目 录

自动化测试工具 sahi 初探.....1

简化合同测试.....8

测试女巫--找到问题的本质篇.....18

测试之前——对测试前质量的考察.....46

以互联网思维做测试.....53

易测性与好的设计之间的关系.....58

第一次编写测试用例总结.....61

测试用例设计方法—错误猜测法.....63

特别的正交测试.....71

自动化测试工具 sahi 初探

作者：郝强

摘要：本文讲述了自动化测试 sahi 的功能特性以及 sahi 的控制器，在 web 应用自动化工具领域我们又多了一项新的选择。

关键词：自动化测试，sahi

1. sahi 是干什么的？

Sahi 简单来讲就是 web 自动化测试工具。其官方网站形容其具备强大的功能，它的主要功能如下：

1. 可以在任何浏览器上录制和回放。

它可以在任何浏览器上录制 web 应用并且可以在任何一种操作系统上进行录制，其同一脚本可以运行在所有的浏览器中。

2. 智能的访问符识别功能。

Sahi 识别元素使用的是简单且稳定的方式，可以动态识别访问符的标识。你也可以使用 ExtJS,ZK,Dojo,YUI 或其它任何的框架来自动化你的应用。

3. 关于 AJAX 不存在超时问题。

Sahi 能够稳定处理非一致页面载入和 ajax, 并且不会出现页面超时失败。使用的脚本很少且易于维护。

4. 内建丰富的报告

报告能够被定制，存储在数据库中，丰富的总结报告以及端到端的报告。

5. 成批快速的并行回放机制

可以在一台机器或分布式运行 sahi 脚本，并且可以成批运行。可以从命令行运行，也可以使用 ant 或其它的持续集成系统中运行。

6. 简单且功能强大的脚本

Sahi 脚本基于 javascript,可以非常容易地与文件系统，数据库，excel, CSV 文件进行交互。还可以从 sahi 脚本中调用 java 代码或库从而使其功能更强大。

7. 内建 excel 框架

使用内建睥 excel 框架可以让 BA 和一些非技术测试人员可以为测试做出更多贡献。进行简单 excel 配置就可以做出能够进行自动化测试的"脚本"。

Sahi 在印度语里是正确或是酷的意思。其官方说法是 Sahi means "Right", "Correct", "Apt" or "Cool" in Hindi。

2. 与其它自动化测试工具功能对比

下表列举了 sahi 开源版本，sahi pro，selenium 的 webdriver 以及 QTP 之间的功能对比，说明一点，对比的出发点是基于 sahi 的功能来讲的：

具备的能力 (Capability)	Sahi OS	Sahi Pro	(Selenium) Webdriver	QTP
支持所有浏览器 (Works on all browsers)	Yes	Yes	Some	Some
支持桌面应用 (Works on desktop applications)	No	No	No	Yes
多浏览器录制支持 (Excellent recorder which works across browsers)	Yes	Yes	No	No
回放可靠性 (Playback reliability)	-	-	-	-
智能访问符 (Smart resilient accessors)	Yes	Yes	No	No
Ajax 和页面载入能够自动等待 （Automatic waiting for AJAX and Pageloads）	Yes	Yes	No	No
同一脚本运行在所有浏览器上 (Same script works on all browsers)	Yes	Yes	Parital	Parital
回放速度 (Playback speed)	Fast	Fast	Fast	Slow
浏览器未获得焦点时脚本停止 （Script breaks if browser not in focus）	No	No	Parital	Yes
可维护性 （Maintenance）	-	-	-	-
很好的录制器和易于识别元素的对象捕捉器 Good recorder and Object Spy for easy identification of elements	Yes	Yes	No	Limited
访问符：对 HTML 实现依赖 Brittle accessors: Too dependent on HTML implementation (use of Xpaths, Css selectors etc.)	No	No	Yes	-
针对每个浏览器分别需要对应的库 Needs separate libraries for each browser (High Maintenance)	No	No	Yes	No
针对每个浏览器新版本需要更新 Needs updates on each new browser version (High Maintenance)	Rarely	Rarely	Yes	Yes
内建详细日志及报告 Inbuilt detailed logging and reporting	Limited	Yes	No	Yes
针对不同浏览器需要编写判定脚本 Needs conditional scripting for	No	No	Yes	Yes

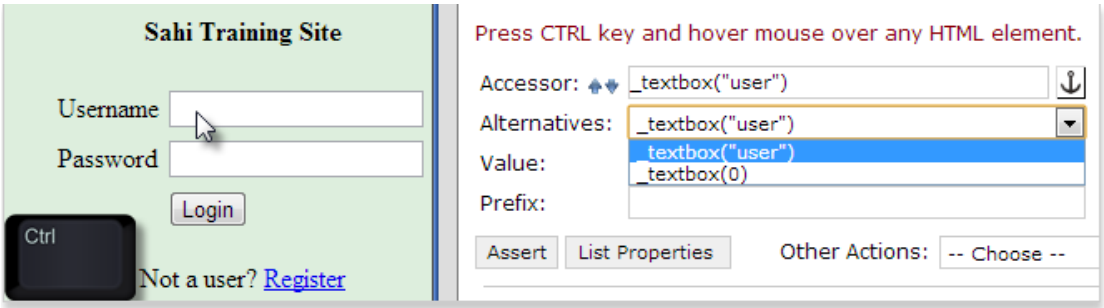
different browsers (High Maintenance)				
编程知识要求 Programming knowledge required (High Maintenance)	Basic	Basic	Advanced	Basic
内建框架 Inbuilt frameworks	No	Yes	No	Yes
测试团队改进要求 Testing team revamp required (High Cost)	No	No	Yes	No
技术支持服务 （Support available）	No	Yes	No	Yes

其实从上表可以看出，单纯看 WEB 自动化测试来讲，sahi 提供的功能确实是很强劲的。而且从实际项目应用的角度来看，确实非常易用。

3. Sahi 的控制器(Controller)

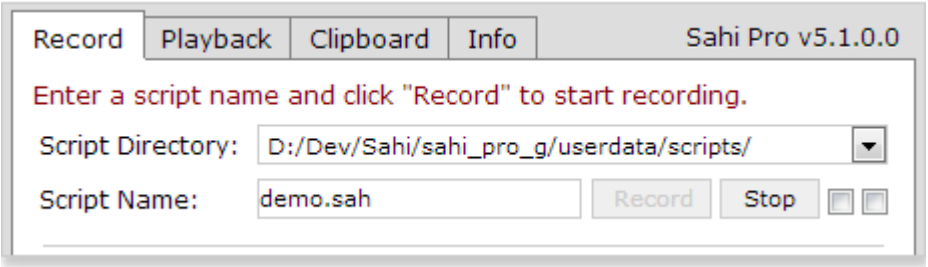
Sahi 的控制器(Controller)功能非常强大，但其主要功能是识别元素,录制脚本，以及回放脚本。

如果想要识别一个元素，你需要按住 Ctrl 键并用鼠标箭头指定你要识别的元素，截取到的访问符如下所示：



我们可以在其下方选择相应的 action,比如说 SetValue，click 等。

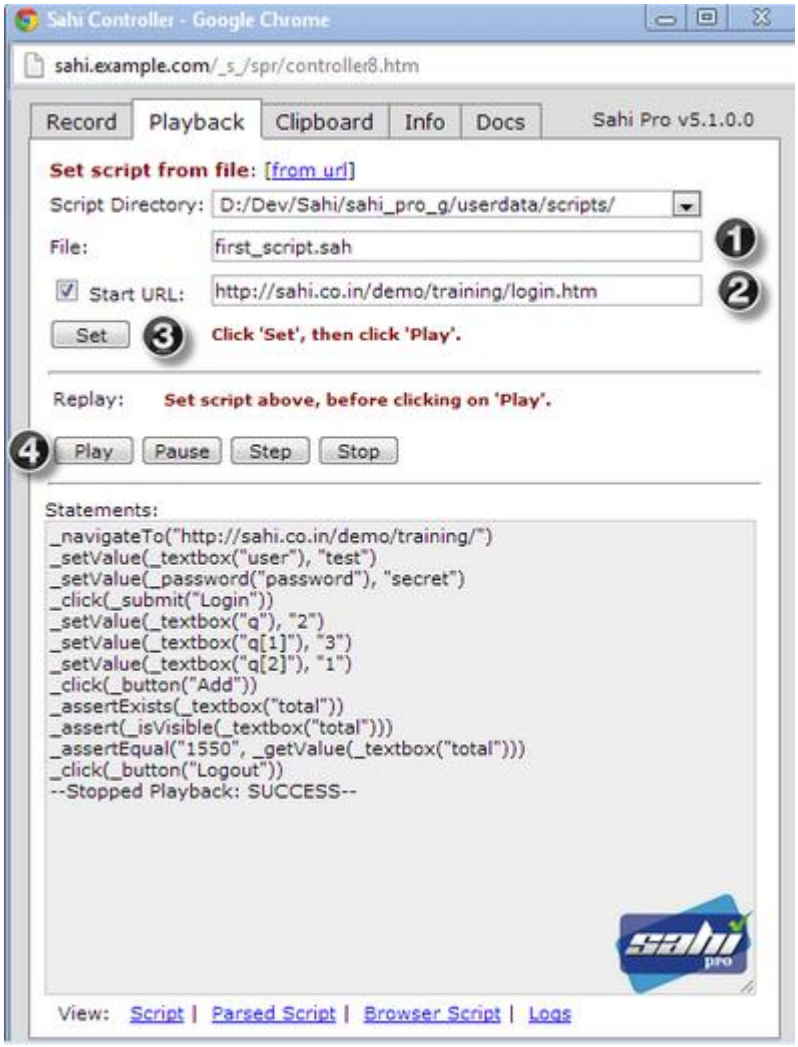
关于录制脚本,首先要选择脚本目录（Script Directory），如下图所示：



之后需要我们在 Script Name 后边的文本框中输入脚本的名字，".sah"是 sahi 脚本的扩展名。

然后点击 Record 按钮，并在 web 页上执行你期望的动作。最后点击 Stop 停止录制。

下面讲一下回放功能，sahi 的回放界面如下图所示：

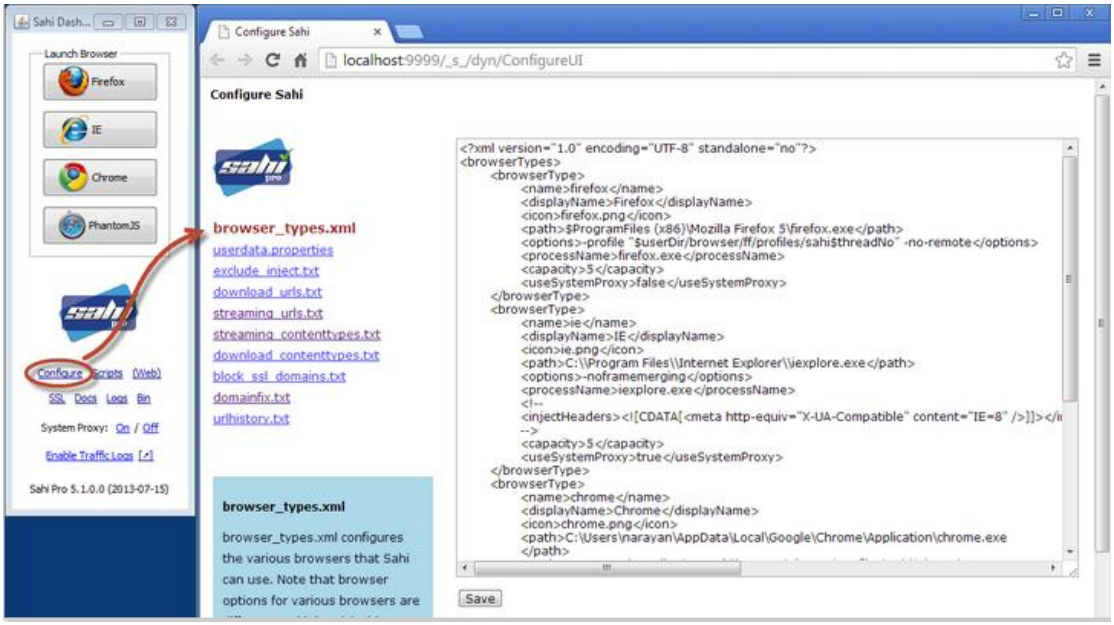


打开这个界面的方法是在页面上按住 Alt 键然后双击。要回放一个脚本我们需要先在 File 后边的文本框中输入要回放的文件名，这里是 first_script.sah。第二步我们要在 Start URL 后边的文本框中输入要测试的 URL 地址。第三步需要我们点击 set 按钮，这时 sahi 会自动将当前的页面导航到您要测试的 URL 地址，如果在第二步中您将 Start URL 前的复选框去掉的话，此步点击 Set 页面不会刷新，且不会导航到要测试的 URL 地址上。最后一步点击 Play 即可以进行回放了。

4. Sahi 配置(Configuration)

Sahi 的面板(Dashboard)列出了所有安装在你系统中的浏览器。Sahi 启动时会尝试在默认安装位置去查找不同的浏览器。假如一个比较特殊的浏览器您已经安装但没有显示在面板上，你就得做如下的配置才可以：

1. 在面板上点击 Configure 链接，这时 sahi 的配置页会被打开。



2. 找到浏览器类型节点<name>firefox</name>

3. 定位到 path 元素并修改其值为在您系统中安装的路径。如 <path>C:\Program Files\Mozilla Firefox\firefox.exe</path>

4. 正常情况下您不需要修改其它值。

5. 点击保存，关闭浏览器并重启 sahi。你将会看到 firefox 显示在你的面板上。

5. Sahi 脚本基础

Sahi 的脚本是基于 javascript 的，但 sahi 中要求变量名必须以$开头,这是唯一不同。

语句：

Sahi 语句就是正常的一行代码，以分号结尾。如：

```
_click(_link("Login"));
```

变量定义：

```
// This is a single line comment
/*
This is a multiline comment.
This has two lines
*/
```

注释：

If 语句：

for 语句：

while 语句：

```
while (condition) {
// statements
}
for (var $i=0; $i<$max; $i++){
// statements
}
}
```

函数(function)：

```
function functionName($parameter1, $parameter2) {
// statements
}
```

一个代码示例：

```
异常处理(TryCatch):
try{
// sahi statements
}catch($e){
// Corrective action
// Can print exact source of error in log
// Can throw the same or another exception
}
```

Sample.sah

```
_include("library.sah");
login("test", "secret");
addBooks(2, 1, 1);
verifyTotal(1150);
_click(_button("Logout"));
Library:
function login($username, $password){
_setValue(_textbox("user"), $username);
_setValue(_password("password"), $password);
_click(_submit("Login"));
}
function addBooks($numJava, $numRuby, $numPython){
_setValue(_textbox("q"), $numJava);
_setValue(_textbox("q[1]"), $numRuby);
_setValue(_textbox("q[2]"), $numPython);
_click(_button("Add"));
}
function verifyTotal($total){
_assertEqual($total, _textbox("total").value);
}
```

作者简介：郝强：软件测试咨询顾问， 12 年的软件测试、测试管理和测试能力改进实践经验。擅长测试团队组建，自动化测试，性能测试以及安全性测试等。

简化合同测试

译者：于芳

合同测试-对在文档中承诺实现的具体的接口和行为的测试，是一种对项目验证至关重要的测试，但是这种测试在扩展多种接口的 Java 类上很难实现。

接口分离法则中，作为可靠的编程法则的一个法则，阐述的观点是客户不应当依赖不使用的接口。也就是说，大的接口应当做到足够小，那样客户才会在实际中使用到这些接口。接口分离法则因此暗示了一层意思：一个应用程序应当有多个小的接口组合在一起来创建更复杂的对象。合同测试认为一个应用程序或者组件的正确性可以通过生产单元测试来保证，也称作隔离对象测试，它通过验证接口的实现情况来确保接口没有违背接口合同对接口的约定。已经有数篇文章在讨论这种测试的优势在哪里；然而，我没有发现任何一篇文章去讲怎样测试合同在几个接口组合到一个单一对象的情况下的实现情况。

合同测试检查由 JAVA 接口和其相关联的文档所定义的合同的实现结果。这个接口定义方法签名，同时文档经常将这种定义扩展到具体定义该接口预期的表现行为。举个例子来说，Map 接口定义了 put()方法，get()方法和 remove()方法。但是可读的文档告诉开发人员如果你要放置一个对象，就必须能够获取它除非调用了移除方法。最后的那句话定义了该合同的两个测试。

另一个例子可以在 Apache Jena Graph 接口里找到，这个接口有一个加法（三倍）的方法。从这个接口上，明显可以看得出这个方法是给该图添加三倍的值。不明确的是在添加三倍的量之后，该图必须报告该加法结果给所有的注册收听者。这个图表合同测试验证了这个行为会发生。

一个更极端的例子是 java.io.Serializable, 这个接口中没有方法需要测试只有文档告诉我们说所有的序列对象必须只能包含序列对象或者实现三个有很具体签名的私有方法（只有两个方法是在 java1.4 之前出现）。除此之外，所有从 Serializable 类衍生的类本身也要是序列化的。去看 Serializable javadoc 找细节。（一个序列化接口的合同测试的例子，在 junit 合同样例中有提供）。对合同测试的使用的基本诉求是他们可以帮助证明代码的正确性。也就是说，如果每个对象接口被定义为一个接口，而每个接口有一个涵盖所有方法及其预期操作的合同测试，那么所有的对象按照接口的定义来看就有模仿其调用的对象的测试，然后运行整个测试套显示对象之间的链接是正常工作并且正确的。

如果我们知道 A 能够调用 B，B 能够调用 C， 那么我们就可以推断出通过转移定律，A 能够调用 C。因此我们可以通过一些工作，证明代码是正确的。

合同测试不会发现配置上的错误。举例来说，如果类 A 使用地图，期望地

图可以接受空的键入值，但是配置说明规定地图的实现不能够接受空的键入值，合同测试将不会检测到这个错误。但是，这个错误是由配置规定中遗漏的需求引起的配置错误。合同测试也不会揭露方法或类的错用。在这个简单的例子中，如果 A 对 B 调用了一个幂函数而不是预期的乘法函数，合同测试就不会发现它。然后，另一边的测试均衡性是---合作测试应当捕捉这种错误。

问题

在最基本的层面上，合同测试说如果你有一个接口 A， 应当有一个测试 AT 测试该接口规定的合同。举例来说，在 Apache Jena 项目中，有一个接口模型，它有一个方法 createResource()。除了返回该资源，执行方法必须确保如果 getModel()方法在返回的资源上被调用，创建该资源的模型会被返回。模型合同测是将会执行该测试任务。

因为 A 是一个接口，AT 必须能够测试任何 A 的实例；因此，它必须是一个有着抽象方法的类，这个抽象的方法能够获取 A 在测试过程中的执行结果或者它是一个具体的有设置器，有能够设置在测试中的 A 的执行结果的设置器的类。

对这个简述的解决方案，假设 ATimpl 是测试下的 A 的具体实现结果，ATimpl 是 AT 的具体实现结果。ATimpl 是一个比较简单的类，它扩展 AT，并且执行抽象的获取或者设置操作。至此，事情变得清晰起来了。然而，与类不同，多个接口可能被一个类执行或者为另一个接口所扩展。这使我们得出一个结论，即一个用例中若有多个抽象测试就必须把这些抽象测试组合起来创建一个完全的合同测试。

假设接口 A 和 B 已被定义，接口 C 是它们的扩展。每个接口都分别有自己的抽象测试：AT，BT，CT。如我们已经看到的，AT 和 BT 很简单并且比较容易写。问题在于 CT，因为 CT 是一个抽象类，基于完全测试的角度它只能从一个基础类衍生，不能既从 AT 衍生又从 BT 衍生。除此之外，遵循 DRY 法则，我们不想再执行 AT 或 BT，尤其是 A 或者 B 的一个变化不会必然地被嵌入执行的 AT 或者 BT 捕捉到。问题变得复杂了，我们认为 C 也许是一个作为一个图书馆或者公用事业项目的一部分发布的一个接口，在这个图书馆或者公用事业项目里应当有整合操作或者扩展那个接口，见图 1 中，由 D 代表的接口。在这种情况下，DT 在 A 或 B 或 C 变化时不应必须录制。变化对上游接口的影响应当限制到只对相关联的测试类有影响。

