目录

浅谈软件企业的 QTP 自动化测试

作者：茹炳晟

QTP 是 HP Quick Test Professional 的简称，主要用于 Windows 应用软件和 Web 应用程序的自动化功能测试。该工具是目前软件自动化测试领域的佼佼者，占有商业市场的绝对主导地位，所以这方面的教材和网络教程都非常多，所以本文不打算涉及与讲解 QTP 的基础知识和使用，而是通过笔者所主持的多个企业级大型 QTP 自动化测试项目（项目开发周期三年以上，自动化测试与开发周期一年半以上），并结合以下几个问题的角度来探讨 QTP 的实际应用经验与最佳实践：

- 什么样的项目（产品）适合用 QTP 实现自动化测试？
- 自动化测试工程师在项目的什么阶段介入最合适？
- 企业的测试能力成熟度可以分为哪几个阶段？QTP 在其中承担什么样的角色？
- 自动化测试团队组织结构的特点有哪些？
- 如何以相对小的开发成本实现 QTP 对于非标准化控件的识别与操作？

一、什么样的项目适合用 QTP 实现自动化测试

简单来说，两句话就可以说明第一个问题，第一，产品或长期项目适合于 QTP 自动化测试，短期项目或一次性项目不适合于 QTP 自动化测试；第二，界面为主要交互方式的项目（产品）适合于用 QTP，基于底层服务或协议的项目（产品）不适合于用 QTP；另外，对于同一产品有多个客户定制版，并且各个定制版本功能基本相同的场合也是非常适合于做自动化测试的，因为这种情况下，自动化脚本的利用率很高。

二、自动化测试工程师在项目的什么阶段介入最合适

理想情况下，自动化测试工程师应该在项目开发的初期就介入到整个过程中。但是任何项目开发的初期都不适合大规模展开 QTP 脚本的开发，因为在这个阶段产品的界面设计还没有稳定，会导致 QTP 脚本的频繁变更，增大不必要的工作量，还会造成测试人员的抵触情绪，不利于自动化测试工作的顺利开展。

但是，这并不是说在项目开发的初期，自动化测试工程师可以不参与到整个项目中。这个阶段，自动化测试工程师应该研究调查 QTP 对于所采用的开发技术的可用性，也就是必须对待开发系统的 QTP 可测试性进行调查，同时自动化测试工程师在这个阶段应该很好地学习被测系统的业务知识。期间主要考虑以下几个方面的问题：

a. QTP 对被测系统控件的支持程度与稳定性

b. 若需要进行 QTP 二次开发，二次开发的难度与时间成本

c. 拟使用的自动化策略与框架

d. 考虑 QTP 上层的平台级测试框架（比如 STAF/STAX 等）的选型与应用策略

e. 分析测试需求，导出自动化测试需求，分析项目各个阶段适合于自动化测试的用例

f. 理解用户需求，充分学习被测系统的业务知识

只有很好地思考关注了以上问题，QTP 自动化测试才有可能发挥出真正的优势。

三、企业测试能力成熟度可以分为哪几个阶段

以笔者的经验来看，企业的测试能力成熟度(基于 QTP 来讨论)可以大致分为以下几个阶段：

阶段 1. 纯手工测试阶段

阶段 2. 在手动测试的基础上，开发适用于具体项目的简单测试辅助工具

阶段 3. 引入 QTP，对于回归度较高的测试用例进行简单的录制与回放

阶段 4. 基于 QTP 使用基于数据驱动的自动化测试

阶段 5. 基于 QTP，对于测试过程中的基本操作进行抽象，产生“原子”脚本与“分子”脚本，并且对对象库 Object Repository（OR）进行统一集中管理与维护

阶段 6. 抽象测试过程中的非业务逻辑(比如测试日志记录，测试结果统计，邮件通知等等)，形成各类 Re-use 脚本，初步形成测试框架

阶段 7. QTP 自动化测试框架的开发与初步应用

阶段 8. 在具体项目中全面推广基于 QTP 的自动化测试框架，严格区分业务脚本和非业务脚本，测试用例的组织基于业务的原子/分子的脚本调用，实现原子/分子脚本开发与测试用例设计的分离

阶段 9. 基于企业产品类型，形成最适合于产品的高层次的平台级自动化测试框架，并不断改进

笔者认为，单纯了解这些阶段的划分对于指导实际工作并没有太大的意义，核心问题是如何缩短从一个测试能力成熟度阶段向更高层次的阶段过渡的周期，加速推进企业测试能力成熟度的上升才是关键。所以我们有必要讨论这些阶段的特征以及从一个阶段上升到另一个阶段所进行的主要活动。这里笔者会根据自己的实际工作经验介绍一些推进企业测试能力成熟度的有效方法。

阶段 1 和阶段 2 属于传统测试的范畴，这里就不展开了。阶段 3 实际上是 QTP 引入的初级阶段，或者说是试用阶段，在这个阶段测试工程师对于 QTP 的使用都基本属于学习探索性阶段，对 QTP 的应用还是主要集中在简单的录制与回放，对 VBScript 的测试脚本处于学习阶段，测试用例相对零散，没有统一的设计与规划。在这个阶段，测试团队对于新技术的学习能力以及团队中是否有相对资深的 QTP 测试工程师起到了决定性的作用，所以如果公司的策略已经决定要发展 QTP 自动化测试，那么这个时候引入资深的自动化测试工程师将会事半功倍。实践表明，这个阶段的内部技术培训可以大幅度提高团队的 QTP 应用水平。

随着测试工程师对于 QTP 学习和使用的深入，数据驱动的测试用例将被逐渐采用，这一方面提高了部分测试用例的效率，另一方面也增强了测试人员对于 QTP 应用的信心。但是随着项目的推进，自动化测试用例的数量越来越多，当被测系统发生变更时，自动化测试脚本的维护工作量也随之增大，往往一个小小的界面或设计变更会导致大量的测试脚本的修改，这个阶段的主要矛盾将集中在如何更好的组织自动化测试脚本结构，提高脚本的可重用性与脚本使用的灵活性。最佳实践是对于测试过程中的基本操作进行抽象，产生“原子”脚本与“分子”脚本，在实际的测试用例中调用这些可重用的“原子”脚本或“分子”脚本，这样可以大幅度提高脚本的重用性，并且可以很好的应对变更。

为了帮助更好地理解“原子”脚本与“分子”脚本的设计思想以及应用场景，这里举个简单的例子，假设我们有以下两个测试需求，一个是登录一个 Web 系统，然后通过输入条件来查询数据，最后退出系统。另一个是登录同一个 Web 系统，然后插入某条数据，最后退出系统。简单的录制/回放就是“一气呵成”，

把整个操作过程录制成两份脚本。但是当用户登录界面变更的时候，两份脚本中关于登录的操作都要修改。当用户的测试用例非常多的时候，显然这个改动的工作量是相当大的。这个时候我们就可以采用“原子”脚本的思想，把登录操作，查询操作，插入操作和退出操作分别设计成四个“原子”脚本，其中登录操作脚本带有两个输入参数,分别是用户名和密码；查询操作脚本以查询条件作为输入参数；插入操作脚本以插入的数据作为输入参数；退出脚本不带参数。然后对于第一个测试需求，我们的测试用例只需依次调用登录脚本，查询脚本和退出脚本；对于第二个测试需求，只需依次调用登录脚本，插入脚本和退出脚本（这里同样可以采用数据驱动的参数化方法）。当用户登录界面变更的时候，我们只需要简单修改登录脚本。测试用例的脚本将不做任何的变动。这就是“原子”脚本的应用场景。至于“分子”脚本是对应于更高一个层面的，我们知道，现在的被测系统很多都是基于业务流的，要完成一个业务流的测试往往需要很多步操作，那么把完成这些操作的“原子”脚本组合起来就构成了“分子”脚本。通常要完成一个高层次的测试用例往往需要多个业务流的配合，这个时候的测试用例就是对“分子”脚本的调用了。简单来说，最基本的操作可以设计成“原子”脚本，对“原子”脚本的调用就构成能够完成特定业务逻辑功能的“分子”脚本，对“分子”脚本的调用就构成了测试用例。当然，测试用例也可以直接由“原子”脚本构成，这取决于测试用例的粒度和被测试系统的特点与复杂程度。

阶段 5 的另一个主要特征是对 QTP 对象库 Object Repository（OR）的统一集中管理。这里需要一套 QTP 对象库的管理流程以及会用到对象库的 sharable 属性，限于篇幅这里就不展开了。简单来说，就是应该从测试流程上保证测试用例设计工程师不能修改对象库中的任何内容，确有修改的需要必须通过申请由专人负责修改，这样做的目的是最大限度地降低 QTP 脚本的对象识别问题。

到了阶段 6，为自动化测试提供服务的非业务功能将被抽象出来，形成独立可重用的脚本，以脚本调用的形式提供给测试系统使用。比如说我们的测试日志记录，测试结果统计，邮件通知功能等等。随着为自动化测试提供服务的脚本的积累，将会逐渐形成基于 QTP 的自动化测试框架雏形。框架雏形将会在项目的使用过程中逐渐完善和成熟，最终演化为适合企业产品类型的自动化测试框架。此时我们将可以严格区分业务脚本和非业务脚本，测试用例的组织基于业务的“原子”或“分子”的脚本调用，实现“原子/分子”脚本开发与测试用例设计的分离。理想的情况是，测试用例设计者将由熟知业务需求的工程师或手工测试工程师来承担，而其完全不用考虑测试用例实现的细节，他只用关注完成一个测

试场景所需进行的业务流程和操作并调用相应的“原子”或“分子”脚本。而由自动化测试工程师来开发“原子”或“分子”脚本，实现最大程度的测试开发与测试用例设计的分离。

对于阶段 9 的高层次自动化测试框架，其实已经不属于 QTP 的范畴了，这个阶段的关注点将集中在整个测试环境的管理和监控，测试用例的组织，管理和分发等，这里笔者极力推荐开源的 STAF（STAX）框架，限于篇幅原因就不展开介绍了。

四、自动化测试团队组织结构的特点

这个话题可以有太多的东西可以讨论，自动化测试团队的人员的知识结构特点，团队成员管理模式的特点等等。这里只简单谈以下两点：

1）对于自动化测试团队的组织，其实有一个很有趣的现象，就是无论开发团队的规模如何，自动化测试团队的规模是相对稳定的。也就是说，对于一个 20 人左右的产品开发团队而言，传统意义上的测试团队（不包括自动化测试）可能需要 5-8 人，而如果开展自动化测试可能需要 6-8 人；而对于一个 200 人左右的开发团队，测试团队（同样不包括自动化测试）可能需要 50-60 人，而自动化测试团队可能还只是需要 10-15 人。由此可见，自动化测试团队的规模不会像传统意义上的测试团队那样，随开发人员规模呈正比增长。笔者认为这是自动化测试的一个很大的优势，尤其是对于规模相对较大的开发团队尤为明显。

2）自动化测试团队的成员应该把自己定位成开发人员，并且自动化测试同样会有自动化测试需求分析，架构设计等过程，所以团队的组织结构可以参考开发团队。

五、如何以相对小的开发成本实现 QTP 对于非标准化控件的识别与操作

对于 QTP 原生不支持的第三方控件，最简单的方式是采购 QTP 的官方插件，比如.NET 插件，JAVA 插件等。但是这个就无形中提高了项目的预算。并且有时候对于一些非主流的第三方控件或者公司自己开发的控件，购买的 QTP 官方插件还是无法提供好的支持，也就是说无法很好的对这类控件进行有效的识别与操作。这个也就是前文第二点中提到的，在项目开发的初期，资深自动化测试工程师必须分析 QTP 的可用性风险并且在必要的时候对 QTP 进行二次开发以规避此类风险。但是进一步想会发现，对 QTP 进行二次开发的代价往往是比较大的，首先是对于测试工程师的技术要求，其次是二次开发的时间成本以及由于自行开发的控件需求变更产生的同步变更问题，再次是控件识别的稳定性问题。那么有没有更好的方法可以解决这类问题。下面就以一个实际项目的技术实例来说明。

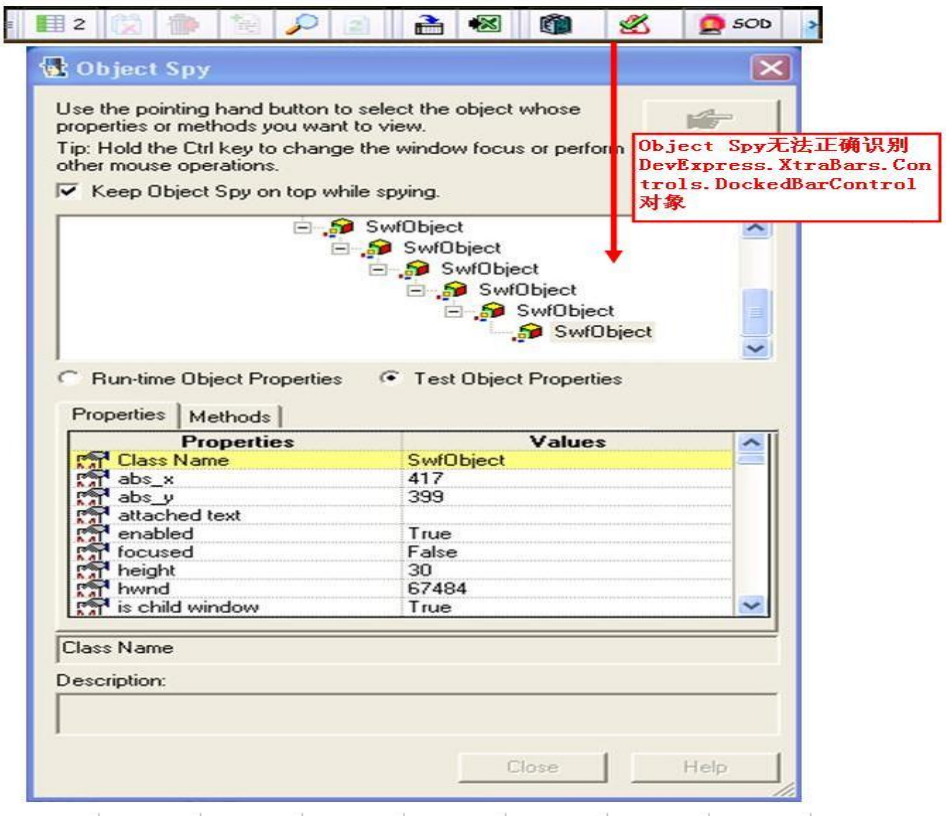


图 1 Spy 无法读取按键对象

在被测试系统的某 GUI 中有 DevExpress.XtraBars.Controls.DockedBarControl 实现的工具栏。该工具栏的 Button 在不同的系统条件下可以分为可用(Enable)和不可用(Disable)两种状态，在测试过程中需要检测 Button 的可用性，并且对于可用的 Button 进行点击操作。DevExpress.XtraBars.Controls.DockedBarControl 是 DevExpress 公司开发的.NET 控件之一，QTP 即使安装了.NET 插件也只能识别到 Microsoft 开发的标准.NET 控件，对于非标准的.NET 控件束手无策。对于 QTP 自动化测试脚本，我们需要对 Button 对象做到两个操作，1 识别按键的状态，可用还是不可用，2 点击按键。

如图 1 左侧可以看到 QTP 的 Object spy 检测到的工作栏的对象 ClassName 是 SwfObject。

而属性 swftypename 中是 DevExpress.XtraBars.Controls.DockedBarControl，这说明了这个对象的真正的属性。

Object Spy 中我们可以看到两类属性，我们可以选择查看 Run-time Object Properties 或者 Test Object Properties。Run-time Object 展现的是运行时本地测试对象的属性或方法。在脚本中可以通过 Object 属性来访问和获取 Run-time Object 的属性或执行其方法。

QTP 为用户提供了两种操作对象的接口，一种就是对象的封装接口，另一种

是对象的自身接口。对象的自身接口是对象控件本身的接口，对象的封装接口是QTP 为对象封装的另一层接口，它是 QTP 通过调用对象的自身接口来实现的。

两种接口的脚本书写格式的差别在于：自身接口需要在对象名后面加 object 再加属性名或方法名，封装接口就不用在对象名后面加 object。对象的封装接口是 QTP 使用的缺省接口，我们录制出来的脚本都是使用封装接口，在 QTP 中输入对象后默认弹出的也都是封装接口。但是封装接口不如自身接口丰富，因为 QTP 只是封装了部分常用的自身接口。所以我们在需要时，可以绕过封装接口，直接调用对象的自身接口。

根据上面的思路，我们可以在 DevExpress 公司的官方网站(http://www.devexpress.com/Downloads/)上下载这些对象的 Help 帮助文档。帮助文档中相当详细的介绍了这些产品的属性以及使用方法。从中我们可以了解到，工作栏的自身接口 VisibleLinks 可以读取到全部的按键，而每个按键则可以用按键在工作栏中的 item 名来识别，按键的可用性可以通过 enabled 属性进行判断。这样我们就可以通过 QTP 的 Run-time Object 来访问和使用这些属性和接口。例如：

```
Msgbox MenuObj.Object.VisibleLinks.item(“ButtonName”).enabled
```

以上的代码通过工作栏对象 MenuObj 的自身接口 VisibleLinks 来获得这个工作栏对象中可见工作栏组件连接（visible bar item links）的一个集合（collection），这个集合中 item(“ButtonName”) 就是 ButtonName 按键。所以，MenuObj.Object.VisibleLinks.item(“ButtonName”)就是指 ButtonName 按键。最终，通过这行代码我们的到的就是 enabled 属性的值，Ture 表示该按键对象可用，False 表示不可用。

对于这个工作栏的按键，我们还有第二个需求，那就是点击。点击这个动作我们为了模拟鼠标点击动作，需要输入坐标。我们利用 Bounds 这个对象的自身属性，Bounds 的返回值格式如下：{X=0,Y=0,Width=688,Height=30}。我们分别抽取 x 和 y 的坐标，算出中间 x 值与中间 y 值，并点击。代码如下：

```
Location=MenuObj.Object.VisibleLinks.item(Arrayitem(0)).Bounds
x=cdbl(Split(Split(Location,",")(0),"=")(1))+cdbl(Split(Split(Location,",")(2),"=")(1)/2)
y=cdbl(Split(Split(Location,",")(1),"=")(1))+cdbl(replace(Split(Split(Location,",")(3),"=")(1),"}","")/2)
SwfwinObj.Activate
MenuObj.click x,y
```

