学写测试用例

原创作者：**FastPoint**　来源：网络

我知道这里有很多朋友刚刚进入测试，为了减少新朋友们对“如何写测试用例”的问题，特别制作了这个教程。我趋向于使用自己开发的应用做案例，这里使用的是 ATM 取款机模拟器，我们使用这个案例来描述书写测试用例的整个过程，如图 1：

图 1

整个界面是比较干净的，跟银行的路边取款机界面没有什么区别，现在我们来看一下如何操作它。首先模拟插卡动作，这个时候 ATM 显示器出现这样的状况，如图 2：

图 2

现在 ATM 显示器提醒我们输入用户密码，继续操作选择小键盘输入用户密码，ATM 显示器显示“******”，点击“确定”按钮，如图 3：

图 3

如果密码正确我们可以进入个人账户主界面，如图 4：

图 4

现在我们随意操作，根据平常我取钱的动作我会先查看一下我当前的账户余额，点击 ATM 显示器右边和“查询”对齐的“<<”help 按钮，进入账户当前余额界面，如图 5：

图 5

哇，我这么挥霍无度的人还有 5000 大元，足够请我的学生吃饭了，呵呵。现在点击和“返回”对齐的“<<”help 按钮，返回个人账户主界面，点击 ATM 显示器右边和“取款”对齐的“<<”help 按钮，进入账户取钱界面，如图 6：

图 6

啊，我要取 2000 元呢，好像当前快捷操作的只有“100”、“200”、“500”，那只好操作“自定义取款”了，点击 ATM 显示器右边和“自定义取款”对齐的“<<”help 按钮，进入账户自定义取款界面，点击小按钮“2”，“0”，“00”，点击“确定”按钮，钱就这么出来了，呵呵。如图 7：

图 7

再检查一下帐户万一这机器不牢靠，扣多了就亏大了，看看：

图 8

跟我一步一步学写测试用例 2

好了，现在案例有了，我们来看看测试用例是什么？下面是对测试用例的关键字解释：

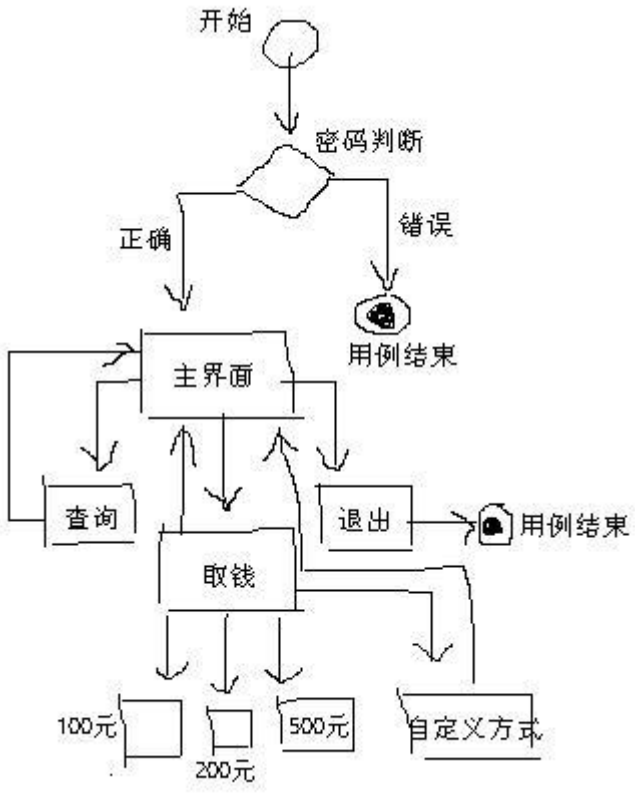
测试用例（Test Case）目前没有经典的定义。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

不同类别的软件，测试用例是不同的。不同于诸如系统、工具、控制、游戏软件，管理软件的用户需求更加不统一，变化更大、更快。笔者主要从事企业管理软件的测试。因此我们的做法是把测试数据和测试脚本从测试用例中划分出来。测试用例更趋于是针对软件产品的功能、业务规则和业务处理所设计的测试方案。对软件的每个特定功能或运行操作路径的测试构成了一个个测试用例。

