注意：**在定义过滤条件时，如下内容应该被考虑：

- 显示在分级列表中的有些项包含子列表。这些项是被一个文件夹包括在一起。双击文件夹，并点击所要选择的项，就能够从子列表中选择一个项。
- 当为一个用户定义条件时，你能够指定当前用户（选择【CurrentUser】）或整个用户组（如：【Developer】）。
- 假如你输入的某个项超过一个单词，必须用一个引号在包含它们。举个例子，若搜索 **Login Boundary** 测试，在条件输入框中应该输入：**“Login Boundary”**。
- 假如你只想输入某个项的一部分，你可以用星号（*）。举个例子，若想在所有测试中搜索包含 Login 单词的测试，在输入框中输入：***Login***。
- 若想从所有的测试中搜索由 Insert New 开头的测试，在输入框中输入：**“insert new*”**。

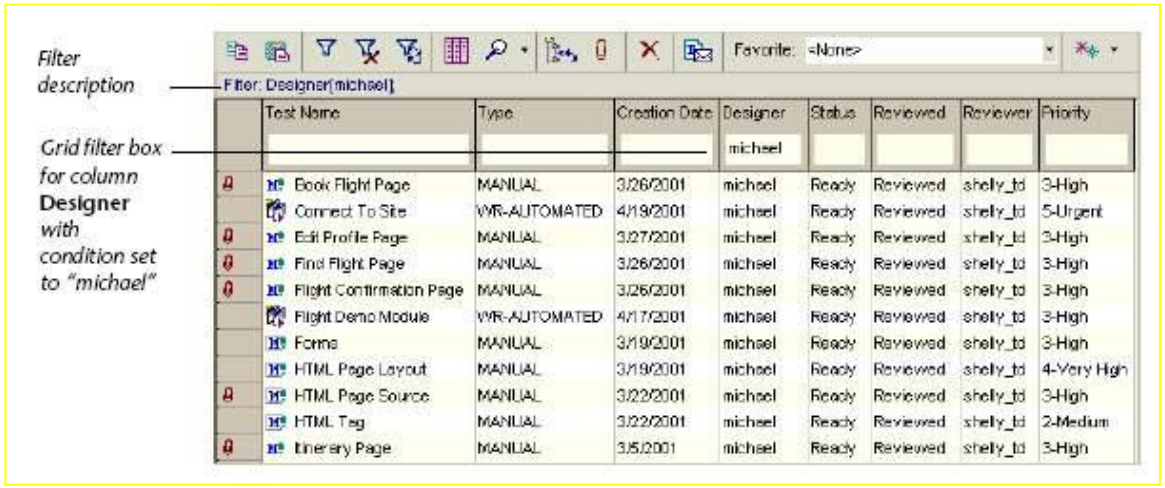
4) 点击 OK 去关闭 Select Filter Condition 对话框。

5) 若想添加交叉过滤条件，点击 Advanced 链接。对于更进一步的信息，请看第 30 页的“高级/交叉过滤记录”

（Advanced/Cross Filtering Records）。

6) 点击 OK 去关闭过滤对话框。

TestDirector 应用这些过滤条件并显示过滤描述。对于一个网格，TestDirector 也将在显示列名称下面的网格过滤框（Grid Filter Box）中显示过滤条件。




**技巧：**对于一个网格，你也可以通过网格过滤框（Grid Filter Box）定义过滤条件。若想显示网格过滤框（Grid Filter Box），请选择 **View>Grid Filters**。假如过滤框为空，则说明过滤条件对此项不适用。你可以直接在过滤框中输入过滤条件或点击过滤框，并点击显示出来的 **Browse** 按钮，在打开的 **Select Filter Condition** 对话框中输入过滤条件。

**注意：**假如你正工作在测试计划树或测试集列表下，你能够保存过滤或分类信息并重新加载你所需要的树或列表。点击 Save 按钮去保存一个过滤设置、点击 Load 按钮去加载一个先前保存的过滤设置、点击 Organize 按钮去重命名，另存或删除过滤设置。假如你正工作在 TestDirector 的任何其它区域，可以使用第五章“使用喜好视图”，来保存作为自己喜爱视图的过滤信息。

## 4.4. 高级/交叉过滤记录

当你在定义过滤条件时，你也能够包括一个 cross filter——关于关联项高级的第二次过滤，如关联的需求、测试、测试集或缺陷。举个例子，在测试计划树中，你能够定义状态为“Open”的测试集作为交叉过滤条件。另外，你可能有一些别的过滤条件，但 TestDirector 仅仅只会显示处于打开状态测试集的测试。