自身接口中还有大量的属性和方法，可以完成非常丰富的功能。解决了自动

化中非标准化对象识别和操作的问题。另外以上的 VBScript 脚本代码可以自行进行封装，以后需要使用时直接调用即可。

总结一下，对于无法识别的.NET 控件，我们可以按照上面的技术实例进行设计，一般情况下，可以解决 70%的左右的控件识别问题。如果以上方式未能解决问题才考虑对 QTP 进行二次开发。当然，对于一些主流的控件，比如.NET，JAVA，在预算允许的情况下也可以直接采购 QTP 的官方插件。

【淘测试专栏】自动化平台 AutoMan 介绍

作者：宝驹

一、AutoMan 的由来

两年前淘宝测试的同学都知道，我们自动化测试工具用的是 QTP，这里我还是需要强列顶一下 QTP 这个工具在对象识别，数据驱动，关键字驱动等方面的强大，然而当大团队运用时，你就会碰到测试脚本维护困难，公用资源难以复用，不能有效进行执行调度，报表资源不能有效展现等问题，并且由于其封闭的架构，扩展一个自定义的功能也需要花费较大的波折，导致自动化覆盖率及自动化实践一直不尽人意。

基于以上的问题，我们在思考，是否有一个语法简单，架构开放，功能强大的开源自动化框架供我们使用呢，此时我们引用了 Watir(Web application test in ruby). AutoMan 大概经历了以下几个阶段：

2009 年 1-3 月，在 watir 的基础上，针对淘宝的实际应用，对 Watir 进行了技术调研，扩展了一些功能，形成了 tcommon 的前身。

2009 年 5-6 月，我们成立天外飞仙项目，形成了在线脚本管理，测试集，测试计划，测试报表等现有大家一直看得到的一些测试管理概念，tcommon 框架也同时发布。

2009 年 8-9 月，借着 twork 的 0.3 版的发布，AutoMan 的前身 TAM 随之发布。

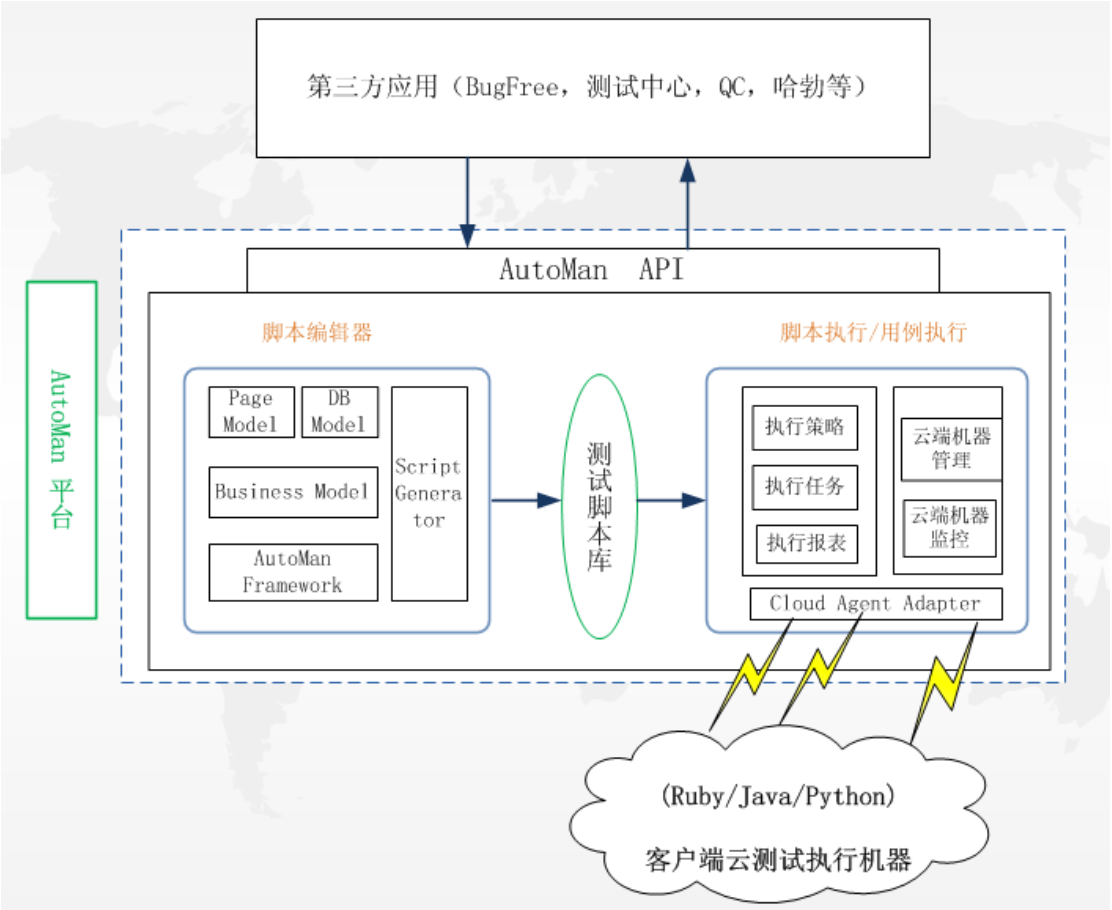
2010 年 4 月，基于对前一阶段深入总结及具体实践中碰到的问题，我们提出了基于页面模型的自动化框架，即 AutoMan Framework， 并在 AutoMan 一期中实现，自动化组内部戏称之为从 Dos 系统到 Windows 系统的跨跃。

2010 年 5-6 月，AutoMan 二期完成，基于 AutoMan Framework，开发了 Page Model，并更进一步产生了 DB Model 的概念，对页面对象，测试数据等做为资源化处理，为后续测试脚本的白话化提供基础。也在此项目中，我们对云测试执行平台进行了优化，稳定性，执行效率方面显著提升。

2010 年 8-9 月，AutoMan 三期进行中，这里也将解决项目中自动化脚本无法集成 AutoMan 的问题，及脚本生成器功能，让你快速生成脚本

二、AutoMan 架构介绍

AutoMan 的整体架构如下图：



1、以脚本为核心的 AutoMan 产品平台

脚本编辑器

针对脚本编辑器，我们的要求是：

- 方法简单好记
- 脚本写得要快
- 应付页面变化
- 快速定位错误
- 项目前期介入
- 测试数据维护方便

也就是说，要降低自动化实施的成本，让更多的项目，更多的用例自动化起来，基于以上的需求，我们开发了基于页面模型，数据模型和业务模型的自动化框架。

页面模型（Page Model）

对于一个用例描述的分析，如对淘宝登录操作，其中有一个用例的描述是这样的：

1）打开浏览器

2）进入登录页面

3）在登录页面的用户名框中输入用户：XXX

4）在登录页面的密码框中输入密码： YYY

5）点击登录按钮

6）进入我的淘宝页面，显示 XXX

从这个例子中我们看到，用例其实就是有:【页面元素+操作+数据】组成的，这里的【页面元素+操作】构成了我们的页面模型，通过 AutoMan 的页面模型，上面的用例就可以直接有脚本来描述：

```
-------------------------------------------------------------------------------
-----
URL_LOGIN = http://login.taobao.com/member/login.jhtml
Ie = IEModel.start(URL_LOGIN)
login = ie.cast(Mms::LoginPage)    #转换到具体页面模型的实例
login.txt_username.set(“tbtest1”)      #设置用户名
login.txt_password.set(“taobao1234”)  #设置密码
loing.btn_login.click                  #点登录按钮
assert_text(“XXX”)                       #进行校验
```

```
-------------------------------------------------------------------------------
```

那如何使这些脚本有效的执行起来？页面模型中就涉及到元素查找和元素操作两部分。首先来看下 loing.text_username 这个是什么？我们实现了一个在线的页面元素维护工具--Page Model,在这里对每一个页面元素进行建模,如下图：

每一个页面有若干个控件组成，每一个控件都有其唯一标志的属性，通过这些属性达到查找对应的元素，在页面查找中非常重要也是在我们实践中非常好用的一点是，我们采用了 JQuery 语法，对于一些不规则元素可以根据上下文关系进行处理，相对于 Watir 的处理能力大大提高了。JQuery（http://jquery.com/)本身也有很多优点：如语法简单，使用快速，文档齐全，开发，UED 非常熟悉等。

有了页面元素的定位后，可以针对页面元素的类型，分别赋予他不同的操作。现在已经定义的类型和操作如下表：

Element类型	对应页面控件tag	后台对应类型	方法	继承父类方法	使用场景
默认	Any（任何tag）	AWatir::AElement	click, text, get, exist?		控件点击，取值等
Button	Button	AWatir::AButton	click	click, text, get, exist?	对button框的点击
CheckBox	checkbox	AWatir::ACheckBox	set,clear	click, text, get, exist?	对CheckBox框的选择和取消选择
Link	Li	AWatir::ALink	click	click, text, get, exist?	对link的点击
TextField	Input,textarea, text	AWatir::ATextField	set	click, text, get, exist?	对普通输入框的输入操作
SelectList	Select	AWatir::ASelectList	set,selected_value, options	click, text, get, exist?	对下拉框的选择操作
no_wait	button,li,span, input	AWatir::ANoWatirElement	click	text, get, exist?	当点击之后有系统弹出框的控件的点击操作
Radio	radio,input	AWatir::ARadio	set,clear	click, text, get, exist?	对Radio框的选择和取消选择
rich_text	body	AWatir::AInnerTextSetElement	set	click, text, get, exist?	对富文本框的输入操作
aliwaypassword	Object	AWatir::AlipayPassword	set		支付宝控件的密码输入
未知（待定）	Object				一旦选了该类型，脚本将不处理该控件

从页面模型中我们期望满足以下的需求：

- 方法简单好记
- 脚本写得要快
- 应付页面变化
- 项目前期介入

由于项目进行中，页面元素的属性变化甚至前期不知道也是很正常的事情，而通过页面模型，可以做到页面元素的属性处于待定状态，也不影响你脚本的编写。

数据模型（DB Model）

也许有人会说，数据对于自动化来说并不是重点关心的内容，自动化更多关心元素的识别，脚本的简单，易读，调试方便快速等。然而你也不得不承认的是

数据问题严重影响了脚本回归中的稳定性，甚至到了数据问题不解决，自动化实施不下去的问题。如前面一个登录的例子：

```
login.txt_username.set("tbtest1")        #设置用户名
```

这里设置用户名为"tbtest1",这里的 tbtest1 这个用户可能包含了很多含义的，如可能是个卖家，可能是个钻石级的卖家，也有可能是个受过处分的卖家。在测试环境中数据是不受保护的，你花了很长时间设置的数据，有人在一秒中内可以破坏得（不是恶意的，是别人不知道这个用户在自动化中的用法）面目全非。从而由于数据原因，导致你的脚本运行无法通过，而跟踪这个脚本却花费了你非常多的时间。以前，数据是在 EXECEL 中准备类似下面的方法：

为了一个数据的完整性，我们针对不同的数据库准备了很多的初始化数据，在脚本运行之前自动运行这些数据，然而，准备这些数据实在是太痛苦了，需要很长的时间，需要很好的眼神，共享性太差等，我们称之为测试人员之痛。针对这个问题,DB Model 模块随之而生。

在 DB Model 模块中实现了如下的功能：

1）在线查找数据

可以在线根据不同的数据库和不同的表进行 SQL 查询操作，实现现有测试数据库数据的查询

2）编辑查询数据，并保存数据列表

根据上面查询到的数据，可以编辑为自己需要的数据，并保存，注意：这个保存的数据不是修改测试数据库中的数据，而是在 AutoMan 库中生成一条维护数据，实现了一个测试数据的快照。

3）对保存的数据实行便利的操作

有了上面的数据后，AutoMan Framework 提供了操作这些数据的 API, 这样实现了操作和维护的分离，也达到维护方便，数据共享(不仅在自动化中使用，手工测试也可以方便使用)等优点，另外借鉴 ActiveRecord 的方法，提供对测试数据库直接操作 API，不需要建立数据连接，操作数据，关闭数据连接等一连串数据操作方法。

◆ 查询：filter

```
ArkDB= DB[“dev_ark_ark”]
ArkDB[:items].filter(:id=>1123).all
ArkDB[:items].filter(:id=>1123, :status=>1).first
ArkDB[:items].filter(:id=>1123, :status=>1)
```

◆ 插入：insert

```
DB["dev_ark_ark"][:items].insert(:id=>1123,:name=”柱石”, :title=>””, ……. )
```

◆ 更新：update

```
DB["dev_ark_ark"][:items].filter(:id=>1123).update(:status=>0)
```

◆ 删除：delete

```
DB["dev_ark_ark"][:items].filter(:id=>1123).delete
```

◆ 直接 SQL：

```
DevArkArkDB["select max(group_ids) from cms_user"].first
```

业务模型（Business Model）

有了 Page Model 和 DB Model 之后，自动化中碰到的最要的两个问题解决了。然而大家也会注意到，有些操作经常会用到，比如登录操作，我们也提供称之为 Business Model 的模块，来共享这些操作，通过以上三者的结合，并基于 AutoMan Framework 来达到生成脚本的快速，准确。

日志查看

另外，为达到快速的调试，AutoMan Framework 提供了快速详细的操作日志，如下：

```
[成功]IE操作，start(http://search.taobao.com/search?q=豆浆机)
[成功]页面模型操作，cast(Buy::SearchAuctionPage)
[成功]lnk_talk_in_bar(去某某某吧讨论).click()操作
[成功]IE操作，attach((?-mix:ba.taobao.com))[标题]=>淘吧
[成功]IE操作，close()
[成功]page_navigator(页面导航条).txt_goto_page(到第几页).set(2)操作
[成功]page_navigator(页面导航条).btn_confirm(确定).click()操作
[成功]页面模型操作，cast(Buy::SearchAuctionPage)
[成功]page_navigator(页面导航条).lnk_page_links(直接到第X页).text()操作
1
[失败]page_navigator(页面导航条).lnk_page_links(直接到第X页)["7"]查找操作，找不到匹配的项！共5个元素。
      m = m.find_elements(AWatir::AElement, '.page-bottom >a:not([class])')["7"];
false
[成功]auction_records(宝贝记录).price(宝贝价格).text()操作
555.00
[成功]auction_records(宝贝记录).lnk_auction_link(宝贝链接).click()操作
[成功]IE操作，attach((?-mix:item.taobao.com))[标题]=>淘宝四年电器销售每时乐 MSL-3218五谷养生机米糊机豆浆机-淘宝F
[成功]IE操作，close()
[成功]IE操作，close()
脚本运行成功。
```

执行管理

执行管理包括执行控制，云端机器控制和云端适配器：

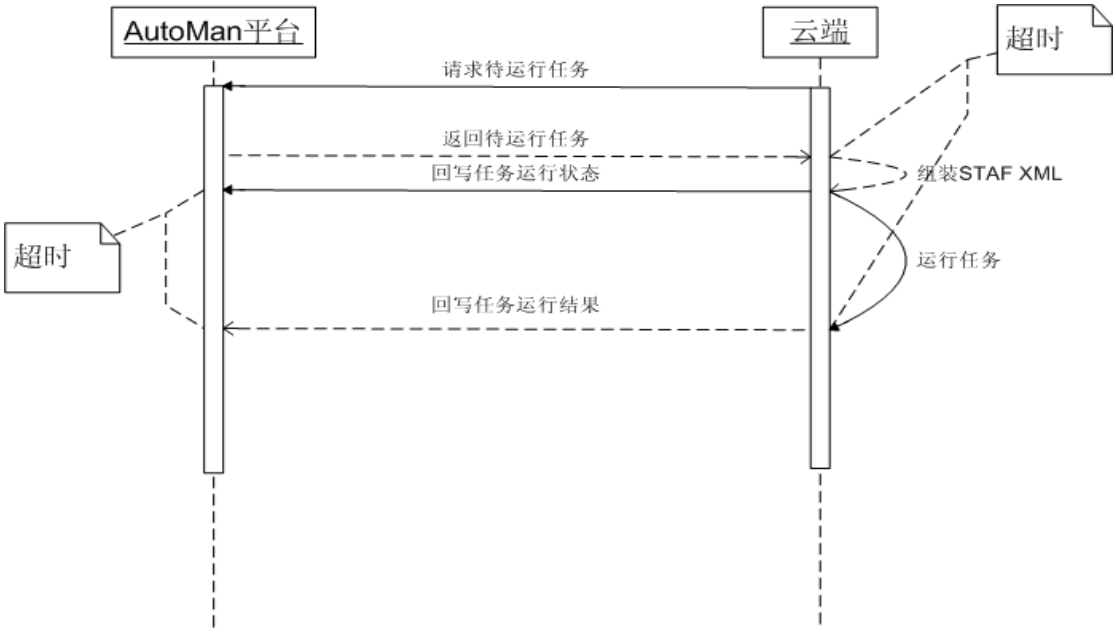
执行控制中分为执行策略，执行任务，执行报表等

● 执行策略

用户可以选择执行哪些脚本，什么时候在什么机器上，什么样的浏览器中执行等方面的选择和设置。

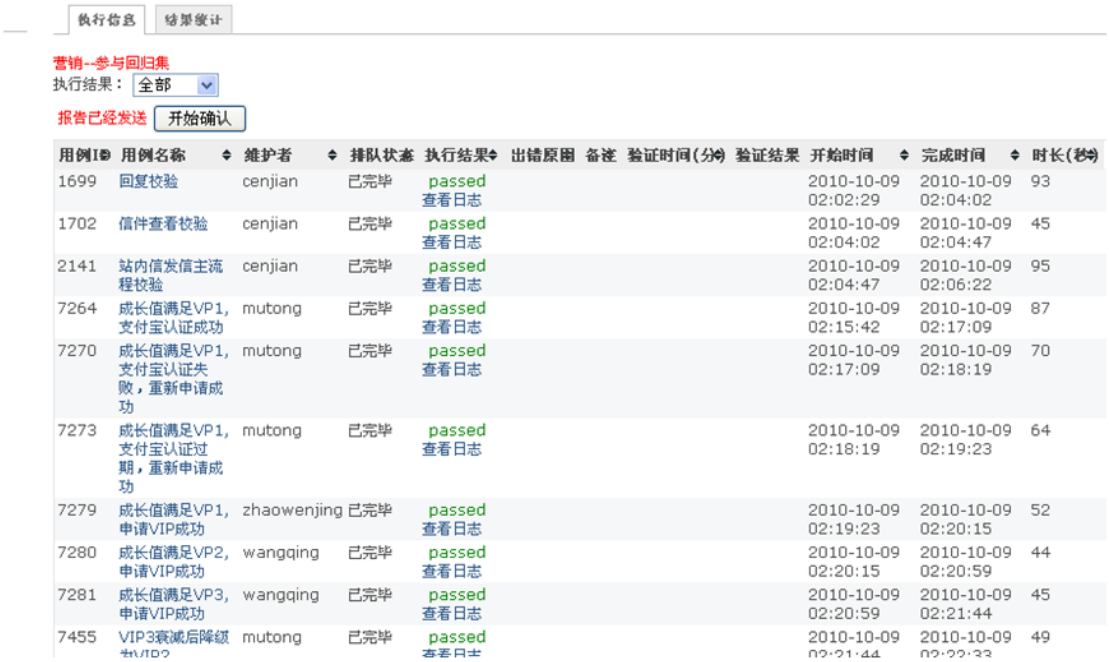
● 执行任务

执行任务我们采用了 STAF 框架，在云端主动不间断的向服务端发送请求，当服务端由匹配的任务生成时，向云端发送执行命令，执行完成后，向服务端发送本次执行日志，执行方案如下图：