以上解释引用 pennychueng，大家可以通过这个联结和他联系。

实际上不同的应用虽然都有测试用例，但是它们的侧重点不一样，今天我们面对的是 ATM 取款机，这样某

些测试用例就要设计的非常“与众不同”了。你现在马上就要动手写吗？No，No，好的设计来自于更多的思维，如果是我我习惯在一张纸上先把业务的流程画出来，它可能是这样的：



看起来有点歪歪扭扭的，当然了这是我想得随手画出，其实这里面肯定有某些方面的逻辑错误和遗漏，不过这样做算是我对要测试物粗浅的理解好了。正规流程是我们先找到这个 ATM 取款机的用例（UserCase），也可以是详细设计文档，也可以是需求规格说明等等，反正你要找到描述这个 ATM 取款机业务逻辑和操作逻辑的文档，不然只是靠想象 100%做不好测试，第一份用例是这样的：

ATM 取款机系统

用例规约

登录 ATM 取款机用例

版本：草案

修订历史记录

日期　　版本 说明　 作者

21/Dec/98 草案 草案版本 Fastpoint

目录

登录 ATM 取款机用例

1. 简要说明

本用例允许普通用户登录 ATM 取款机系统。本用例覆盖用户密码后台验证。

本用例的主角是普通用户。

2. 事件流

ATM 取款机初始化完毕插卡后，本用例就开始使用了。

基本流 - 输入用户密码

1. 初始界面，等待用户密码输入。
2. 普通用户点击键盘“1”。
3. 普通用户点击键盘“2”。
4. 普通用户点击键盘“3”。
5. 普通用户点击键盘“4”。
6. 普通用户点击键盘“5”。
7. 普通用户点击键盘“6”。
8. 系统后台验证普通用户密码，正确。
9. 系统切入 ATM 取款机普通用户个人帐户界面。
10. 系统后台验证普通用户密码，错误。
11. 系统显示普通用户个人帐户密码错误，返回步骤 1。