问题然后变成怎样执行测试套，执行这样一种测试套，提供一个启动类，数个抽象测试能被发现和包括入内的测试套的执行。

解决方案

我提出的解决方案是去创建一个允许标准的 JUnit 测试引擎来发现和整合多

个测试执行到一个单一的测试套中的框架。这个框架引进了三个新的注解和一个类：

@Contract, @Contract.Inject, @ContractImpl, 和 ContractSuite 类:

@Contract 适用于测试类，规定该类是对另一个类的合同测试。所以，在我们的例子中，类 AT 应当有注解@Contract(A.class)。

@Contract.Inject 注解表示该测试类的获取和设置方法。

@ContractImpl 注解适用于具体的测试类的执行。它识别在测试中的类。在我们的例子中，CTImpl 应当有注解@ContractImpl(CImpl.class)。

@ContractTest 注解替换标准的 JUnit @Test 注解来阻止 JUnit 在配置不合理的情况下运行合同测试。

ContractSuite 类与标准的 JUnit @RunWith 一起使用来表明该类定义了一套合同测试来运行。在我们的例子中，CTImpl 应当有注解@RunWith(ContractSuite.class)。

@Contract 和 ContractSuite 实例

下面列出的样例代码是经过轻微地重整的，以实现一组目标。我想定义一个抽象测试（CT），是接口 C 的合同测试。我想要一个那个类 CImplTest 的具体的实现，该类仅仅执行在 CT 中定义过的测试，还有一个具体的 CImpleContractTest 测试，执行像 AT 和 BT 测试那样的 CImplTest 测试。

假设 A 和 B 有类似的 AT 和 BT 类通过@Contract 注解与其相关联。当 JUnit 运行 CImplContractTest 测试时，它使用 ContractSuite 类。那个类发现所有被 CImpl 实现的接口然后定位它们的相关联类。它使用由 ContractImpl 注解规定的接口 C 的实现。

```
C, extending A and B
public interface C extends A, B {}
// a single abstract CT test -- tests just the C interface (not A or B)
@Contract(C.class)
public abstract class CT<T extends C> {
...
}
// implements the single CT test -
// Tests just the implementation of interface C
public class CImplTest extends CT<CImpl> {
...
}
// implements the suite of CT tests
// Tests interface C
// as well as the A and B interfaces
@RunWith(ContractSuite.class)
@ContractImpl(CImpl.class)
public class CImpleContractTest{}
```

生产者介绍

在前面的讨论中讲到引入测试中的类，合同测试系统实际上使用了一个 IProducer 接口来定义一个可以创建测试中类的新实例的对象，并提供了一种机制来在测试执行后做清理工作。@Contract.Inject 注解被用在一个为该测试类返回 IProducer 的方法上。这个接口定义如下：

```
public interface IProducer<T> {
public T newInstance();
public void cleanUp();
}
```

实例延伸

@ContractTest 注解被添加上来是因为几个测试执行者试图举例说明并且执行基本的测试类，而其他人不想。（看起来 JUnit 类应当使用它们自己的合同测试。）对这个注解的使用阻止了这些测试执行者汇报假的失败信息。

```
// define the contract test
@Contract(C.class)
public abstract class CT<T extends C> {
private IProducer<T> producer;
@Contract.Inject
protected abstract IProducer<T>
setProducer(IProducer<T> producer) {
this.producer = producer;
}
protected final IProducer<T> getProducer() {
return producer;
}
@After
public final void cleanupCT()
{
producer.cleanUp();
}
@Test
public void testCMethod() {}
}
// a test that only runs the contract test
// This is useful for debugging the contract test.
```

```
public class CImplTest extends CT<CImpl> {
public CImplTest() {
setProducer( new IProducer<CImpl>() {
@Override
public CImpl newInstance() {
return new CImpl();
}
@Override
public void cleanUp() {
// does nothing
}
});
}
}
// A contract suite.
// run as a contract suite
@RunWith(ContractSuite.class)
// is the suite for CImpl
@ContractImpl(CImpl.class)
public class CImplContractTest {
// the producer for the CImpl
private IProducer<CImpl> producer =
IProducer<CImpl>() {
@Override
public CImpl newInstance() {
return new CImpl();
}
@Override
public void cleanUp() {
// does nothing
}
};
```

```
// method to inject our test instance into test classes
@Contract.Inject
public IProducer<CImpl> getProducer() {
return producer;
}
}
```

当 JUnit 运行 CImplTest 测试的时候，只有 testCMethod()方法会运行。然而，当 CImplContractTest 类运行的时候，所有的 AT，BT 和 CT 测试都会运行，使用来自 CTImplContractTest 的生产者来创建一个 CImpl 的实例，这个实例是 C，B 或者 A 的实例，视测试而定。

实际运用

前面已经讲过，Apache Jena 有一个叫做 Graph 的接口。此时，Apache Jena 实验性新测试在用 Junit 合同包来测试其框架和为整合者开发 Graph 接口新实例提供打包的测试编写。

Graph 是一个 Jena 项目的扩展点，它使得整合者能够快速添加额外的存储类型和策略。期望整合者将创建这个类的实现，因为这个类在 Jena 应用程序架构中扎根很深。

那么执行者正确地执行该接口的合同测试就是必需的了。目前 Apache Jena 团队提供集合供执行者几个运行的测试。

然而，没有一个明确的测试用例是测试完整的 Graph 合同的。如果实验性测试为 Jena 项目接受的话，那么他们会提供一个 GraphContract 图表合同测试和一个抽象的运行 IProducer<Graph>的 GraphProducer 图表生产者。有这个抽象的 GraphProducer 测试的原因是有些图表必须关闭来使他们停止工作，但是该测试接口可能产生数个图表。

AbstractGraphProducer 追踪那些被创建的图表然后在清理过程中将他们关掉。它也有扩展点来处理图表关闭后的另外的操作。

AbstractGraphProducer 代码类似于下面的这些代码：

```
public abstract class AbstractGraphProducer<T extends Graph> implements
IProducer<T> {
/**
* List of graphs opened in this test.
```

```
*/
protected List<Graph> graphList =
new ArrayList<Graph>();
/**
* The method to create a new graph.
*
* @return a newly constructed graph of
* type under test.
*/
abstract protected T createNewGraph();
@Override
final public T newInstance() {
T retval = createNewGraph();
graphList.add(retval);
return retval;
}
/**
* Method called after the graph is closed.
* This allows the implementer to perform
* extra cleanup activities, such as deleting
* the file associated with a file-based graph.
*
* By default this does nothing.
*
*
* @param g The graph that is closed
*/
protected void afterClose(Graph g) {
};
@Override
final public void cleanUp() {
for (Graph g : graphList) {
if (!g.isClosed()) {
```

```
}
afterClose(g);
}
graphList.clear();
}
}
```

这个图表接口的执行者只需要包含 Jena 测试类到到他们的项目测试中然后编写一个简单的合同来运行测试类。

```
// A contract suite.
// run as a contract suite
@RunWith(ContractSuite.class)
// is the suite for MyGraphImpl
@ContractImpl(MyGraph.class)
public class MyGraphContractTest {
// the producer for the MyGraph
private IProducer<MyGraph> producer =
new AbstractGraphProducer<MyGraph>() {
@Override
public MyGraph createNewGraph() {
return new MyGraphImpl();
}
};
// method to inject our test instance into test classes
@Contract.Inject
public IProducer<MyGraph> getProducer() {
return producer;
}
}
```

运行这个测试会执行所有使用与 MyGraph 类的合同测试。如果 Apache Jena 更新了他们的测试用例，他们将会被执行。MyGraph 开发者不需要知道 Graph 接口扩展了 GraphAdd，也不需要开发者关心那个图表实现的接口的组织架构。与开发者相关的合同是 Apache Jena 会提供能测试图表接口等级分类的合同测试。 对开发者来说，优势在于他/她不需响应接口的重构行为，图表签名方法变

化的情况除外。

而且，因为合同测试来源于开发接口的团队，开发者可以比较确定测试是正确的，而且如果代码通过了测试，就将会在环境中正确地运行。

测试女巫--找到问题的本质篇

作者:王平平

摘要： 将统计学_6 sigma 应用到测试技术中，需要使用的 6 sigma 工具如下 Minitab，DOE(Design of Experiments)，主效应图，交互作用图，柏拉图，相关性。

关键字：测试技术，统计学，6 sigma

一、 前言：

女巫，就是左手里拿着一个魔法杖，右手拿着一个水晶球。如遇到很难解决的问题，念一串咒语，挥舞一下魔法棒，问题就解决了！如想预知未来，看看水晶球就都晓得了!！麻烦看客使用 Kimi 的口气："好酷耶！"是的，我如果是测试女巫该有多酷！姐姐我一直不喜欢数学，包括相关的学科例如统计学。但是 7 年前公司组织学习 6 sigma，在听完老师在介绍 6 sigma 之后，立刻非常感兴趣，哎呦喂，这不是就是女巫的魔杖以及水晶球吗？通过数据的搜集，然后使用专业的工具，使用特定的方法进行分析，就能找到问题的本质原因（这就是魔法杖）；根据当前数据的分析甚至可以预测以后可能发生的事（这就是水晶球）。老师还举了一个特别典型的例子，某汽车著名品牌的售后得到一个很奇怪的投诉：一个用户开车去买冰激凌，每周六去买冰激凌，有时车子启动就需要两次才可以成功；如果不是周六去买冰淇淋，车子的启动就不存在这个问题听起来像是一个恶搞，车子难道因为不喜欢周六去买冰淇淋吗？但是此公司并没有对于这个奇怪的现象置之不理。而是派去了测试工程师拜访客户以及冰淇淋店去搜集各方的资料，经过测试工程师整理各方资料，并使用 6 sigma 汇总分析才知道：原来本质的原因是：

客户周六有时购买香草口味的冰淇淋，有时购买其它口味的冰淇淋，而香草口味的冰淇淋在周六需要等待很久的时间，其它口味的冰淇淋是立等可取的，所以还是与发动机熄灭多久时间后重启发动有关系。这个例子与我们的工作有很大的相似处，对于我们测试中发现的 bug，不同的人员对于找出 bug 产生的真正原因差别真的很大。有的测试人员就只能将问题定位到类似周六买冰淇淋的深度；但是有的测试人员就可以进一步分析，将此问题定位到类似发动机熄灭时间的深度。原来可以有一个这样的学科可以帮助我们快速准确的定位找到一个问题的真正原因，这实在是一个太好的学科：因为不仅给大家提供了目标，而且还给大家提供了如何研究进而如何找出影响实现此目标的真正原因（这就是女巫的魔法棒^_^）。而且可以根据当前数据的分析可以对未来的工作有一定的借鉴作用（这就是女巫的水晶球^_^）。搞清楚了上述背景，顿时对学习 6 sigma 充满了兴趣！

6 sigma 中工具和方法很多，我们这次主要根据找 bug 真正原因为例，介绍

DOE，柏拉图，主效应，交互作用，相关性这些方法，并着重介绍如何将这些方法应用到我们软件测试中。

二、 上述这些方法的基础知识介绍

1. DOE 介绍

1) DOE 基础知识：

DOE 的全称为 Design Of Experiment,故名思议对于试验进行合理的安排，以较小的试验次数，较短的试验周期，获得理想的科学的值得信赖的结论。它是一套方法，使用此方法以及相关工具可以研究多个因子与响应变量的关系，从数学角度可以这样理解 Y=（X1，X2，X3）即 X 即为因子，Y 即为响应变量。即找到真正影响 Y 的 X 是什么。

一言蔽之：通过它你可以根据你的想法来创造有意义且有效率的试验。

2) DOE 方法介绍

DOE 是一套流程供 Follow，以下简单介绍一下这个流程如下：

a) 确定目标

在产品设计或者制造时，发现某些指标不能满足我们的需求；且运用我们的经验是无法准确找到哪些原因导致这些问题的。这时候需要使用 DOE。

如果从软件测试角度来说明，我们发现一个问题（即 bug），会很影响到产品的品质。但是我们发现这个 bug 不是每次都能复现，因为软件工程师需要让我们找到这个 bug 的复现步骤，并抓取 Log。

我们通过自己定位 bug 的经验去尝试找 bug 的步骤，但是很难找到这些具体的步骤，这时就可以考虑使用 DOE 协助我们。

b) 分析现有的流程

我们如果选用了 DOE 进行分析，就要考虑当前的工作流程是否合理，而不能只着眼于当前的问题能否尽快解决。DOE 的分析是比较透彻的，我们应该将它的战果扩展到流程这样的深层剖析层面上来，要思考为什么之前的工作流程不能解决当前的难题，当前的流程到底存在什么问题。

c) 收集数据并筛选因素

对现有流程进行了思考，接下来我们就要考虑筛选因子，即找出可能的 X 是什么。这一步尽量利用之前已有的数据，尽量减少试验的成本。在已经知道可能因素的基础上，还需要进一步根据这些因子进行搜集数据，分析，进而进行进一步的筛选，就像搜寻犯罪分子一样，要一步步的缩小疑犯的范围。注意此步骤会有一定的循环，直到你认为已经找到了真正的原因。

d) 分析实验数据

既然 X 已经确定了，我们就可以使用工具对各个 X 进行分析，进一步确认

这些 X 之间是否存在交互作用，哪个 X 的影响力最大等。

e) 回归测试

即基于上述的反复试验，我们已经找到了 X 且也知道哪些 X 对 Y 的影响最大，我们就要重新做个试验确认我们的研究成果是否正确。这一步很重要，必须要验证一下我们的研究成果，这才是一个完整的 DOE 试验，否则即使你找到了真正原因，都会存在一个疑问：这个原因真的是本质的原因吗？

f) 制定新流程

问题已经解决了，我们就要开始反思，这个解决方法对于现有的流程是否有借鉴作用，进而制定一个新的流程。

2. 柏拉图

它是一个寻找影响产出（Y）的原因，用从高到低的顺序排成矩形，用于很直观的看出来哪个因素对产出（Y）的影响最大。不知大家听说过 80/20 法则没有（有时也成为二八原则），即 80%的结果取决于 20%的原因。使用此工具就可以将关键因素筛选出来，也就是说柏拉图式寻找主要因素的攻击和方法。

3. 主效应

它的作用就是：考虑单个因子对结果的影响。例如减肥此事件：

响应变量 Y 为减肥因子为：A 因素(分两个水平：吃零食和不吃零食)

对于 A 的主效应就是：固定 B 的因素的条件下，只看吃零食和不吃零食对减肥效果的影响。

B 因素(分两个水平：运动和不运动)：固定 A 的因素的条件下，只看运动和不运动对减肥效果的影响。

我们找 4 个都是 200 斤的胖子进行试验，分别接受试验一个月结果如下：

胖子 1　　吃零食且运动的人：180 斤

胖子 2　　不吃零食且运动的人：100 斤

胖子 3　　吃零食且不运动的人：250 斤

胖子 4　　不吃零食且不运动的人：150 斤

对于 A 的主效应就是只看是否吃零食对于减肥的影响

对于 B 的主效应就是只看是否运动对于减肥的影响

B1 的意义定义为运动　A1 的意义为吃零食

B2 的意义定义为不运动　B2 的意义为不吃零食

A 在 B1 的效应的意义就是：运动时，不吃零食 100 斤；吃零食 180 斤。

这时就可以看出 A 对于"减肥"此响应变量 Y 的影响：不吃零食更有助于减肥。

A 在 B2 的效应的意义就是：不运动时，不吃零食 150 斤；吃零食 250 斤

这是就可以看出 A 对于"减肥"此响应变量 Y 的影响：不吃零食更有助于减肥。

B 的效应的研究同 A，这里不再赘述。

4. 交互作用

即把两个不同变量的数据在同一个图上显示出来，看其变化趋势是否有关联，比如同增同减，或者变化趋势相反。当一个自变量的效果在另一个自变量的每一个水平上不一样时，我们就说存在着自变量的交互作用。

此时两个线会交叉；如无关联，则两个线是几乎平行的。

我们还是以上面用于说明主效应的有关减肥的例子来说明交互作用的例子：

主效应只是研究单个因子对于响应变量 Y（减肥）的影响，没有研究两个因子（是否吃零食和是否运动）是否有交互作用，如果两个有交互作用则说明 A（是否吃零食）在 B1（运动）和 B2（不运动）的表现是不一样的，即：在 B1 的条件下是不吃零食，对减肥有作用；但是在 B2 的条件下是吃零食，对减肥有作用。

这种情况下对于因子的研究就不能仅仅限于主效应的研究了，需要将此两个因子的交互作用(即运动*零食)作为第三个因子，使用 DOE 进行更为复杂的分析。

5. 相关性

是衡量两个变量之间的关系的一种方法，衡量两个变量之间的关系的方法有两类：第一类是函数关系，即当变量给定之后就有唯一的 Y 与之对应。第二类就是相关性，即变量与变量之间存在着非确定性的依赖关系

即两个变量存在联系，但是是非确定性的依赖关系，如一般来说身材比较高的人体重就会比较高，但是这并存在这样的函数关系，函数关系是：身体高的人，体重一定会高。

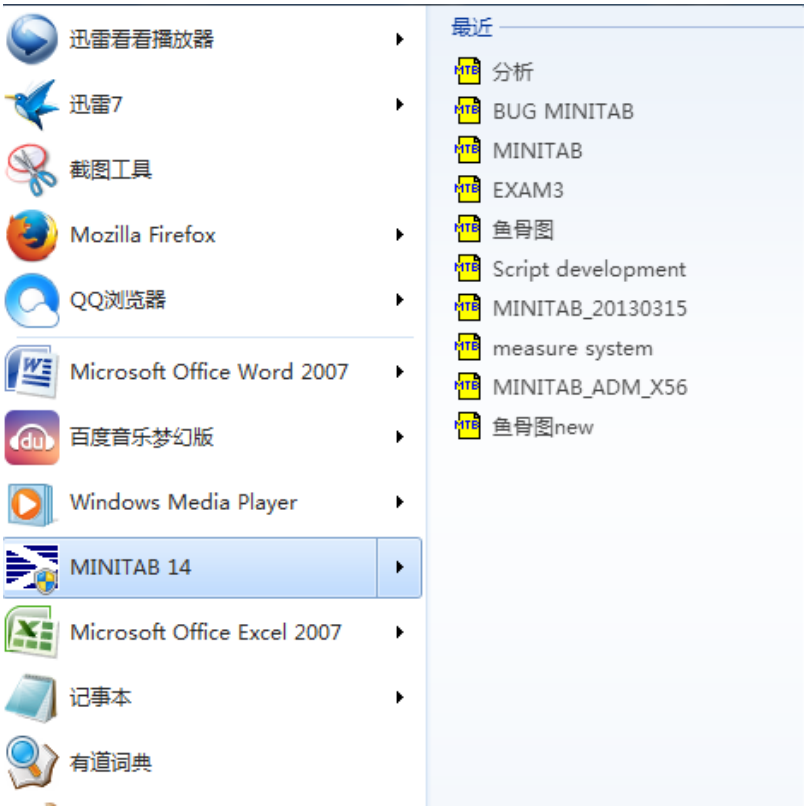
正相关：即相关系数为正数，说明一个变量随着另一个变量的增加而增加。

负相关：即相关系数为负数，说明一个变量随着另一个变量的增加而减少。

三、 6 Sigma 常用工具介绍

1. 柏拉图

这个功能是软件 Minitab 上有关"质量工具"的其中一个工具(Minitab 是 6 sigma 实施的一个软件工具)，安装成功的 Minitab 如【图 1】