● 执行报表

每次执行报表：



营销--参与回归集

执行结果：全部

报告已经发送 开始确认

用例ID	用例名称	维护者	排队状态	执行结果	出错原因	备注	验证时间(分钟)	验证结果	开始时间	完成时间	时长(秒)
1699	回复校验	cenjian	已完毕	passed 查看日志					2010-10-09 02:02:29	2010-10-09 02:04:02	93
1702	信件查看校验	cenjian	已完毕	passed 查看日志					2010-10-09 02:04:02	2010-10-09 02:04:47	45
2141	站内信发信主流程校验	cenjian	已完毕	passed 查看日志					2010-10-09 02:04:47	2010-10-09 02:06:22	95
7264	成长值满足VP1，支付宝认证成功	mutong	已完毕	passed 查看日志					2010-10-09 02:15:42	2010-10-09 02:17:09	87
7270	成长值满足VP1，支付宝认证失败，重新申请成功	mutong	已完毕	passed 查看日志					2010-10-09 02:17:09	2010-10-09 02:18:19	70
7273	成长值满足VP1，支付宝认证过期，重新申请成功	mutong	已完毕	passed 查看日志					2010-10-09 02:18:19	2010-10-09 02:19:23	64
7279	成长值满足VP1，申请VIP成功	zhaowenjing	已完毕	passed 查看日志					2010-10-09 02:19:23	2010-10-09 02:20:15	52
7280	成长值满足VP2，申请VIP成功	wangqing	已完毕	passed 查看日志					2010-10-09 02:20:15	2010-10-09 02:20:59	44
7281	成长值满足VP3，申请VIP成功	wangqing	已完毕	passed 查看日志					2010-10-09 02:20:59	2010-10-09 02:21:44	45
7455	VIP3衰减后降级	mutong	已完毕	passed					2010-10-09	2010-10-09	49

问题分析报表：

除这些执行报表外，我们还提供自动化的用例数，用例的增长数，及每条产品线等报表查看功能。

- 云端机器控制，

自动化测试需要大量的执行机器，这些机器的稳定运行，正确配置本身也是花费非常大的工作量的，包括云端机器监控和管理，能有效监控机器有效状态，并对云端机器可进行在线的标准化配置，如 AutoMan 包的升级，Host 的设置等。

- 云端适配器（Cloud Agent Adapter）

未来的脚本可能不仅局限于 Ruby 一种，因此需要有一个云端处理的适配器，使平台不局限于对 AutoMan Framework 的执行，也可以执行 Java，Python 等各类工具，STAF 框架很好的帮助我们实行了与语言，平台的无关性。

2、服务于外部的开放平台

AutoMan 自身是一个集脚本编写，脚本管理，执行，报表于一体的一个平台，但也没有覆盖测试的方方面面，如对环境的布暑，用例的编写等，为更好地与其他平台的协作，我们开放一些接口 AutoMan API，供第三方应用的调用，如 BugFree，QC。同时也希望能从第三方应用中受惠，使我们有更多的精力集中到 AutoMan 平台中，使之更为专业。

三、AutoMan 的后续发展

AutoMan 做为一个产品，后续还有更多的事情需要我们去完善的，重点也如框架图中所画的，分为 AutoMan 脚本编辑和 AutoMan 执行两大块去运作和发展。

1、脚本编辑器

目标是让脚本快速生成，需要对 Page Model,DB Model 操作体验上的不断优化，也有对脚本编辑器的改进，还有对 AutoMan Framework 需要横向扩展，如对 FireFox，Chrome 等浏览器，对 Win32 程序的支持等。

2、AutoMan 脚本执行

脚本的执行，需要跟脚本具体使用场景相结合，如在项目中如何有效使用脚本，脚本和用例的关联，各类统计报表的生成，执行任务更加高效稳定，客户端机器的有效配置，管理等，每一项都和用户的使用习惯和操作高效性有关，对我们的用户分析提出了很高的要求。

3、AutoMan 应用

根据来源于社区，造福于社区的精神，AutoMan 不仅希望在淘宝能够有效的使用起来，也希望能造福于整个测试界，让 AutoMan 承载的淘宝测试思想不仅在淘宝能开花结果，在外面的世界中也能绽放，这样也会要求我们的产品更开放，代码质量更专业，运营模式更科学等。AutoMan 的如何应用，有待于更多人更多的想象力！

构建 SilkTest 脚本无人守值解决方案

作者：蓝天伟

摘要：在我们的自动化测试项目中，经常遇到的一个困扰是脚本通过率无法令人满意，本文将以 SilkTest 为例讲解如何构造高质量的自动化测试方案。

关键词：SilkTest，无人守值，自动化测试

问题的提出：

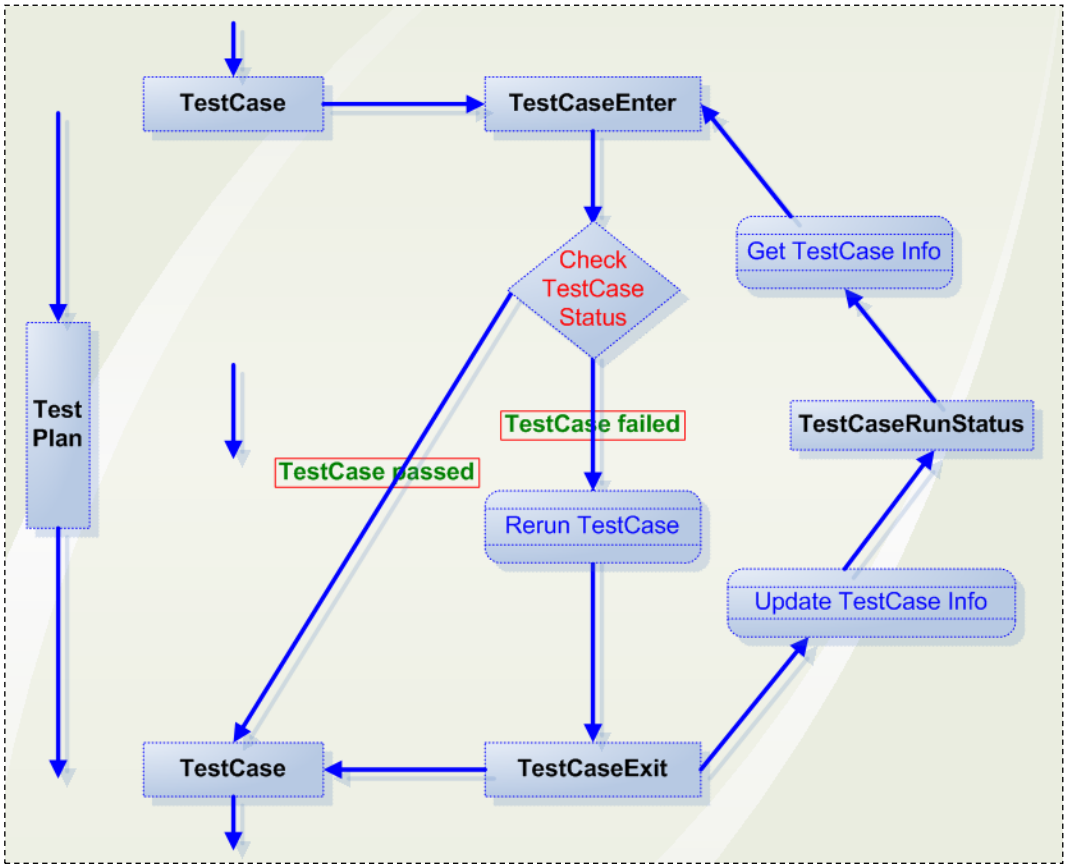
曾经在做一个 SilkTest(以下简称 ST)项目过程中，遇到这样的问题：由于产品和 ST 开启 Extension 后偶尔会起冲突，导致产品不可预期的 Crash（手工无法重现），从而导致 ST 就在那 Hang 住了，从而导致后续的脚本无法得到运行，这里出现第一个问题：如何保证脚本可持续运行？而当我们解决了单次脚本可持续运行的问题，同时又出现了另一个问题，因为产品和 ST 的冲突导致产品偶然性的 Crash，而当我们再次运行时，又会发现脚本又是可以通过的。那么当脚本运行完后，我们得到的结果其实并不能真正反应产品的 Defect 情况，而且跑完脚本后，我们还需要再次运行那些 Fail 掉的脚本，而且往往很多是非 Defect 造成的，可能源于脚本的不稳定（对于这个问题，我建议大家还是最大程度增强脚本的稳定性），工具的不稳定，网络的不稳定等等原因引起，这些问题导致我们需要花非常多的时间去手动的运行或检查这些 Fail 掉的 Case，出现的第二个问题是：如果保证我们得到最最接近产品真实 Defect 的测试结果？我相信这个问题也同样出现在其他的自动化工具中。

解决方案

一、解决思路

对于以上问题，我们引入无人值守解决方案，对于无人守值，我们的定义是：自动处理脚本运行过程中遇到的错误和异常；消除非产品问题导致脚本失败的用例（即提高脚本的通过率）。对于第一个解决办法是自动处理错误和异常，一般的做法是 a. 尽量减少用例之间的关联；b. 对用例脚本在运行前进行清理，而对运行后进行恢复 c. 少数情况下可以编写小型的监控工具来处理一些非预期窗口的弹出（该方法在以往的项目中被证实是非常高效的）。对于第二个提高脚本的通过率，我们的解决办法是利用 RE-RUN 机制，这也是本文介绍的重点。

二、Re-Run 机制流程图



三、Re-Run 机制核心代码

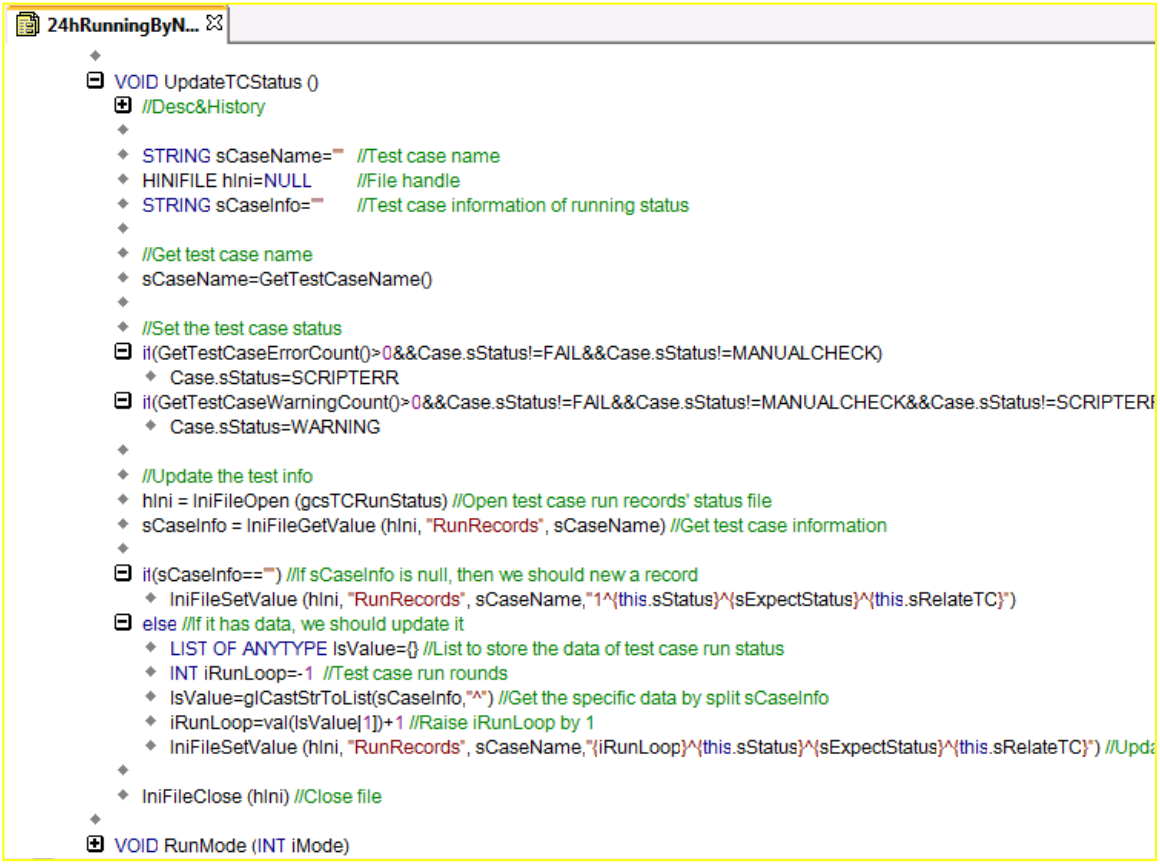
该代码主要包括 2 部分：在执行用例前检查是否运行当前用例 CheckIfRun 和在执行完用例后更新当前用例的状态到状态表 UpdateTCStatus，如下图：

24hRunningByN...

```
use "generic_library.inc"
use "projectSetting_library.inc"

//For 24h running by nobody
const STRING gcsTCRunStatus         ="{gcsTestDir}\Result\TestCaseRunStatus.ini"
BOOLEAN gbRunBy24h                  =TRUE
BOOLEAN gbRunNormal                 =TRUE

const STRING PASS                   ="Pass"
const STRING FAIL                   ="Fail"
const STRING UNRUN                  ="Unrun"
const STRING SCRIPTERR              ="ScriptErr"
const STRING WARNING                ="Warning"
const STRING MANUALCHECK            ="ManualCheck"


winclass wcCase
    //Const Definition
        STRING  sStatus=PASS
        STRING sExpectStatus=PASS
        STRING sRelateTC="NULL"

    //Method Definition
        VOID CheckIfRun (BOOLEAN gbRunBy24h optional)

        BOOLEAN CheckTCRelated (STRING sTCName)

        VOID UpdateTCStatus ()

        VOID RunMode (INT iMode)

window wcCase Case
```

24hRunningByN...

```
    VOID CheckIfRun (BOOLEAN gbRunBy24h optional)
        //Desc&History

        //If gbRunBy24hRunning is null, then we use traditional mode to run test case
        if(gbRunBy24h==NULL)
            gbRunBy24h=FALSE

        if(gbRunBy24h)
            HINIFILE hIni=NULL          //File handle
            STRING sCaseInfo=""         //Test csae information
            INT iRunLoop=0              //The round of running test case
            STRING sCaseName=""         //Test case name

            sCaseName=GetTestCaseName() //Get test case's name

            hIni = IniFileOpen (gcsTCRunStatus) //Open test case run status' records file
            sCaseInfo = IniFileGetValue (hIni, "RunRecords",sCaseName ) //Get test case information
            IniFileClose (hIni) //Close file

            //Determine running the test case or not
            if(sCaseInfo!=NULL&&sCaseInfo!="")
                LIST OF STRING lsValue={}
                lsValue=glCastStrToList(sCaseInfo,"^")
                if this.CheckTCRelated(sCaseName) //To check all the related test cases has passed or not
                    if(!gbRunNormal) //Check the run mode is normal or not
                        Case.RunMode(++iRunLoop) //it will run slowly one by one times
                    else
                        Case.RunMode(1)
                else  //If test case expect status equal with actual status, then the test case will be skipped
                    print("√√√Passed!")
                    exit

```



四、Re-Run 机制的使用

首先我们需要在 Basestate 文件中增加两个函数 TestCaseEnter 和 TestCaseExit（关于这 2 个函数，具体可参看我的另一篇文[SilkTest］关于 SilkTest 中的场景恢复函数），具体代码如下：