备选流

1. 密码输入错误内部计数超过 3 次，普通用户个人帐户封存。
2. 密码后台验证。

特殊需求

特殊需求将在下次迭代中确定。

前置条件

1．插卡

在本用例开始前，普通用户要登录插卡。

后置条件

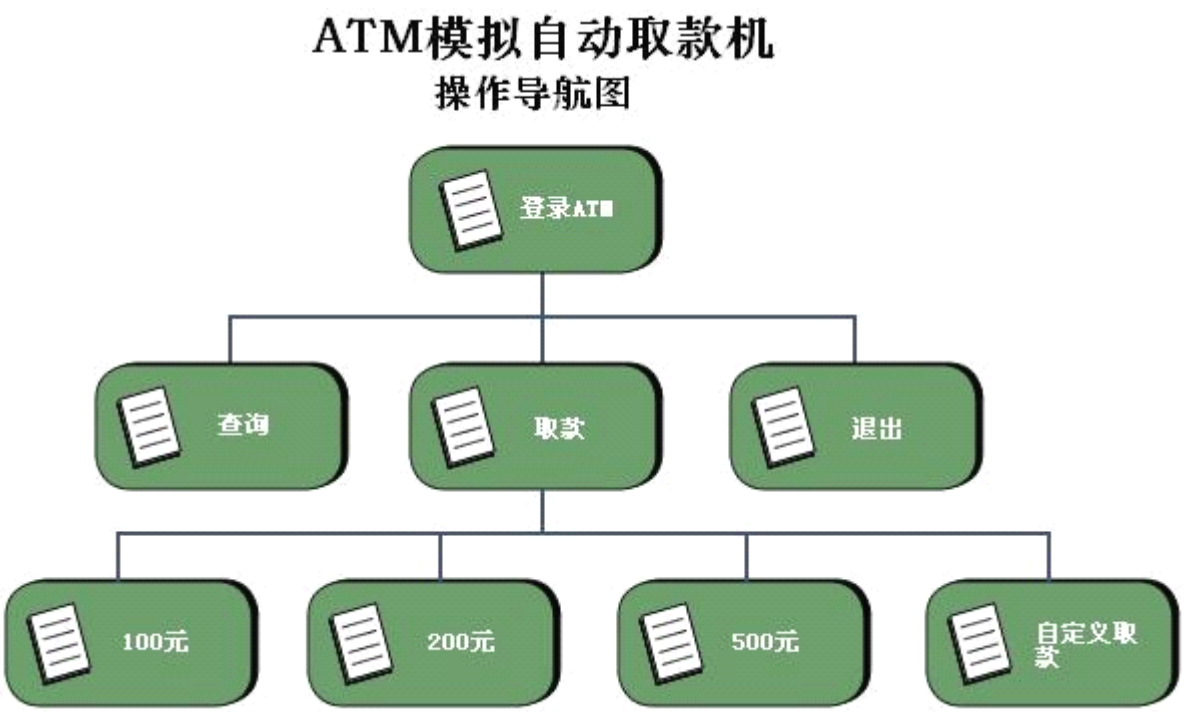
后置条件将在下次迭代中确定。

扩展点

业务用例的扩展点将在精化阶段中确定。

跟我一步一步学写测试用例 3

先来看一下整个 UI 界面的导航图，如图所示：



首先我要说明一下，测试用例也是分很多种的，既然我个人非常推崇朴实测试用例，所以这一次我也朴实的、快速的构造关于 Login ATM 的功能级测试用例吧。

ATM 取款机系统

测试用例

登录 ATM 取款机功能测试用例

版本：草案

修订历史记录

日期　　版本 说明　作者
21/Dec/98 草案 草案版本 Fastpoint

目录

登录 ATM 取款机功能测试用例

1. 简要说明

本用例针对普通用户登录 ATM 取款机系统的功能操作测试。本用例不包含用户密码后台验证测试。

本用例的主角是普通用户，已知密码设定“123456”为正确。

2. 操作顺序

ATM 取款机初始化完毕插卡后，本测试用例就开始使用了。

基本操作顺序 - 输入用户密码

1. 初始界面，等待用户密码输入。
2. 普通用户点击键盘“1”。
3. 普通用户点击键盘“2”。
4. 普通用户点击键盘“3”。
5. 普通用户点击键盘“4”。
6. 普通用户点击键盘“5”。

7. 普通用户点击键盘“6”。
8. 系统后台验证普通用户密码，正确。
9. 系统切入 ATM 取款机普通用户个人帐户界面。

备选流

1. 初始界面，等待用户密码输入。
2. 普通用户点击键盘“2”。
3. 普通用户点击键盘“3”。
4. 普通用户点击键盘“4”。
5. 普通用户点击键盘“5”。
6. 普通用户点击键盘“6”。
7. 普通用户点击键盘“7”。
8. 系统后台验证普通用户密码，错误，等待继续输入。

备选测试数据

序号 测试数据 期望值 实际值
01 123456 T
02 234567 F
03 00.564 E

特殊需求

特殊需求将在下次迭代中确定。

前置测试条件

1. 插卡

在本用例开始前，普通用户要登录插卡。

后置测试条件

后置测试条件将在下次迭代中确定。

扩展点

用户密码输入错误三次，系统返回 ATM 取款机普通用户个人帐户界面。

跟我一步一步学写测试用例 4

如果是一个软件的 UI，那么在它的形成之前肯定有一份特殊的构造文档，我找了半天终于在机器上找到了原来为 ATM 取款机写的《创意设计概要》，文档内容是这样的：

ATM 取款机系统

创意设计概要

登录 ATM 取款机 UI 设计

版本 1.0

修订历史记录

日期 版本 说明 作者
21/Dec/98 1.0 初始版本 Fastpoint

目录

登录 ATM 取款机 UI 设计

1. 简介

1.1 目的

本文档将说明在 ATM 取款机系统的用户界面 (UI) 设计中所采用的标准。

1.2 范围

本文档包括在此 Web 站点中使用的所有 UI 元素。

1.3 定义、首字母缩写词和缩略语

请参见词汇表。

1.4 参考

2. 概述

ATM 取款机的可视化元素采用同真实银行 ATM 机器一致的外观元素，除此之外所有和 ATM 动作联动的硬件功能由软件模拟绘制，大体上，它将保持同真实银行 ATM 机器一致的视觉和操作效果。

