泽众软件测试网
电子期刊

SPASVO

第二十九期
2014年9月

最常见的移动App Bug——崩溃的测试用例设

测试数据管理框架

移动APP性能——如何保证高质量

了解测试用例设计的要领

使用元数据设计测试用例

测试我们的移动星球_移动自动化测试

更好的测试用例设计

你要怎么切你的披萨?

上海泽众软件电子期刊

2014 年 9 月 第二十九期

主办单位：上海泽众软件科技有限公司

联系电话：021-61079698 传真：021-61079698 转 8017

意见反馈：fangmh@spasvo.com 投稿：fangmh@spasvo.com

公司地址：上海市普陀区曹杨路 450 号绿地和创大厦 18 楼 1801 室

邮政编码：200063

公司主页：www.spasvo.com

目录

最常见的移动 App Bug——崩溃的测试用例设..... 4

测试数据管理框架..... 5

移动 APP 性能——如何保证高质量..... 12

了解测试用例设计的要领..... 14

使用元数据设计测试用例..... 17

测试我们的移动星球_移动自动化测试..... 18

更好的测试用例设计..... 22

你要怎么切你的披萨?..... 24

泽众软件——专业软件测试门户
2014 年 9 月 第二十九期 电子杂志期刊

最常见的移动 App Bug——崩溃的测试用例设

介绍：

我们的日常生活中对移动设备越来越多的使用意味着移动 App 测试这个主题已成为需要考虑的一个无法避免的问题。根据最近的调查研究，用户难以容忍有 bug 的移动 App。

移动 App Bug 的影响是用户体验差、App 的商店评级下降、用户换用竞争对手的 App，声誉和信誉损失、最后销售量减少，如果它是一个付费 App 的话。

移动 App 测试与传统台式机测试相比有一定的复杂性。这些复杂性可以被分类为：

环境（大量的设备，各种移动 OSs，适应频繁 OSs 变化） 。

设备（触摸式和非触摸式设备，有限的内存容量，电池耗电量） 。

网络（不同的网络和运营商，在不好或无网络的情况下的 App 行为，离线支持） 。

可用性（方向，触摸，多触摸，缩放，分页和导航的局限性，各种干扰，如来电，来电短信，闹钟，和低电量警报） 。

所有这些手机专有的复杂性需要新的针对移动 App 测试的测试用例设计方案。

最常见的移动 App Bug：

为了确定最常见的移动 App Bug，进行了一次研究，其结果发表在国际测试会议上[1] 。

为了这个目的，准备了一次在线调查思考参与者的移动测试经验并发表在移动 App 开发和测试相关的专业社会团体内。

有针对性的参加本次调查的主要有移动 App 测试人员和开发人员。结合几个结果，最常见的移动 App Bug 在对调查结果进行统计分析后确定。

根据调查的结果，移动 App 崩溃是最常见的移动 App Bug ，这是预料中的结果，因为很容易发现一个移动 App 崩溃。Android OS 上一个写着“强制关闭错误”的弹出窗口跳上屏幕；当发生崩溃时，iOS 中 App 屏幕突然消失消失。最坏的情况下，App 崩溃可能会导致系统故障，操作系统崩溃。