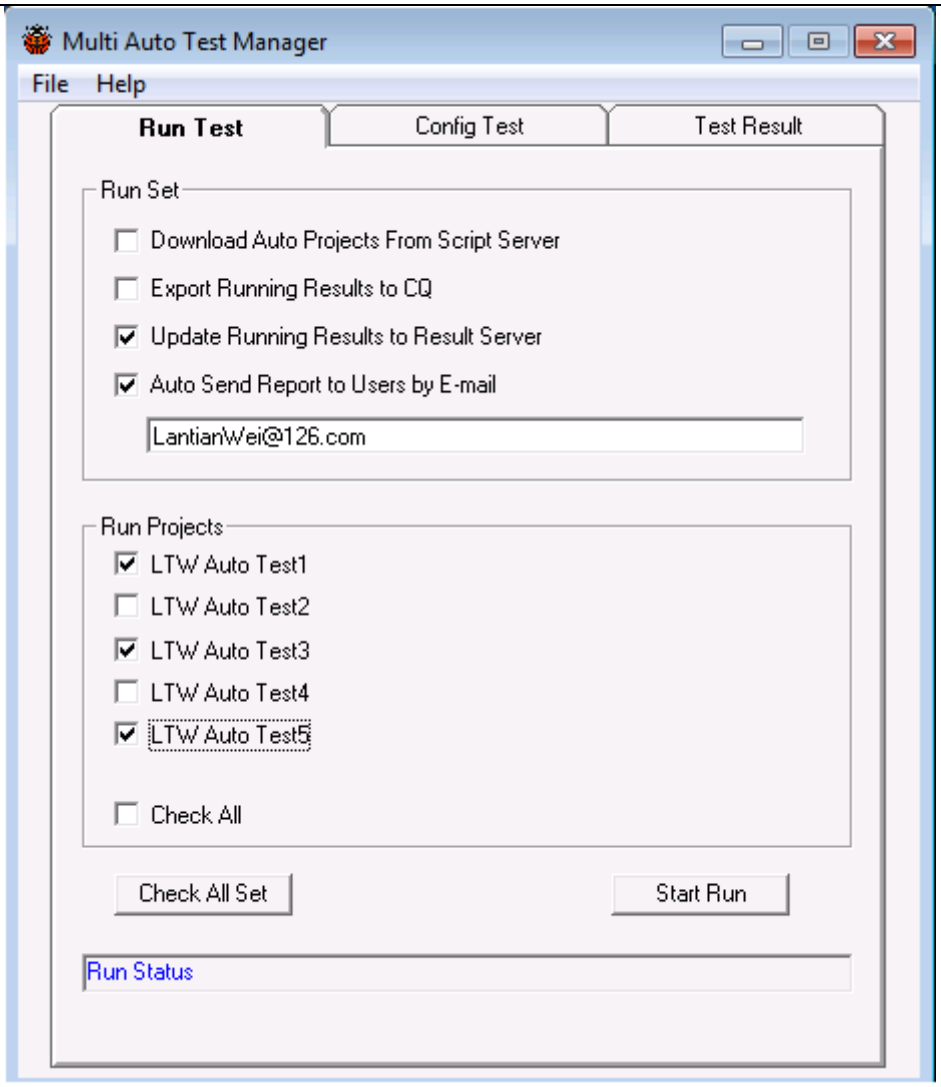
```
TestCaseEnter()
    Case.CheckIfRun(gbRunBy24h) //Call the method to use the solution

TestCaseExit (BOOLEAN bExcept)
    if(bExcept)
        ExceptLog ()
    Case.UpdateTCStatus() //Update test case run status
```

然后我们需要在用例脚本文件. t 文件引入 24hRunningByNobody.inc 文件，现在当我们再运行 Plan 文件，你会发现它会自动运行那些 FAIL 的用例了。当然如果你想进行一些额外的设置，比如对那些上下文关联的用例使用该方案，我则需要到 Case 里面进行设置，下面为我工作中的一实例：

```
//TC010 Tag/Model Variable Mapping
    testcase CQ002743300() appstate BaseState

    testcase CQ002743320() appstate BaseNone

    testcase CQ002743350() appstate BaseNone
        //Case Steps Description

        //Case Const Definition

        //Case Initialization
            Case.sRelateTC="CQ00274332"
            Case.sExpectStatus=WARNING

        //Case Script
            Log.Step("1. Change the 2ed mapping's type from 'Tag to Model' to 'Model to Tag'.")

            Log.Step("2. Close Organizer. On spreadsheet, Tag Variables table, modify variable 'Go Feed.Pressure' to 8501 kPa.")

            Log.Step("3. Hit 'Automatic Tag Update Disabled' icon.")

            Log.Verify("Verify it should be changed back to 8500 kPa automatically")

            Log.Step("4. On Model Variables table, change GO.Feed.Pressure to 8502 kPa.")

            Log.Verify("Verify the relative tag changes accordingly")

        //Case Recover/Clear

    testcase CQ002743360() appstate BaseNone
```

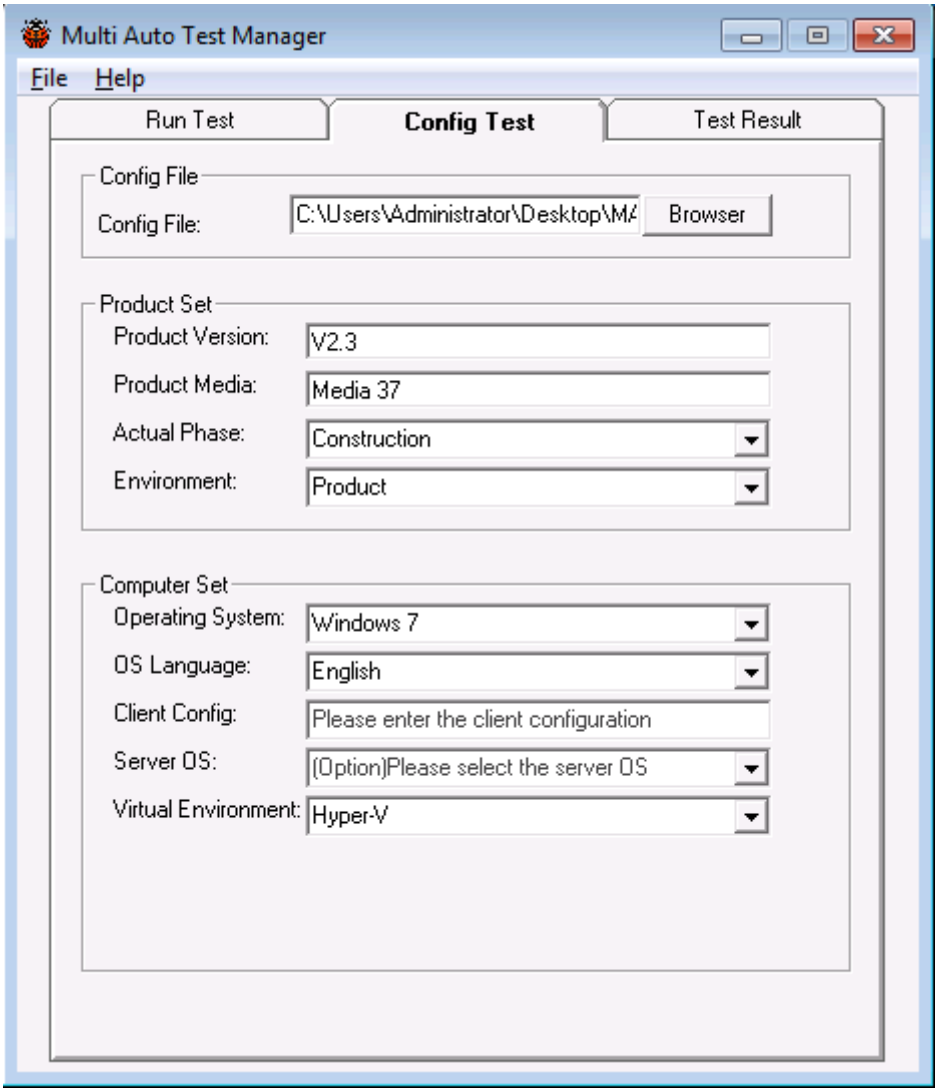
五、其他扩展

为了更好地利用该解决方案，我们开发了一 ST 的管理工具—Multi Auto Test Manager，如下图：



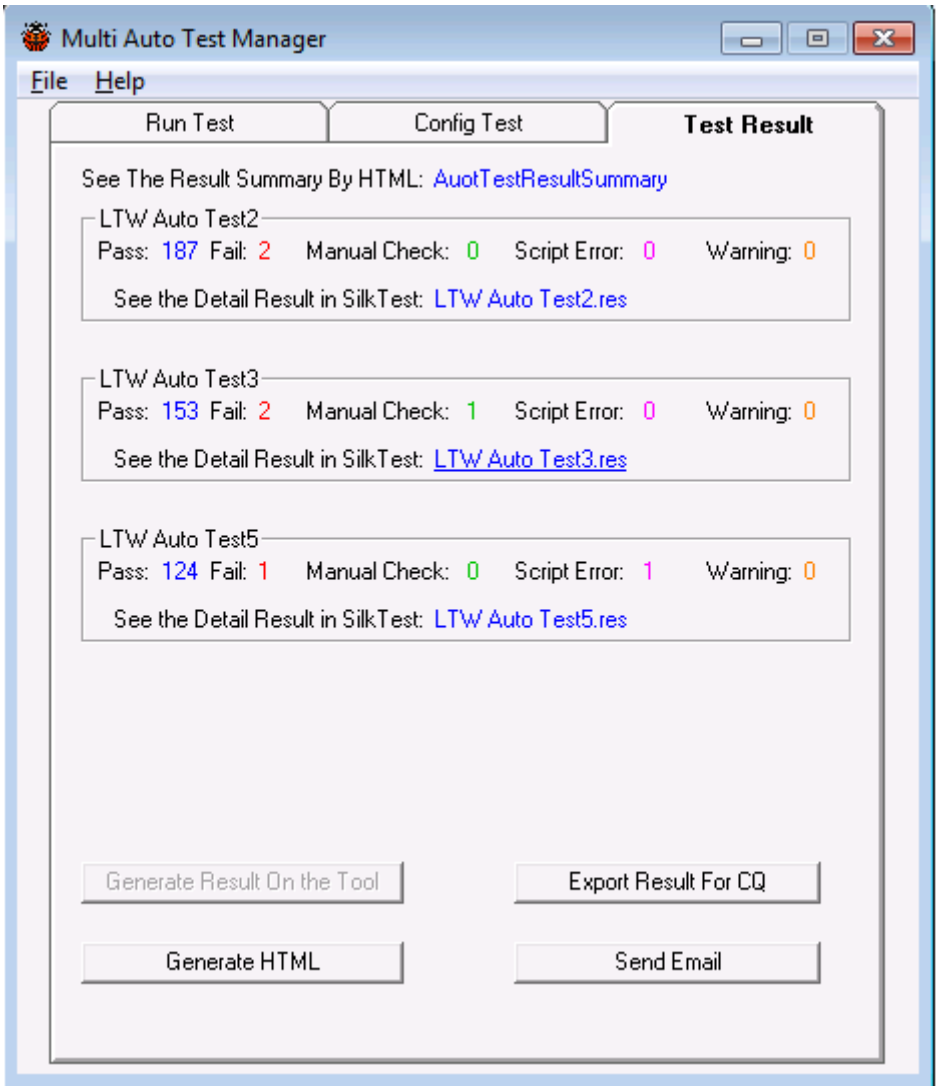
功能简介：

1. 从脚本服务器下载脚本到本地；
2. 导出测试结果到 CQ；
3. 上传详细测试结果到结果服务器；
4. 放送测试结果简报给相关测试人员；
5. 支持同时运行多个自动化项目的脚本



功能简介：

1. 通过配置文件导入自动化项目脚本信息；
2. 配置产品信息；
3. 配置测试机信息



功能简介：

1. 生成结果简述；
2. 通过 HTML 查看测试结果；
3. 通过 SilkTest 查看测试结果；
4. 导出测试结果到 CQ；
5. 产生 HTML 测试结果报告；
6. 发送测试结果报告

六、总结：

根据我们对以前的项目统计，一般按照我们传统的方式运行，脚本的通过率在 75%左右；但利用 Re-Run 机制后，我们项目的脚本通过率可以保证在 95%以上，从而可以节省大约 20%的用例的执行时间（这个时间其实是非常可观的，特别是对拥有成百上千的用例来说！），如果我们再配合 Multi Auto Test Manager 执行脚本，那么我们将会节省很大一部分时间。该解决方案其实还有一些不完美的地方，大家可以根据自己的实际需要进行完善。

Web 测试用例的设计过程

作者：文青山

摘要：

本文在个人工作经验和测试用例设计经验的基础上，总结出测试用例设计过程的思想，并就这一思想做了简单的阐述，另外也提供了一个综合此思想和用例设计方法的实例。

关键词：测试用例设计过程、需求与用例的关系

记得曾经作为一个刚入行的新手时，接到设计某个系统的测试用例的任务时，自己一筹莫展了很久。虽然不少书籍或资料上都比较好的讲到了测试用例的设计方法和原理，但是如何将这些理论转换为工作中的利器，如何有效的开展测试用例的设计工作，作为一个测试新手来说书上所讲的原理未免过于高深，所起到的实际指导作用也微乎可微。本文在个人测试工作经验和测试用例设计经验的基础上，总结了各个前辈们提出的一些观点，简单的阐述了个人有关测试用例设计过程的思想，以期帮助正在迷茫的同行们，另外也希望能与大家交流以弥补自身的不足。

一、认识测试用例设计流程，建立全局思想

1、用例设计上的一些认识

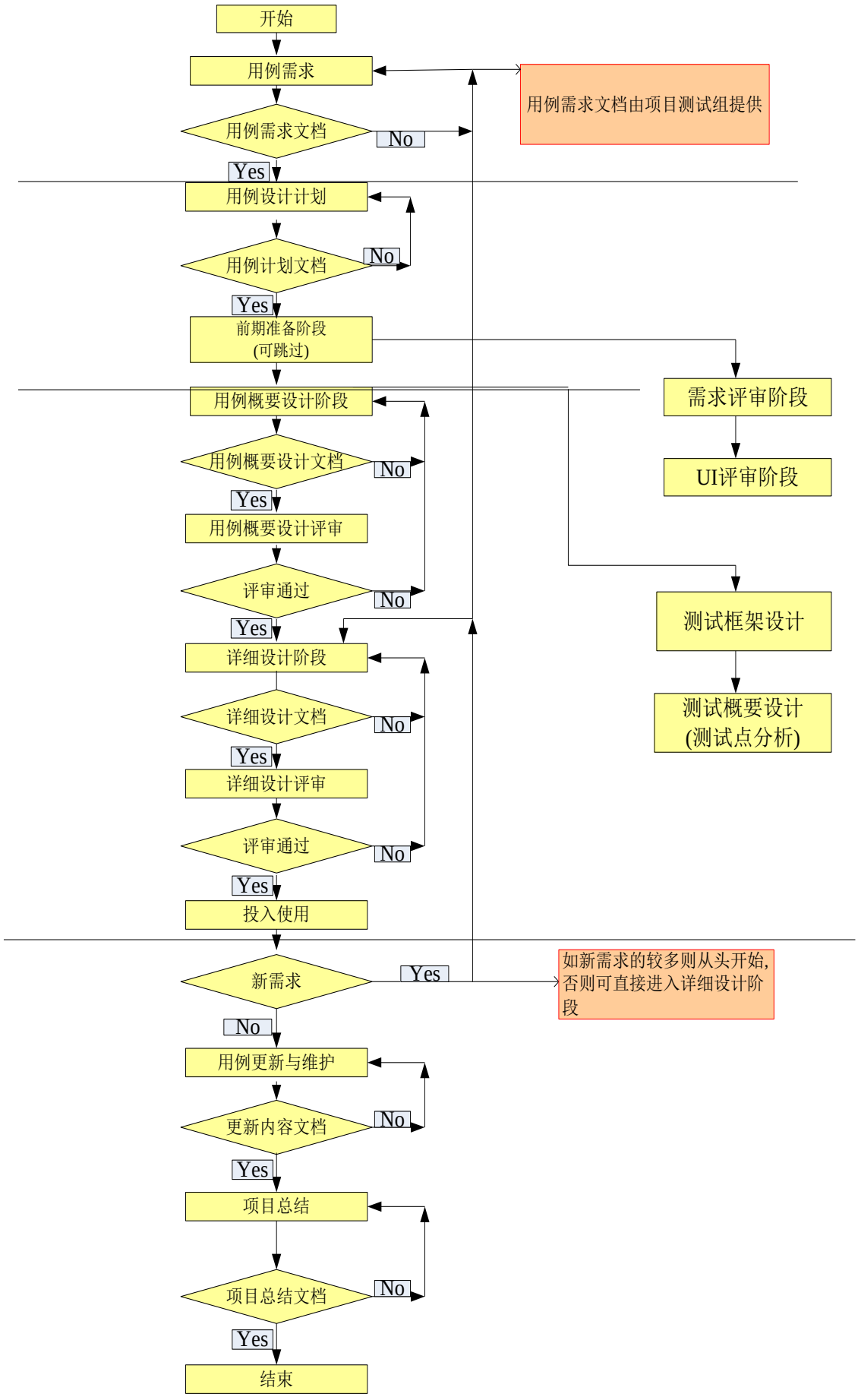
一直以来，无论是书面上还是个人对测试用例设计的理解上，首先提到设计测试用例，往往想到的都是具体方法，而对站在流程的高度上来思考测试用例的设计工作的资料和文章我所接触的都比较少。另外，在测试用例设计过程中，有不少才接触的人，往往无从下手，在什么时候该干什么事，也并不清楚，或者不明白如何有效地使用测试用例的设计方法去构建整个系统的测试用例。另外，还有一种普遍的认识，认为测试用例只需要照着 Web 页面一个一个功能点写下来就 OK 了，不需要考虑过多因素，但往往在测试过程中这些功能点却没有发现错误，而忽略的部分往往能发现较多的缺陷。这些现象和原因，个人认为都是由于没有建立测试用例设计整体思想或所建立的测试用例设计整体思想有偏差而造成的。

那么测试用例设计工作的完整流程应该是什么样子的呢？目前这个还没有看到相关权威文章或者书籍给予介绍，笔者在工作和学习中总结了一个测试用例设计的整体流程的例子，至于是否具有很强的实用性和较强的指导性目前还不得知晓，在些引用此流程只是做理论上的禅述。

2、用例设计的完整流程(图)

下文提供了一个测试用例设计的工作流程，此工作流程是为科室测试用例小组的工作安排而专门设定的，不具有广泛性，本文将依此流程，做一些测试用例

在全局基础上进行设计的个人思想，并在图后对一些可能存在的疑问的地方做了一些阐述，仅供参考。


开始
用例需求
用例需求文档由项目测试组提供
用例需求文档
No
Yes
用例设计计划
用例计划文档
No
Yes
前期准备阶段
(可跳过)
需求评审阶段
UI评审阶段
用例概要设计阶段
用例概要设计文档
No
Yes
用例概要设计评审
评审通过
No
Yes
测试框架设计
测试概要设计
(测试点分析)
详细设计阶段
详细设计文档
No
Yes
详细设计评审
评审通过
No
Yes
投入使用
新需求
Yes
如新需求的较多则从头开始,否则可直接进入详细设计阶段
No
用例更新与维护
更新内容文档
No
Yes
项目总结
项目总结文档
No
Yes
结束

3、什么叫测试用例需求？

个人认为测试用例的设计是根据测试系统的类型而决定的，测试用例的设计应该根据所测系统的大小、类型、特别是测试策略的安排而进行设计。此处所讲的测试用例需求，更多的从测试项目组的工作策略的开展来考虑的。个人认为测试用例需求，分整体需求和功能点需求和其它需求。比如，从整体上测试用例需要分为几种类型的用例，详细用例需要满足那些功能点或其它特点，而这些要求，在测试用例评审时，就可以作为用例评审的通过的标准。