3. 直观地功能（风格）

ATM 取款机的用户操作功能，例如插卡、打印票据等功能由软件模拟，整体外观设计趋于保守。排除一切花哨不切实际的元素，例如 ATM 的广告效果。

4. 确定色彩方案

采用冷色和中性色，亮色或暖色可用作强调。

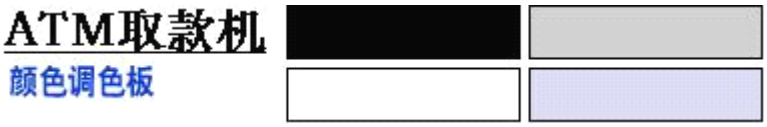


图 1 - ATM 取款机调色板

在 ATM 取款机上，将利用色彩来区分背景和活动业务区域。在 ATM 取款机中，模拟机身背景为标准的灰色。所有的正文文字都是黑色（警告除外，采用红色），相对于各种活动业务背景颜色都为白色。

5. 字体

功能操作字体设定：字体 Dialog,样式无格式，大小 12。
业务提示字体设定：字体 Dialog,样式粗体 ，大小 16。

6. 屏幕布局

屏幕长宽限制在 640 个象素 X 480 个象素，坐标 X 0 、Y 0、宽 640、高 480。活动业务区长宽限制在 270 个象素 X 210 个象素，坐标 X 30 、Y 30、宽 270、高 210。