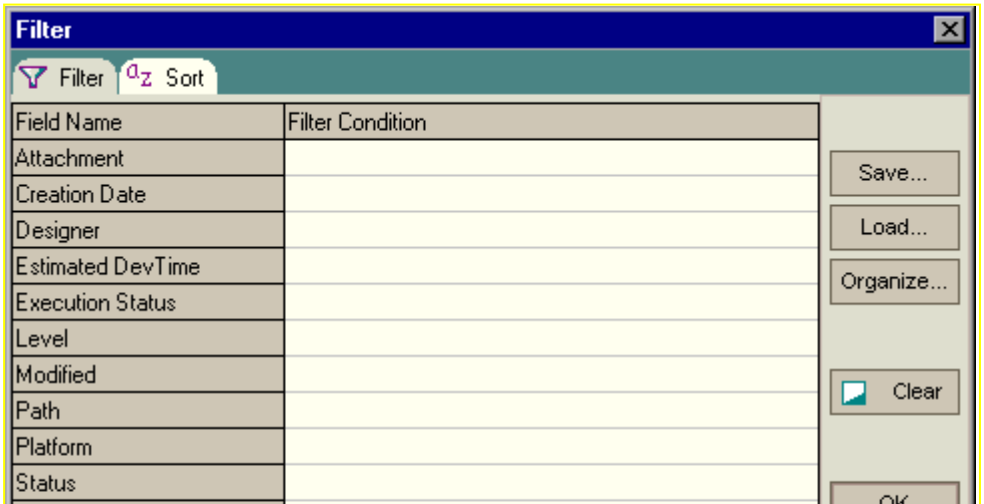
下面的表格简要介绍了交叉过滤（Cross Filter）在各个

TestDirector 模块中的有效性。

| 模块（Module） | 交叉过滤（Cross Filter） |
|---|---|
| 需求（Requirements） | ● 关联缺陷：通过缺陷和相关的测试覆盖过滤需求。<br>● 关联测试：通过测试覆盖过滤需求 |
| 测试计划（Test Plan） | ● 关联测试集：通过包含测试的测试集来过滤测试。<br>● 关联需求：通过需求覆盖来过滤测试。<br>● 关联缺陷：通过相关联的缺陷来过滤测试。 |
| 测试实验室——测试集列表（Test Lab-Test Sets List） | ● 关联缺陷：通过缺陷过滤测试集。<br>● 关联测试：通过包含的测试过滤测试集。 |
| 测试实验室——执行网格（Test Lab-Execution Grid） | ● 关联需求：通过覆盖了所选择的需求的测试来过滤测试实例。<br>● 关联缺陷：通过相关联的缺陷来过滤测试。 |
| 缺陷（Defects） | ● 关联测试集：通过相关联的测试集来过滤缺陷<br>● 关联需求：通过覆盖了所选择的需求的测试来过滤缺陷。<br>● 关联测试：通过相关联的测试来过滤缺陷。 |