移动 App 崩溃原因：

为什么移动 App 经常崩溃？App 崩溃有几个原因：从平台或环境到开发问题。

一些崩溃原因（排名不分先后） ：

设备碎片化：由于设备极具多样性，App 在不同的设备上可能有表现不同。

带宽限制：带宽不佳的网络对 App 所需的快速响应时间可能不够。

网络的变化：不同网络间的切换可能会影响 App 的稳定性。

内存管理：可用内存过低，或非授权的内存位置的使用可能会导致 App 失败。

用户过多：连接数量过多可能会导致 App 崩溃。

代码错误：没有经过测试的新功能，可能会导致 App 在生产环境中失败。

第三方服务：广告或弹出屏幕可能会导致 App 崩溃。

移动 App 崩溃的测试用例设计：

测试用例是移动测试最重要部分之一。

准备和执行预先定义的针对移动 App 崩溃的测试用例将简化和加速移动 App 崩溃的测试。

一些通用的触发移动 App 崩溃的测试场景，如下：

1 验证在有不同的屏幕分辨率，操作系统和运营商的多个设备上的 App 行为。

2 用新发布的操作系统版本验证 App 的行为。

3 验证在如隧道，电梯等网络质量突然改变的环境中的 App 行为。

4 通过手动网络从蜂窝更改到 Wi-Fi ，或反过来，验证 App 行为。

5 验证在没有网络的环境中的 App 行为。

6 验证来电/短信和设备特定的警报（如警报和通知）时的 App 行为。
7 通过改变设备的方向，以不同的视图模式，验证 App 行为。
8 验证设备内存不足时的 App 行为。
9 通过用测试工具施加载荷验证 App 行为。
10 用不同的支持语言验证 App 行为。
显然，还会有更多的导致 App 崩溃的 App 特定场景。

结论：
在这项研究中，展示了针对移动 App 崩溃的通用测试案例。
如果移动测试团队在他们的测试场景中准备并执行这些测试用例，那么早在开发周期就可以找到崩溃相关的 Bug。 然后，开发团队将阐明崩溃原因，并找出一个解决所有 Bug 的通用方法。最后，App 质量和用户满意度就会增加。

测试数据管理框架

如果所有的测试员都有一个共同的难题，那大概就是管理他们的测试数据了。无论你在测试中扮演什么角色或你是哪种类型的测试员，要使得你的测试数据正确还蛮难的。已经做过不少不同领域的项目，我们总结：没有一个单独的解决方案可以管理测试数据。事实上，甚至没有这样一个单独的测试数据管理问题。理由很简单：测试数据应该要满足你的（基于你的业务，规章，结构及可用环境等因素的）特定测试需求。所以，从测试数据管理的角度，没有哪两个情况是一样的。

你该如何管理测试数据？

测试是在压力下进行的，因为迭代开发模型被更频繁地使用，且改变的时间一直在减短。结果，让你的测试数据恰当的可用时间也在压力之下，于是对恰当测试数据管理的需求不断在增加。另一方面，我们注意到在我们的日常工作中，越来越多的公司开始寻找方法解决他们的测试数据管理问题。但是，另一方面，我们又不得不意识到，接受这一挑战时，我们并没有什么可做的。我们不能求助于一个覆盖测试数据管理所有方面的文件程序。我们也不能使用一个测试数据管理工具应对测试数据管理的所有方面。这是不可能的，因为这种流程或工具不存在。顶多现存流程和工具只覆盖一些测试数据管理问题。那么，下一个题就是：“如何缩小这个差距？”我们能创建一个仪器来把测试数据管理作为一个整体来解决而不论准确情况吗？或者换句话说，我们能创建一个对实际帮助我们解决测试数据管理问题来说足够具体且同时被应用于任意（测试）项目的工具吗？

我们已经确定，寻找一个通用的解决方案并不可取。于是我们就想到或者我们不应该一开始就找解决方案，而应该试着把重点放在该如何更好地理解手边的测试数据管理问题。理想情况下，加强了理解，最后就可以想出一个按部就班的设计并实施既定测试数据管理问题的解决方案的方法。

创建一个测试数据管理框架的想法诞生了。因为该框架并不明确限定于任何特定的测试数据管理解决方案，所以它应该适用于任何给定情况。

测试数据管理框架详解

测试数据管理框架需要包含两大部分。一部分记录一个开发组织对测试数据管理的需求，另一部分创建一幅满足那些需求的测试数据管理实践的路线图。图1列出了测试数据管理框架的要素。