4、测试用例分类与测试策略的关系？

个人始终认为，测试用例的分类应按照测试工作策略的安排而进行设计，为不同策略提供多个方式的选择，下面举几个例子，用来说明策略与用例的关系。

鹰眼监测分析系统：此系统总共只有 5 个页面，每个页面所有功能仅为显示或查询，很显然此系统规模很小，并且此系统使用范围为内部使用，且内部使用也仅有 3 至 4 人，另外，由于上面的原因此系统不做性能、安全测试。所以测试组采取的策略为，仅对界面、功能、兼容性进行测试即可，再加上项目很小，用例也相对较少，则我们就可以将系统的测试用例整合在一起，每次测试执行所有用例，1~3 轮即可停止测试。

Pantheon 系统：此系统共有三个子系统，且主系统分多种权限，且子系统与子系统、权限与权限之间存在着某种联系。另，供用户使用的系统的网络环境为外网环境，主系统的环境为内网环境。根据需求和测试策略，我们大体上需要做界面测试、控件测试、功能测试、业务逻辑测试、兼容性测试、性能测试、易用性测试、安全性测试（供用户使用的系统）、回归测试。那么测试用例的设计，可划分为界面测试用例、控件测试用例、功能测试用例、业务逻辑测试用例、性能测试用例、安全性测试用例、易用性测试、用户体验测试等（兼容性测试和回归测试，可用策略安排，选取主要用例类别进行测试；易用性测试和用户体验测试不在考虑范围），将测试用例按照上述方法，列出多个模块，为系统测试时多次迭代的主要重点倾注于功能测试、业务逻辑测试做好铺垫，其它用例类型，由于 Web 的特点以及稳定性可做适当的减少工作量，这样测试用例类型的层次明显，在安排的策略时也能根据用例的类型进行灵活性和重点性安排。

5、测试用例需求文档的实例

在此还是拿 Pantheon 系统来举例(仅考虑系统测试阶段)

整体用例类型：界面测试用例、控件测试用例、功能测试用例、业务逻辑测试用例、易用性检查表

详细用例应具有的特点：

A、用例应该覆盖所有测试需求点（这个很重要）

B、用例描述应具体，能高度概括测试描述的内容（如，验证登录最大位数错误）

C、测试用例按照规定的模板编写

D、测试编号为英文_数字

E、覆盖了所有需求文档中的描述

F、覆盖了所有功能点的描述

G、按照用例设计方法进行了等阶类划分、边界值、错误猜测等进行设计

F、其它请参照《测试用例评审检查点》

6、用例设计计划

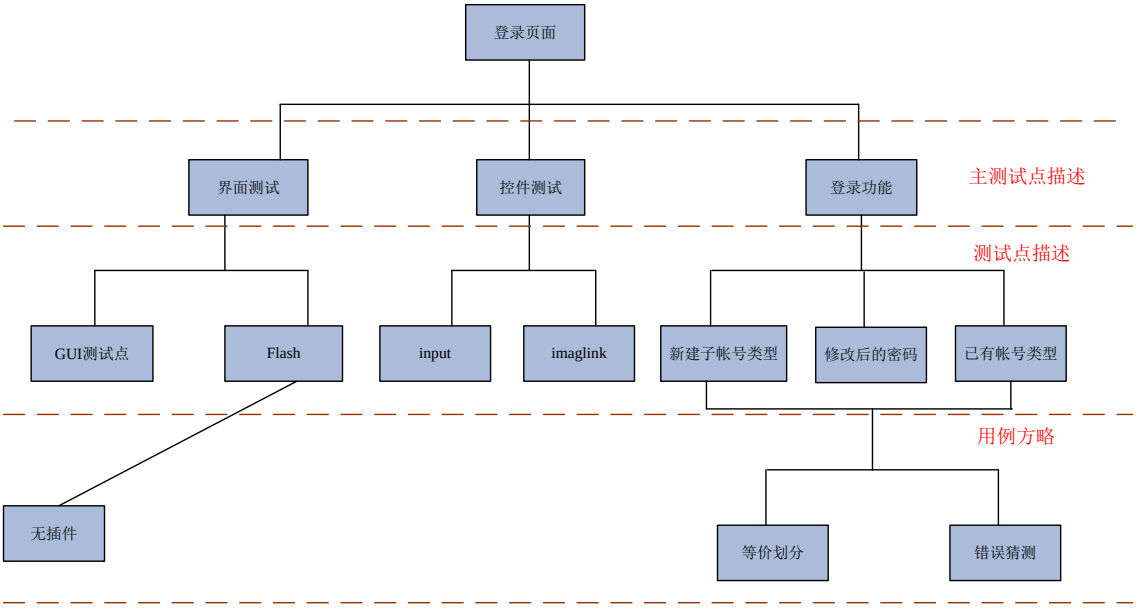
用例设计计划应在测试总体计划的范围内，并在这个时间范围内略有盈余。另外，测试用例设计计划需要安排好人员模块的分工、互查角色的定义、评审的人员的职责和注意点、以及通过可以正式投入使用的标准等信息，为测试用例设计做好统筹安排。

7、什么叫用例概要设计？

原本概要设计是软件设计过程中的一种方法，其主要任务是把需求分析得到的 DFD 转换为软件结构和数据结构。个人为了说明测试用例整体设计，引用了这种说法，而且在实际工作中引入了此类方法，以经验来说，此种方法用来设计用例具有很强的指导性，在用例评审时效率也相当高，并且可较好的验证页面功能点是否完整及用例设计策略的恰当性。那我所说的用例概要设计是指什么呢？用例概要设计是指将页面测试点以及与其它页面的联系，呈倒推层次形的图形描述，并且对该功能点将采取的测试方法（用例具体设计方法）进行大体描述。

8、用例概要设计实例

以新鹰眼登录页面为例，下面提供了一个用例概要设计图



9、基于同行评审原则的测试用例评审

测试也会产生缺陷，而测试产生缺陷的主要阶段个人认为就是在测试用例设计时，为了防止其缺陷的产出，我们往往采用内部评审和联合评审的方式来控制它。关于测试用例的评审请参考我的一篇博文《从同行评审的基本准则来看测试用例的评审过程》。

二、了解需求与用例的关系

理想中的测试用例按要求是能够追溯到需求文档中的需求，我们通过测试用例的归类，来佐证需求文档中的设计点得到了很好的实现，当需求改变时，我们可以清晰的知道那些功能点或者用例得到了影响。简而言之，我们通过此类方式，可以了解需求到测试用例的发展历程，同时也能通过用例来佐证需求。关于需求与用例的关系，朱少民先生的《全程软件测试》这本书有较详细的描述，大家可以去参考一下。

1、从需求追溯至单个用例的好处？

关于这种方式所建立的好处，上面已经进行了一些叙述，下面再罗列几条，更加详细的解释请参考相关书籍。

A、通过此，可判断用例的覆盖率（可强用力地佐证覆盖功能，设计点，需求率）

B、通过此，增加用例修改的直观性以及可维护性

C、通过此，可判断模块、功能点修改对需求的影响，增加了用例的可控性

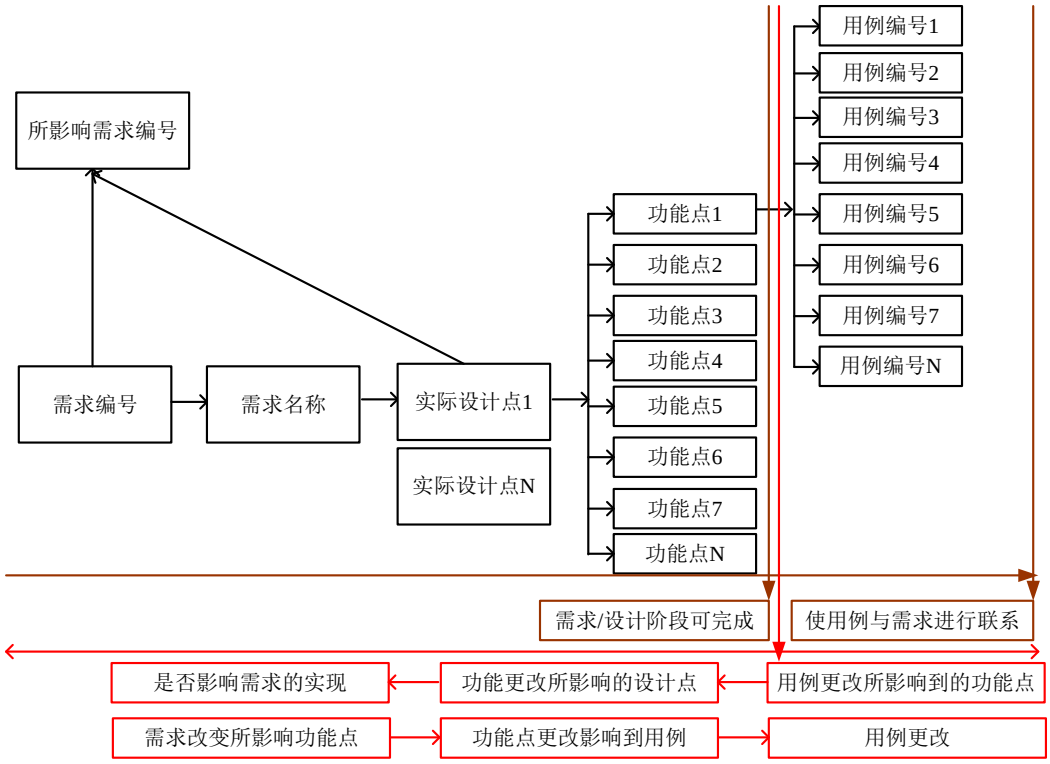
D、通过此，可判断需求更改之后，所影响到范围

E、通过此，为软件的质量评估提供了数据支持

F、通过此，可知用例结果对软件的影响等

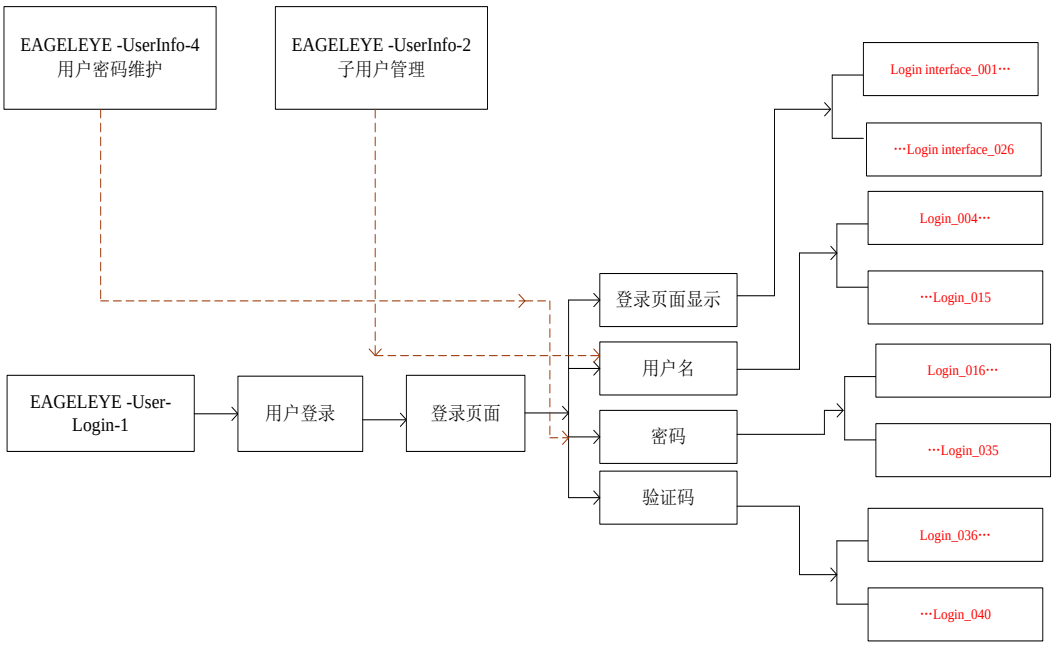
2、建立用例基线的方略图

此类方略建立的方式有多种，个人在此提供一个粗略的方略图和测试用例 ID 模板，仅供参考，不具有广泛性，其实用性也没有得到佐证。



3、建立基于用例基线的实例

以新鹰眼登录页面为例，下面提供了一个基于需求而建立的用例基线图



当然从实际使用的角度来看，上面这种图略法并不能很好的地体现这一思

想，而且所用时间较多，画这个图只是想更好地解释这一思想，具体操作可在测试用例文档中给予体现。基于这一思想的用例模板可参考我的博文《从需求追溯到测试用例》中的示例，URL 地址：http://www.51testing.com/?uid-287227-action-viewspace-itemid-202062。

三、测试用例的设计方法

（1）测试用例的设计方法已经有人做了很好的总结，在此就不重复，更多实例和讲解，请参考由“开着拖拉机上班”在 51testing 论坛上提供的《史上最全的测试用例设计方法总结》的文档，URL：http://bbs.51testing.com/viewthread.php?tid=101113&highlight=%CA%B7%C9%CF%D7%EE%C8%AB%B5%C4%B2%E2%CA%D4%D3%C3%C0%FD%C9%E8%BC%C6%B7%BD%B7%A8%D7%DC%BD%E1。另外，曾做过一个测试用例设计方法在 Web 项目中的实例的培训文档，有需要的同行可以 E-mail 我（wolaizhinidexin@163.com）。

（2）关于用例设计方法的个人观点

记得有人说过，一条成功的用例是能发现一个从来没有发现的缺陷。个人觉得，诸多用例设计方法没有最好，大家不可过多专注于这些方法的理论研究，良好的用例设计应该是采用多种方法来综合使用的，甚至有时用不成方法的方法来生成的用例，发现的缺陷也有可能是惊天地泣鬼神的。另外，在测试过程中我始终认为测试用例只是参照，依据用例展开发散思绪，是测试人员应该常常做的。如能做到，依据用例而飘然于用例之上，那么 web 测试对你也就不存在什么难度了。

Myers 曾提出了使用各种测试方法的综合策略，下面列出以供参考：

1）在任何情况下都必须使用边界值分析方法，经验表明用这种方法设计出测试用例发现程序错误的能力最强。

2）必要时用等价类划分方法补充一些测试用例。

3）用错误推测法再追加一些测试用例。

4）对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度，如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

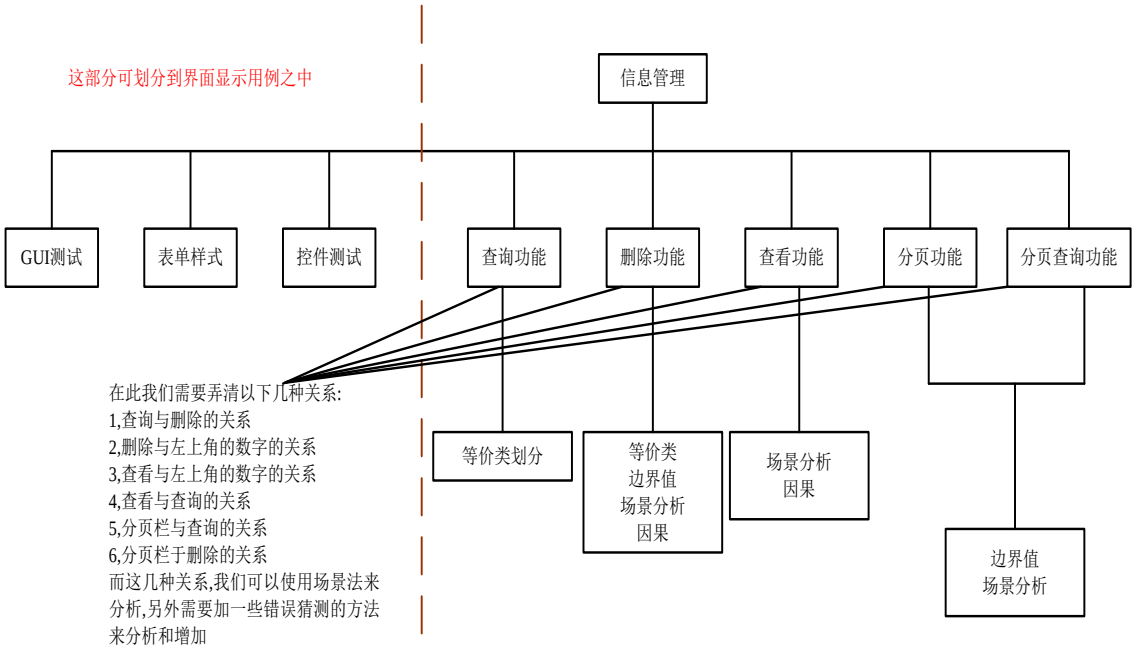
5）如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

四、综合运用的实例

在此提供一个综合运用测试用例设计过程思想和测试用例设计方法的实例，希望能给新手们带来帮助。（Pantheon_用户系统，信息管理页面）

1）按照第一节中提到的流程，我们应该弄明白这里的测试需求，但由于我们没有这个，所以在此省略。

2）画出用例概要设计图，并大体上给出用例设计方法的方向



3）下面我们针对每个功能点，一个一个的来分析。

GUI 测试、表单样式、控件测试，这些功能点，我们可以套用 Web 检查表，进行一定的修改，就可用来使用，所以这部分在此就省略了。

查询功能：

据 Web 页面（正常的应该是测试需求点文档）来分析，查询栏隐藏着以下条件：

●报警时间必须正确

●车牌号、报警类型可选择或不选择

●报警时间可填写或不填写

依照上面的分析，我们可以划分出以下分类，将类别与类别进行组合，即可

组成查询功能的测试用例，详细用例在此省略。另外，再配合一些错误猜测，可进一步完善查询功能的用例（如：选择正确条件之后，点击分页栏）。

A	无效等价类	期待结果			
	起始时间晚于截止时间	提示相关信息			
	有效等价类				
	C	无效等价类		无效等价类	期待结果
		起始时间选择，截止时间不选择 起始时间不选择，截止时间选择	B项	车牌号、报警类型不选择	显示起始时间至当前时间内的所有信息 显示截止时间以前的所有信息
				有效等价类	
				车牌号选择，报警类型不选择	显示起始时间至当前时间内该车的所有信息 显示截止时间以前该车的所有信息
				车牌号不选择，报警类型选择	显示起始时间至当前时间内所有报警类型 显示截止时间以前的所有报警类型信息
				车牌号选择，报警类型选择	显示起始时间至当前时间内该车，该报警类型的所有信息 显示截止时间以前的该车，该报警类型的所有信息
		有效等价类		无效等价类	期待结果
		起始时间早于截止时间 起始时间等于截止时间	B项	车牌号、报警类型不选择	显示该段时间内的所有信息 显示该时间点内的所有信息
				有效等价类	
				车牌号选择，报警类型不选择	显示该段时间内该车牌号的所有信息 显示该时间点，该车牌号的所有信息
				车牌号不选择，报警类型选择	显示该段时间内所有报警类型信息 显示时间点，所有报警类型信息
				车牌号选择，报警类型选择	显示该段时间内该车牌号和所选择警类型的信息 显示时间点、该车牌号和所选择警类型的信息