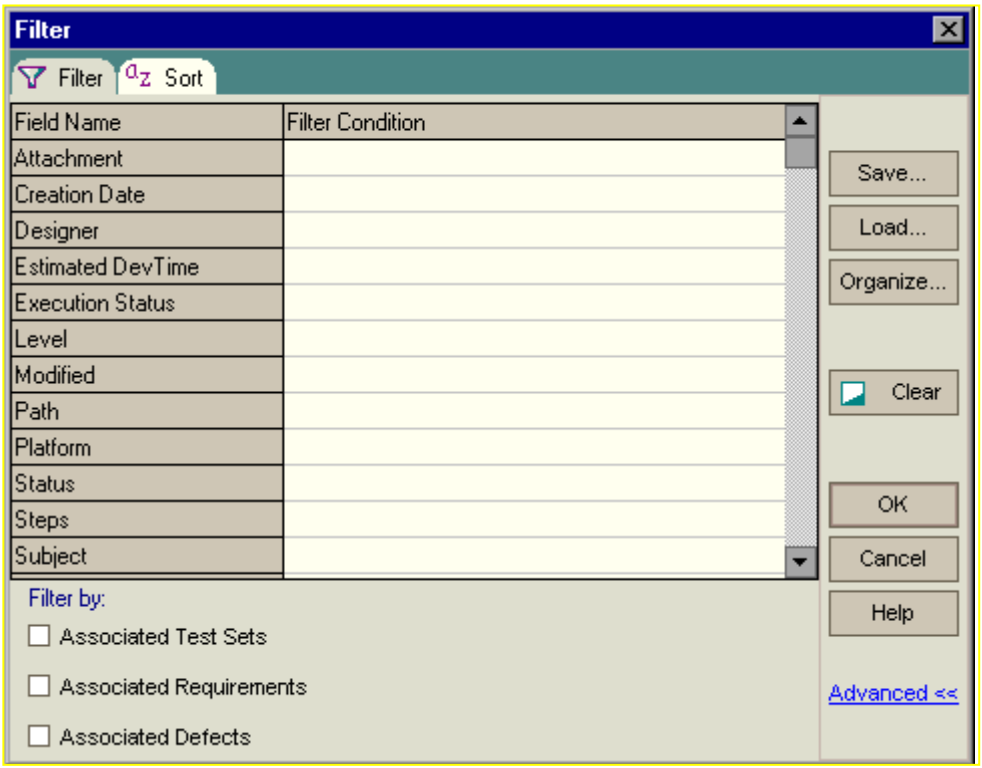
**定义交叉过滤：**

1) 点击 **Set Filter/Sort** 按钮  。过滤对话框被弹出，并显示过滤属性页。



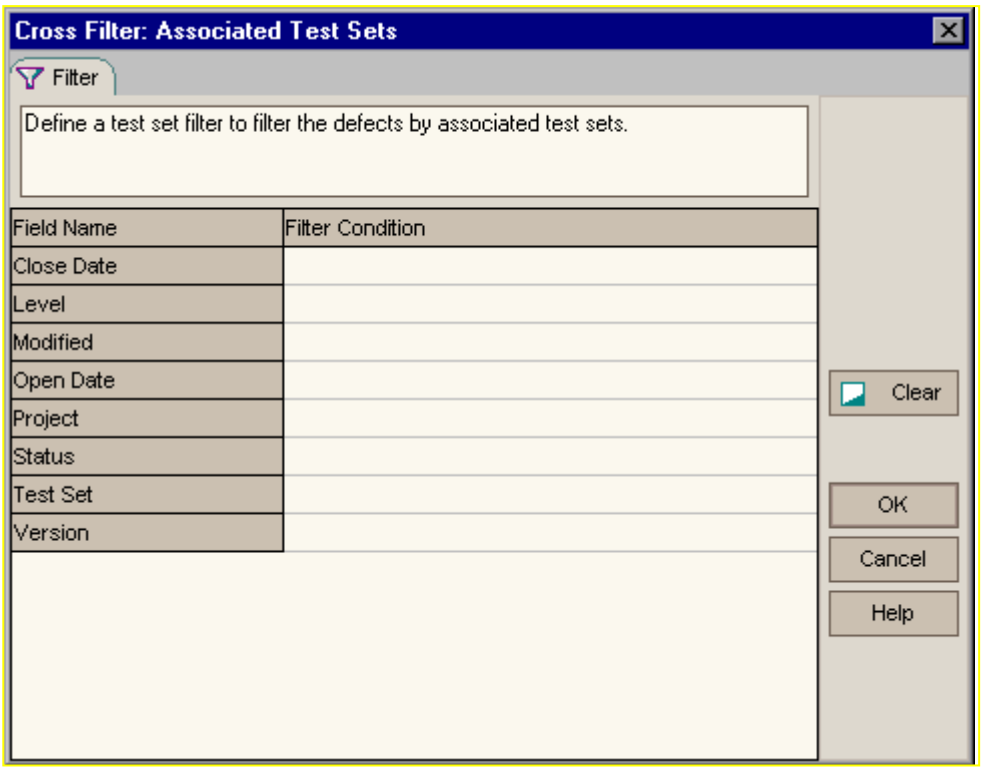
关于定义过滤条件的更进一步信息，请查看第 27 页的“过滤记录”（Filtering Records）。

2) 点击 **Advanced** 链接去显示 **Cross Filter** 选项。




3) 在 **Filter by** 下面，选择一个高级过滤的复选框。例如：选择“Associated Test Sets”复选框，然后点击相应的链接，Cross

Filter：Associated【Filter】对话框将被弹出。




4) 用你准备过滤规则定义一个交叉过滤。对于更详细信息，请查看 27 页的“过滤记录”（Filtering Records）。

5) 点击 OK 按钮去保存你的改变并关闭 Cross Filter 对话框。

6) 点击 OK 按钮去保存你的改变并关闭 Filter 对话框。

## 4.5. 记录分类

默认情况下，记录是以它们被添加的顺序进行显示的。当你设置记录的分类属性后，它们的显示顺序根据 ASCII 分类顺序（ASCII Sort Order）而定。ASCII 分类顺序首先会认为以字符或空格开始的记录先于以数字开始的记录，接着会考虑大写字符，最终考虑小写字符。