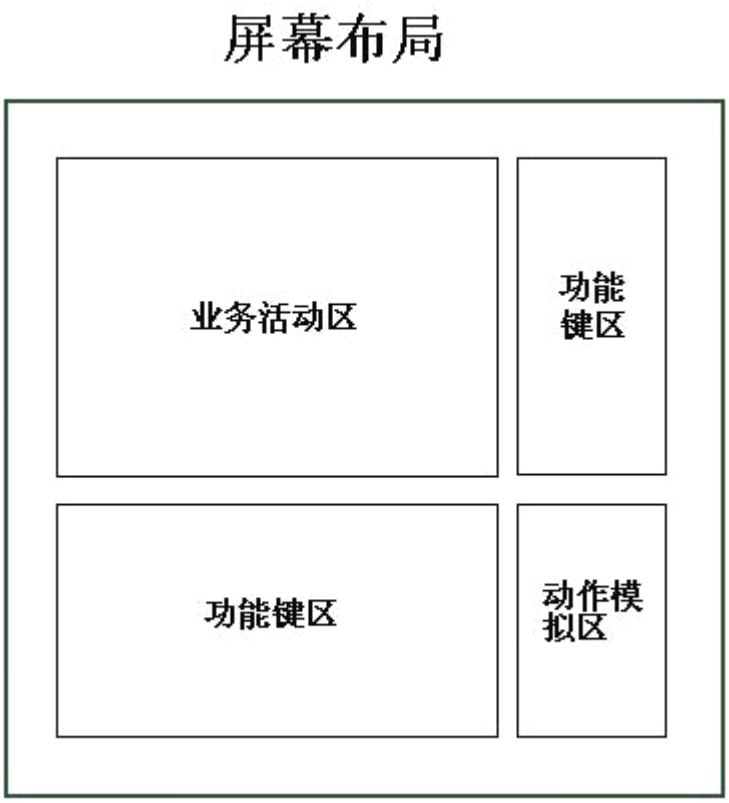


图 2 - 屏幕布局标准

功能区块将包含帮助键、小数字键盘。动作模拟区块将包含所有用户触发行为键。

7. 图形标准

ATM 取款机取消动画和其它图像设置。

8. 其它标准

在下次迭代中确定。

9. 个性化元素

警告字体采用红色。

10. 结论

该 ATM 取款机应该易于操作、浏览，并具有较快的反馈速度。

这个文档就是 ATM 取款机的 UI 设计用例了，它将成为 UI 测试用例一份重要依据。

跟我一步一步学写测试用例 5

现在我们就要针对《创意设计概要》作一些 UI 测试设计，首先对 UI 做层次分析，区分不同的效果 UI 界面整，如图所示：

上面的图片作为正常功能界面，单独抽取 UI 验证元素。


ATM模拟器
用户输入密码错误,剩余2次!
<
<<
<<<
<<<<
1
2
3
取消
4
5
6
7
8
9
.
0
00
确定
插卡(正确)
插卡(错误)
账户冻结
网络正常
网络异常

上面的图片作为异常功能界面，单独抽取 UI 验证元素。

在构造 UI 测试用例之前首先声明一下，UI 测试用例不可能在第一时间内完成，随着项目的扩大 UI 测试用例也相应的做调整。一般来说，做一份公共的 UI 测试用例是非常必要的，可以在每个功能测试用例前导入此份文档，扩大测试覆盖率。

ATM 取款机系统

ATM 取款机 UI 测试用例

公共 ATM 取款机 UI 测试用例

版本：草案

修订历史记录

日期　　版本 说明　作者
21/Dec/98 草案 草案版本 Fastpoint

目录

公共 ATM 取款机 UI 测试用例

1. 简要说明

本用例针对 ATM 取款机系统的 UI 元素测试。本用例非关联 UI 元素外功能验证测试。

本用例的主角是普通用户。

2. UI 验证元素分类

2.1 UI 验证元素分类 A - 父界面窗体

1. 父界面窗体-MixSize-[640, 480]。
2. 父界面窗体-MineSize-[0, 0]。
3. 父界面窗体-Name-ATM 模拟器。
4. 父界面窗体-布局-Center。
5. 父界面窗体-Font-Dialog 12 无格式。
6. 父界面窗体-BackGround-[236,233,216]。
7. 父界面窗体-ForegRound-[0,0,0]。
8. 父界面窗体-Border-（无边框）。
9. 父界面窗体-IcoSet-atm.ico。

2.2 UI 验证元素分类 B - 数字键盘按钮

1. 数字键盘按钮-MixSize-（例子）[39, 27]。
2. 数字键盘按钮-MineSize-（例子）[39, 27]。
3. 数字键盘按钮-Text-（例子）3。
4. 数字键盘按钮-布局-Center。
5. 数字键盘按钮-Font-Dialog 12 无格式。
6. 数字键盘按钮-BackGround-[236,233,216]。
7. 数字键盘按钮-ForegRound-[0,0,0]。

8. 数字键盘按钮-Border-[XPEmptyBorder]。
9. 数字键盘按钮-IcoSet-（例子）Null。

2.3 UI 验证元素分类 C - 功能键盘按钮

1. 功能键盘按钮-MixSize-（例子）[49, 25]。
2. 功能键盘按钮-MineSize-（例子）[49, 25]。
3. 功能键盘按钮-Text-（例子）<<。
4. 功能键盘按钮-布局-Center。
5. 功能键盘按钮-Font-Arial 14 无格式。
6. 功能键盘按钮-BackGround-[236,233,216]。
7. 功能键盘按钮-ForegRound-[0,0,0]。
8. 功能键盘按钮-Border-[XPEmptyBorder]。
9. 功能键盘按钮-IcoSet-（例子）Null。

2.3 UI 验证元素分类 D - 模拟功能键盘按钮

1. 模拟功能键盘按钮-MixSize-（例子）[93, 23]。
2. 模拟功能键盘按钮-MineSize-（例子）[93, 23]。
3. 模拟功能键盘按钮-Text-（例子）插卡(正确)。
4. 模拟功能键盘按钮-布局-Center。
5. 模拟功能键盘按钮-Font-宋体 12 无格式。
6. 模拟功能键盘按钮-BackGround-[236,233,216]。
7. 模拟功能键盘按钮-ForegRound-[0,0,0]。
8. 模拟功能键盘按钮-Border-[XPEmptyBorder]。
9. 模拟功能键盘按钮-IcoSet-（例子）Null。