删除功能：

由于删除是在表单中进行的，所以我们可以分析出以下几种场景，并对这几种场景采用相关方法，得出用例。另外，再配合一些错误猜测，可进一步完善查询功能的用例（如：删除后再查询）。

分析场景	边界值	等价划分	
		无效等价类（即选择"否"）	有效等价类（即选择"是"）
未查询时，删除	首页，删除单个	仍处于当前页，总数没有变化	总数少1，序号仍累加，仍处于第1页
	首页，全部删除		总页数少1，总数少10，序号仍累加，仍处于第1页
	中间页，删除单个	仍处于当前页，总数没有变化	总数少1，序号仍累加，仍处于所选页
	中间页，全部删除		总页数少1，总数少10，序号仍累加，仍处于所选页
	末页，删除单个	仍处于当前页，总数没有变化	总数少1，序号仍累加，处于末页
	末页，全部删除		总页数少1，总数少10，序号仍累加，处于末页
	全部删除		无分页栏，提示相关信息
查询后，删除	首页，删除单个	仍处于当前页，总数没有变化	查询内容中，总数少1，序号仍累加，仍处于第1页
	首页，全部删除		查询内容中总页数少1，总数少10，序号仍累加，仍处于第1页
	中间页，删除单个	仍处于当前页，总数没有变化	查询内容中总数少1，序号仍累加，仍处于所选页
	中间页，全部删除		查询内容中总页数少1，总数少10，序号仍累加，仍处于所选页
	末页，删除单个	仍处于当前页，总数没有变化	查询内容中总数少1，序号仍累加，处于末页
	末页，全部删除		查询内容中总页数少1，总数少10，序号仍累加，处于末页
	全部删除		返回初始状态，即显示所有信息
删除对左上角数字的影响	原因	结果	
	删除未查看的报警信息	减N	
	删除已查看的报警信息	无变化	

查看功能：

查看跟删除有些类似，要据相关方法进行分析，就可得知下面所列出的信息，然后对其进行综合，即可得出相应的用例，具体用例略。

分析场景	边界值	结果
未查询时,查看	首页查看	打开查看页,关闭后页数以及所处页无变化
	中间页查看	打开查看页,关闭后页数以及所处页无变化
	末页查看	打开查看页,关闭后页数以及所处页无变化
查询后,查看	首页查看	打开查看页,关闭后页数以及所处页无变化
	中间页查看	打开查看页,关闭后页数以及所处页无变化
	末页查看	打开查看页,关闭后页数以及所处页无变化
查看对左上角数字的影响	原因	结果
	查看未查看的报警信息	减N
	查看已查看的报警信息	无变化

分页功能：

分页栏采用边界值和场景分析，即可得出相关用例，下面只是列出了一部分分页栏的用例，详细的用例可以参考 web 检查表中的检查点。另外，可以配合错误猜测法来完善用例（如：删除后，点击分页栏）

分析场景	边界值	结果
未查询时,翻页	点击1	无法点击
	点击>1,<n+1	跳转到所点击的页
	点击n+1	跳转到末页
查询后,翻页	点击1	无法点击
	点击>1,<n+1	在查询内容中,跳转到所点击的页
	点击n+1	在查询内容中,跳转到末页

分页查询：

分页栏查询采用场景分析，边界值以及等价类划分，即可得出下面的分析内容，注意“非数字字符”我们还可以进一步采用等价类划分的方法进一步分析，在此就剩略了。另外，关于分页栏查询输入框，往往很多时候开发忘记对非法字符进行判断，所以我们在此可进行一些错误猜测。如输入″ style=″background:url(javascript:alert(21881))″

分析场景	边界值	有效等价类	期待结果
未查询时,分页查询	首页 中间页 末页	>0,<n+1	跳转到指定页
		无效等价类	
		<0	跳转到首页
		>n	跳转到末页
		非数字字符	跳转到首页
分析场景	边界值	有效等价类	
查询时,分页查询	首页 中间页 末页	>0,<n+1	在查询内容中,跳转到指定页
		无效等价类	
		<0	跳转到查询内容中的首页
		>n	跳转到查询内容中的末页
		非数字字符	跳转到查询内容中的首页

将以上分析内容，给予一定的排列和组合，按照测试用例的模板，就可以生成详细用例了。另外，查询功能中的报警时间，无效等价类少分析了一种情况，大家可以想想是什么情况？

五、详细测试用例书写规范

A、用例概要，需采用“验证+概述步骤思想”的语言，如“验证“删除”用户对分页栏的影响。

B、用例中，凡是涉及到按钮、链接、页面或引用用例的语言，均要使用引号（“”）。

C、标点符号必须统一为以下内容，编号后面用点号，其它用中文标点符号（,、“”、。、:、;；等）

D、期待结果与步骤相对应。

E、只有一个步骤或期待结果时，也必须使用编号。

F、步骤描述中，使用语言链接步骤（如点击步骤 1->步骤 2->观察…..）。

G、用例中，用语应精练，有很强的指引路线的作用，尽量减少使用类似 N 多个或 H 小时的用语。

H、在 Excel 中，相同初始条件或其它内容，不允许合并同类项。

M、用例编号，以主导航栏中的英文名称+数字。

六、参考

[1]朱少民：《全程软件测试》

[2]E 测工作室：《QTP 项目应与进阶》

[3]51testing：开着拖拉机《史上最全的测试用例设计方法总结》

[4]我的博客（URL：http://www.51testing.com/index.php?uid-287227）《从需求追溯到测试用例》《手机测试策略与 Case 的关系》《测试用例小组团队规则书》《兼顾兼容性测试的冒烟测试和系统测试过程》

提高你的测试超能力

——秘密工具添加到你的工具箱中

作者：杨俊

第一个作为栏目特色出现在 *www.StickyMinds.com* 上。

我三岁时，最喜欢的卡通片是神奇四侠。妈妈发现后，她转换了频道，不准我看它。太暴力了吧，我猜。她强迫我看罗杰斯先生的节目，希望作为一种纠正，我想-但是我的心灵已经中毒太深。尽管我是冒险坚持偷看神奇四侠，“现在是猛击时刻。”还是成为我那几个月来最爱说的话。

几天前，我意识到，当我测试时，我像一个超级英雄。1969年播下的种已经开满了花。对探索性测试我最喜欢的是，当我使用X射线影像来扫描一个物品时，找出它的瑕疵时我的感觉。我可以像克拉克肯特一样，扮演成普通用户身份，但我也可以在需要时跨越高楼大厦。拥有一对仿生耳，超强度，能够扭曲时间和控制自然力量。

是什么给了我这些神奇的力量，超越了普通的测试者？一组特殊的工具。什么工具？他们未出现在马里克和福特的优良测试工具(http://www.testingfaqs.org/tools.htm)名单中。我所讲的工具甚至不会被当作测试工具来出售。但它们给你的力量是很强大的。

X射线影像，超强度，时间控制

一些工具能让你看到看不到的东西。

在约六英寸的你的处理器内放置一个晶体管收音机，调到大约AM975的地方，你其实能听到芯片在工作的声音。 当程序运行呈现类似被挂起的状况时，它能帮助你判断程序到底是真的卡住，还是说只是处在忙碌处理中。我是从格雷格教皇这里学到的这个办法。

运行 Restorator（共享软件，60美元）， 它可以提取每一个应用程序的资源，包括所有对话框，菜单文本，还有存储在字符串资源中的错误消息。这工具，不仅为你学习应用程序功能和做测试概述提供方便，而且对程序更新时做简要说明和鉴别出资源是否已经改变很有帮助。

使用InCtrl5（免费软件），能帮助你及时获知哪些文件和注册表项做了改变。可以使用它来做安装测试或只是找出正在运行的应用程序是如何修改文件的。 如果你想实时监控这些东西，Winternals 做了一系列像Filemon，Regmon这样价格便宜的系统监控工具，它们能分别监控到所有文件系统和注册表变化活动。

其它一些工具能让你做一个普通用户做不来的事情。比如EZ Macros（30美

元）能让你做光记录和回放。它是全功能GUI测试工具的替代品，非常轻便和便宜。此外，Kleptomania（50美元） 可以复制通常不允许被复制的对话框，图形显示文本和错误消息。然后Print Key Pro（20美元）能为你精确截图。

我最喜欢的工具之一是 Spector（75 美元）,它在我测试过程中，每一秒就能给屏幕截一下图。这个工具是相对非侵入的，不需要你来操

心“我刚才做了些什么操作?”因为它那像录像机一样的界面可以让我重新准确看到任何时候的屏幕上的样子。有一次，我无意中录了一个星期的个人电脑使用情况。直到我注意到Windows目录下有奇怪，大量的数据文件时我才意识到它正在记录。

我绝对最爱的测试工具就是Perl了。如果你有任何编程的潜能，考虑学习Perl吧。它是一个非常强大的操作数据语言（例如当你需要为测试做分析或创建文件时），而且它有许多可用的模块，可以实现许多很不错的功能，例如控制应用程序，监控系统资源，模拟web浏览器。最棒的是，Perl是免费的。

寻找一个工具的秘密特性

这些工具都有一个共同点：它们都低于100美元，它们除了做测试用，还能做其他事情。比如，Spector, 就能通过电脑上的截屏，暗中监视到配偶是否有欺骗情况，小孩有没做坏事（它的一连串截屏实际上就像是一个男人在偷窥一个女人的画面）。Restorator 可以用来让你定制你的应用程序。

要找到这些有用的工具，你需要更有创造性地去搜索它们，而不仅仅是在雅虎搜索框里键入“test tool”的字样。要时不时浏览一下 Download.com，寻找有用的工具. 我在 Google 上通过搜寻“resources extract”找到了Restorator。除了下载，还要核对工具是否带有开发工具包，O/S 资源包，MSDN 库，技术书籍附带的 CD。

光盘，供应商网站，或请教程序员们最常使用的工具是什么。有些工具是操作系统自带的，但需要你去寻找才会发现他们的存在。

如果您发现任何其他又酷又便宜的工具，请告诉我。我正在尝试编辑一个大的工具列表。所以，让我们一起行动起来吧！消灭所有 bug ！

单据测试技巧

——ERP 软件测试

作者：薛继国

摘要：

ERP 软件测试在整个软件开发周期内非常重要，ERP 软件包含的功能非常广泛，展现的形式也非常复杂，其中以单据、账表居多，不同的功能对应的测试方法各不相同，但对于同类型的 ERP 软件相同的功能测试方法技巧大致相同，本文重点对 ERP 软件重要组成部分－单据 的测试技巧进行详细讲解，以采购订单为例展开说明。

关键词：单据、表头、表体

一、单据在软件中的作用

单据在软件中是重要的组成部分，是满足一个业务流程不可缺少重要环节，一个单据能够形成一个流程多个分支或多个流程多个分支的业务，在一个流程中起到承上启下的作用。

二、单据特点：

单据的重要程度决定单据的复杂度比较高；由于组成部分内容多决定了功能涉及面比较广。

三、测试方法：

单据的测试可以从单据组成及功能上来进行测试设计

1、工具栏测试：一般分为输出区按钮、单据操作区按钮、表体操作区按钮、辅助区按钮

● 工具栏按钮显示：单据的不同状态下工具栏按钮显示不同，按钮的状态不同，比如：单据处在审核状态下“审核、修改”按钮置灰，但不同的单据控制也不相同，需要根据实际需求设计要求来判断。

● 工具栏按钮操作：工具栏上按钮正常可操作状态下操作是否有反映。

● 操作后实现相应功能要求：按钮操作是否可以实现相应功能，比如：单击“增加”按钮后单据是否处于可编辑状态，单击“帮助”按钮后是否弹出帮组界面，单击“退出”后是否退出单据界面。

● 工具栏按钮的快捷键操作：每种单据都有或多或少的快捷键操作，测试操作快捷键是否正确，比如：在单据界面中按“F1”弹出帮助，“F6”保存单据，具体该单据有哪些快捷键需根据一致性规范要求来测试。

2、输入项测试：包括字符型、日期型、数字型等

● 正常输入：各种输入框提供操作员进行数据录入，每种项目输入正确值后判断是否正确，是否可正确输入，比如：日期型输入框输入 2007-02-28

● 非法输入测试：不符合类型输入、非法字符输入，比如：数值型输入框录入字符 abc 应不允许，应控制不能录入或录入后给出提示，且提示确定后焦点返回输入框

● 极限值测试：主要针对数值型和字符型字段的测试，比如：存货的数量多为 float 类型，通常长度为 8 位，单价多位 money 类型，长度为 8 位，两个字段全部输入 8 位长度后程序是否报错，且金额＋数量×单价，此时计算出的金额超出范围是否异常，是否有溢出判断；对于字符型如表体自定义项输入，如果自定义项长度为 30 位，全部输入满 30 位后是否可以正常保存，是否报错

● 参照测试：大多数输入项都可以参照录入，比如：供应商参照，参照按钮是否可用，参照后弹出是否为供应商档案界面。

● 快捷键测试：基础档案的数据一般都可以进行“F2”快捷参照，按“F2”是否可正确弹出参照界面；“ctrl+F2”快捷参照，是否可以正确参照显示存货+自由项

● 输入后带入信息测试：特别是属性较多的字段输入后会带出相关的属性信息，比如：存货参照录入后会带出存货名称、规格、自定义项等属性信息

● 输入项控制测试：相关联的输入字段有控制关系，比如：输入部门 A 后，如果此时输入业务员只能输入属于部门 A 的业务员；如果先输入业务员，如果业务员有对应部门，此时自动带入部门；输入供应商后自动带入该供应商的税率（如果设置了税率）。

3、非输入项测试：系统自动产生后显示的字段

3.1 表头区：

3.1.1 单据号：该单据号可通过格式设置为自动编号不可修改，此时增加单据后会按规则自动生成单据号，且不允许修改，需要测试自动生成的号是否正确；也可以通过格式设置为自动生成可以手工编号，详细见格式测试。

3.1.2 默认显示字段：部分表头字段新增时会带出默认值，测试是否可以正确带入默认值，比如：单据类型

3.2 表体区：一般是指制单人、审核人、变更人、关闭人等，这些字段是在对单据进行审核、变更、关闭后系统自动显示的字段，需要测试显示是否正确

4、权限测试：功能权限、数据权限

● 功能权限：是指在系统中可设置为操作员是否可以操作单据的权限，包括查询、录入、审核、关闭、变更等权限

（1）需要分别测试每种操作有权限和没有权限是否控制正确

（2）需要测试各种操作组合后的权限控制是否正确

（3）需要测试年结、升级后权限是否丢失，详细见升级、年结测试

● 数据权限：是指单据中的个别字段的权限，包括操作员、部门、业务员、存货等权限，这些权限都是针对操作员来讲的，可在系统中设置某操作员有哪些操作员、部门、业务员、存货的权限，那么没有权限的不可以录入或者查询。

（1）操作员：如果操作员有其他操作员权限，那么可以看到其他操作员增加的单据，如果没有则看不到

（2）部门：如果操作员有某部门的权限，则可以输入该部们，且可以看到具有该部们的单据，否则不能输入也看不到有该部们的单据

（3）业务员：如果操作员有某业务员的权限，则可以输入该业务员，且可以看到具有该业务员的单据，否则不能输入也看不到有该业务员的单据

（4）存货：如果操作员有某存货的权限，则可以输入该存货，且可以看到具有该存货的单据，否则不能输入也看不到有该存货的单据

只要操作员没有其中一种字段的权限，那么就看不到这张单据

表：1

	操作员	部门	业务员	存货	单据查询
查询权限	√	√	√	√	√
	√	√	√	×	×
	√	√	×	×	×
	√	×	×	×	×
	×	×	×	×	×
	×	√	√	√	×
	×	×	√	√	×
	×	×	×	√	×
	×	×	×	×	×