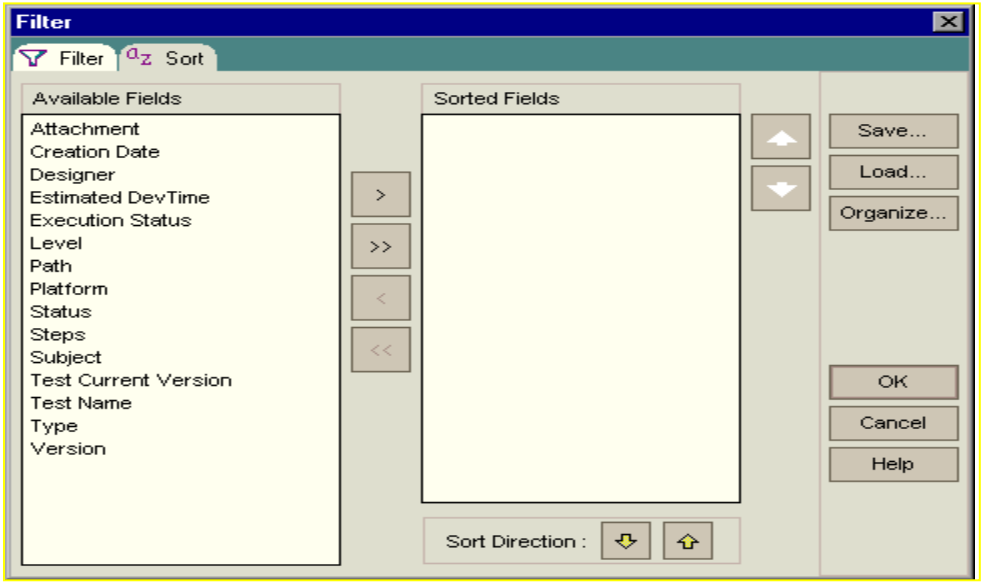
举个例子，假如在测试网格中的 Tester 列被标识为最高的分类优先级，记录将以显示在 Tester 列的名称根据 ASCII 分类顺序进行分类。假如 Test Name 被标识为次优先级，则先按 Tester 列的名称进行分类，对于同名的 Tester 列，再按 Test Name 列进行分类。

注意：默认情况下，记录是按等级顺序定义在测试计划树中，只有当记录定义了根据主题进行分类后，记录才会按字母顺序进行排列。

对记录进行分类（To sort records）：

1) 点击 Set Filter/Sort 按钮，过滤对话框将被弹出。

2) 点击 Sort 属性页标签。




**Available Fields** 中包含了所有能够显示在列表中的域名称。

**Sorted Fields** 中包含了当前已经标识了分类优先级的域名称。

3) 选择一个域名称并点击 Arrow 按钮(<和>),将它们在 **Available Fields** 和 **Visible Fields** 间移动。点击双向箭头按钮(<<和>>),将所有的名称从一个列表框移动到另一个列表框。注意，你也可以点击列表名在两个列表框间进行拖动。

4) 在 **Sorted Fields** 中，使用向上和向下箭头，设置域名称的显示顺序。注意，你也可以直接向上或向下拖动域名称。

5) 在 **Sorted Fields** 中，选择一个域名称并点击 **Sort Direction** 按钮，从而设置此域是以升序还是降序显示。

6) 点击 **OK** 去应用分类顺序设置。

**注意：**假如你正工作在测试计划树或测试集列表下，你能够保存过滤或分类信息并重新加载你所需要的树或列表。点击 Save 按钮去保存一个过滤设置、点击 Load 按钮去加载一个先前保存的过滤设置、点击 Organize 按钮去重命名，另存或删除过滤设置。假如你正工作在 TestDirector 的任何其它区域，可以使用第五章“使用喜好视图”，来保存作为自己喜爱视图的过滤信息。

### 4.6. 刷新并清除设置

你能够刷新清除 TestDirector 数据的过滤和分类设置。

-  点击 Set Filter/Sort 箭头并选择 Refresh，或点击 Refresh Filter/Sort 按钮来刷新在 TestDirector 网格或树中的数据。
-  点击 Set Filter/Sort 箭头并选择 Clear，或点击 Clear Filter/Sort 按钮来清除在 TestDirector 网格或树所有的过滤和分类优先级设置。