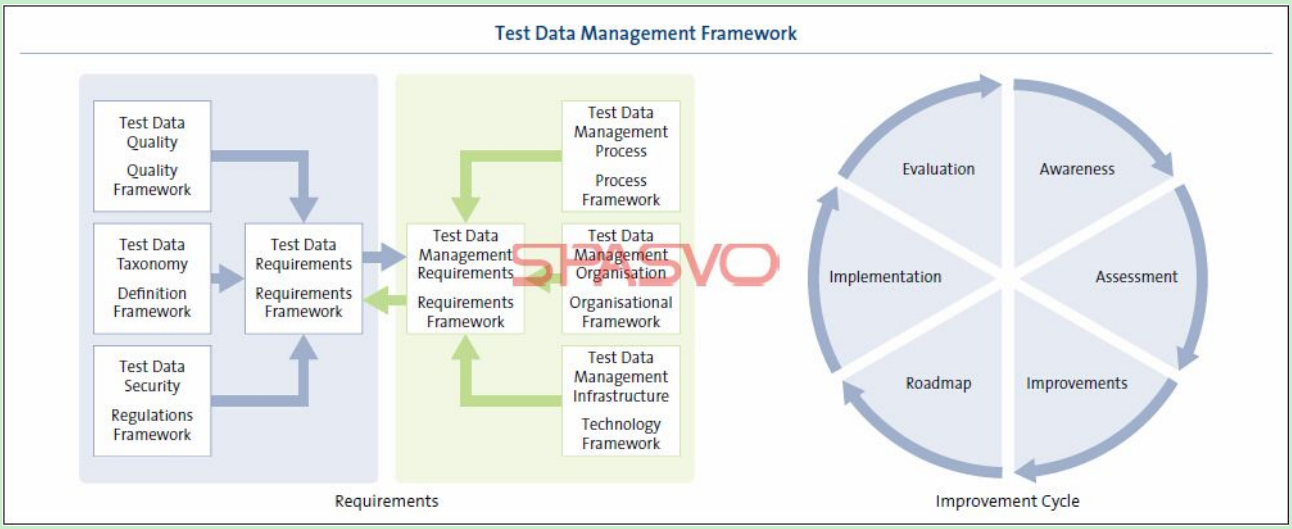


图 1. 测试数据管理框架详解

测试数据管理框架的第一部分叫做需求框架。它由测试数据管理的一系列要求组成，且能让你证明并引出一个组织的需要用来管理测试数据的准确需求。测试数据管理框架的第二部分被称为提升周期。它由一个周期的按部就班的用以设计并实施测试数据管理提升的方法组成。框架的两部分紧密合作。需求框架允许你决定目前的测试数据管理实践及偏好，提升周期允许你去设计并实施可以导致期待情况的测试数据管理提升。或者，反之亦然，当引入了测试数据管理提升，需求框架就可以对所实现的提升结果进行评估。

更具体点儿，测试数据管理框架包含三个子框架。

▪需求框架明确区分测试数据本身的实际需求及实际管理测试数据的需求。这就生成两个子框架。

▪测试数据需求子框架专注于测试数据的特性。它以一种测试数据应该是怎么样的通用方法表现。相对地，测试数据管理需求子框架再一次以一种通用方法专注于测试数据该如何管理。

▪最后，测试数据管理提升周期子框架要看一个组织该怎样决定其当前测试数据管理实践，其期待（将来）测试数据管理实践，及一个线路图的。接下来，我们详细说说每一个子框架吧。