【图 1】

以下文字即为介绍如何使用柏拉图

1) 搜集数据并将数据输入工作表中，如【图 2】

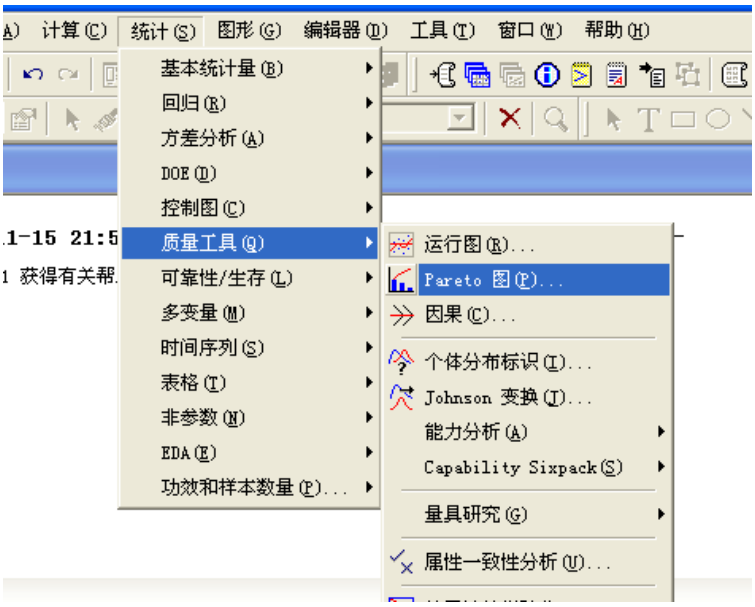
即生产出来的某类产品有多种不良，将这些不良种类以及相应的数量填入工作表中，注意：柏拉图分析至少要有 4 个以上因子（例如【图 2】中的"项目"至少要有 4 个以上，因子太少，分析没有实际意义）

工作表 1 ***

↓	C1-T	C2	C3
	项目	数量	
1	缺角	56	
2	划痕	87	
3	变形	24	
4	污染	45	
5	虚点	12	
6	其他	9	

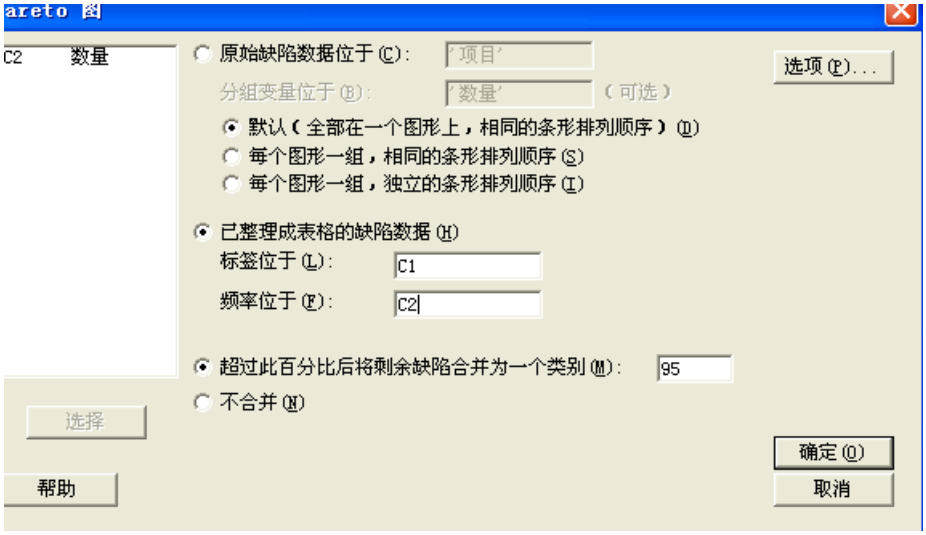
【图 2】

2) 选择 Minitab 中的"统计"->"质量工具"->"Pareto 图"



【图 3】

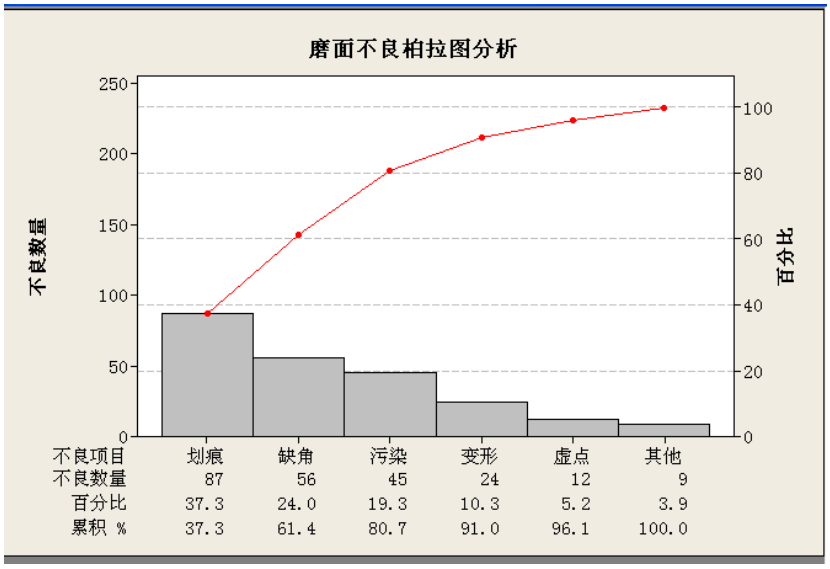
3) 出现如【图 4】界面，并点击"已整理成表哥的缺陷数据"标签位于此栏位设置为"项目"，频率位于设置为"数量"，点击"确定"按钮



【图 4】

4) 可以得到如【图 5】要会解读这张图：百分比即为说明各个因子的不良数量所在的累积百分比，也可以查看"累积 %"此栏位。

从此图可以看出：划痕，缺觉以及污染这些因子导致的不良数量占所有不良数量多的 80.7%。这样就找到了关键因子是什么。



【图 5】

2. 主效应图

单个因子对响应输出（即 Y）的影响，直线的斜率越大，对输出（即 Y）的影响越大。

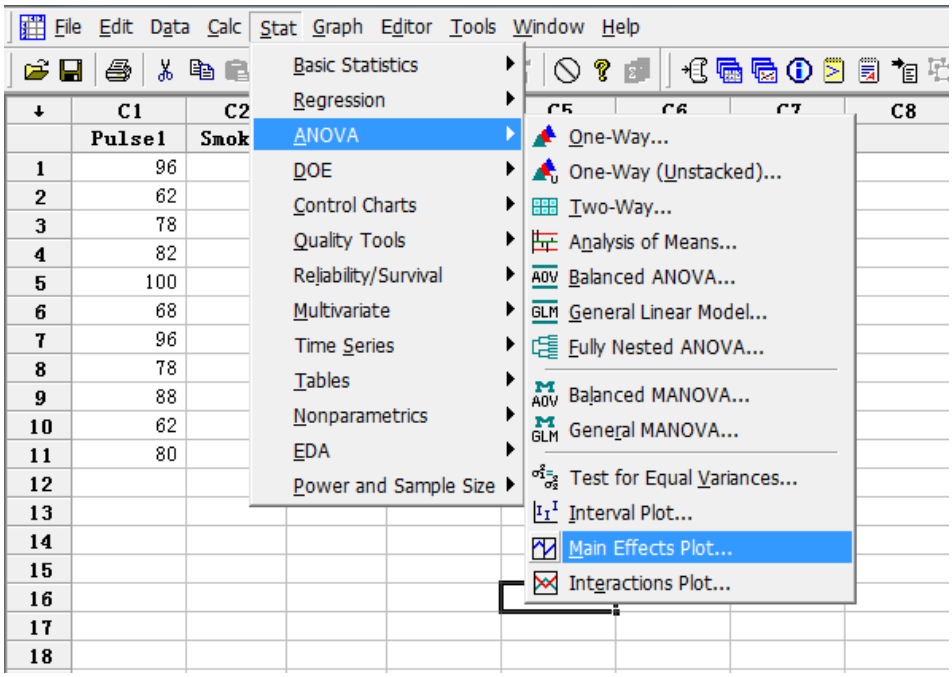
例如我们需要研究一个抽烟对人类的脉搏是否有影响这样的课题，

搜集如【图 6】的资料即 11 个人的脉搏资料，smokes 中的"1"代表不抽烟，"2"代表抽烟。

C1	C2
Pulse1	Smokes
96	2
62	2
78	1
82	2
100	1
68	2
96	2
78	2
88	1
62	1
80	2

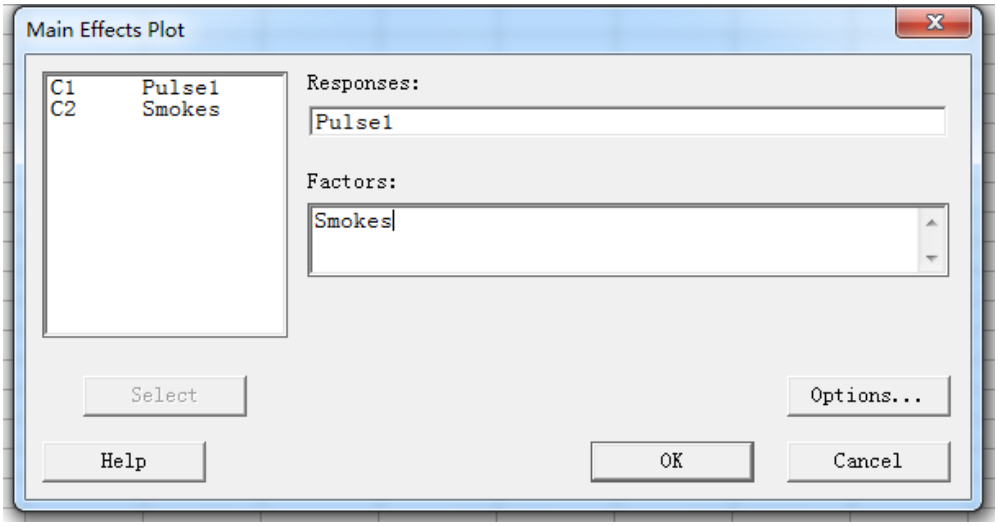
【图 6】

1) 选择"统计(Stat)"->"方差分析(ANOVA)"->"主效应分析(Main Effects Plot)"如【图 7】



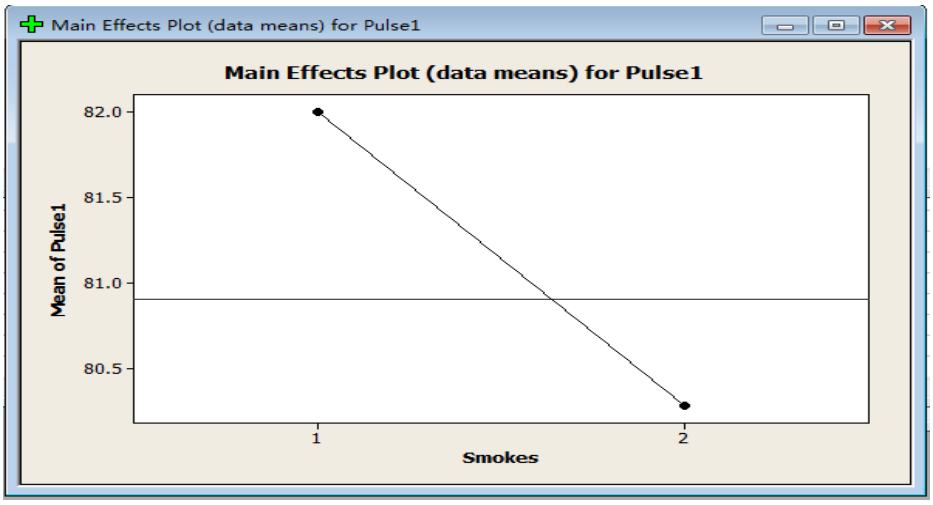
【图 7】

2) Responses:即为响应输出（Y），Factors:即为因子（X）如【图 8】



【图 8】

3) 点击 OK 后出现如【图 9】，就可以看出抽烟对于脉搏有很明显的影响。



【图 9】

3. 交互作用图

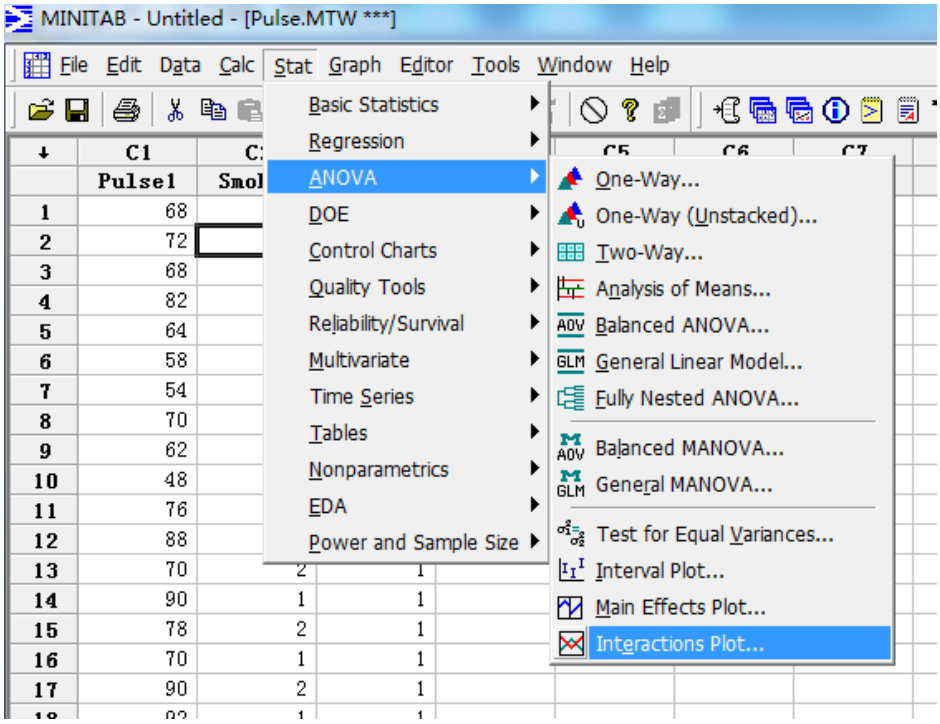
例如我们研究影响人类的脉搏的两个因子：性别和是否抽烟是否有交互作用

1) 搜集如【图 10】的资料，Smoke 为 1 代表抽烟，2 代表不抽烟；Sex 为 1 代表男性，2 代表女性。

↓	C1	C2	C3
	Pulse1	Smokes	Sex
1	64	2	1
2	58	2	1
3	62	1	1
4	66	1	1
5	64	2	1
6	74	2	1
7	84	2	1
8	68	2	1
9	62	2	1
10	76	2	1
11	90	1	1
12	80	2	1
13	92	1	1
14	68	2	1
15	60	2	1
16	62	2	1
17	66	1	1
18	70	1	1
19	68	1	1
20	72	2	1
21	70	2	1
22	74	2	1
23	66	2	1
24	70	1	1
25	96	2	2
26	62	2	2
27	78	1	2

【图 10】

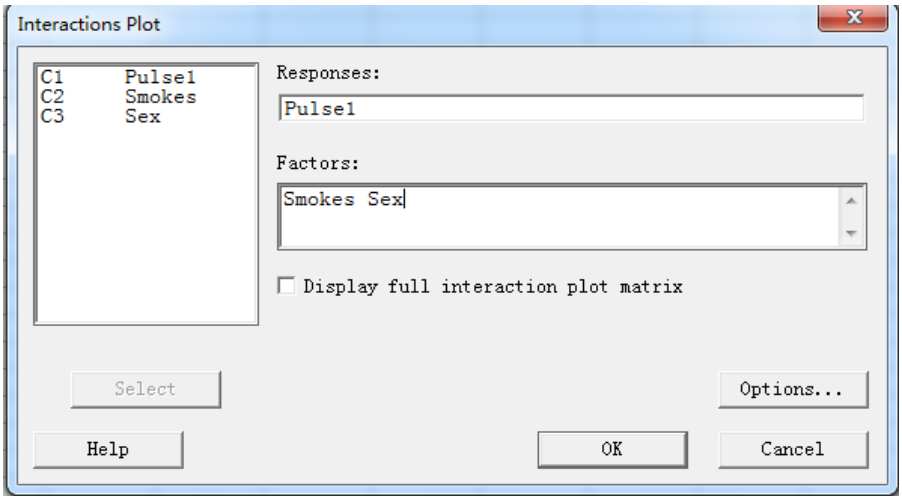
2) 选择"统计(Stat)"->"方差分析(ANOVA)"->"交互作用分析(Interactions Plot)"如【图 11】



【图 11】

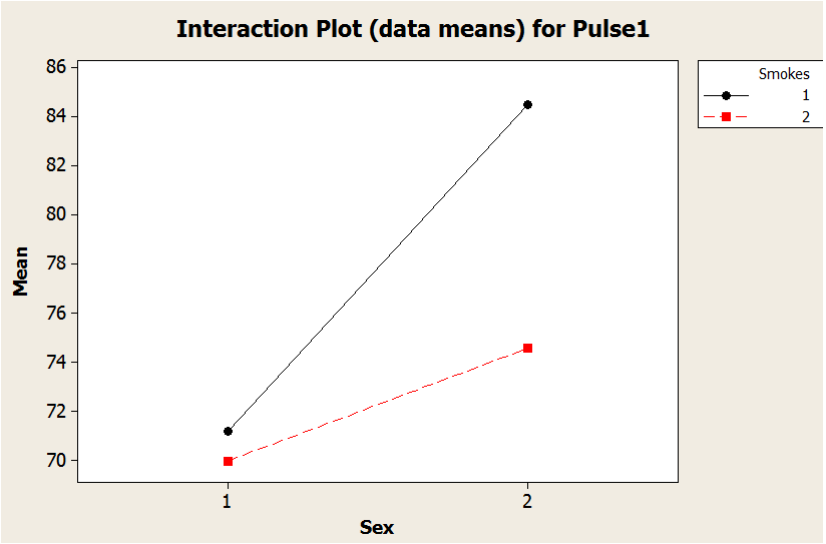
3) Responses:即为响应输出（Y），Factors:即为因子（X）

如【图 12】



【图 12】

4) 点击 OK 键后出现如【图 13】从此图中可以看出，性别和是否抽烟存在很明显的交互作用。



【图 13】

4. 相关性(Correlations)

例如我们研究一部分人类的身体和体重此两个因子是否存在相关性。

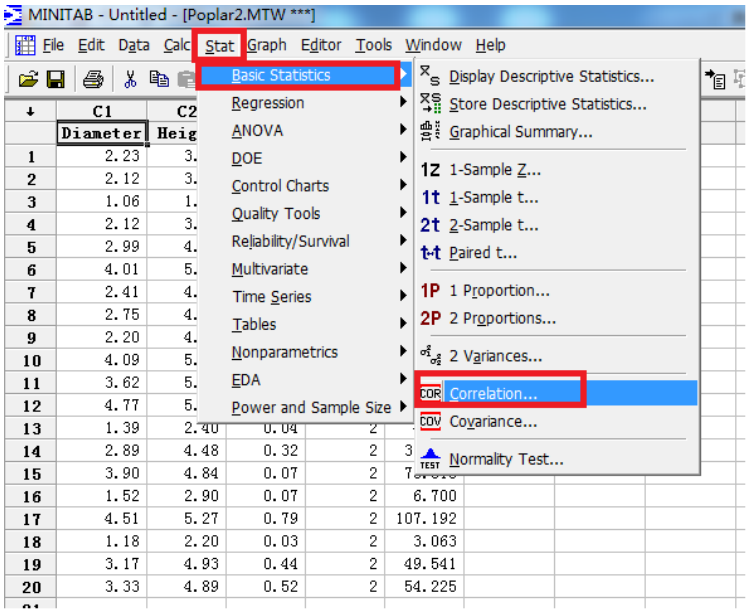
1) 搜集如【图 14】的资料

C1	C2	C3	C4	C5
Diameter	Height	Weight	Site	D2H
2.23	3.76	0.17	1	18.698
2.12	3.15	0.15	1	14.157
1.06	1.85	0.02	1	2.079
2.12	3.64	0.16	1	16.360
2.99	4.64	0.37	1	41.482
4.01	5.25	0.73	1	84.421
2.41	4.07	0.22	1	23.639
2.75	4.72	0.30	1	35.695
2.20	4.17	0.19	1	20.183
4.09	5.73	0.78	1	95.852
3.62	5.10	0.60	2	66.832
4.77	5.54	1.11	2	126.051
1.39	2.40	0.04	2	4.637
2.89	4.48	0.32	2	37.417
3.90	4.84	0.07	2	73.616
1.52	2.90	0.07	2	6.700
4.51	5.27	0.79	2	107.192
1.18	2.20	0.03	2	3.063
3.17	4.93	0.44	2	49.541
3.33	4.89	0.52	2	54.225