### 4.7. 将数据保存到文件中

你能够将网格中的内容保存为 Text 文件、Microsoft Excel 电子表格、Microsoft Word 文档、或 HTML 文档。

**保存数据到文件中：**

1) 在网格上点击右键，并在右键菜单上选择 **Save As**。
2) 选择一个文件类型：**Text File**、**Excel Sheet**、**Word Document** 或 **HTML**，保存网格结果对话框将被弹出。
3) 在 **Save in** 框中，选择此文件保存的路径。
4) 在 **File Name** 框中，输入此文件名称。
5) 点击 **Save** 按钮。

## 5. 添加附件

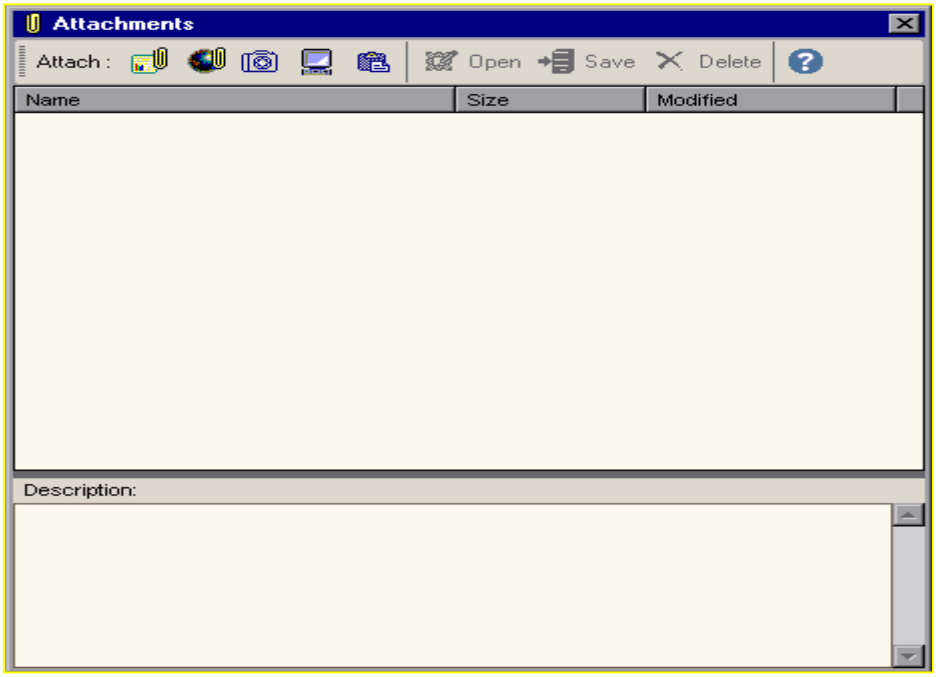
你能为需求、主题文件夹、测试、测试步骤、测试集、测试运行或缺陷添加附件。本章描述如下几个部分内容：

- 贴附文件（Attaching a File）
- 贴附URL（Attaching a URL）
- 贴附快照（Attaching a Snapshot）
- 贴附系统信息（Attaching System Properties）
- 贴附剪贴板图像（Attaching an Image from the Clipboard）
- 管理附件（Managing Attachments）

### 5.1. 关于添加附件

遍及整个测试过程，你都可以添加附件来更好的阐明你的工作，这些附件可以是文件、URL、应用程序快照、从剪贴板拷贝的图像或

系统信息。你是通过 Attachments 对话框来管理附件的。




下面的表格描述了在 TestDirector 的各个模块，怎样去打开 Attachments 对话框。

| 添加附件到…… | 所在模块 | 处理过程 |
|---|---|---|
| 需求 | 需求 | 从需求树上选择一个需求，并点击 Attachments 按钮或选择 View>Attachments。 |
| 测试或主题 | 测试计划——测试计划树视图 | 从测试计划树上选择一个主题文件夹或测试，点击 Attachments 标签页。 |
| 设计步骤 | 测试计划——测试计划树视图 | 从测试计划树上选择一个测试，点击 Design Steps 标签页。在此标签页上选择一个步骤，并点击 Attachments 按钮。 |
| 测试 | 测试计划——测试网格视图 | 从测试网格上选择一个测试，并点击 Attachments 按钮。 |
| 测试集 | 测试实验室 | 选择一个测试集，并点击 Test Set Properties 标签页，然后点击 Attachments |

| | | |
|---|---|---|
| | | 链接。 |
| 测试步骤 | 测试实验室——手动测试运行 | 在一个手动测试运行期间，点击 Exec Steps 按钮。选择其中一个步骤并点击 Attachments 按钮 |
| 测试运行 | 测试实验室——执行网格或执行流图 | 在 Test Run Properties 对话框中，点击 Attachments 标签页。 |
| 缺陷 | 缺陷 | 从缺陷网格上选择一个缺陷，并点击 Attachments 按钮。 |