借用“销售渠道”说明

我们将借用一个假定的“销售渠道”应用来说明测试数据管理框架，考虑以下假设：

▪客户驻留在一个 SQL2008R2数据库，机会和员工驻留在一个 Oracle11g 数据库。

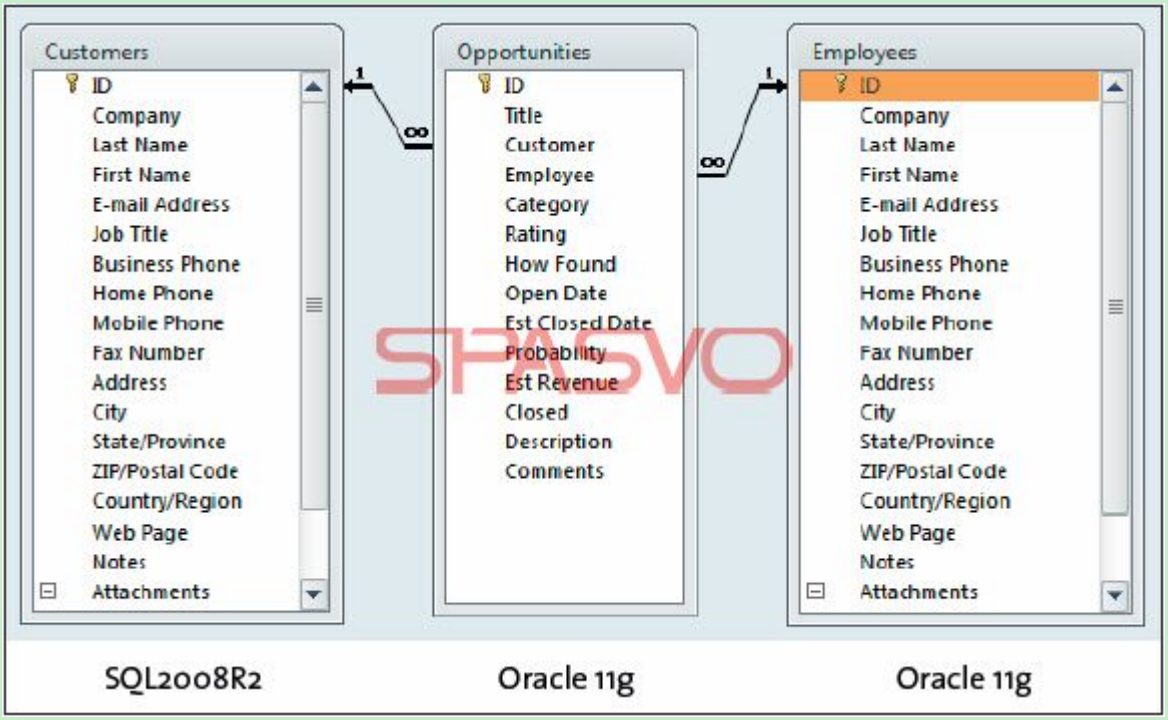


图2. “销售渠道应用”的数据模型

▪生产表中当前记录的数量

▪客户：1000

▪机会：6700

▪员工：60

▪过去测试“销售渠道”会导致客户数据隐私的重大问题，结果，管理决定当所有客户数据从生产环境中选择并被加入任何开发或测试环境中时必须被隐藏。

测试数据需求

测试数据需求子框架注重测试数据本身的特性。它包括另外四个解释适当测试数据应该遵循的需求的子框架，需求框架简单列出了一些基本使用于（测试）项目且是测试数据需求子框架重点的通用测试数据需求。通用测试数据需求为评估一个组织的测试数据需求提供指导。一些需求或许不适用于某特定情况。如果是这样，那么它们很容易被忽视。如果不是，，它们就应该由将要设计的测试数据管理解决方案来实现。定义框架，质量框架和规章框架分别提供现存不同种类测试数据及测试数据应该遵循的质量与规章标准的更深入的信息。它们提供解释需求框架中通用测试数据需求所需的背景信息。