2.3 UI 验证元素分类 E - 活动业务区

1. 活动业务区-MixSize-（例子）[32767, 32767]。
2. 活动业务区-MineSize-（例子）[270, 210]。
3. 活动业务区-Text- Null。
4. 活动业务区-布局-Center。
5. 活动业务区-Font-Dialog 12 无格式。
6. 活动业务区-BackGround-[236,233,216]。
7. 活动业务区-ForegRound-[0,0,0]。
8. 活动业务区-Border-[EtchedBorder]。
9. 活动业务区-IcoSet-（例子）Null。

2.3 UI 验证元素分类 F - 活动业务区文字

1. 活动业务区文字-MixSize-（例子）[48, 26]。
2. 活动业务区文字-MineSize-（例子）[48, 26]。
3. 活动业务区文字-Text- （例子）200 元。
4. 活动业务区文字-布局-（例子）Center。
5. 活动业务区文字-Font-Dialog 18 无格式。
6. 活动业务区文字-BackGround-[236,233,216]。
7. 活动业务区文字-ForegRound-[0,0,0]。

8. 活动业务区文字-Border-（无边框）。
9. 活动业务区文字-IcoSet-（例子）Null。

2.3 UI 验证元素分类 G - 用户帮助文字

1. 用户帮助文字-MixSize-（例子）[179, 18]。
2. 用户帮助文字-MineSize-（例子）[179, 18]。
3. 用户帮助文字-Text- （例子）注意：只接受 50 元和 100 元的倍数。
4. 用户帮助文字-布局-（例子）Center。
5. 用户帮助文字-Font-Dialog 12 无格式。
6. 用户帮助文字-BackGround-[236,233,216]。
7. 用户帮助文字-ForegRound-[0,0,0]。
8. 用户帮助文字-Border-（无边框）。
9. 用户帮助文字-IcoSet-（例子）Null。

2.3 UI 验证元素分类 H - 用户 Flash

1. 用户 Flash-MixSize-（例子）[416, 250]。
2. 用户 Flash-MineSize-（例子）[416, 250]。
3. 用户 Flash-Text- Null。
4. 用户 Flash 字-布局-（例子）Center。
5. 用户 Flash-Font- Null。
6. 用户 Flash-BackGround-[236,233,216]。
7. 用户 Flash-ForegRound-[0,0,0]。
8. 用户 Flash-Border-[EtchedBorder]。
9. 用户 Flash-IcoSet-（例子）Flash.gif。

3. 特殊要求

下次迭代增加

4. 扩展点

下次迭代增加

跟我一步一步学写测试用例 6

跟我一步一步学写测试用例 5 中展示的是系统 UI 部分的公用测试用例，实际上这块检查内容应该被最先动态化，所谓动态化就是让静态测试用例中所有的检查元素用程序的方法来实现。

对于公用 UI 测试用例的复合测试策略，如图：

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertParentWindows(); Create Simulate-FK-001 Test Programme;
策略	1. 构造一个可以进行 UI 录制的测试驱动程序。 2. 构造一个界面元素 Assert 机制。 3. 构造测试驱动系统内错误抛出机制。

一般复合测试策略定义了执行针对测试项目的可能支持手段，相对于 跟我一步一步学写测试用例 5 其他复合测试策略基本上跟第一份差不多，如图：

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertNumberKey(); Create Simulate-FK-002 Test Programme;
策略	1. 构造一个可以进行 UI 录制的测试驱动程序。 2. 构造一个界面元素 Assert 机制。 3. 构造测试驱动系统内错误抛出机制。

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertFunctionKey(); Create Simulate-FK-003 Test Programme;
策略	1. 构造一个可以进行 UI 录制的测试驱动程序。 2. 构造一个界面元素 Assert 机制。 3. 构造测试驱动系统内错误抛出机制。