## 5.2.贴附文件

TestDirector 能够让你去贴附文件。

1）在 Attachments 对话框，点击 File 按钮  ，Open 对话框将被弹出。

2）选择一个文件名并点击 Open。

文件名称、文件尺寸和修改日期会连同一起显示在附件列表中，与文件程序相关联的图标显示在文件名称前面。

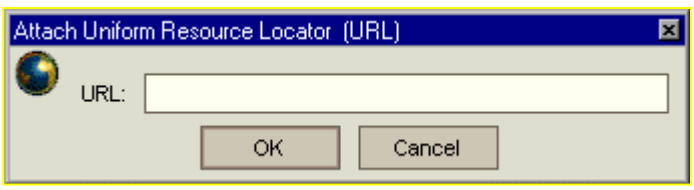
3）在 Description 栏中输入任何与所附文件相关的信息。

## 5.3.贴附URL

TestDirector 能够让你去贴附 URL。能够是任何有效的 URL，如：HTTP、FTP、Gopher、News、Mailto、File 等。

**贴附一个 URL：**

1）在 Attachments 对话框，点击 URL 按钮  ，贴附 URL 的对话框将被打开。




2）在 URL 对话框中输入一个有效的 URL，并点击 OK。

此 URL 将显示在附件列表中，系统默认的 Web 浏览器图标显示在 URL 前面。

3) 在 Description 栏中输入任何与所附 URL 相关的信息。

## 5.4.贴附快照

TestDirector 能够让你去贴附你应用程序的图像。

1) 在 Attachments 对话框，点击 Snapshot 按钮  ，Snapshot 的对话框将被打开。


2) 拖动 Camera 图标到你想要捕获的对象上。捕获的图像将被显示在 Snapshot 对话框中。

3) 你能够改变你所捕获图像的缩放级别。

- 点击 Zoom Out 或 Zoom In 去放大或缩小图片。
- 点击 Normal 返回到原始状态。

4) 点击 Attach。

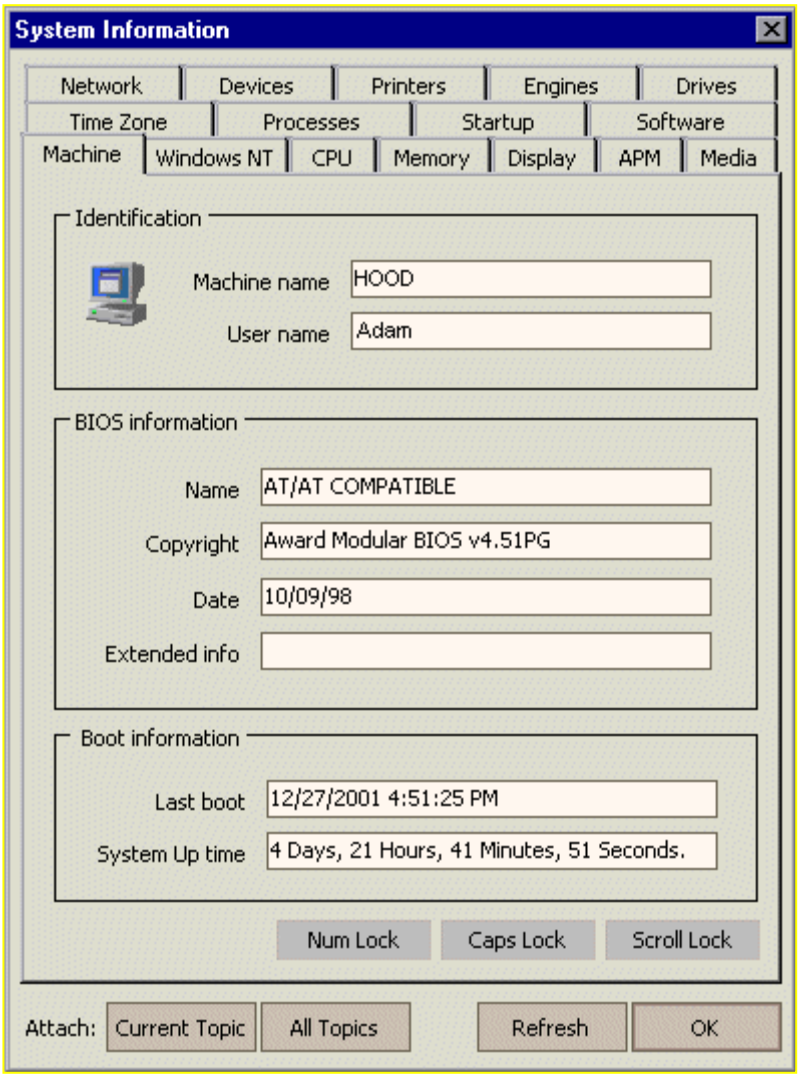
TestDirector 会为此图像统一分配一个文件名，且扩展名为.jpg。文件名显示在附件列表中，且图像图标显示在文件名旁边。文件的尺寸和修改日期也同样显示在附件列表中。