【图 14】

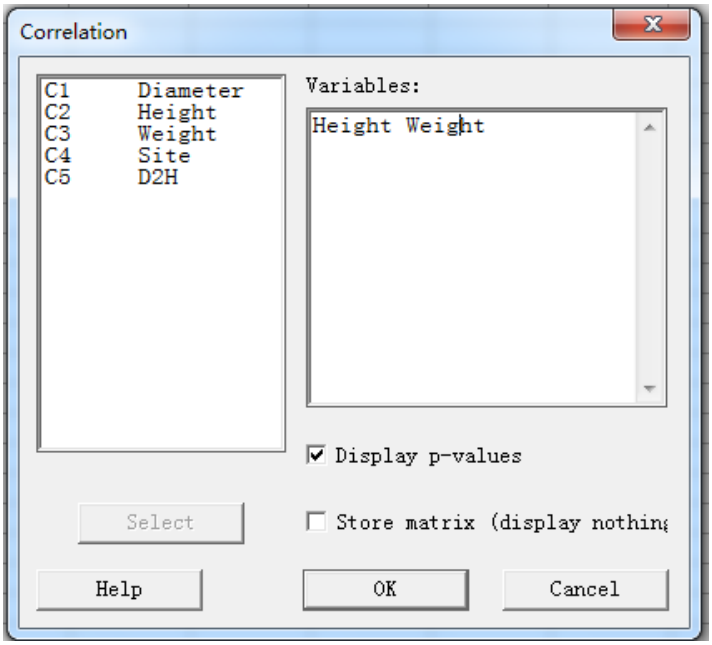
2) 选择"统计(Stat)->Basic Statistics（基本统计）->Correlation（相关性）"

如【图 15】



【图 15】

3) 选择身高体重到 Variables（变量）中，如【图 16】



【图 16】

4) 点击 OK 键即出现分析结果，如【图 17】:

即 0.813 为相关系数，系数越大，说明相关性越大

P-Value 小于 0.05 说明两个变量有相关性。

此两个参数，P-Value 用于说明是否有相关性

Correlation 用于说明相关系数是有多大

```
Welcome to Minitab, press F1 for help.

Results for: Poplar2.MTW

Correlations: Height, Weight

Pearson correlation of Height and Weight = 0.813
P-Value = 0.000
```

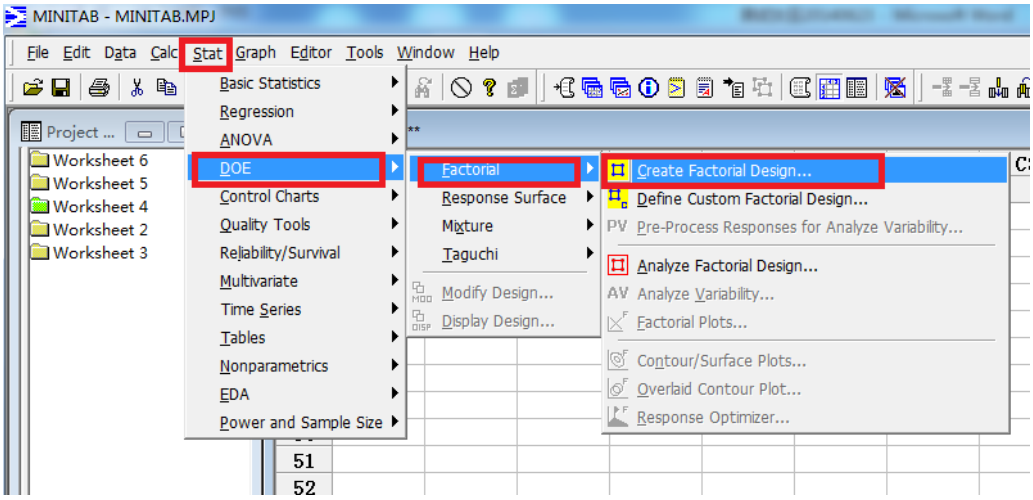
【图 17】

5. DOE

以上 5 个工具均是分析现有资料，筛选因子，当因子确定好了，就可以开始着手设计实验了，以下会按照 Minitab 的操作，说明一下，6 Sigma 的 DOE 工具如何使用。

1) 设计实验

按照以下路径进入设计实验界面"Stat(统计)->DOE->Factorial(因子)->Create Factorial Design（创建因子设计）"如【图 18】

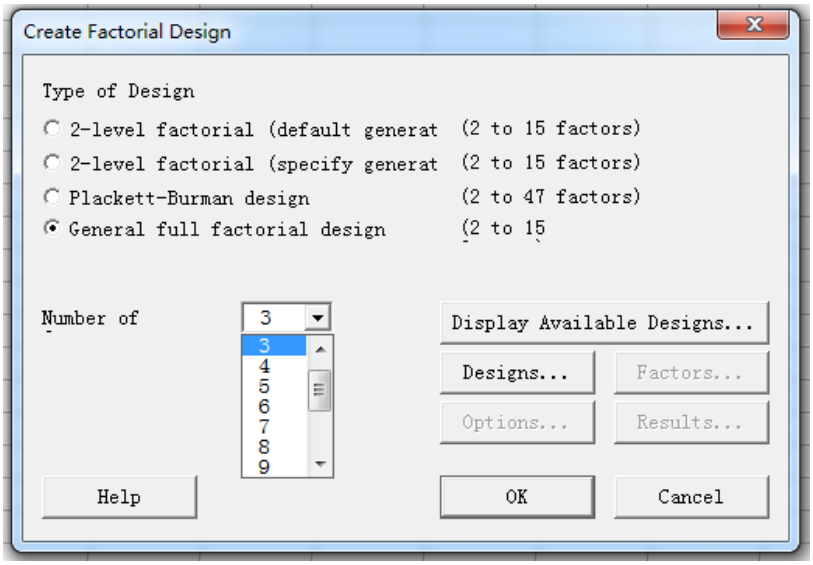


【图 18】

2) 注意 2-level 的类型都是只有两个 level 的试验，例如减肥的两个因子：零食和运动，它们都有两个水准：吃零食和不吃零食；运动和不运动。这样的实验设计就是选 2-level 的设计，如果有一个因子是多于两个水准的，就只能选 General Full factorial design。

例如减肥还有另外一个因子：身高，身高的水准有三个：高，中，低，这样就需要选择 General full factorial design。

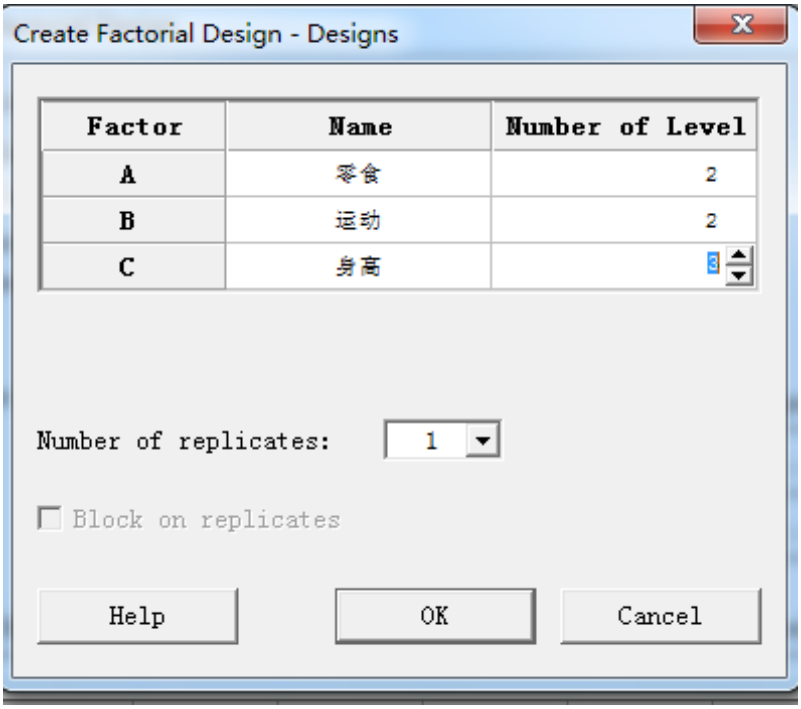
下拉菜单选择因子的个数，如果减肥只有零食和运动这两个因子就选择 2，如果还会新增身高这个因子，就要选 3。如【图 19】



【图 19】

3) 点击【图 19】中的 Designs 出现如【图 20】根据你的实验设计

输入因子名和因子对应的水准数量，例如零食有两个水准：吃零食和不吃零食，所以 number of level 为 2。



【图 20】

4) 点击 ok 即会出现 DOE 帮你设计的试验如【图 21】

一定要约定好：

零食 1：代表吃零食，2 代表不吃零食

运动 1：代表运动，2 代表不运动

身高 1：代表高，2 代表中，3 代表低

C1	C2	C3	C4	C5	C6	C7	C8
StdOrder	RunOrder	PtType	Blocks	零食	运动	身高	体重
6	1	1	1	1	2	3	
5	2	1	1	1	2	2	
11	3	1	1	2	2	2	
10	4	1	1	2	2	1	
12	5	1	1	2	2	3	
2	6	1	1	1	1	2	
3	7	1	1	1	1	3	
4	8	1	1	1	2	1	
9	9	1	1	2	1	3	
8	10	1	1	2	1	2	
1	11	1	1	1	1	1	
7	12	1	1	2	1	1	

【图 21】

5) 此时根据这样的组合进行试验：

即需要找 12 位胖子原始体重都是 200 斤，根据零食，运动，身高的水准要求进行试验 1 个月，例如第一行：

零食 1：吃零食；运动 2：不运动；身高 3：比较低的身高

按照这个要求试验 1 个月。

其它行都是按照这个要求进行试验，将一个月后的体重填到"体重"此栏位中，如【图 22】

C1	C2	C3	C4	C5	C6	C7	C8
StdOrder	RunOrder	PtType	Blocks	零食	运动	身高	体重
6	1	1	1	1	2	3	230
5	2	1	1	1	2	2	225
11	3	1	1	2	2	2	180
10	4	1	1	2	2	1	190
12	5	1	1	2	2	3	175
2	6	1	1	1	1	2	165
3	7	1	1	1	1	3	160
4	8	1	1	1	2	1	220
9	9	1	1	2	1	3	150
8	10	1	1	2	1	2	145
1	11	1	1	1	1	1	166
7	12	1	1	2	1	1	155

【图 22】

6) 此时就可以开始分析了，首先看零食，运动，身体分别与体重的相关性分析。如【图 23】可以看出身高与体重无相关性，因为 P-Value 很高，且相关系数很低

Correlations: 零食, 体重

Pearson correlation of 零食 and 体重 = -0.498
P-Value = 0.099

Correlations: 运动, 体重

Pearson correlation of 运动 and 体重 = 0.813
P-Value = 0.001

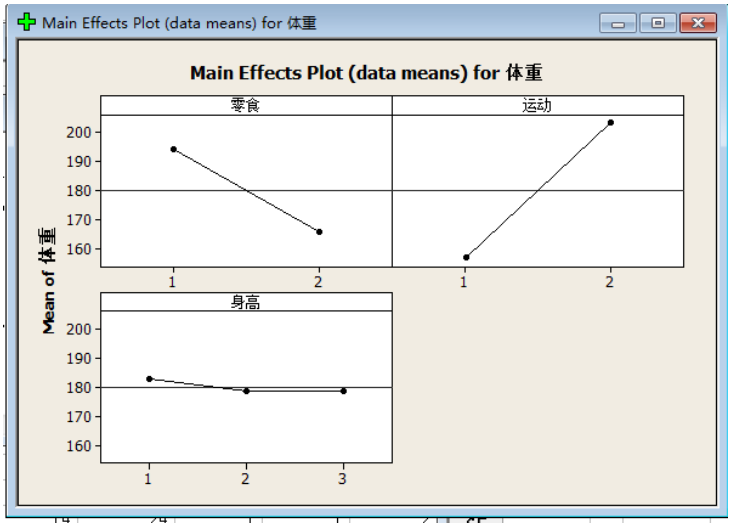
Correlations: 身高, 体重

Pearson correlation of 身高 and 体重 = -0.057
P-Value = 0.860

【图 23】

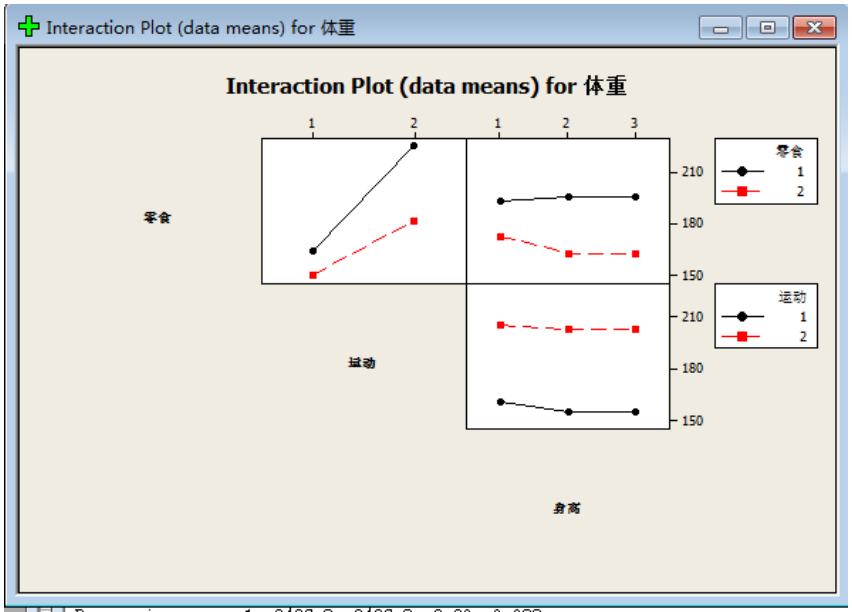
7) 进一步使用主效应图分析如【图 24】可以看出零食和运动的主效应非常大，而身高的主效应很不明显，这与相关性研究的结果一致。

且"不吃零食"体重会有明显的下降，"运动"体重也有明显的下降。



【图 24】

8) 交互作用分析如【图 25】可以看出零食和运动有很强的交互作用，身高

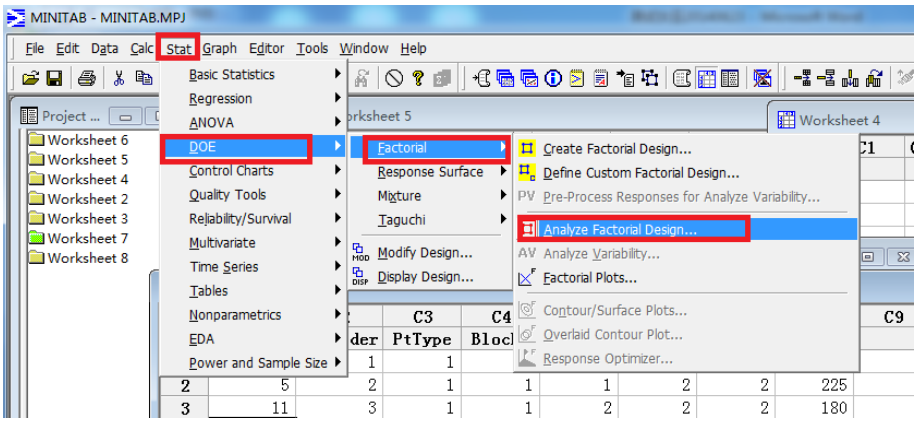


和零食，以及身高和运动都无明显的交互作用。

【图 25】

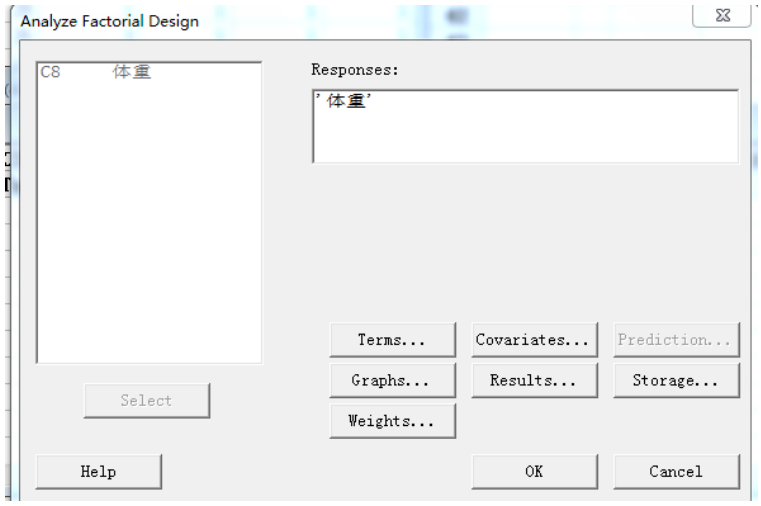
从以上 6） 7） 8）步骤的分析可以看出，可以将身高这个因子删除掉，使用 DOE 进一步分析。

9) 选择"Stat（统计）->DOE->Factorial(因子)->Analyze Factorial Design(分析因子设计)"如【图 26】



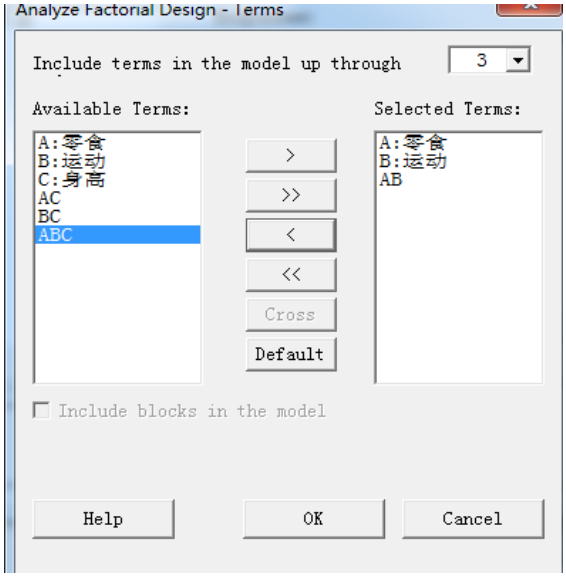
【图 26】

10) Response 即响应变量选择"体重"如【图 27】



【图 27】

11) 点击 Terms,注意在 Selected Terms 将身高移除，因为我们刚刚研究身高与体重并无关系，不需要进一步研究。如【图 28】，点击 OK



【图 28】

12) 因子选择完毕后，分析结果如【图 29】，注意 SeqSS 是贡献度，即通过这个值可以看出哪个因子对响应变量：体重多的影响大。

如【图 29】运动的 Seq SS 为 6486.8

零食的 Seq SS 为 2436.8

零食和运动的交互作用为 660。

综上研究：想要减肥首选应该是运动，其次是不要吃零食。

即运动对减肥的影响最大，其次是零食。

General Linear Model: 体重 versus 零食, 运动

Factor	Type	Levels	Values
零食	fixed	2	1, 2
运动	fixed	2	1, 2

Analysis of Variance for 体重, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
零食	1	2436.8	2436.7	2436.7	82.14	0.000
运动	1	6486.8	6486.8	6486.8	218.65	0.000
零食*运动	1	660.1	660.1	660.1	22.25	0.002
Error	8	237.3	237.3	29.7		
Total	11	9820.9				