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertMockFunctionKey(); Create Simulate-FK-004 Test Programme;
策略	1. 构造一个可以进行 UI 录制的测试驱动程序。 2. 构造一个界面元素 Assert 机制。 3. 构造测试驱动系统内错误抛出机制。

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertActiveBusines(); Create Simulate-FK-005 Test Programme;
策略	1. 构造一个可以进行 UI 录制的测试驱动程序。 2. 构造一个界面元素 Assert 机制。 3. 构造测试驱动系统内错误抛出机制。

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertActiveBusinesFont(); Create Simulate-FK-006 Test Programme;
策略	4. 构造一个可以进行 UI 录制的测试驱动程序。 5. 构造一个界面元素 Assert 机制。 6. 构造测试驱动系统内错误抛出机制。

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertUserHelpFont(); Create Simulate-FK-007 Test Programme;
策略	7. 构造一个可以进行 UI 录制的测试驱动程序。 8. 构造一个界面元素 Assert 机制。 9. 构造测试驱动系统内错误抛出机制。

复合测试策略

约束条件	被测试系统 UI 已经处于用户可视、可用状态
异常抛出	如果约束条件未真，则测试驱动抛出错误信息
场地限制	无
标记	Marker 01-1100-001 & assertUserFlase(); Create Simulate-FK-008 Test Programme;
策略	10. 构造一个可以进行 UI 录制的测试驱动程序。 11. 构造一个界面元素 Assert 机制。 12. 构造测试驱动系统内错误抛出机制。

然后根据测试驱动的编辑器给出对应的测试脚本，一般来说好的测试驱动不用用户重新定义或者自己编写测试脚本，所有的数据全部根据已经存在的测试用例元素抽取，如图：

```
#-------------------------------------------------------------------------------
#   TestCase: TS20051104_TestCase012
#   TestRequirement: TQ05a、TQ05b、TQ06
#
#   TestScript: 01-1100-001
#   Write:Fastpoint
#-------------------------------------------------------------------------------

#启动被测试程序体
Launch ATM.Mock.show( String[] args);

#等待系统屏幕显示窗口
Wait for FrameShowing(ATM);

#取得项目对应测试库
LoadTestDataBase for TCD("Oracle.Project.TestData" );

#Simulate-FK-001
Assert $ UigetObject.MixSize equals "640, 480";
Assert $ UigetObject.MineSize equals "0, 0";
Assert $ UigetObject.Name equals "ATM 模拟器";
Assert $ UigetObject.布局 equals "Center";
Assert $ UigetObject.Font equals "Dialog 12 无格式";
Assert $ UigetObject.BackGround equals "236,233,216";
Assert $ UigetObject.ForegRound equals "0,0,0";
Assert $ UigetObject.Border equals "无边框";
Assert $ UigetObject.IcoSet equals "atm.ico";

#Simulate-FK-002
Assert $ UigetObject.MixSize equals "39, 27";
Assert $ UigetObject.MineSize equals "39, 27";
Assert $ UigetObject.Name equals "[?0-9]";
Assert $ UigetObject.布局 equals "Center";
Assert $ UigetObject.Font equals "Dialog 12 无格式";
Assert $ UigetObject.BackGround equals "236,233,216";
Assert $ UigetObject.ForegRound equals "0,0,0";
Assert $ UigetObject.Border equals "XPEmptyBorder";
Assert $ UigetObject.IcoSet equals "Null";

……

#本脚本执行全部结束
Terminate;
```

实际上动态用例的好处在于对回归测试的支持，前期测试用例设计快速复制和测试用例自动化对加快测试速度起了很大的推动作用。

缺点在于：UI 测试的公共缺点就是当 UI 元素变化后，原来可用的测试脚本将失效，如果变动巨大那么可能造成所有关联 UI 测试全部失效。所以说这里会采用一种新的探测技术“UI 效能监控”，不过跟本学习贴没什么牵扯就不用说了。