5) 在 Description 栏中输入任何与所附快照的相关信息。

## 5.5.贴附系统信息

TestDirector 能够让你贴附计算机系统的信息。

1) 在 Attachments 对话框，点击 System Info 按钮，System Information 对话框将被打开。



2）若仅想贴附单个标签页上的信息，请点击此标签页并点击 Current Topic 按钮。

3）若想贴附所有标签页的信息，请点击 All Topic 按钮。

TestDirector 会为此信息统一分配一个扩展名为.tsi 的文件名。文件名显示在附件列表中，且图标显示在文件名旁边。文件的尺寸和修改日期也同样显示在附件列表中。

4) 在 Description 栏中输入任何与所附文本文件相关的信息。

## 5.6.贴附剪贴板图像

你能够将拷贝到剪贴板中的图片贴附到 TestDirector 中。

1）拷贝图像到剪贴板中。

2）在 Attachments 对话框，点击 System Info 按钮  。

TestDirector 会为此信息统一分配一个扩展名为.jpg 的文件名。文件名显示在附件列表中，且图标显示在文件名旁边。文件的尺寸和修改日期也同样显示在附件列表中。

## 5.7.管理附件

你能够通过 Attachments 对话框查看、修改和删除附件。

**查看附件：**

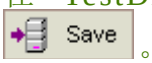
1) 在 **Attachments** 列表中，选择一个附件。附件相应的描述信息将显示在下面的 **Description** 框中。

2) 双击此附件并点击 **Open** 按钮 

。附件将在相应的应用程序中打开。如，URL 将在系统默认的 Web 浏览器中打开。

**修改附件：**

注意：当你在相应的应用程序中打开附件的时候，TestDirector 是拷贝附件到你客户端机器的本地目录。在对此附件作修改后，你需要对它进行保存两次。首先是通过打开它的应用程序对本地拷贝进行保存，然后点击 TestDirector 中的 Save 按钮将本地拷贝保存到 TestDirector 工程中。

1) 在附件列表中双击一个附件，此附件的一个本地拷贝将在相应的应用程序中打开。
2) 修改此附件。
3) 在打开的应用程序中保存此附件。注意这保存的只是你附件的本地拷贝。
4) 关闭附件。
5) 在 TestDirector 的 **Attachments** 对话框中点击 **Save** 按钮 

。此次是将附件的本地拷贝保存到 TestDirector 工程中。

**删除附件：**

1) 在 **Attachments** 列表中，选择你准备删除的附件。可以利用 **Ctrl** 键一个选择多个附件。
2) 2. 点击 Delete Selected 按钮 

，并点击 Yes 确认。

## 6. 使用喜好视图

喜好视图（Favorite View）是按照你的设置执行的 TestDirector 窗口视图。TestDirector 允许你保存喜好，并可以在需要的时候重新加载它们。本章将描述如下内容：

- 添加喜好视图（Adding Favorite Views）
- 组织喜好视图（Organizing Favorite Views）

### 6.1. 关于使用喜好视图

你能够通过选择某种设置来决定 TestDirector 窗口的容貌。能够保存测试网格、执行网格、缺陷网格、所有的报告和图表、以及文档引擎的 Favorite Views。这些设置可能包括为网格列应用一个过滤、在报告中对域进行分类或设置一个图像的外观。你能够为了以后的使用而保存一个 Favorite View，并且可以加载它在以后任何适当的时候。

可以在公共文件夹或私有文件夹保存 Favorite View。在公共文件夹保存的视图可以被所有用户访问。在私有文件夹保存的视图仅能被创建者访问。

从 Favorite 列表中选择一个 Favorite View，并将其加载到 TestDirector 窗口中。

**注意**：在执行网格中，与别的地方而言，对 Favorite View 的操作有一点细微的差别。取代 **Favorite** 按钮的是，你使用菜单栏命令：选择 **View > Favorites** 去显示 Favorite View 命令，选择 **Add** 去增加一个新的视图，选择 **View** 去加载一个存在的视图，选择 **Organize** 去组织你的视图。