S = 5.44671 R-Sq = 97.58% R-Sq(adj) = 96.68%

Residual Plots for 体重

【图 29】

四、 应用到实际工作中_找到问题的本质

1. 确定目标

在以往做的项目中，发现无法复现的 bug 会产生较多负面影响

1) 增加 DVT 较多的工作量：测试 4 版，每版测试 30 次。

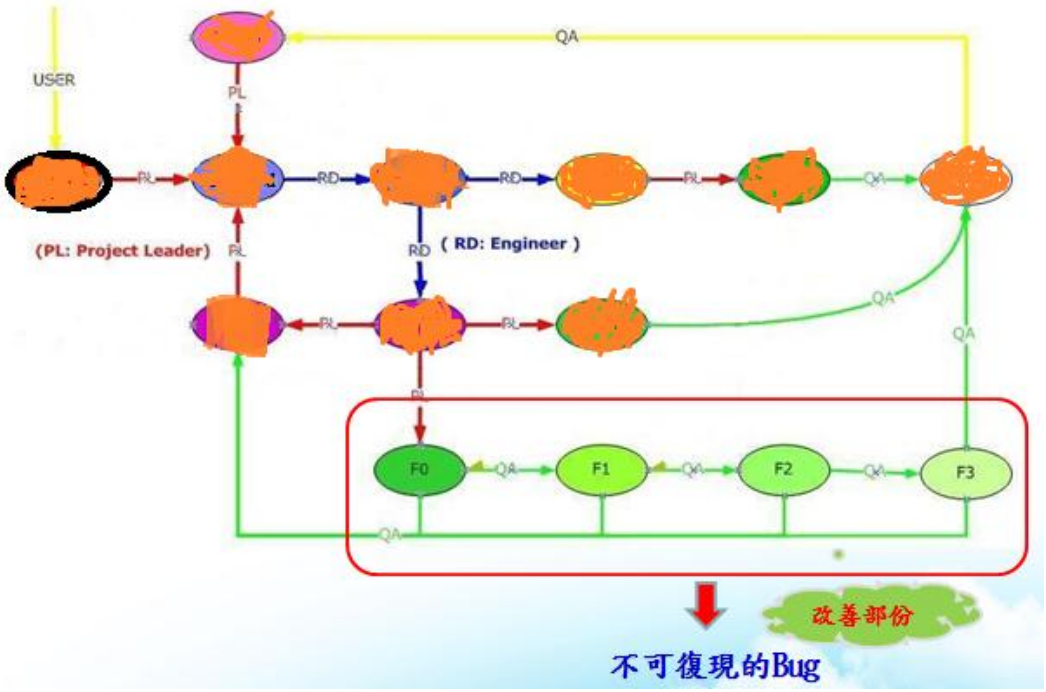
2) 降低 RD 解决 bug 的速度：无法复现的 bug 会进入观察期(至少 4 周)， 在观察期内的 bug, RD 一般不会进行解决

3) 增加项目的风险：一些隐藏较深的 bug 测试 120 次也未能复现，但是还有被客户发现的风险。

2. 分析现有流程

如【图 30】此流程为软件测试发现的 bug 的生命流程，

出于公司保密的要求，将其它的状态隐藏起来；我们需要研究的就为红框标出的流程：如果遇到不可复现的 bug，将会将 bug 的状态改为 F0 然后从 F0 状态改为 F1(需要验证 bug 30 次)，同样的道理将 bug 的状态 F1 改为 F2，以及 F2 改为 F3，最后将 F3 改为最后的状态。这是对于无法复现 bug 的当前处理流程。



【图 30】

3. 收集数据并筛选因子

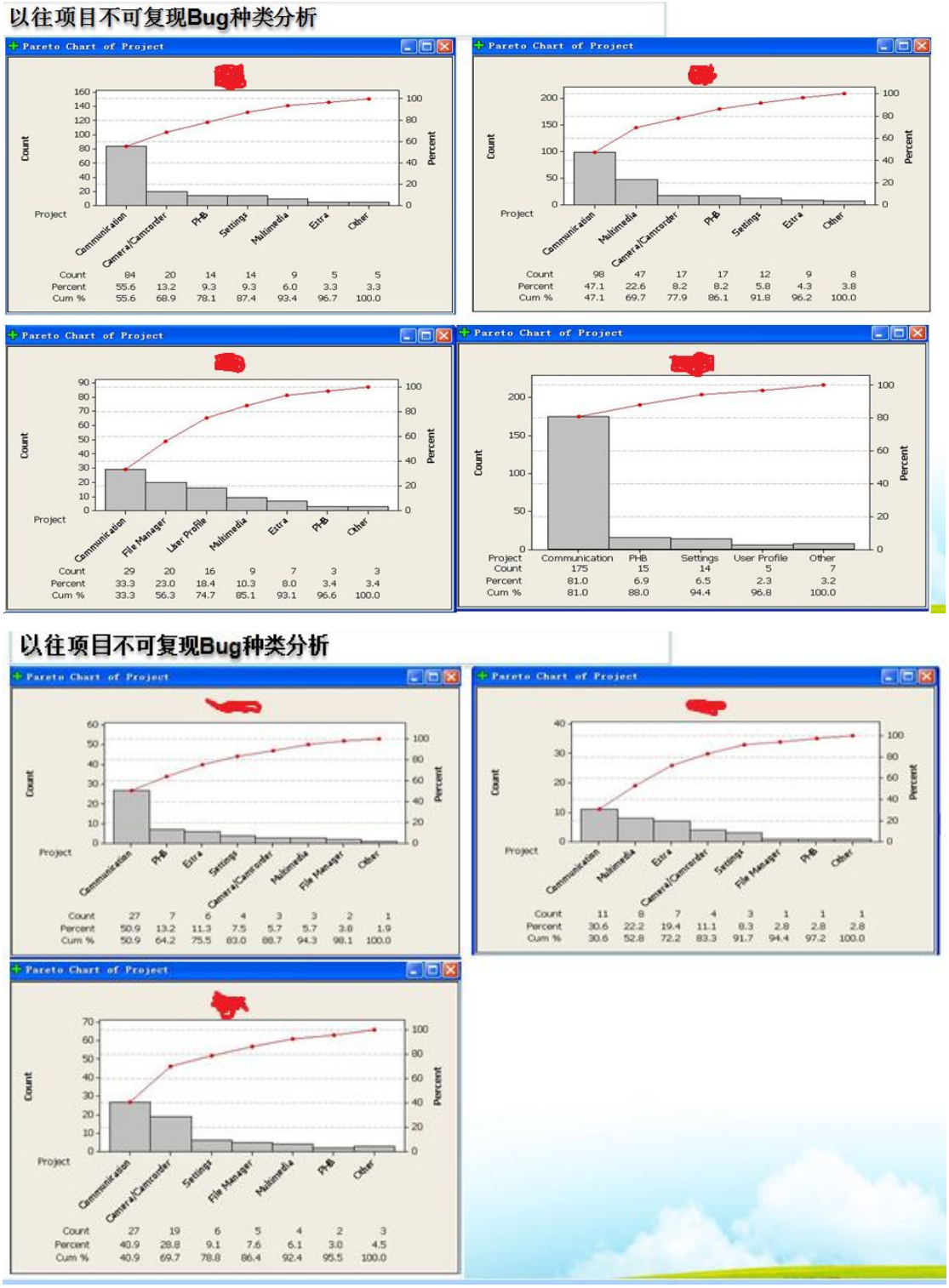
1) 搜集 9 个项目的各个不同种类不可复现 bug 的数量如【图 31】

↓	C1-T	C2	C3	C4	C5	C6	C7	C8
	Project	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
1	Communication	27	11	27	175	84	98	29
2	Camera/Camcorder	3	4	19	0	20	17	2
3	Extra	6	7	1	3	5	9	7
4	File Manager	2	1	5	2	4	5	20
5	Multimedia	3	8	4	2	9	47	9
6	PHB	7	1	2	15	14	17	3
7	Settings	4	3	6	14	14	12	1
8	User Profile	1	1	2	5	1	3	16

【图 31】

2) 使用柏拉图来分析不可复现的 bug 比较多的集中在哪里，如【图 32】

从以往的 7 个项目，不可复现 bug 的柏拉图分析结果看，Communication 此类别是不可复现 bug，最多的一个种类。所以接下来我们应该针对 Communication（通讯相关）的 bug 进行进一步分析。



【图 32】

3) 寻找某一个项目的通讯的无法复现的 bug,按照 DOE 的方法进行分析，查看能否找到真正的原因。如【图 33】

如果遇到无法复现的问题，不仅要明确当前知道的一些条件，而且在做实验

明确试验目的

Bug描述

1. 进入Web UI->Wi-Fi Settings->Basic
2. 修改某3个字段（Network Name，AP Isolation, MAX Station Number）的某些参数(这3个字段不一定同时更改)保存后，无法搜寻
3. 必须要重新Download版本才可以搜索到路由器

试验前的准备：

试验设计前先回答如下问题：

我想解决什么问题？
现况为何？
想要得出什么结论？
是否有意义？以及是否具备相关资源

前需要进行一些准备，主要是按照以下逻辑思考 4 个问题，如【图 33】

【图 33】

4) 以下的步骤为回答"试验前的准备的 4 个问题"如【图 34】:

明确试验目的

试验设计前先回答如下问题：

我想解决什么问题？->
无法可以找到“终端无法搜到路由器”此现象产生的确切的条件
现况为何？->
目前只知道有3个选项有可能是此现象产生的条件
想要得出什么结论？
希望可以找到“终端无法搜到路由器”此现象产生的确切的条件
是否有意义？以及是否具备相关资源
此问题的现象十分严重，会使路由器彻底无法工作，且无法恢复且并不是很复杂的操作就能产生此问题，如果不知道真正的原因，会使此Bug隐藏，增加项目的风险

【图 34】

5) 将我们已知的条件转换为 DOE 能识别的参数，如【图 35】

有可能影响

响应变量和影响因子的确定

参数定义：

1. 响应变量Y：终端无法搜到路由器
2. 自变量X: SSID, AP Isolation, MAX Station Number
3. X的level: SSID->2，AP Isolation->2 MAX Station Number->10

【图 35】

6) 试验设计，使用 DOE 的工具进行设计试验如【图 36】

实验设计

StdOrder	RunOrder	PtType	Blocks	SSID	AP Isolation	C MAX Station number
36	1	1	1	2	2	6
24	2	1	1	2	1	4
22	3	1	1	2	1	2
23	4	1	1	2	1	3
1	5	1	1	1	1	1
39	6	1	1	2	2	9
8	7	1	1	1	1	8
7	8	1	1	1	1	7
37	9	1	1	2	2	7
18	10	1	1	1	2	8
9	11	1	1	1	1	9
5	12	1	1	1	1	5
28	13	1	1	2	1	8
27	14	1	1	2	1	7
10	15	1	1	1	1	10
14	16	1	1	1	2	4
20	17	1	1	1	2	10
11	18	1	1	1	2	1
35	19	1	1	2	2	5
13	20	1	1	1	2	3
19	21	1	1	1	2	9
33	22	1	1	2	2	3
32	23	1	1	2	2	2
4	24	1	1	1	1	4
29	25	1	1	2	1	9
6	26	1	1	1	1	6
3	27	1	1	1	1	3
12	28	1	1	1	2	2
31	29	1	1	2	2	1
25	30	1	1	2	1	5
34	31	1				4
16	32	1				6
40	33	1				10
2	34	1			1	2
17	35	1	1	1	2	7
26	36	1	1	2	1	6
30	37	1	1	2	1	10
15	38	1	1	1	2	5
38	39	1	1	2	2	8
21	40	1	1	2	1	1

一共进行40次的实验

【图 36】

关于试验数据的说明，请看如【图 37】

试验设计说明

本试验参数说明

响应变量Y:1个 终端无法搜到路由器

自变量X:3个SSID, AP Isolation, MAX Station Number

SSID (1->不更改SSID 2->更改SSID)

AP Isolation (1->Disable 2->Enable)

MAX Station Number（数目为1-10）

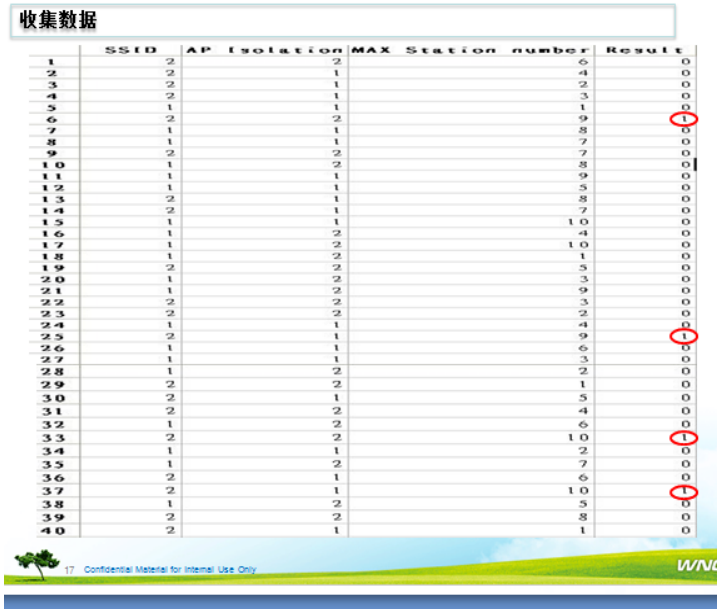
X的Level : SSID->2 AP Isolation->2 MAX Station Number->10

即3个因子，每个因子的个数不一样

所以选择General full Factorial Design

【图 37】

7) 根据 DOE 设计的试验进行实际复现 bug 得出结果如【图 38】



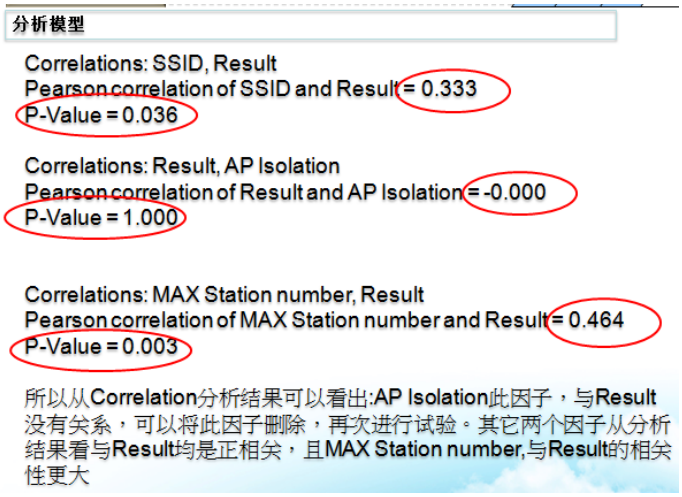
收集数据

	SSID	AP Isolation	MAX Station number	Result
1	2	2	6	0
2	2	1	4	0
3	2	1	2	0
4	2	1	3	0
5	1	1	1	0
6	2	2	9	1
7	1	1	8	0
8	1	1	7	0
9	2	2	7	0
10	1	2	8	0
11	1	1	9	0
12	1	1	5	0
13	2	1	8	0
14	2	1	7	0
15	1	1	10	0
16	1	2	4	0
17	1	2	10	0
18	1	2	1	0
19	2	2	5	0
20	1	2	3	0
21	1	2	9	0
22	2	2	3	0
23	2	2	2	0
24	1	1	4	0
25	2	1	9	1
26	1	1	6	0
27	1	1	3	0
28	1	2	2	0
29	2	2	1	0
30	2	1	5	0
31	2	2	4	0
32	1	2	6	0
33	2	2	10	1
34	1	1	2	0
35	1	2	7	0
36	2	1	6	0
37	2	1	10	1
38	1	2	5	0
39	2	2	8	0
40	2	1	1	0

17 Confidential Material for Internal Use Only

【图 38】

8) 使用相关性此工具来分析通过经验获得的因子，是否与问题可以复现相关，如【图 39】



分析模型

Correlations: SSID, Result
Pearson correlation of SSID and Result = 0.333
P-Value = 0.036

Correlations: Result, AP Isolation
Pearson correlation of Result and AP Isolation = -0.000
P-Value = 1.000

Correlations: MAX Station number, Result
Pearson correlation of MAX Station number and Result = 0.464
P-Value = 0.003

所以从Correlation分析结果可以看出:AP Isolation此因子，与Result没有关系，可以将此因子删除，再次进行试验。其它两个因子从分析结果看与Result均是正相关，且MAX Station number,与Result的相关性更大

【图 39】

9) 删除不相关因素，简化 bug 的步骤如【图 40】

简化模型

Bug描述改进

1. 进入Web UI->Wi-Fi Settings->Basic
2. 修改2个字段（Network Name， MAX Station Number）的某些参数保存后，无法搜寻不到路由器
3. 必须要重新Download版本才可以搜索到路由器

Bug频率->Rare

参数定义：

1. 响应变量Y：终端无法搜到路由器
2. 自变量X: SSID, MAX Station Number
3. X的level: SSID->2 , MAX Station Number->10

【图 40】

10) 重新使用 DOE 工具设计复制 bug 的试验如【图 41】

简化模型和验证试验

	StdOrder	RunOrder	PtType	Blocks	SSID	MAX Station number
1	34	1	1	1	2	4
2	21	2	1	1	1	1
3	7	3	1	1	1	7
4	23	4	1	1	1	3
5	40	5	1	1	2	10
6	27	6	1	1	1	7
7	37	7	1	1	2	7
8	6	8	1	1	1	6
9	25	9	1	1	1	5
10	15	10	1	1	2	5
11	10	11	1	1	1	10
12	22	12	1	1	1	2
13	14	13	1	1	2	4
14	28	14	1	1	1	8
15	38	15	1	1	2	8
16	32	16	1	1	2	2
17	16	17	1	1	2	6
18	13	18	1	1	2	3
19	3	19	1	1	1	3
20	1	20	1	1	1	1
21	29	21	1	1	1	9
22	33	22	1	1	2	3
23	18	23	1	1	2	8
24	12	24	1	1	2	2
25	20	25	1	1	2	10
26	39	26	1	1	2	9
27	31	27	1	1	2	1
28	35	28	1	1	2	5
29	9	29	1	1	1	9
30	11	30	1	1	1	1
31	19	31	1	1	1	9
32	36	32	1	1	1	6
33	24	33	1	1	1	4
34	26	34	1	1	1	6
35	5	35	1	1	1	5
36	2	36	1	1	1	2
37	30	37	1	1	1	10
38	8	38	1	1	1	8
39	4	39	1	1	1	4
40	17	40	1	1	2	7

一共进行40次的实验

21 Confidential Material for Internal Use Only

【图 41】

11) 按照 DOE 设计的试验，进行复制 bug，结果如【图 42】

简化模型和验证试验

	SSID	MAX Station number	Result
1	2	6	0
2	2	4	0
3	2	2	0
4	2	3	0
5	1	1	0
6	2	9	1
7	1	8	0
8	1	7	0
9	2	7	0
10	1	8	0
11	1	9	0
12	1	3	0
13	2	8	0
14	2	7	0
15	1	10	0
16	1	4	0
17	1	10	0
18	1	1	0
19	2	5	0
20	1	3	0
21	1	9	0
22	2	3	0
23	2	2	0
24	1	4	0
25	2	9	1
26	1	6	0
27	1	3	0
28	1	2	0
29	2	1	0
30	2	5	0
31	2	4	0
32	1	6	0
33	2	10	1
34	1	2	0
35	1	7	0
36	2	6	0
37	2	10	1
38	1	5	0
39	2	8	0
40	2	1	0

改进后Result的说明：

统计实际测试发现：AP Isolation此参数确实不是此Bug产生的条件，可以将此测试条件删除

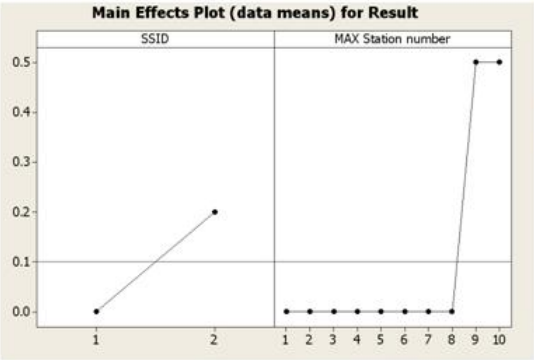
【图 42】

4. 分析实验数据

1) 使用主效应图分析如【图 43】

验证试验结果分析

使用DOE进行分析_主效果图：



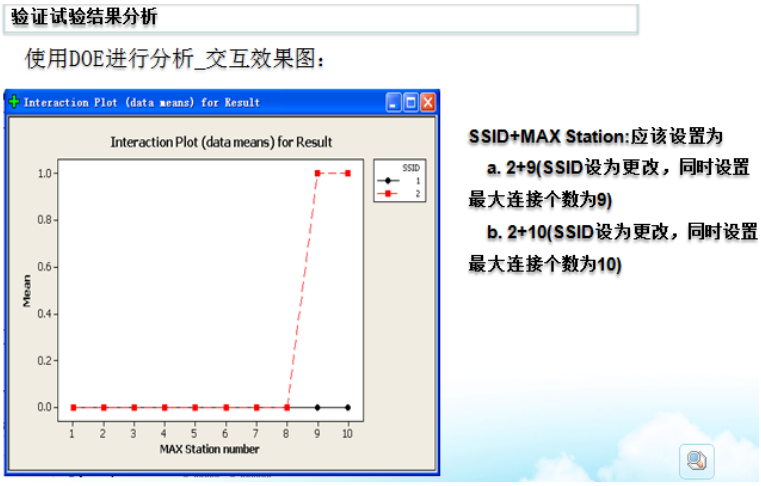
可以看出两个关键因子的参数应该设置为：

1.SSID：2(代表需要更改SSID)