5、打印测试

● 所见即所得测试：默认格式下，单据上看到的内容应和预览、打印出纸张的内容一致

● 打印格式测试：通过格式设置后，打印或者预览出的内容的格式应和格式设置的一致

● 多页测试：测试表体多页情况打印到纸张上的内容是否正确，是否页间隔有丢失数据

● 套打测试：套打测试一般是按规定的套打纸进行测试，经过设置可以正确打印到规定的套打纸上，如果怎么设置都打不到纸上的正确位置，那么这就有问题了。

6、格式设置测试

● 显示格式：每种单据都应有一个默认格式，可以自定义格式模板，也可以将自定义模板设置为默认模板

（1）设置的自定义模板为默认模板，然后进入单据是否按设置的默认模板显示

（2）在单据中可以切换各种显示模板，应能正确按切换的模板显示

（3）增加的模板在单据中应可以看到，删除增加的模板在单据中应不能看到，前提该模板不是默认模板

● 单据字段格式设置：可以设置单据中各种非预制的字段属性，比如：长度、是否可编辑、是否必输等，设置后进入单据测试是否按设置的控制。

● 打印格式：每种单据都一个默认格式，可以自定义打印格式模板，也可以将自定义模板设置为模板模板。

（4）设置的自定义模板为默认模板，进入单据打印预览、打印时是否按默认模板显示

（5）默认模板不可以删除，非默认模板可以删除

7、表体计算关系测试

● 主辅计量单位换算：测试主辅计量之间的计算关系是否正确

（1）固定换算率：

◆ 修改数量===>件数＝数量/换算率

◆ 修改件数===>数量＝件数×换算率

◆ 变换其他换算率单位===>件数＝数量×换算率

（2）浮动换算率（以数量为主，数量不变）：

◆ 修改数量===>件数＝数量/换算率

◆ 修改件数===>换算率＝数量/件数

◆ 修改换算率===>件数＝数量×换算率

（3）浮动换算率（以件数为主，件数不变）：

◆ 修改数量===>换算率＝数量/ 件数

◆ 修改件数===>数量＝件数×换算率

◆ 修改换算率===>数量＝件数×换算率

● 数量单价金额换算：通常单据表体单价有含税、不含税之说，因此计算方法不同

（1）表体无税：

◆ 修改无税单价===>无税金额＝数量×无税单价，税额＝无税金额×税率，价税合计＝无税金额＋税额，含税单价=价税合计/数量

◆ 修改数量===>无税金额＝数量×无税单价，税额＝无税金额×税

率，价税合计＝无税金额＋税额，含税单价=价税合计/数量

◆ 修改无税金额===>无税单价＝无税金额/数量，税额＝无税金额×税率，价税合计＝无税金额＋税率，含税单价=价税合计/数量

（2）表体含税

◆ 修改无税单价===>含税单价＝无税单价*（1+税率），价税合计额＝数量*含税单价，无税金额＝价税合计/(1+税率)，税额=价税合计-无税金额

◆ 修改数量===>价税合计＝数量*含税单价，无税金额＝价税合计/(1+税率) ，税额=价税合计-无税金额，无税单价＝无税金额/数量

◆ 修改无税金额===>价税合计=无税金额*（1+税率），含税单价=价税合计/数量，税额=价税合计-无税金额，无税单价=无税金额/数量

8、显示测试

- 单据菜单名称显示测试
- 标题显示测试
- 表头项目名称测试
- 表体列表体显示名称测试
- 表尾显示信息名称测试

9、提示测试

- 输入非法内容后提示信息测试，比如：输入非法或不存在的档案后弹出的提示内容是否正确
- 非法操作提示信息测试，比如：已审核的单据修改弹出提示信息，已关闭的单据弃审弹出信息
- 未设置打印机时进行打印、预览等操作弹出提示信息测试
- 所有的弹出的提示信息需要测试信息的内容是否易懂、是否与需求规格说明书要求一致

10、效率测试

- 进入单据效率：单击菜单后显示单据界面的时间
- 档案参照效率：档案参照后带入的时间
- 保存效率：单击保存按钮后到完成的时间
- 审核、弃审效率：单击审核按钮后到完成时间
- 翻页效率：单击翻页按钮后到显示上下张单据时间
- 所有效率需与规格效率说明书中的一致，不能大于要求的效率值

11、需求验证测试

- 根据需求规格说明书验证产品功能是否实现，是否与需求一致

12、易用性验证测试

● 根据易用性标准说明书验证产品功能是否满足标准，是否符合一致性规范

13、多语测试

● 多语测试主要针对多语环境、多语门户的测试，多语环境是指测试用机是多语系统，多语门户是指门户可以显示多种语言，登陆时可以选择多语中的一种进行登陆

● 多语测试项目是单据菜单多语、标题多语、表头项目名称多语、表体列名多语、表尾名称多语、所有操作弹出的信息多语、右键菜单多语

● 测试以上项目的语言应与登陆的语言一致，且内容应与需求规格说明书要求一致

● 测试单据在此种语言中各种操作是否正常，比如：简体环境下单据可保存，其他语言环境就会报错无法保存。

14、接口测试

● 参照上游单据控制

（1）一对一：一张上游单据参照生成一张本单据

（2）多对一：多张上游单据参照生成一张本单据

（3）多对多：多张上游单据参照生成多张本单据，这里涉及到拆单、拆记录。拆单：是指一张上游单据有多条表体记录，按不同记录生成不同的本单据；拆记录：是指一张上游单据的一个表体记录的数量被拆分生成多张本单据，此处测试比较复杂，需要测试拆单、拆记录后回写是否正确，是否正确回写到上游对应的单据上。

（4）被参照的上游单据不能弃审、删除，只有删除生成的本单据后才能弃审、删除

（5）超量生成控制：如果不允许超量，生成单据的数量大于来源单据是不允许保存，反之可以保存

（6）回写上游单据数据测试：参照生成本单据后会回写上游单据“已生成数量”，测试是否回写的数量正确

● 被下游单据参照控制

（1）一对一：一张本单据生成一张下游单据

（2）多对一：多张本单据生成一张下游单据

（3）多对多：多张本单据生成多张下游单据，这里的拆单、拆记录同上，测试要点同上。

（4）被参照的本单据不能弃审、删除，只有删除生成的下游单据后才能弃审、删除本单据

（5）超量生成控制：如果不允许超量，生成下游单据的数量大于本单据是不允许保存，反之可以保存

（6）回写本单据数据测试：参照生成下游单据后会回写本单据的“已生成数量”，测试是否回写的数量正确

● 联查测试：

（1）联查数据测试：可以向上联查上游单据，可以向下联查下游单据

（2）联查权限测试：联查时测试权限控制是否正确，如果控制数据权限，操作员如果没有对应的操作员、部门、业务员、存货其中一种权限，那么联查时将看不到数据。

15、表体右键操作测试

● 右键显示菜单多少、显示名称是否与要求一致

● 右键操作某菜单是否有反映，是否与菜单的功能一致

● 右键显示多语是否正确

16、表头排序、模版选择操作测试

● 排序条件界面是否正确，多语显示是否正确

● 排序条件界面是否可以正确设置排序规则，界面按钮操作是否正确

● 依据排序规则排序结果是否正确

● 模版显示是否正确（包括多语）

● 模版选择后单据是否按选择的模版显示

17、升级、年结测试

● 单据：

（1）界面显示的信息应与与升级、年结前一致

（2）单据的状态应与升级、年结前一致

（3）单据的格式应与升级、年结前一致

（4）单据的权限应与升级、年结前一致

● 升级、年结后是否可以进行后续操作

（1）单据是否可以被下游单据参照

（2）如果未审核是否可以修改审核操作；已审核是否可以弃审操作

（3）单据是否可以制单、结算、记账操作

（4）如果有审批流：是否可以继续走审批流进行审批；如果存在预算控制，是否可以继续走预算控制审批

（5）未审核的单据是否可以删除、修改操作

（6）是否可以进行逆向操作，逆向操作控制是否正确

启发式风险为基础的软件测试

作者：成韩丽

这是基于风险的测试：

1、制作一个风险高低次序表。

2、对每一个风险进行测试。

3、如果风险消失并且新的风险产生，调整你的测试，使得测试仍然集中在在最新的结果上。

有什么问题吗？现在你知道了什么是基于风险的测试，我就可以集中精力在这片文章中了，这篇文章将会说明为什么你想要这么做和怎样可以做的更好。

一、为什么要进行基于风险的测试？

作为一个测试者，有些事情是你必须做的，这些事情取决于你所从事的项目，你的工作环境等等。但是不论你是做什么的，你的工作都包括找出产品中的重大问题，风险就是一个可能会发生的问题。一个问题越有可能发生，那么它发生后的影响就会越大，与这个问题相关的风险也就越高。因此，基于风险的测试是有根据的。如果你同意这前提，那么你将会非常想知道有关“基于风险的测试”，而不仅仅是觉得是多余的。是否所有的测试都是基于风险的？

要回答这个问题，想想看食物。我们都必须食用食物才能活着，但是如果说我们是“基于食物活着”好像有些奇怪。在一般环境下，我们并不认为我们是靠一顿接着一顿的食物活着的。很多人都不会对我们食用的食物保持相同的记录，或者说都不会谨慎的将我们的日常活动和食物联系起来。尽管如此，当我们吃的过多时，或者对食物过敏时，又或者当我们的食物快要用完时，我们将会明确的对我们下一餐进行很好的计划，这就和风险测试是一样的。

仅仅因为测试是由风险推动的，风险并不是为了组织测试的过程的直接的必需品。标准的测试方法只是含蓄的针对风险有些计划，你可以管理这些风险，通过组织测试功能、需求、结构模块，或者甚至是预先确定的永远不会改变的测试，这尤其当你面对的风险很容易理解或者失败的总风险不太高的情况下是正确的。如果你想更加的确定你是否在适当的时间进行了适当的测试，基于风险的测试将会对你有所帮助，在执行任务的测试本身方面，基于风险的测试关注和证明测试效果，当其他的组织过程需要更多的时间或者需要超过你可以给予的资源，那么就用基于风险的测试吧。

如果你为正在进行测试的产品失败的代价非常高负责任的话，你可能会采用非常严谨的风险分析。这种方法适用于统计模型，或者全面分析风险和失效模式。

我从来没有在过一个这样的项目中，该项目中我们认为对成本进行严格的控制是正确的，因此，所有我所知道的都是我经历过的。在这一方向一本写的很好的、易于理解的书是《安全临界计算机系统》，由Neil Storey所著。还有一本统计测试技术的由John Musa编写的《软件可靠性工程》。还有一种风险分析那些较小已被写入风险序列中的，这种类型的分析一直都是提供给你的，不需要进行统计，我称这种类型的为启发式风险分析。

二、启发式分析

用启发式的方法寻找一个问题的解决方案是一个有效的并且不需要常用的方法。这个方法可以追溯到希腊的哲学家，是George Polya在由他所写的著名的《如何解答它》一书中引进的。“启发式的推理不仅仅是关注结果的和严谨的，而仅仅是临时的和不严谨的，它的目的是发现当前问题的解决方法。”

启发式的方法常常出现在诸如自由问答、建议，或者向导中。一个启发式的检查表并不是和活动检查表一样的，例如，在活动检查表中，你可能会在一个缺陷报告中包括“再生的步骤”，而启发式检查表的目的不是控制你的活动，而是帮助你考虑更多的可能性和关注于问题的各方面。

一个非常好的在软件开发需求分析阶段的启发式方法，看看《探索需求：设计之前的质量》，由Don Gause和Gerald M. Weinberg所编写。

三、分析的两种方法

让我们先来看看一些为了探索软件风险的启发式方法。我认为风险分析无外乎就是由内至外或者由外至内，这两种是相辅相成的方法，每种都有各自的优势。

四、由内至外

首先，了解有关的详细情况，并确定与之相关的风险。通过这种方法，你研究产品，并且不断重复问自己：“怎么可能会出错呢？”。更具体的说，针对产品的每一个部分，问以下的三个问题：

- 漏洞：在这个组件中存在着何种缺点或可能失败的因素？
- 威胁：在这个组件中存在着何种输入或情况使得漏洞可能会触发失败？
- 受害者：谁或者说什么将会被潜在的失败影响，且如果发生情况有多糟？

这种方法需要敏锐的技术洞察力，但又不仅仅依靠你的洞察力。我之前做过使用由内至外风险分析方法最成功的就是与开发者制作的“石头汤”，我带来石头（启发法），他带来汤。

正如这样的情况：在一个典型的分析会议我们找到一个空的会议室有块很大的白色写字板。我问：“这个写字板如何使用？”，开发者于是在上面画了很多压扁的盒子、曲线箭头、不规则的柱体和其他半清晰的标记。他边画边讲有关产品的事情。同时，随着开发者的描述我试图快速在脑海中模拟它的构造。在我想

我明白了那个过程或者说理解了如何测试它，然后我把自己理解的解释给他。那个白板是一个很重要的支撑物，因为在我吸收消化那些信息的时候很容易迷惑。当我不记得解释的思路时，我可以皱眉故作神秘，指着任何图随即的部分，这样说：“我还不清楚这个部分是如何工作的”。

同样的，当我理解了内部构造，我就会找出潜在的漏洞、威胁和受害者。更确切的说，我会让开发者带着以下的问题找出它们：

- 【指着一个盒子】如果在此框中函数失败？
- 永远不会在错误的时候调用这个函数？
- 【指着图的任何一部分】在这里你做过什么误差校验？
- 【指着一个箭头】这个箭头的准确意思是什么？如果它断掉了会发生什么？
- 【指着一个数据流】如果这个信息从这个到那里的时候以错误的状态流出，你如何知道？会发生什么？
- 这个过程能承受的最大负载是？
- 这个过程的外部的元件、服务、状态或者构造都是相互依赖的吗？
- 这里的任何资源或者构造图会不会被其他的过程修改或者影响？
- 这是一个完整的图吗？你没有遗漏什么吗？
- 你把这些组合在一起的时候如何测试他们？
- 你最担心的是什么？你认为我应该如何测试？

这并不是一个完整的问题列表，但它是一个良好的开端。同时，正如开发者所说，我注意听他们是按照信仰还是程序工作。我注意听任何在他的声音中没有把握的或者关注的音调，犹豫不定的或者他选择的词语，这些都可以反映他是否想通了需求、设计或者执行过程中的所有问题。混淆或者模棱两可暗示潜在的风险。当我们识别出一个风险时，我们也会讨论为了评价和掌控那些风险如何测试。

通常像这样的会议持续一个小时，然后我带着对功能的理解离开，以及一个详细的风险列表和有关测试的策略。我所执行的测试就是交谈的结果，不仅仅是致力于风险，而且是驳倒或者证实开发者有关产品的故事。

这个方法有非常好的优势，但是他需要对开发者和测试者高效的沟通技巧，以及主动积极肯干的彼此合作的精神。你可以在开发者不在的情况下致力于这种分析方法，但是你自己就必须担负着学习、建模和分析系统的重担。

有内至外方法是一种风险分析的直接形式，它要求“什么风险和这个事情有联系？”，由内至外方法是由外至内方法的反面，它要求“什么事情和这种风险有联系？”。

五、由外至内

用一开始设定的潜在风险表格和具体的详细的状况进行对比。这是一个比由内至外更普遍的方法，并且更加的容易。通过这种方法，你商定一个预先确定的风险列表，并确定他们是否适用于此时此地。那个预先设定的列表有可能是书面的或者只是以前的一些经验在你的脑海中迸发出的火花。我使用三种列表：

● 分类质量标准

这种分类是为了针对不同种类的需求设计的。如果需求和这种分类的各项指标不符合将会发生什么？如何尽最大的努力去进行合理的测试，以确定他们是否符合“足够好”的标准？

√ 能力：它能够完成所需的功能吗？

√ 可靠性：在所有的需求环境下它会运行良好和抵抗失败吗？

√ 可用性：针对真正使用产品的用户它简单易用吗？

√ 性能：它的运行速度和反应如何？

√ 易安装性：在其目标平台上它能够很简易的安装吗？

√ 兼容性：它能够在外部的部件和配置中很好的运行吗？

√ 保障性：对客户提供产品支持的费用是否经济？

√ 易测性：在产品被测试的时候是否易测呢？

√ 可维护性：它构造、安装或者增加功能是否经济呢？

√ 便携性：它携带到港口或者在别处技术重用是否经济呢？

√ 本地化：用其他的语言出版该产品是否经济呢？

我拼凑起来的这个有很多的来源，包括ISO9126标准、惠普的FURPS的列表和其他一些来源。关于这个列表没有什么权威性，只是包括了我在桌面应用程序测试的时候有用的一些部分。我记得这个列表是用缩写CRUPIC STeMPL。为了记住它，大声说缩写和想象它是罗马尼亚一个冰球运动员的名字。稍加练习你将会在任何你需要它的时候在你的脑海里召唤出那个名单。

● 通用风险名单

通用风险是指对于任何系统他们都是普遍的。这是我最喜欢的通用风险：

√ 复杂：不成比例的巨大、复杂、或令人费解。

√ 新功能：任何事情在产品中都没历史记录。

√ 更改：没有任何已经更改过或者“改善”的部分。

√ 上游依赖：任何事情的失败将导致级联系统的其余部分失败。

√ 下游依赖：什么事情是可能导致其余部分失败的特别敏感的系统。

√ 关键：任何部分的失败可能造成重大的损害。

√ 精确：所有的部分必须满足的要求全部符合。

√ 大众：任何部分将会使用很多次。

√ 战略：任何部分对于你的事业来说都有特殊的重要性，比如一个功能可以让你在竞争中脱颖而出。

√ 第三方：在产品中使用的任何东西，在项目外面开发。

√ 分布：不论分散在任何时间或空间，各个元素仍然必须携手合作。

√ 机动：任何已知的都有很多问题。

√ 最近的失败：任何部分都有最近的失败历史记录。

● 风险目录

一个风险目录是属于一个特定的领域的风险列表，在风险目录中的每一个风险线是这样一个句子：“我们可能会遇到……”。风险目录是由测试者按照相同的技术测试模式一遍又一遍的重复推动的。你也可以将仅仅只是把你在测试期间遇到的问题分类整理后，放在一起形成风险目录。下面有一个安装目录的一部分风险的例子：

√ 错误的文件安装

◇ 临时文件没有清理

◇ 升级后没有清理旧文件

◇ 安装不需要的文件

◇ 未安装需要的文件

◇ 正确的文件安装在错误的地方

√ 文件崩溃

◇ 旧文件替代了新文件

◇ 用户数据文件在升级时发生崩溃

√ 其他应用程序崩溃

◇ 和其他产品进行共享的文件被更改

◇ 属于其他产品的文件已经被删除

√ 硬件配置不正确

◇ 硬件配置对于其他的应用程序是不正确的

◇ 硬件配置没有为安装程序进行配置

√ 屏幕保护程序破坏了安装

√ 没有不兼容的应用程序检测

◇ 目前执行的应用程序

◇ 目前已安装的应用程序

√ 安装悄悄更换或更改关键文件或参数

√ 安装过程很缓慢

√ 安装过程需要用户始终一直关注

√ 安装过程混乱

◇ 用户界面是非正统的

◇ 用户界面非常容易用错

◇ 提示消息和操作指南混乱

一个非常普遍的风险目录例子，查看《测试计算机软件》中的附录A，作者是Cem Kaner、Jack Falk和 Hung Nguyen。

1）确定你想要分析的是哪一个组件或者哪一个功能。你是在查看整个产品、一个单独的组件，还是组件列表？

2）确定你所关注的规模。我习惯使用一般、较高和较低几个级别。任何事情都可以被推测为一个一般的风险，除非我有理由相信它是一个较高的或者较低的风险。使用一个规模对你是非常有意义的，但是要小心模棱两可的范围，或者比其本身显示出更加客观的规模。

3）收集你所需要分析对象的信息（或者带有信息的人）。显然，你为了分析它需要知道一些有关它的情况。当我对一个产品进行由外至内的分析方法时，我会收集任何便于收集的信息，在分析时制定一个标记，然后去向比我更加专业的人询问，而后得到他们的分析评论。做这件事情的另外一个方法是把所有人同时召集到一个房间里在会议中分析。