部分 Favorite View 命令仅仅对某些用户组有用。用户组的权限是由系统管理员决定的。更详细信息，请查看《TestDirector 管理员手册》。

## 6.2. 添加喜好视图

你能够添加视图到喜好视图列表中。

**添加一个视图：**

1) 点击 **Favorites** 箭头  ，并选择 **Add to Favorites**，Add Favorite 对话框将被弹出。


2) 在 **Name** 框中，输入一个视图名称。
3) 在 **Location** 单选框内：
   - 选择 **Private** 去增加视图名到你的私有文件夹。在此文件夹内的 Favorite Views 仅仅只能够被你访问。
   - 选择 **Public** 去增加视图名到一般文件夹。在此文件夹内的 Favorite Views 能够被所有用户访问。

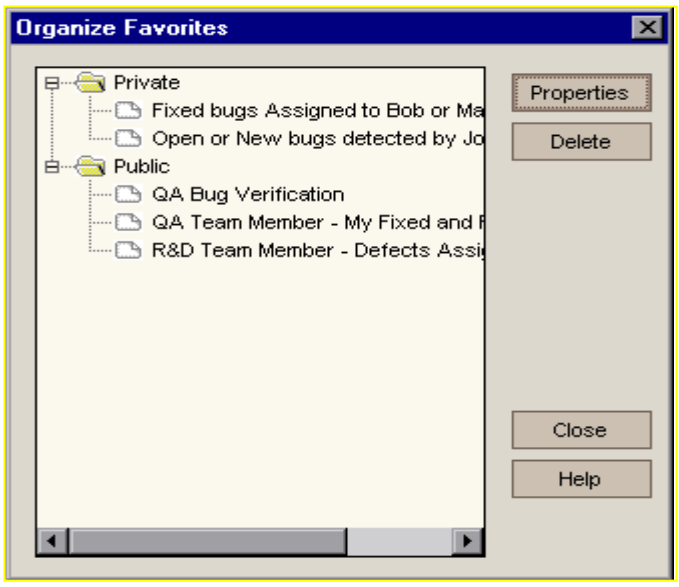
那些已经存在的喜好被显示在 **Existing Favorite** 框内。

4) 点击 **OK** 按钮，新的视图名称被添加到 Favorite 列表。

## 6.3. 组织喜好视图

你能够通过删除视图和改变视图属性来组织喜好视图列表。

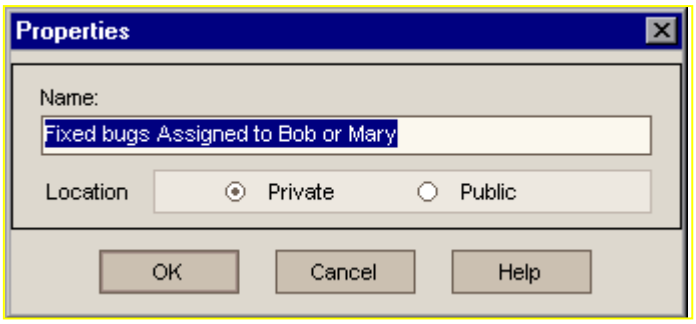
点击 Favorites 按钮  并选择 Organize Favorites，来打开 Organize Favorites 对话框。




**修改视图属性：**

你能够对显示在喜好视图列表中视图的属性进行修改。

1) 在 **Organize Favorites** 对话框中，选择一个视图并点击 **Properties** 按钮。Properties 对话框将被弹出。




2) 在 **Name** 框中，为视图输入一个新的名称。
3) 在 **Location** 单选框内：
   - 选择 **Private** 去增加视图名到你的私有文件夹。在此文件夹内的喜好视图仅仅只能够被你访问。
   - 选择 **Public** 去增加视图名到一般文件夹。在此文件夹内的喜好视图能够被所有用户访问。
4) 点击 **OK**。
5) 点击 **OK** 去关闭 **Organize Favorites** 对话框。

### 6.4. 删除视图：

你能够删除喜好视图列表中的视图。

1) 在 **Organize Favorites** 对话框的视图列表上，选择一个视图名称并点击 **Delete**。
2) 点击 **OK** 确认。
3) 点击 **OK**，关闭 **Organize Favorites** 对话框。