2. MAX Station:9和10（代表最大连接个数需要设置为9或者10）

【图 43】

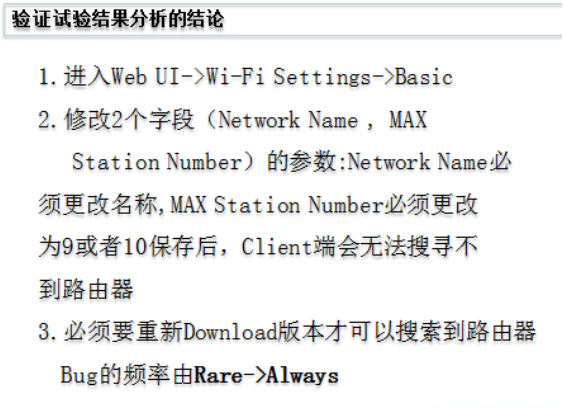
2) 使用交互作用图分析如【图 44】



【图 44】

5. 回归测试

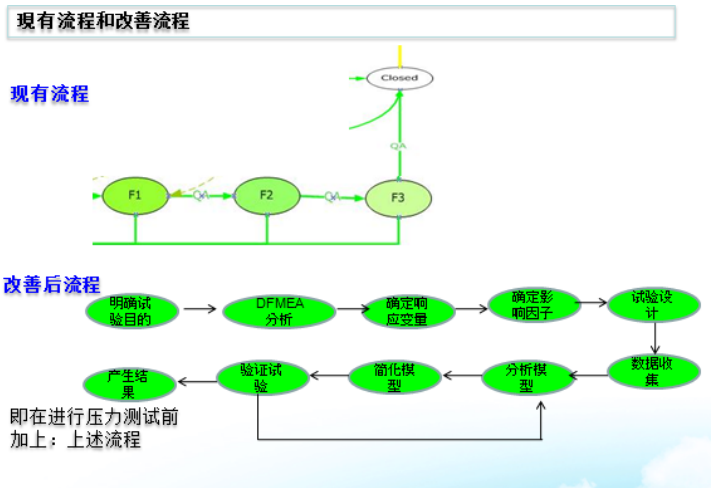
如【图 45】



【图 45】

6. 流程改进

如【图 46】



【图 46】

五、 总结

统计学如何应用到工作中乃至生活中，看起来是一件"高大上"的事情，其实真正用起来还是很亲民的。总结一下这次学的理论和工具：其实是可以分成两大类：筛选因子工具和 DOE 的分析工具。筛选因子工具分别为："柏拉图"，"主效应"，"交互作用"，"相关性"。"柏拉图"是用于从众多因子中找出少数关键因子时使用。"主效应"用于找到哪些因子对于响应变量影响比较大，对于影响很小的因子可以删除。"交互作用"用于确认因子之间是否有交互作用，如果有交互作用在 DOE 的分析中，也要将它们的交互作用作为一个新因子进行分析，例如"零食"和"运动"是有交互作用的，在 DOE 分析时，就要将此交互作用（即零食*运动）作为因子进行一起分析。"相关性"是用于确认各个因子与响应变量之间是否有相关性。上述 4 个工具一起分析可以比较安全的筛选因子。筛选因子过后，就可以使用 DOE 进行分析了。

事实上 6 sigma 还有很多其它的工具还未介绍，后续我就打算按照这样的流程：根据我们软件测试所需要研究的问题为出发点，再此过程中遇到需要使用的工具和相关理论，对于新的工具和理论，就一一进行介绍。

参考文献：

[1]有关 DOE 基本概念介绍的网站：http://wiki.mbalib.com/zh-tw/DOE

[2]有关"Correlation"的介绍的网站：http://tech.casd.cn/wzym/0157/g10157/g1sxn930.htm

测试之前——对测试前质量的考察

作者：殷亮

引言：博观而约取，厚积而薄发

甚至在最先进的国家，也存在许多缺陷与不完善之处，存在许多妨碍社会资源以更有效的方式进行配置的障碍。对这些障碍的研究构成了我们的现实问题。

——阿瑟•赛斯尔•庇古(Arthur Cecil Pigou)[①]

测试之前

二十七年前，软件管理大师杰拉尔德•温伯格（Gerald M. Weinberg）在其著作《Perfect Software》中曾调侃那些不懂测试的开发者："我们在开始测试之前没有任何问题。我们完全符合进度计划。是测试弄糟了所有事情。"[②]

现在，大多数从业者都已经清楚了问题所在：弄糟了所有事情的不是测试，而恰恰是据称"没有任何问题"的"测试之前"。是上游工作的低劣质量，导致了最终测试周期的无限拉长与测试资源的超额投入。正如传统测试文献所经常强调的：一个缺陷在不同阶段修复的成本差异——即随着开发生命周期的向前推进，修复前期引入的一个缺陷的单位成本将呈指数增长。这一观点的意义在于它很好地揭示了项目失控的一个重要方面：即在软件前期质量还很糟糕的情况下，人们过早地由"测试之前"过渡到了正式的测试阶段。这样，当那些原本可以在前期被发现与修复的缺陷跟随着软件包一齐流入到测试环节时，必将导致更多轮次的测试与更高昂的修复成本：人们不仅投入了额外的测试资源，同时开发资源的投入却未减反增（因为对缺陷的修复本身就是另一个引入错误以及新缺陷的机会）。从社会学的角度来看，这是一种低效的社会安排，是"非帕累托最优"（Non-Pareto optimal）的：一方的福利降低了，但另一方的处境却并未因此而得到任何改善。

因此，我们如何正视"测试之前"的美好假象，以及如何有效地防止测试资源被滥用，已经不只是一个出于对测试团队的尊重与保护的简单问题，而是一个影响到项目整体资源投入的管理问题——因为测试资源的滥用必将导致开发资源的更多投入，在其它条件不变的情况下，这两类资源投入的增加都唯有通过日程的向后推移来弥补，进而使项目毫无悬念地驶向死亡之旅。但遗憾的是，如今似乎只有极少数的软件组织较好地解决了这一问题，而其它绝大部分的软件组织仍在持续的焦虑与不断的失望中，继续背负着测试资源滥用带来的沉重负担，同时还得一再相信：这一次测试之前是"真的"没任何问题的。

度量困难

既然我们相信绝大多数企业已经知道测试之前质量控制的重要性，而且也给予了测试之前的质量足够的重视，那么我们接下来就需要解释：为什么人们愿意控制测试之前的质量，但测试之前的质量却总得不到控制？

传统测试文献对缺陷修复成本的论述，与质量管理大师菲利浦•克劳士比（Philip B. Crosby）的质量思想是一致的：均认为缺陷越早得到修复投入成本越低。克劳斯比强调："最便宜的做事方式就是一次把事情做好"，因而得出"质量是免费的"这一著名结论[③]。然而现实情况是：在软件开发领域里，几乎从没有过"一次把事情做好"——尽管计算机科学先驱哈兰.米尔斯（Harlan D. Mills）和其他学者认为，他们可以完美地构建系统而不需要测试，但迄今为止，仍没人能在实际的软件项目中令人信服地证明这一点。因为人在感官能力与脑力上的局限性，以及只有着极为有限的存储处理能力和同样有限的接收反馈通道，再加上一些不可忽略的心理因素与环境压力，这使得在绝大多数情况下，要求他人一次将事情做好无疑是不太现实的。

如此一来，我们只能退而求次：不要求一次做好，但质量必须"足够好"，为难以避免的错误留下人性化的空间。但这样并没有让我们的问题难度随之降低，因为这时我们还必须给测试之前定义一个相应的质量标准：它不是绝对的"零缺陷"，但它必须是"足够好"的。定义这个标准的困难在于，首先，我们并不知道该让这个标准落在"零缺陷"之后的什么地方才算合适——不同的软件项目规模、不同的复杂性，定义一个固定允许的缺陷数显然是不公平、也是不合逻辑的；其次，质量也并非是仅凭缺陷数便能客观度量的——不同的严重级别、不同的错误类型，在增加了度量的复杂性的同时，也为人们针对缺陷的争论与讨价还价埋下伏笔；最后，由于测试原本就是对软件质量的度量，所以除非我们最后执行完测试得到结论，否则这一度量又必然是不完全的。以上种种原则，使得我们想要为测试之前的软件质量进行严谨而完整的度量变得极为困难：因为它不仅复杂，而且还存在着极为高昂的执行成本。

为了低成本地对测试前质量进行度量，常见的做法是：测试团队试图抽取测试用例的一个子集（比如整个测试用例的 5%），要求送测的版本必须完全通过后才启动正式测试。但我不明白的是，人们普遍选择的这一做法究竟是认为可以通过 5%的测试用例的结果推断出总体质量，还是希望它能督促开发人员构建高质量的软件呢？如果是前者，那么我相信这是出于对统计学的误解——是的，我们的确可以从样本推断出总体，但这样做的前提是样本必须是随机的、而且还应当进行多次取样。另外，如果我们认为这样就能督促开发人员构建高质量的软件的话，那这可能就是严重低估了人的"足智多谋"，低估了他们学习规则与利用规则的潜能。诺贝尔奖得主迈克尔·詹森（Michael C. Jensen）甚至指出，绩效机

制通常还会“鼓励雇员花费精力去影响那些度量和考核产出机制的设计与应用”[④]。在时间压力下，我们完全有理由相信我们仍能得到近乎完美的 5%——只不过剩下的 95%有可能会让我们极其失望。

就像我们所观察到的，这一常见方法也没能有效地帮助企业改善软件的前期质量。软件自身的复杂性与人体机能的限制，使得我们不得不放弃“一次做好”的苛刻要求；而“足够好”这一标准它在定义上的困难与度量上的高昂成本，又使得我们即使想退而求次也并不容易。而那些为了阻止对测试资源的滥用、由测试团队单方面筑成的质量壁垒，往往有如沙丘般脆弱、形同虚设；有时我们可以选择将壁垒进一步加强，但结果也许只会导致更多的资源被浪费、更多的冲突被挑起，仍然改变不了测试之前的质量问题。

激励干扰

上文中我们探讨到了人们为何不能一次做好，以及质量的度量困难，但度量的困难并不是测试之前质量问题的直接原因，因为度量的困难在那些优秀的软件企业里也都是存在的。并且，度量只是一种评估、一种测试，度量并不会改变质量本身。因此，我们必须将分析转至上游的开发团队，从那里去寻找测试之前质量问题的根源。

在进行分析之前，我想有一点应该是可以肯定的：尽管我们常常对开发团队送测的质量不满，但我们并不用质疑企业管理层对开发团队的质量意识的强调，与此类似的，我们也并不用质疑管理者对开发进度的重视。因为同时关注项目的质量与进度，这在所有做项目的企业里，都是再平常不过的事。现今的管理思想中，人们予以执行与跟进非常高的评价，因而我们不断地拆分目标与计划，直到它易于跟进与监督为止。软件专家史蒂夫•迈克康奈尔（Steve McConnell）认为，将项目拆分成可跟踪的里程碑，可以提高项目的可视性与降低项目风险[⑤]。这样不至于到了项目计划接近尾声时，才发现项目需要延期。

于是，在软件项目中就有了一些很“自然而然”的常见的标准里程碑：比如传统瀑布项目中的产品定义、需求说明、概要设计、详细设计、编码、系统测试、产品发布等。在敏捷中，尽管这样僵化的里程碑划分已不复存在，但通常还是会有个明确的系统测试阶段，毕竟每个迭代完成后的版本只是“潜在可发布”的。至少可以说，仍然会有那么一个或多个时间点——迭代内的或迭代外的——在这些时间点上，将发生软件包从开发团队到测试团队的正式交接。我的观点是，在其它条件不变的情况下，只要存在着这样的点，测试之前的质量就极有可能存在问题；而这些点对外界可视性越高，问题发生的可能性就越大。

这是因为，当人们需要同时考虑多个目标时，就总会面临如何在多个目标之间正确地分配时间与精力的问题。我们在上文中已经提到了，对于开发团队，他

们通常都面临着两个方面的监督，一个是质量方面，一个是时间方面；也就是说，管理层一方面关注软件的质量，另一方面还关注里程碑的完成情况，比如是否准时送测等等。同时我们也已经讨论过，相对而言质量目标是最难监督的，而时间目标却恰恰相反。所以，如果里程碑临近、但项目进度又赶不上日程时，就有可能导致开发团队的行为扭曲——团队会投入一切可利用的资源确保能赶上里程碑，包括掩饰项目真实进度、满足送测条件、使各项工作看上去已经完成……然后才会考虑质量问题。这几乎是大多数软件企业开发过程的写照：在可度量的里程碑目标与难以度量的质量目标之间，人们优先选择里程碑目标；在即将度量的当前里程碑目标与很久以后才度量的交付目标之间，人们优先选择当前里程碑目标——这样至少让现在看上去没任何问题——而在未来，即使前景再糟，自认幸运的我们也总是相信自己能力挽狂澜。

甚至在日常中，人们也会作出这样地选择：尽可能的将有限的精力分配到能产生更多效益的活动、或者易于被监督的活动中。这两点并不矛盾，对于雇员而言，同样的一份精力，投给那些易于被监督的活动显然要比投给不容易被监督的活动要更划算、效益更高。管理层越是容易监督、越是关注，就越是会促使雇员将精力都投入到这些活动上，而忽略其它活动。以此我们推断，当管理层对测试点加大关注时，必然会导致开发团队对日程的重视和对质量的忽视，因为此时满足日程目标的收益已进一步增大，而度量的困难早已使在质量方面的投入回报降低。在压力下，人们会倾向于选择投入一切资源确保准时送测，因为达到这一目标并无它法；而质量问题由于度量困难，则完全可能通过事后的政治手段或是雄辩的口才来“提升”——也许比投入资源来提升质量更有效。

同样，在其它领域里，我们可以普遍看到：销售代表们总是愿意花更多的精力在能立即产生销售额的活动上，而不是在学习产品知识或是维护企业形象等方面“浪费时间”；采购员更在意零件物料成本的降低，而不是零件质量；以及大多数教师注重学生学业成绩的提高，而不是学生社会能力与思维创造力的提高等等。就像经济学理论所假设的，人们总是向着个人效用最大化的目标前进着，这一目标不一定会与管理者所期望的目标一致，甚至往往都是不一致的。尤其是那些在雇员们看来进一步付出也不会有等同回报的活动，在这些活动上，我们就不大可能期望雇员们会给予它足够的关注。

数学证明

实际上，我们可以将上述结论阐述得更正式、更一般化、也更严谨一点：如果管理者无法监督雇员在两种或多种任务之间的时间与注意力的分配，那么其结果必然是——要么雇员们花费在每一种活动上的时间与注意力的边际收益率必

须相等，要么边际收益率较低的活动将得不到任何时间与注意力。

也就是说，每当人们在考虑下一单位精力的分配时，总是会优先考虑边际收益率较高的活动，直到这些活动的收益率降至其它活动的边际收益率之下。我们可以用数学很轻易地给出证明：

这个问题实际上就是给定雇员的效用函数 $U=U(x,y)$，在约束 $c=x+y$ 下求最优。其中 x 与 y 分别为雇员在两项活动上的投入水平，U 为不同投入水平下的雇员收益，另外假设两项投入水平之和是一常量 c(雇员的时间与注意力在给定时间里是固定的，比如一天八小时)。

这一最优化模型的拉格朗日函数是：

$Z=U(x,y)+\lambda(c-x-y)$

作为一阶条件，我们可以得到如下一组联立方程：

$Z_\lambda= c-x-y=0$

$Z_x= U_x-\lambda=0$

$Z_y= U_y-\lambda=0$

后两个方程实际上就向我们表明了 $U_X=U_y=\lambda$ 的关系，也就是雇员分配自己的时间与注意力时，遵循了边际收益相等的原则。

忘记测试之前

上面的简单模型足已揭示管理工作中所面临的复杂性了，人们在工作中如何分配时间与注意力，往往并不如管理者想象的那么简单。最终的分配方案取决于人们对各个活动收益的评价，而这些收益又受活动成果能否被准确度量的影响。对于那些不容易度量或监督的活动，就无可避免地会被人们投以较少的精力，或者以次充好。在容易监督与不容易监督的活动之间，在关注现在还是关注未来之间，人们的选择总是坚定的。

那些对送测点给予过度关注的管理者，只不过犯了这么一个不起眼的小错误——甚至单独看来都不算是错误的错误，但结果却是致命的：测试之前的假象在继续，测试资源的滥用也在继续，就像一个打不开的结，让整个团队都为之疲惫不堪，甚至完全看不到将来。而这个结又是如此的隐蔽，因为管理者往往想不到是这里出了问题；但他（她）必须解开它，他（她）必须忘记测试之前。也就是说，我们应当明确地告诉项目团队，我们只在乎项目最终交付时的质量与时间，如果有必要，我们甚至还应明确地强调，我们并不关心任何的中间时间点，包括项目何时送测等等——虽然这些时间点也许是重要的，但这些时间点的成功并不代表着项目的最终成功。就像在统一过程（UP）中，我们不再划分传统的编码阶段与测试阶段，而是将它们合并成单一的构建阶段一样，虽然这些时间点很容易评估，但我们并不想因为这些时间点而忽视了更有价值的信息，更不想因为这

些时间点错失了真正能挽回项目的机会。

这也就意味着，在一个理性的软件企业里，项目经理必须是一个真正的管理者，而不应该仅仅只是一个资源协调者。项目经理必须对所需使用的资源有明确的预算、并通过预算合法地获得这些资源，开发资源如此，测试资源亦应是如此。这样，项目经理才有足够的权力履行他作为一个成本中心的义务：在确保质量的情况下，最小化资源投入——以一个管理者的身份对整体投入负责，而不是以一个协调者的身份只对计划中的时间部分负责（在这种情况下，他必然也是无法负责的）。一般而言，在能考核结果的时候，我们不必、也不应该考核过程。

小结

如果我们对前面讨论的那些命题进行严谨的科学分析，那么可能会发现无论是缺陷修复成本假说，还是克劳斯比的质量思想，实质上都只不过是一种同义反复（tautology）：它们不可能被现实证据驳倒——因为事情如果已被一次做好，那么就不再有机会度量它没一次做好的成本了。在专业领域中，“同义反复”一词有着极为强烈的贬意色彩，“同义反复”意味着提出的假说无法被证伪，而“可被证伪”却是成为科学理论的基本条件。但在这里我有必要澄清一点：对于这些经典假说，我们并非有意对其责难——作为有用的同义反复，它们仍然是质量领域里最伟大的论述，它们使我们能以一种全新的方式认识到缺陷的预防与前期质量的重要性。如同生物学中的适者生存、经济学中的消费者效用最大化与科斯定理、以及会计学中的复式记账法，尽管这些都是典型的同义反复，但却并不妨碍它们成为各自学科理论中的核心。我们分析这些经典假说的本质，只是试图指出当前我们的困窘所在——为什么我们知道一次做好的投入是最经济的，但却总是无法一次做好？

这些无疑是同义反复类假说所不能回答我们的，正如日适者生存一说并不能告诉我们如何进化一样。同义反复类假说为我们指明了前进的方向，也许还指明了到达时的特征，但并未告诉我们如何达到它。换而言之，在我们寻求改进测试之前的质量的过程中，仅仅懂得将缺陷尽可能地修复在早期是远远不够的——因为企业中的组织结构、激励制度，以及团队的心理因素都增加了这一路上的艰难险阻，任何一个软件组织都不能忽略它们、假装它们并不存在。

今天，对于测试之前的质量控制，我们已取得了长足的进步，单元测试的普及，以及单元测试工具的持续发展，使得快速地度量软件质量已成为可能（尽管仍不完美）。这在很大程度上改变了对开发团队的激励结构：因为对质量的不重视已能相对容易地在早期监督出来。但本文所提出的激励干扰问题和资源所有权问题却似乎并没有随年月的流逝而有明显的改善，它们仍影响着许许多多的企业，也仍困扰着一代又一代的管理者。有时我们也许只能感叹：在人类浩瀚的知

识星河中，技术类知识要远比管理类知识幸运，因为它们是如此地耀眼、它们家喻户晓，人们歌颂它们、追捧它们、传播它们；但是我们永远都不能忽视这些争相辉映的灿烂星光背后，还有管理智慧的朴实星光，那点点星光，只为寻找真理的双眼而明。

注释：

① Arthur Cecil Pigou. 《Wealth And Welfare》. 1912.