4）查看每一个列表中的风险范围，并确定其重要性。对于每一个范围询问“在这个范围中我们有问题吗？如果是这样，是多大的风险？”记录你的想法。思考一些可以支撑你的想法的特殊原因。如果你是在一个会议中进行这项工作，你要询问“我们怎样知道这是或者不是一个风险呢？为了做出一个更好的风险预估我们必须要了解什么呢？”。

5）如果有任何其他的风险并没有出现在你的列表，记录它们。特殊的风险将会在这一阶段出现在你的视野中。

6）记录任何影响你分析风险能力的未知因素。在这个过程中，你将会经常感觉很为难。例如，你可能会想知道是否有特定的组成部分，尤其的复杂，也许它一点也不复杂。为了能够确定这些你需要知道什么呢？正如你在分析的过程中，这些可能帮助你一项一项的建立一个信息列表。在某些时候，去获取这些更新你的信息和风险列表。

7）在风险发布时进行复核。以一个所有部分都是同等风险的风险列表而告终这是非常普遍的，这可能的确就是真实情况。另一方面，它可能是你的关注发布是偏斜的，因为你不愿意对要测试什么不测试什么做出艰难的选择。无论你是于哪种风险分布告终，都进行复核，通过几个简单的同等风险的例子，询问这些风险是不是真的平等。考虑到一些风险程度不同的例子，并询问是否真的是有一

定道理，花费更多的时间在高风险上花费较少的时间在低风险上。证实那些重要性的风险分布是正确的。

我推荐这个分析应该包括不同角色的不同类的人参与，例如：技术支持、开发和市场。

六、组织基于风险的测试三种方法

不论你是使用由外至内、由内至外还是一些交叉的方法进行分析，我可以提出三种不同交流方式的风险，并组织关于这些风险的测试：风险检查表、风险/任务矩阵、或者组件的风险矩阵。

● 风险查看表

这种方法可能是组织基于风险的测试最简单的方法。一个风险查看表仅仅就是一个定期检讨风险，你要问自己的测试是否揭示出这些问题。如果你觉得自己没有足够的关于了解风险产品问题的最新资料，那么就做很多的测试，以收集更多的信息。

● 风险/任务矩阵

风险/任务矩阵是由一个有两列的表格组成。左边是一个风险清单；右边是一个可以减轻和其他风险相关的任务的列表。按照风险的重要性排序，将最重要的风险放在最顶端。像这样思考矩阵中的每一行“如果我们担心风险X，那么我们应该在任务Y上投入更多的时间。”

风险/任务模型用于对测试资源进行交涉时是一个非常有用的工具。我喜欢在下面的情况使用这个技术，管理者不接受比较次等的测试，但也不提供足够测试人员测的工作。矩阵有助于使得管理者的期望符合有限的资源。当你在解释没有足够的资源的影响时，这个方法将会很容易得到测试资源。

这个方法的缺点是一些任务可以减轻不止一个风险。另外，一些减轻的任务成本高或者花费很多的时间，它们实际上在帮助发现问题期间产生的价值，不如对项目造成的问题。尽管如此，它仍然是基于广泛项目基础的揭示风险和测试本质关系比较简单的一种方法。

● 组件的风险矩阵

组件的风险矩阵是由三列组成的表格。将产品分解成30或40个区域或组件，这些组件可以是物理代码，例如“安装程序”；功能，例如“输出”或数据，例如“剪贴画库”。换句话说，一个组件就是需要测试的任何事物。在每一行最左边的列，列出一个组件。在最右边的列，列出所有知道的在这个组件中显现出重要意义的启发式风险（如果一个启发式风险对于所有的组件都是平等的，不要重复列出）。在中间列，列出对于一个风险的“较高”、“较低”或“正常”的判断。

组件	风险	启发式风险
输出	正常	分布式的、大众的
报告生成	较高	新的、战略上的、第三方、复杂性、关键性
安装	较低	大众的、可用性、更改性
剪贴画库	较低	复杂的

这个矩阵将会在交流和判定哪个组件将会更加有效的方面帮助你。我有一个常规的规则，那就是较高风险的项将会比平常项产生多于两倍的效果，依次的平常项比低风险的项产生多于两倍的效果。当然，这仅仅是一个近似的取值。

启发式风险也包括在这个表格中，因为它能够在你进行风险判断的时候帮助发现问题，但是请谨记——没有一种特定的关系存在于启发式风险和一个特定判断之间。你可能会发现你处于这样的情况之中，你会为了一个组件的风险高于另外一个而争执，即使第一个组件有比第二个组件有更多的启发式风险推动。风险分析仅仅只是一个大概评估影响风险的因素，并不是准确的计算它们。

随着项目的收益，你将会把焦点关注在测试不同的组件以及它们联系的风险等级上。这个方法的缺点是它仅仅是关注在增加测试需要的高风险上，而不是在降低测试需求的那些因素上。你可能需要将次等因素的风险添加到矩阵当中，但是我发现这样会使得整个矩阵变得太复杂。

七、做所有工作

经常保持这样的想法：你的风险分析将是不完整的和在某种程度上不正确的，并且有可能是错误的。所有你所拥有的仅仅是在项目开始初期对于风险的传闻。随着项目的推进，你获得更多的产品信息，你应该判断你的测试效果和你对风险的最准确评估是否匹配。此外，需要处理次等的风险分析，不要把基于风险的测试作为你的测试工作的唯一形式。抽出至四分之一的精力在不关注风险的地方，例如实地测试、代码覆盖测试或功能覆盖测试。这就是所谓的多元化办措施的原则：使用多样化的方法，因为没有一个启发式的方法可以一直有效。

最后，如果要我选择基于风险的测试工作两个生命要素，我会说是经验和团队精神。经过一段时间，任何产品线或技术都会显现出其特有的问题（如果你注意你在工作时所发现的问题），从中得到学习，然后尽力邀请有不同观点的不同人参与到风险分析的过程中。

如果有一个以风险为基础的测试魔术，它注意到了魔术的迹象和线索，包括所有在你周围的，展现问题所有地方。有些人不自觉地考虑这个问题，也许就好

了。但是当一个问题被你忽视时，因为你没有进行全面的彻底的测试，你将会需要解释你为什么这么做和你做了什么。管理部门可能会猜测是你工作不细心，然后认为所有的测试都是不完整的。在这个时候，如果拥有它的风险列表或者矩阵将是很好的情况，基于风险的测试，你可以向管理者展现你尽力用现有资源做到最好，他们会为了这些赞赏你的。

我感谢Cem Kaner和Brian Lawrence在我写这篇文章时给予的建议和批评。

浅析性能测试中的 3+1 原则

作者：尘中的尘

摘要：

在性能测试中有很多的总结，其中大部分是性能测试工具的使用，对于测试设计总结偏少，且这些总结和具体行业关系较为紧密，导致跨行业测试人员比较难于理解。笔者根据多年测试经验，整理出对性能测试设计、测试执行以及数据分析的 3+1 原则（3+1 原则是指量、全、深+快）。笔者认为只要在测试过程中紧紧把握住 3+1 原则，就具体了做好性能测试的基础。对于性能测试新手而言把握住这些原可以快速入门，性能测试老手则可以作为参考，对性能测试更深入的理解。

关键词：性能测试 测试分析 测试设计 测试用例 测试报告

一、性能测试中“量”原则

测试设计是测试工作的重点，在性能测试设计中通常要考虑很多的因素，如果测试设计遗漏往往会导致重测。笔者根据多年测试经验总结出一个核心，就是“量”。性能测试和功能测试的最基本区别也就是性能测试有一定规模的“量”，比如系统有一定数量的用户，有一定的负荷。

在测试设计中，对“量”的分析上，可以从系统范围和局部范围两个维度进行思考。系统范围是指将整个产品作为被测对象，主要根据产品实际的使用环境来分析涉及的“量”，然后在测试环境中进行模拟，可以模拟实际环境相同中的“量”，也可以根据测试目的按照一定的算法扩大“量”。局部范围是指以产品中某一个局部模块或者功能等为范围，主要靠测试人员的分析得到，根据系统外部输入的量，分析出被测局部范围有哪些“量”，然后再针对此局部范围有针对性的进行测试，这种方法可以弥补系统范围的不足，但需要对系统有较深入的了解。

针对不同的被测对象，涉及到“量”的测试点也不相同，下面根据笔者工作经验和相关文档整理而成，读者可以根据实际情况进一步扩展或细化，这些是性能测试的基础，建议联合有经验的测试人员和开发人员做一次全面的梳理，并形成文档在后续测试工作中使用和检查。

业务量：一个系统往往可以处理多种业务类型，这里的量就是指系统支持的业务类型数量。在产品实际使用时这些业务是并发的，所以在测试过程中也要同时进行模拟。相对于其他类型的“量”而言，业务类型量是比较少的，但可以说是最关键的。比如在交换机系统中可以提供语音电话、短消息等业务，这些业务是可以并发的，因此测试时就需要同时模拟这两种业务。

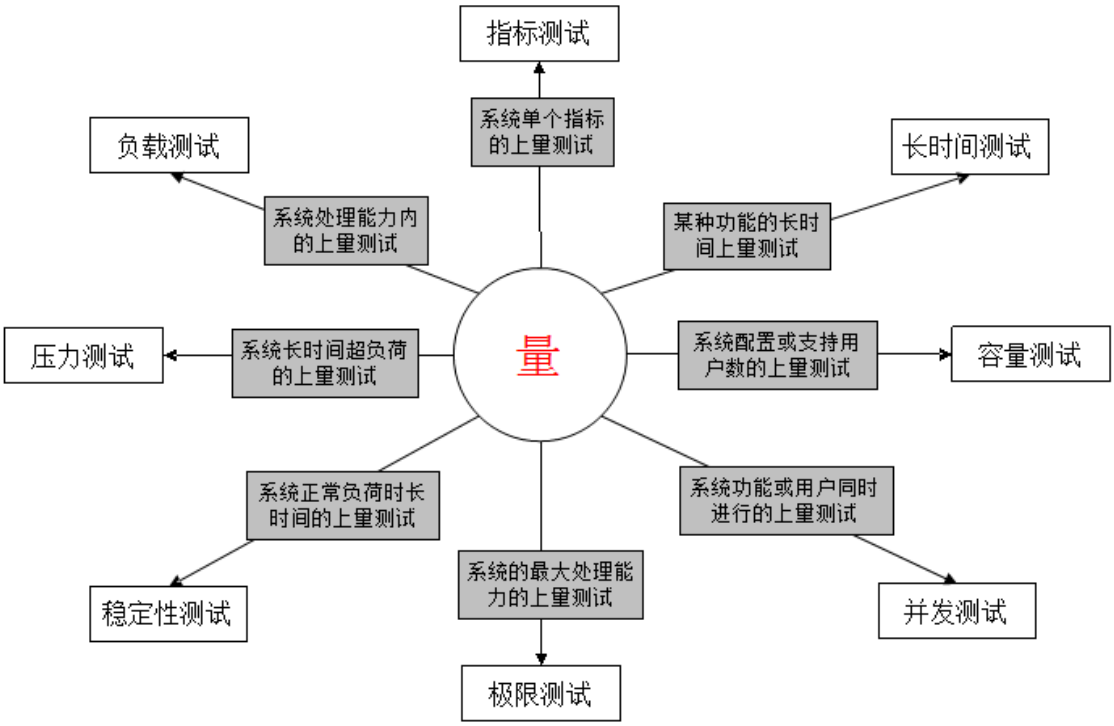
负荷量：这里的量是指系统处理的流量，系统能够提供多种业务或者功能，且可以并发，在实际的使用环境中不同业务的用户数不同，因此其对系统产生的负荷也不一样。在测试时，对于系统范围的测试通常是由外部用户的行为决定的，数据来自于现场收集或根据现状进行的趋势分析。比如登陆网站的用户中，同时进行软件下载的用户有 500 个，同时进行网页浏览的用户有 1000 个。对于局部范围的测试，通常是将外部的负荷转化为对应的局部负荷，然后通过工具进行模拟得到。比如直接通过测试桩的方式进行测试，而不需要模拟外部用户的行为。

配置量：性能测试环境可能没有功能测试复杂，但其环境要求和配置通常都是要求上量的，对于被测系统通常要测试典型配置和最大配置，这里的配置包括软件配置和硬件配置。比如在数据库性能测试中，需要在数据库中添加一定规模的数据，然后测试其增、删、改、插等操作的响应时间。

用户量：用户包括静态用户和动态用户，用户量系统中要有一定数量的这两类用户。静态用户是指用户在系统中驻留，但并发起没有相关的业务，动态用户是指用户正在使用系统功能。比如在通信领域，交换机中会通常有些用户都会驻留在系统中，但并没有发起业务，有些用户则正在进行通话、上网等。前者可以称为静态用户，后者可以称为动态用户。

时间量：这里是指测试时间不能太短，因为系统上量后，有些缺陷需要经过时间或者次数的积累才会爆发。比如软件中的内存泄露问题，通常短时间内内存无法全部泄露，在较短时间内停止测试则会导致缺陷无法被触发。具体测试时间有多长，需要根据被测系统的实际情况而定。

在很多的性能测试总结以及分类中，将性能测试进一步划分为负载测试、压力测试、稳定性测试、极限测试、并发测试、容量测试等等。对于这些分类，笔者并不表示反对，但对于性能测试新手而言，通常不太容易理解和记忆，因为每种分类的差异并不大，而且业界也没有一个确切的定义，通常每个人总结的意思差不多，但细节又都不一样。笔者将经常提到的性能测试分类和“量”的维度相结合，抽取关键因素表示如下图，只要把握住“量”的原则，无论哪种分类都可以很容易的理解。



二、性能测试中“全”原则

笔者这里的“全”主要是针对性能测试用例而言，正规的公司一般都有固化的性能测试模板，虽然每个公司的模板存在差异，但都会包括：预置条件、测试步骤和预期结果三个部分。笔者总结“全”的原则是针对预置条件和测试步骤中的测试结果观察和数据收集而言。

有些公司对性能测试用例要求比较宽松，用例描述是否清晰、全面都依赖于测试人员，这样会导致用例的可继承性差、测试人员和设计人员理解不一致、出现缺陷复现困难、设计遗漏导致测试结果无效等等问题。

预置条件主要是用来说明测试用例是在什么样的背景下执行，结合前面“量”的分析，常见的背景条件包括硬件配置、软件配置、测试工具准备等等。在测试设计时要考虑需要何种背景，是否有必要要求海量配置、复杂环境等，避免浪费人力物力。

用例的测试执行主要是观察系统反应是否符合预期，对于性能测试不仅仅要关注结果，还需要关注过程，对于数据的收集也是这样，最好收集全面详细的过程数据和结果数据。通常系统外部的表现可以通过人来观察，但对于系统内部是否有缺陷就要根据测试数据的分析得到。因此用例描述一定要列举所有的观察点，所有的数据收集点。笔者在多年的测试过程中，发生过多次由于观察不全、数据记录不全导致无法判断测试用例是否通过，最后被迫重新测试的情况。

因为性能测试耗时耗力，成本较大，测试要尽量做到最好一次能够观察到全

部观察点，收集所有数据。为了避免观察和数据收集遗漏，建议根据被测系统特点，清理出需要所有观察点和收集数据。

数据收集通常和观察点是一致的，数据收集的主要目的是避免数据丢失，便于后期测试分析和缺陷定位等工作。根据笔者多年经验，性能测试观察点通常包括两个部分，具体每个产品每个行业都不一样，一是系统自身的统计数据，依赖于产品的开发情况，有些产品提供统计、打印、告等功能；二是测试工具的统计和分析，比如响应时间、吞吐量、CPU利用率，内存利用率，磁盘I/O状况等。

三、性能测试中“深”原则

这里的“深”包括两个维度，一是对系统的了解要“深”，二是对缺陷的分析要“深”。

性能测试通常都是黑盒测试，如果是大的系统，大的公司，分工更细，性能测试人员对系统的理解就更加肤浅。《软件测试经验与教训》中有个观点“黑盒测试并不是基于无知的测试””，笔者在多年测试中深有体会。要想做好性能测试，首先要深入理解系统的架构，系统的机制，其次是根据系统的实际情况，可以有针对性相关产品的知识，因为通常一个系统的组成部分都不会是某个公司自己开发的，都是使用了其他公司的一些产品，这样对这些产品知识的了解也比较重要。

通常性能测试中发现的缺陷都是系统中深层次的问题，这些问题对于开发人员来说，定位比较困难，有些缺陷甚至很难重现。因此测试人员要对测试过程进行仔细观察以及对测试结果数据进行初步分析。这里的分析要有一定的深度，当然不是深入到发现缺陷的跟因，但分析结果能够引导开发人员进行问题的定位。笔者多年的经验中感觉到通常开发人员也搞不清楚哪个地方出了问题，在判断大的方向上测试更占优势。

这样的要求对于一般的测试人员来说都较难达到，需要经过长时间的积累。对于所发现的缺陷，要清楚缺陷的根因和解决方法，经过多次积累之后，分析就会更加准确，同时对后续的性能测试设计能力也有很好的提升。

四、性能测试中“快”原则

性能测试中“快”的原则主要应用在下面两个维度：

一是测试经验的固化，避免测试设计、观察等遗漏。建议性能测试人员根据本文提到的“量”、“全”、“深”的维度将经验固化成模板或者工具，便于经验的传承，减少测试的重复和遗漏。

二是性能测试的自动化，包括性能测试环境构建和测试执行以及测试分析的自动化。相对于功能测试而言，性能测试的基础工作量可以说是成倍增加，这些工作可以说是没有任何产出的，只有性能测试用例的执行才具备真正的价值。因此相对于功能测试而言，自动化对性能测试效率的提升价值更大。

五、说明

本文总结是根据笔者经验整理而成，力求做到通用，但由于行业、产品等的差异以及自身水平的限制，无法一一说到，读者可以根据 3+1 原则自己分析整理。文中错误之处，欢迎批评指针。

顺便谈一下笔者理解的性能测试人员应该具备的能力，包括如下4点：一是性能测试设计方法，比如手机的性能测试、Web的性能测试，这些性能测试都有各自的特点；二是对测试工具的熟练使用，比如业界常用的LoadRunner；三是行业知识以及被测系统的理解，比如IT、金融、通信等；四是相关基础知识的熟悉和了解，比如Linux操作系统、Oracle数据库等。