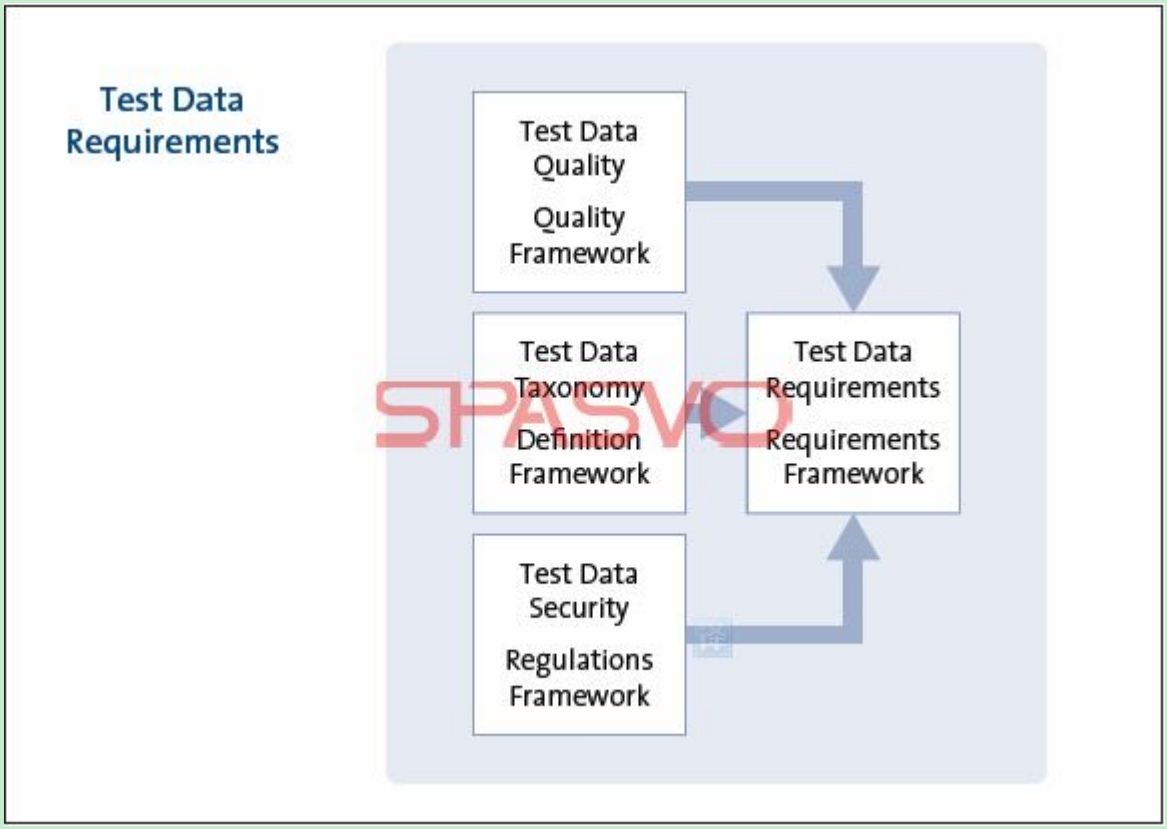


图3. 测试数据需求子框架

测试数据管理是个复杂的任务且当试着满足测试数据需求时可以想出许多不同的测试数据策略。制定一个关于与手头（测试）项目相关的每个（测试）数据需求的单独测试数据策略大概太耗时了。因此，我们希望能够为更大的（测试）数据组选择测试数据策略。这样我们就需要一个机制来定义可以以同样方式对待的（测试）数据组。定义框架或测试数据分类系统提供了这个机制且是建立在以下三方面上的：

▪测试数据特性（例如测试数据类型，生产相似性，一致性，统一性，数量）

▪测试目标（组件测试，系统测试，验收测试）

▪测试环境（DTAP 模式）

借用“销售渠道”说明

对于一个单元测试，开发人员只需要一些来自每个表格的记录（比如：10名员工，100次机会，100名客户）以充分覆盖代码。但是对于测试性能，环境必须是类似生产的，这意味着在一个专门环境中照搬所有表格。对验收测试业务，测试数据（如：外国客户，不同状态下的机会（打开的/关闭的），不同国家的员工）中包含所有不同种类的情况也很重要。无论哪个测试环境，有一致的测试数据意味着你不能只选一个表格获取数据的。在我们的例子里，客户和员工都与机会相关联，所以所有这些表格中的记录都要被挑选。最后，在每个（测试）项目里建立测试数据是一项很重要的活动。没有恰当的测试数据，就无法执行一个单独的测试用例。

但是接下来又有问题了。什么是恰当的测试数据？什么时候我们用的测试数据质量够了？质量框架来回答。框架里，当测试数据满足以下需求时，我们觉得测试数据适合测试目的（且是高质量的）：

▪测试数据符合适用于你公司内的通用数据质量属性（如：准确性，完整性，可达性等）

▪测试数据覆盖测试需求

▪测试数据反映真实生活数据

每个公司都要处理他们以安全方式处理的数据。根据法律，个人数据必须受到保护而不被无意使用，被认为机密的非个人数据不该泄漏出去。无论这个责任最初目的是什么（国际立法或仅仅是出于自身利益），公司受到的来自暴露出去的敏感数据的伤害都相当大。规章框架解答了该如何管理测试数据（和测试环境）以满足相关测试数据安全需求。理想情况是，该政策可以成为公司安全政策，测试政策或质量政策的一部分。