② Gerald M. Weinberg. 《Perfect Software: And Other Illusions about Testing》. 1986.

③ Philip B. Crosby. 《Quality Is Free: The Art of Making Quality Certain》. 1980.

④ Michael C. Jensen. 《Foundations of Organizational Strategy》. 1998

⑤Steve McConnell. 《Rapid Development: Taming Wild Software Schedules》. 1996.

以互联网思维做测试

作者：郝闯

题记：当写下这个标题的时候，由于引用了这个最近很火的词汇“互联网思维”，我很想给自己说一句“no zuo no die”。但是还是想要表达完，请各位看官先看完，之后随便喷。

这次我们谈谈方法论，方法论用于指导实践，选择好的方法论，能使得实践事半功倍。

子曰“温故而知新”，我们要谈测试这个话题，当然也得还是先来回顾一下。

在软件开发不断发展和成熟过程中，衍生了很多的开发模型：边做边改模型、瀑布模型、演化模型、增量模型、螺旋模型、喷泉模型、敏捷模型-SCRUM...不用说太久远的东西，就结合实际，用早先些年的，但目前还在使用的一些来说。我们大家比较熟知的有比较经典的瀑布模型、CMMI 之类这些，我们统称为传统测试模式。

而后这些年，RUP、TDD、ATDD、Agile、Scrum、XP、Lean 越来越火，可谓各领风骚数几年，这一类我们且称之为现有测试模式；

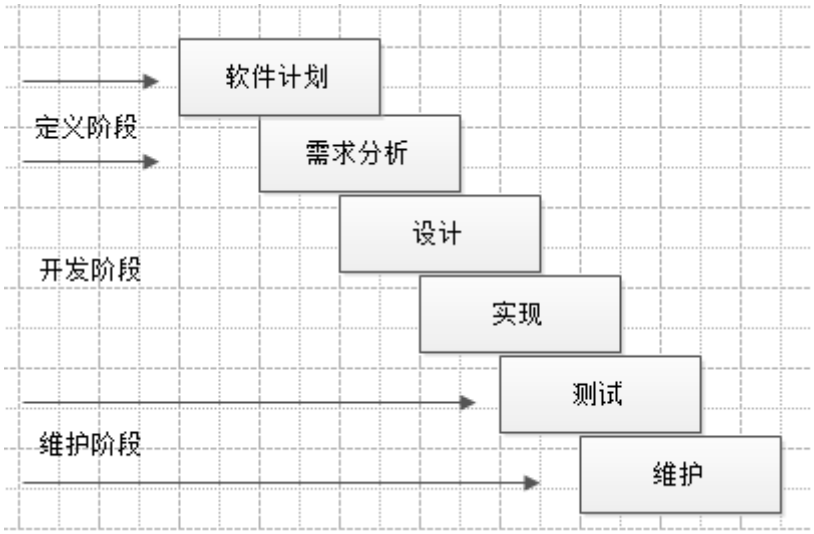
而近一两年以来，移动互联网、大数据、云计算、O2O、互联网金融等行业越来越火，测试也相应的随之变迁，测试模式和方法论也有一些概念出来，云测试、DevOps、BDD、Exploring Test 可谓是正在如火如荼，我们姑且称为测试演进模式；

下面我们从这三类来简要谈一谈一些测试相关的话题。

一、传统测试模式

回到本源，传统软件开发过程中，测试是软件开发中的一个环节。经典的瀑布模型在软件工程中占有重要的地位，它就是这样的，如图一所示：

（图一）



瀑布模型提供了软件开发的基本框架。它的规则从上一项活动接收该项活动的工作对象作为输入，利用这一输入实施该项活动应完成的内容给出该项活动的工作成果，并作为输出传给下一项活动。我们可以看到，它思路很清晰，将一个比较复杂的协作事情工序化，功能的实现与设计分开，问题化简，也方便分工协作。这种结构化的分析与设计方法，将软件生命周期划分为软件计划、需求分析、设计、编写、测试和维护六个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。我们可以看到，它确实很清晰，很实用。

但是同时，我们可以发现，其实这并不高效，资源利用率并不高。因为定义阶段，只有需求人员在忙，而开发阶段，只有研发在忙，到了维护阶段，只有维护人员在忙。这种如同流水线一般的工作，也要求前面的需求分析必须精准，合理，做需求分析的人必须具备很强的能力。而且，每一个阶段都需要有严格的确认和评审，在任何一阶段出现问题，可能就得返工，再来一次。这在传统行业是可行的，但是在软件行业，新技术日新月异，用户需求千变万化，对于经常变化的项目而言，瀑布模型价值很低，很难施展。

很简单，像目前，遇到突发事件，电商年终促销，近期世界杯期间，一个产品从策划到上线可能要求要在很短的数个小时时间内完成，这种需求，如果还在瀑布模式下，几乎很难。

二、现有测试模式

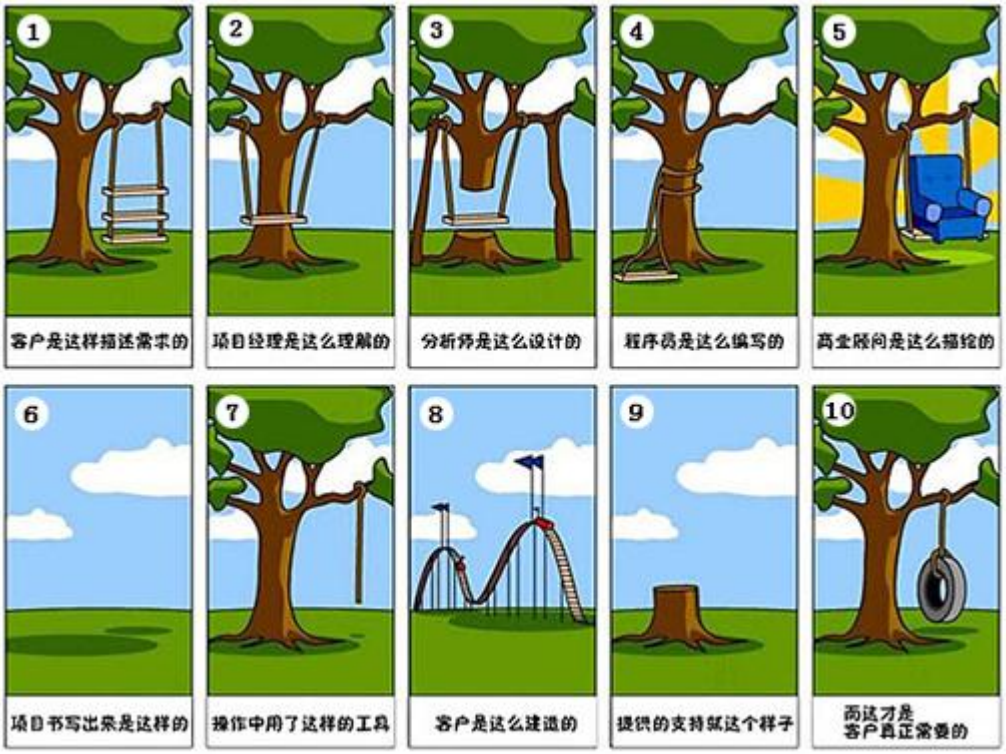
当然，现在越来越多的公司再引入 Scrum、Lean 等敏捷精益。笔者曾经经历过敏捷精益的转型，通过实践发现，确实还是可以带来了不少的改变，当然这个需要公司领导层的支持。通过梳理整个软件研发流程，引入可视化看板，实例化需求，持续集成、自动化测试、自动化构建与部署等，以 Sprint 为开发周期进行迭代研发，结合实际去做改进和优化，可以做到让项目相对透明较好的运转。

不过这个要求项目成员及时响应，遇到问题及时反馈和响应，另外需要团队的坚持，周期性的反思和总结经验。

现阶段，业界不少公司都是这么做的，这种方式较之之前传统已经有了很大的进步和提升。但是，互联网世界的发展太快，Google CEO 埃里克·施密特告诉我们的反摩尔定律：“反过来看摩尔定律，一个 IT 公司如果今天和 18 个月前卖掉同样多的、同样的产品，它的营业额就要降一半。”这个对于所有的 IT 公司来讲，是非常可怕的，因为一个 IT 公司花了同样的劳动，却只得到以前一半的收入。当然也就逼着公司必须赶上摩尔定律所规定的更新速度。也就是产品不仅需要创新，也需要快。而现有测试模式基本可以做到这些，但还需要继续加强，寻找更好的测试模式。

（图二）

三、测试演进模式



而近一两年，移动互联网、O2O、互联网金融等来势汹汹。可以说“风口已经准备好了，就等猪到位了”而做好这种快速的小而美公司的创新和研发，测试业同样需要更新知识。但是，一直以来，我们可以看到测试和需求之间，总是有各种的不对称，如上图二。

尤其移动互联网的今天，要应对快速地迭代，快速地创新，小而美的打造 MVP（最小可用产品）产品，需求更是很关键，对测试同学们而言，如何把控，如何规划你整个测试更是尤加重要。而针对互联网行业创业型的小公司，面对新的一切与未来，你该如何下手？这里我简单谈一下自己的肤浅认知，主要是在互

联网行业测试过程中，你需要注意和做些什么。

竞品分析

作为测试人员，在资源或条件允许的情况下，一定要自己动手研究竞品，因为现在不像以前传统模式下，会有专门的需求分析人员，来给你写一份完整清晰，条理清楚，逻辑可循的需求说明书。这些新时代的产品经理可能丢给你的只是一个粗糙的所谓的 PRD，其实你看上去，就是极端简要的描述，连相应的 UE 或者 UI 高保真都没有。请问，这时候你怎么办？所以，研究竞品，只有研究竞品，才能够在你面对一个模糊需求的时候，有迹可循。当然，如果你细心一点，多研究一些，甚至可能会发现产品经理描述的需求本身存在逻辑问题。恭喜你，这时候，你已经可以和产品经理去沟通这个需求的问题。而不要小看这些，这中活动本身也是一种测试，只是测试的足够早而已。

用户参与

有时候，我们产品会在实际上线后进行 A\B 测试，而这本身就是在实际的一种测试和验证。A\B 测试本身会覆盖到一些交互设计方面的测试，另外可以倾听到真实用户的需求。通过一次的 A\B 测试也许能提升更好的转换率。让用户帮忙测试真实的页面元素属性，一点点让产品做得更好。

数据收集

一定要注意多去收集你所负责产品的用户数据，只有收集真实的用户反馈信息，听到用户使用产品过程中真实的反馈，你才能够更好的理解他们的使用场景。而注意了，用户的抱怨，用户所提的问题，以及他们的使用场景，本身也是最真实的一种测试，只不过有些环境，在实际研发过程中，是你没有测到，或者没法模拟。遗漏在了市场上面，用户真实测到了，而且反馈给你了，一定要注意去分析，如何在后续的迭代中避免这种问题。以用户为中心，与用户互动，让用户参与，收集用户反馈的数据，并且拿来进行数据分析。不论是大量数据还是采样数据，只要是真实用户数据，其实都是有一定价值的。

用户体验

在互联网模式，很多产品是免费模式的，但是用户体验要做到极致，还不能打任何折扣。须知，在移动互联网这块的用户都很挑剔，没有回头客。你必须得在每个环节都做到最好，多终端，多平台，多网络环境下，各种复杂场景下的测试。一切都是围绕着用户，以用户为中心的测试。通过永远的 Beta 逐渐去磨炼好的产品。

结束语

好了，其他的我们不去讲太多，练好一个扎实的测试技术基本功当然是前提，只有你能先把测试的基本功练扎实，才能去做一些更有趣的事情。重复性的枯燥

的事情，都可以留给机器去做，代码实现之后，定义好流程或者调度原则，自动化去执行，而你可以安心的做测试分析，数据分析，用户行为分析之类，岂不是更有趣？

最近网上流传这一个段子，是这么讲的：“化缘的改叫众筹了，算命的改叫分析师了，八卦小报改叫自媒体了，统计改叫大数据分析了， 大忽悠改叫互联网思维了，做耳机的改为可穿戴设备了 ，IDC 的都自称云计算了，办公室出租改叫孵化器了，圈地盖楼改叫科技园区了，放高利贷改叫资本运作了，借钱给朋友改叫天使投资了。”

这话讲的有点苛刻，不过越是高大上的事情，事实也是来自和发源于一些实实在在的东西，无中生有那是只有游戏中才会发生的事情。但是对于一个新概念，如果你一直没搞明白怎么回事，倒不妨用这种段子娱乐式最简单的去理解。做好测试，勿忘初心。

易测性与好的设计之间的关系

译者：于芳

好的单元测试用例中发现的问题揭露了设计和代码上的缺点。

最近这段时间，对单元测试的介绍急剧地增多。而在上世纪 90 年代和 2000 年早期，这一习惯似乎十分无力。通常就是这样，人们采用一项新的技术，回溯当年，各机构们也在从结构化设计方法转向以对象为中心的设计方法，他们将所有的焦点放在至关重要的权利上，同时丢弃看起来不能与新愿景整齐地融合的例行做法。因此单元测试就被忽略了。

但是，很少但是要说我们从过去的 10 年中学到了什么，那就是我们学到了单元测试是一个非常重要的学科。测试帮助我们更好地来写代码，形成回归基础使得重构和让添加成为特色更简单。有人讨论单元测试的一个十分细微的影响。处在单元层面的可测性和好的设计之间似乎有着某种极为紧密的联系。几乎是公认的说法，难测的代码必有设计问题。当你修复了设计上的问题时，测试就变得简单了。

让我们来看一个例子。

在图 1 中，我们有一个类，这个类看起来设计不错。有一个公共方法，evaluate()，它是用户与类接触的点。它将其工作传给一系列私有方法。很明显，这样的一个类没有任何错误，RuleEvaluator 做了我们想让它做的工作。现在，我们应当怎样为这个类写单元测试用例呢？我们应当能够举例来说明，用多个变量来调用 evaluate()方法然后检查结果。但是假如我们想测试 getNextToken()方法呢？

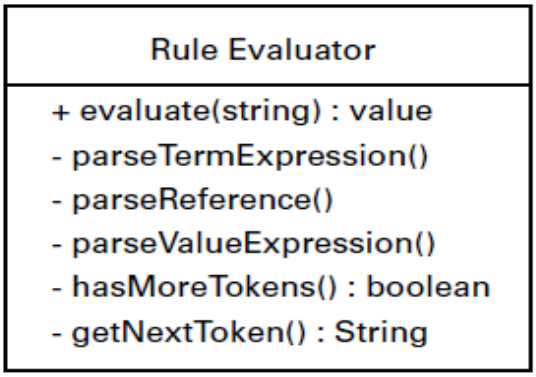


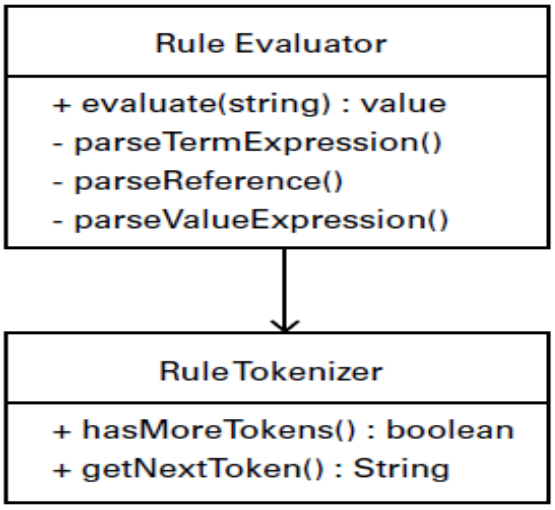
Figure 1: A typical class.

图 1. 一个典型类

getNextToken 方法是一个私有方法。理论上来看，我们应能为运用

getNextToken 方法的 evaluate 方法写测试用例，但是有时候很难衡量输入量，尤其是我们想测试一下深度调用方法的具体案例。如果我们按照这条路径走的话，我们写的用例可能就是间接的而且不具有很好的可读性。人们经常遇到这个问题，然后想着这只不过是说测试是项困难的工作的一个例子罢了。但是让我们再看一看。

RuleEvaluator 有问题吗？结果还真是有。RuleEvaluator 有三个不同的责任：评估，检索和标记。当代码在该状态下可以经常运行良好，稳定不变，各自的责任会变得紊乱起来，而该类也变得难以维护。让我们看看当我们开始修复设计上的问题时会发生什么。



An improved version.

图 2：改善版

图 2 展示了那个我们抽取一个类做标记后的系统情况。我们的测试状况现在好多了。getNextToken 方法是一个新类上的公有方法，我们可以很容易地完成对它的测试。改进设计后同时解决了测试上的问题。

这个例子也许会惹恼你。你可以认为无需重用 getNextToken()或者 hasMoreTokens()方法中的代码，将他们拉进其类中是无意义的，因为这无意义地增加了类的数量。但是作这个动作的确让代码更有模块化的概念。我们可以在将来的某个点重用 RuleTokenizer 方法。至少，现在很清晰，我们应当在标记功能代码的哪些地方做改变。我们的代码相关性更加少，改变其中一个而不影响另一个更容易。

如果这是可测性不高是设计差的一个信号的唯一例子的话，这只会被认为是侥幸，但实际不是这样。通常来说，单元测试的痛是有地方出错了的表示。有时这种痛很明显。举例来说，如果你必须创建很多对象才能得到你想测的那一个对象，这常常是过度耦合的一个表现。其他的情况更微妙。我们来仔细观察一些例子。

其他指示性的表示

大部分人认可的是单元测试应当分开来执行。我们应当能够在不影响其他任何测试的状态的情况执行每个单元测试。当我们按照这个方式行不通时，就以不可名状的难以重现的失败结束了测试。经常需要有大量的调试工作才能发现这些问题的根源。而这指示出来的设计上的问题是总体反复无常的状态。我们很多年来已经知道全局变量常有问题。单元测试就是痛的证据，而再一次，如果我们在设计问题上做功课，那么单元测试就会变得容易了。

我们的每个测试都应当当作好像他们是在一个密封的容器里一样测试的这样的事实经常帮助我们看到其他的设计问题。我经常注意到当团队试图让其系统里的关键类变得独立可测时，他们发现很多对象在被摧毁后并不释放资源。在该程序中，这种事情也许从来没被注意到过。但是，当你在跑成千上万个单元测试用例时，资源泄露很快就变得明显起来了。

最盛行的测试挑战之一也指向设计问题。团队们经常发现在第三方 API 中做单元测试很困难。一个经典的例子是用户界面密集的代码。开发者常将计算代码放在事件处理器中。能够单独测试这些计算片段也许不错，但是这几乎不可能，因为你只能通过出发用户界面的事件来练习代码的可测性。这个设计问题在于缺乏不同功能分开的考虑。一旦计算考虑从用户界面考虑分离开来，计算功能就被简单地测试了。

对我们来说，把事物分成小块比较容易去理解，而当这些小块与小块之间是彼此独立的也有益于理解。也许单元测试为什么是设计好坏的一个探测器的原因是因为，它，本质上是一个认知过程。当我们写测试用例时，我们是在经历一个推论的过程，而我们也在测试代码中让这个推论结果明晰。那些对测试来说难以理解的项目片段在没有测试的情况下很可能也难以理解。

这是一个很奇怪的局面，是不是？痛并不好玩。我们可以创建加工流程来减少单元测试的很多痛，但是到那个时候我们还会有什么？到时候也许就会玩完了，因为没有更多的能够触发我们去发现更好的设计的反馈。

第一次编写测试用例总结

作者：王远平

摘要：对第一次编写测试用例总结和感想

关键字：软件测试、用例、总结

毕业两年了，软件测试是我的第二份工作。在之前一家公司是做 JAVA 开发的，开发系统一个个的小模块，开发完后对自己开发的功能做单元测试，后面被派到其他部门支援，帮助他们写测试文档和测试他们的系统，就是在这个时候对测试有了基础的认识，对测试产生了兴趣，亲自实践才觉得测试不是之前想象的那么枯燥无味、那么没有技术含量，测试也是需要测试思想和测试方法的，所以也为我后面选择软件测试这个方向做了铺垫。

现在的公司是自己做产品的，产品比较成熟，测试流程也比较规范。我很有幸已转正就当了公司核心产品的测试负责人，因为之前有开发经验，也顺理成章地接了这个产品的自动化开发。一切都挺顺利的，自以为凭借之前一点点的测试经验和对测试的兴趣可以很好地把工作做好，可是当我第一次写用例的时候，才知道我是测试领域的菜鸟。下面是我对第一次写用例的总结：

1、软需没有评审完整，评审时没有将软需中一些主要流程和分支没有写清楚的问题提出来，导致写用例时还要和开发沟通是怎么实现的，最后用例按开发实现的写了，发现开发实现和需求是不一致的。针对这个问题，我学到的经验是软需一定要评审完整，有什么不清楚的地方一定要叫开发人员在软需中写清楚，还有测试用例要按照软需来写，也提一个建议，希望产品经理、项目经理也来评审软需，因为最清楚功能做成什么样怎么做的是他们，这样可以使软需更完整。

2、有些该测试的点没有考虑全面，单个功能的用例写了，隐含的功能不能写全面，这个可以将功能分为多个步骤，分析各个步骤在执行的时候可以进行什么操作，这个操作对这个功能有什么影响，如果有影响就把这个操作写在用例中。

3、对于多种情况多种分支的不知道如何取舍，因为功能包含多个条件，每个条件又有多种情况，如果每个条件都把全部情况写进去用例就会非常多，这时候只需要每个功能点都验证到，每个条件覆盖一种情况就好。

4、一个用例中验证的点太多，步骤不连贯，这次出现比较多的错误就是经常把有确定和取消的操作放在同一个用例里面，因为一个步骤不能同时点确定又点取消，点完确定后又要返回之前的操作再点取消，这样不连贯的情况要分在两个用例中写，同时写用例时一定要先想好怎么写，心里面要有条理，写用例前可以把每种情况先列出来，这样不会遗漏，思路也比较清晰。

经过这次写用例，在前辈们那里学到了很多写用例很好的方法，我会吸取这

次经验教训，争取下次写得更好的。做好每一个行业都没有想象中那么容易，就像软件测试，或许它的门槛比较低，但是如果进去以后没有好好学习专业知识、掌握工作流程、寻找进一步的发展方向的话，也将会被这个行业淘汰。所以我们要做的是虚心学习，不断地进步，才不会被淘汰。

测试用例设计方法—错误猜测法

作者：陈耿哲

很多软件测试从业者用到的黑盒测试用例设计方法大多是等价类划分法、边界值分析法、判定表法、因果图法和正交试验法等，其实还有一种方法不得不提到，那就是错误猜测法，这对资深测试人员尤为重要。因为随着在产品测试的实践中对产品的了解和测试经验的丰富，使用错误猜测法设计的测试用例往往非常有效，可以作为测试设计的一种补充手段。并且积累的经验越丰富，方法使用效率越高。那么到底什么是错误猜测法呢，下面我们将通过定义和实际测试案例来加深对错误猜测法的认识。

首先，我们先来看看错误猜测法的定义：有经验的测试人员往往可以根据自己的工作经验和直觉推测出程序可能存在的错误，从而有针对性的进行测试。它的要素共有三点，分别为：经验、知识、直觉。关于如何使用的问题，我们提炼出两点：

1．列举出程序中所有可能有的错误和容易发生错误的特殊情况；

2．根据他们选择测试用例。

我们知道经验是错误猜测法的一个重要要素，也就说带有主观性，那么这就决定了错误猜测法的优缺点，首先我们来看优点：

1．充分发挥人的直觉和经验

2．集思广益

3．方便使用

4．快速容易切入

对应的缺点有：

1．难以知道测试的覆盖率

2．可能丢失大量未知的区域

3．带有主观性且难以复制

综上所述，我们可以总结出错误猜测法的适用方式：

1．先用其他方法设计测试用例

2．使用错误猜测法补充用例

通过上面的理论知识，我们知道了错误猜测法的定义、优缺点及适用方式，接下来我们将通过一个实际案例来加深对错误猜测法的认识。

实例：测试**学院的课程搜索输入框

既然是用错误猜测法，那么我们首先列出可能导致搜索结果出错的情况，如下：

1．单个空格，多个空格

2．字符串前面有空格

3．字符串后面有空格

4．转义符 "\n"

5. Null

6. 特殊字符

7．通配符 *

8.空串，很长的字符串

以下是测试执行过程：

1、单个空格，多个空格

这里测试的是输入单个空格时的搜索结果

上面测试的是在搜索框中输入一个空格的情况，点击搜索按钮后，显示结果为“相关结果 0 条”，根据结果得知空格被接受了，故这里为一个缺陷。同理，我们输入多个空格，显示结果为“相关结果 125 条”，不符合预期输出，所以这也是一个缺陷，测试执行过程截图如下：

这里测试的是输入多个空格时的搜索结果

2、字符串前面有空格　　字符串后面有空格

关于空格还有另外一种情况，就是字符串中带有空格，我们分别在搜索框中输入“ a”

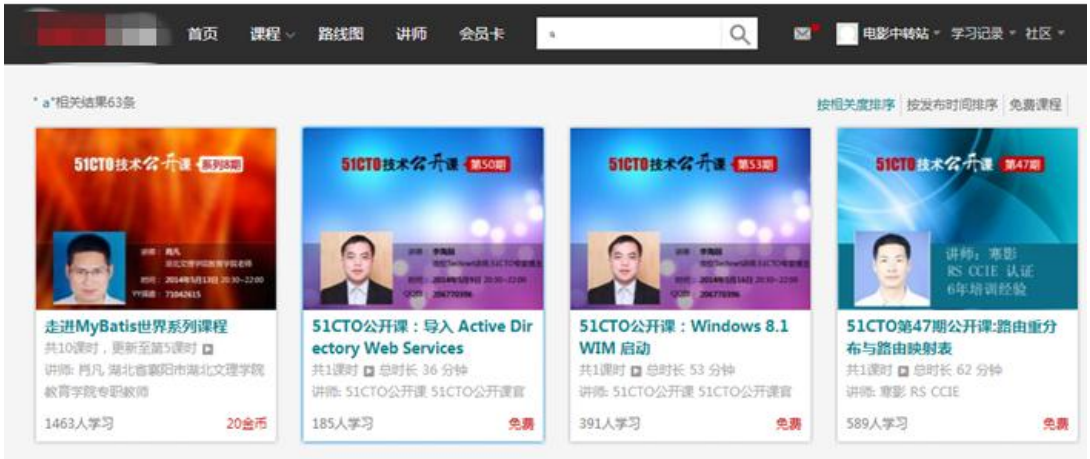
“　　a”

“a ”

“a　　”

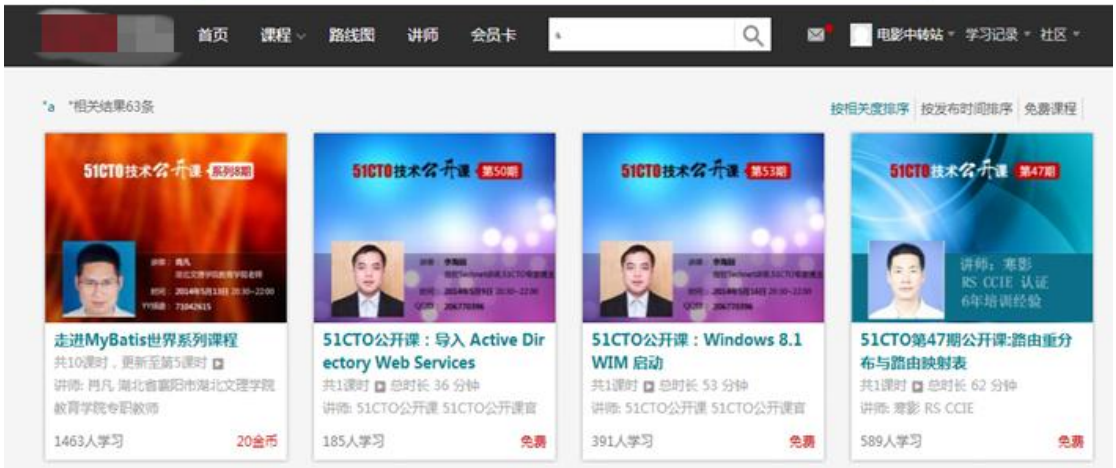
具体的执行结果截图如下，从结果中分析得知，这也是一个缺陷。

这里测试的是在字母“a”前面输入一个空格时的搜索结果



这里测试的是在字母“a”前面输入多个空格时的搜索结果

这里测试的是在字母“a”后面输入一个空格时的搜索结果

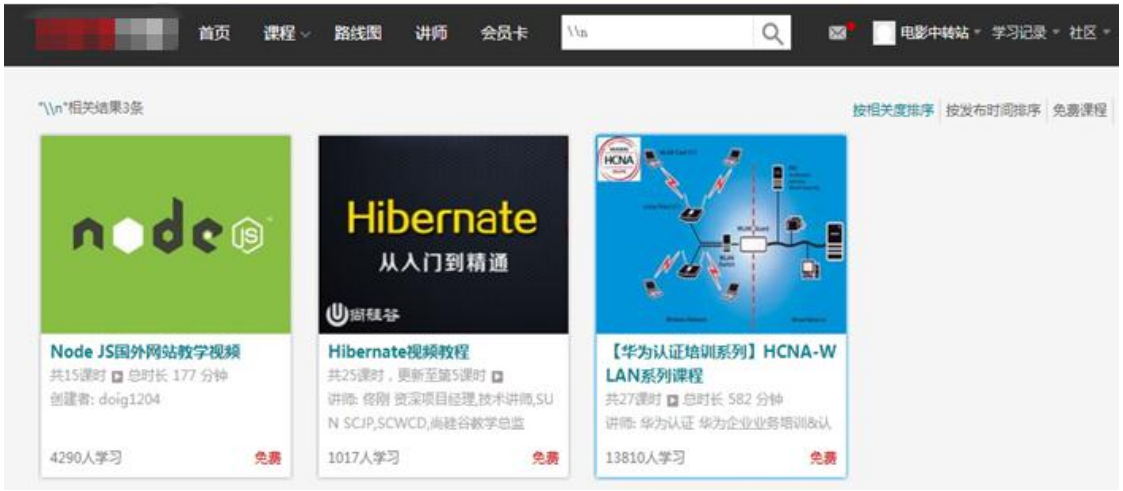


这里测试的是在字母“a”后面输入多个空格时的搜索结果

3、转义字符　　"\n"

我们输入转义字符“\n”,点击搜索按钮后，搜索框中的“\n”多了一个“\”，并在结果中显示出课程，故这里为一个缺陷。

这里测试的是输入转义字符“\n”时的搜索结果



4、Null

这里测试的是输入NULL时的搜索结果

在搜索框中输入 null，显示“null 相关结果 0 条”，符合预期结果，故这里不是一个缺陷。

5、特殊字符　'

在搜索框中输入一个单引号 ‘　，搜索出部分课程，这与预期输出不相符，故这里为一个缺陷。

这里测试的是输入一个特殊字符‘ 时的搜索结果

**6、通配符　*　**

这里测试的是输入一个通配符 * 时的搜索结果

当输入通配符时，实际输出结果与预期结果一致，所以这不是缺陷。

7、空串，很长的字符串当我们未在搜索框中输入字符串时，直接点击搜索按钮后，并未有任何显示，这符合预期输出。

空串：将光标在搜索框中单击一下，然后点击搜索按钮，并未有任何变化

输入一个较长的字符串，搜索结果显示“相关结果 40 条”，与预期输出不一致，所以这里为一个缺陷。

输入一个较长的字符串，点击搜索按钮后显示“相关结果40条”

首页
课程
路线图
讲师
会员卡
电影中转站
学习记录
社区
*1234354675676878798912343512343546756768787989123435467567678787989123435467567678787989123435467567678

输入一串足够长的字符，点击搜索后导致内存崩溃，显示500

500 Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request. Sorry for the inconvenience.
Please report this message and include the following information to us.
Thank you very much!

URL: http://edu.51cto.com/index.php?
do=search&m=index&q=1234354123435467567612341234354675676878798912343512343546756768787989123435467567687879891234354675676
Server: web41
Date: 2014/06/15 06:23:26

Powered by Tengine

经过以上的测试步骤，可以对缺陷进行总结：

1. 单个空格　多个空格
2. 字符串前面多个空格　字符串后面多个空格
3. 转义字符
4. 单引号　‘
5. 过长的字符串

至此，我们再对错误猜测发做一个总结，希望能对大家有所帮助：

错误猜测法，没有用到任何特殊的方法，只是利用直觉和经验猜测出错的可能类型，然后编写测试用例来暴露这些错误。基本思想是列举出可能犯的错误或错误易发情况的清单，然后依据清单来编写测试用例，并且在阅读规格说明时联系程序员可能做的假设来确定测试用例，也就是说规格说明中的一些内容会被忽略，要么是由于偶然因素，要么是程序员认为其显而易见。

特别的正交测试

作者：吴晨昊

一次面试一个品学兼优的女孩子，刚从 51testing 学习完毕，来我们单位面试，由于我本人对于统计学比较有兴趣，所以就问了她一个统计学的方法，什么叫做正交测试。

她的回答非常流利，而且说这是 51Testing 老师所教。但是我却说了她说得并不完全正确，我一时也没有说清楚为什么不正确，也许这就是直觉！ 面试完之后，我立即翻书找出答案，等再去找她的时候她已经离开了。于是我心里一直都忐忑不安，总感觉如果这样结束面试是非常没有礼貌，也没有价值的。

下午的时候，我鼓起勇气向 51Testing 老师询问了她的 QQ，然后很顺利的加了 QQ，并把我的结论告诉了她，她也认同之后，我这才放下了心。

虽然最终，我们单位并没有录用她，但是我还是觉得她很优秀，就和面试过的许多优秀的女孩子一样，我感觉她们都是思维敏捷，品行端正，品性善良的，我的做法也应该是正确的，我希望她能找到一个适合她的好工作！那我想这是当然的啦！

女孩所说的是---正交表中任意 2 列都是全排列。我翻书后发现，虽然任意 2 列的数值覆盖到了全排列的所有可能情况，但是在某些正交表中（不是所有的）会出现重复的情况，所以说不可以说成为任意 2 列是全排列。

那应该怎么描述这个正交表的特性呢？

$L_8(2^7)$表—任意 2 列的组合会出现重复，比如第一列和第二列中的实验号 1 和实验号 2 都是(1,1)，实验号 3 和实验号 4 都是(1,2)，而全排列是不可以有重复的数字出现的。

水平 列号 / 实验号	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

$L_9(3^4)$表—任意 2 列的组会都没有出现重复：

水平 列号 实验号	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

我由此总结一下正交表的特性：

1：表中任何一列不同的数字出现的次数相等，如果任何两列没有重复的数对，那么表中任何一列不同的数字出现的次数都是此正交表的水平数，如果任何两列有重复的数对，重复一次，则任何一列不同的数字出现的次数就为水平数的两倍，以此类推。

2：表中任何 2 列中，每种数对出现的次数相等。如果(1,2)有 2 对，那么(2,3)也有 2 对，以此类推。

3：表中任何 2 列中，去除重复的数对，则这 2 列的数对其实是有放回的全排列。

以下是无放回的全排列和有放回的全排列之间的区别：

对于 $L_9(3^4)$ 表

取任意 2 列无放回的全排列—显然这不是正交表，因为对于这 2 列，无放回的全排列是不可能出现（1，1），（2，2），（3，3）的：

实验个数是水平数*（水平数-1），此例就是 3*2=6。

水平 列号 实验号	1	2
1	1	2
2	1	3
3	2	1
4	2	3
5	3	1
6	3	2

取任意 2 列有放回的全排列—这 2 列符合正交表的特性：

实验个数是水平数的平方。此例就是 3^2=9。

水平 列号 实验号	1	2
1	1	1
2	1	2
3	1	3
4	2	1
5	2	2
6	2	3
7	3	1
8	3	2
9	3	3

对于正交测试，平时我们用到的并不多，我想原因是：

1、在测试中，大多数情况之下，都需要设计测试用例覆盖所有的测试点，而正交测试规定了测试数据，虽然均匀分布，但无法使得所有测试点都覆盖全面，导致意义不大。

如：对于正交表 $L_9(3^4)$

不使用正交表，测试用例数量是：81，使用正交表，则测试用例数量是：9.

2、测试时间匆忙，很少会想到选择正交表来测试。

虽然正交测试有它的缺点，但是在特定的场合，通过统计学的正交分析，我们可以利用它对于软件进行某些分析，辅助我们对于软件的认识和理解。

以下我就描述一下正交测试法的某些用处：

正交测试就好像一个黑盒测试，通过对于输入数据的各种组合运算，得到输出数据：

如果每一个因子都有 3 种水平，要做到全部覆盖测试用例，那么测试用例的个数，就是

3^3=27 个测试用例。

而使用了 $L_9(3^4)$ 正交表，就可以使用 9 个测试用例进行分析。

它的作用：

1、得出各个因子对于结论的影响力的大小。（可以找出影响力最大的因子和最小的因子）

因子 1：学历（1：硕士及以上，2：本科，3：大专及以下）

因子 2：年龄（1：40 及以上，2：30~39，3：29 及以下）

因子 3：是否结婚 （1：已婚，2：未婚，3：离婚）

结论：幸福指数 (结论为模拟数据)

	学历	年龄	是否结婚	幸福指数
1	1	1	1	80
2	1	2	2	85
3	1	3	3	70
4	2	1	2	75
5	2	2	3	68
6	2	3	1	78
7	3	1	3	65
8	3	2	1	75
9	3	3	2	80
T1	235	220	233	
T2	221	228	240	
T3	220	228	203	
R（极差）	15	8	37	

T1 是每一列的数字为 1 的幸福指数相加，T2 是每一列的数字为 2 的幸福指数相加，以此类推。

R 是 T1，T2，T3 中最大的数减去最小的数的差。

从中，我们可以看到，因子"是否结婚"的 R 最大，而因子"年龄"的 R 最小。

这说明，对于幸福指数的影响力最大的因子，是"是否结婚"，其他的影响力都不如它大。

这样，在测试的时候，我们可以着重考虑这个因子，而进行测试，具体如何测试，则可以按照具体情况，具体分析。

2、推断出特定的测试用例。

而在是否结婚这个因子中，我们发现 T2 的值最大，是 240，所以我们可以知道未婚的幸福指数相比较而言是最大的。因此，因子 3 选择 2：未婚。

在年龄这个因子中，T2 和 T3 是一样大的，都是 228，所以想取幸福指数大的年龄，"2：30~39，3：29 及以下"都是可以的。因此，因子 2 选择 2，或者 3。

以此类推：因子 1 选择 1：硕士及以上。

如果现在要求我们设计测试用例，来测试最大的幸福指数配比，对于这三个因子，我们可以设计的测试用例是（1，2，2）或者（1，3，2），这样的设计出来的幸福指数是非常高的可能性是非常大的。

而我们看到，（1，2，2）在正交测试中，是有这个测试用例的（实验号 2），但是（1，3，2）是没有的，因此，这样设计的测试用例显然不属于正交测试，但的确是运用了正交测试法，而推断出的测试用例。这也就是正交测试法的另一个运用，推断出测试用例。

希望这些计算方法对于您的测试有所帮助。