借用“销售渠道”说明

可以用三种方法按要求隐藏客户数据：

▪搞乱基于模式的公司名（比如用 X 或 Y 代替特性）

▪用不乱但虚构的数据（如 John Tester， Teststreet 10 in 1000 Testland）替代敏感数据

▪在像东大街一样的地方加入任意数量

▪基于计算程序用自己的数据代替现存数量

测试数据管理需求

测试数据管理需求子框架解释该如何管理测试数据。其结构与测试数据需求子框架很相似。它也包含四个子框架。需求框架相似地列出了一些通用测试数据管理需求。流程框架，组织框架和基础设施框架各自提供关于专门用于测试数据管理背景的典型管理方面（流程，人，技术）的更深入信息。他们提供解释需求框架中通用测试数据管理需求所需的背景信息。

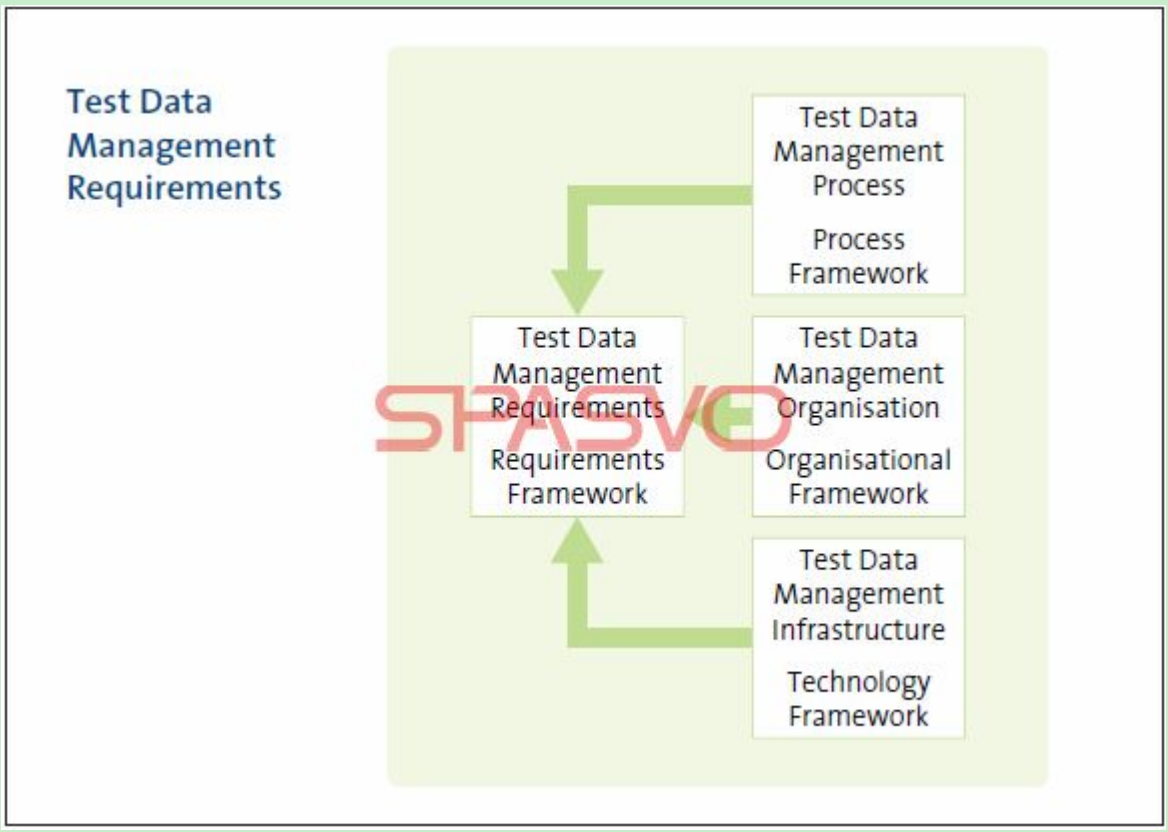


图4. 测试数据管理需求子框架

管理你的测试数据是必须包含在你的整个测试流程中。因此流程框架简化了这整个流程及在其之中执行的不同测试活动的测试数据生命周期（定义，建立，使用，净化等）。从管理的角度，这可以确保测试数据管理活动在恰当的时间执行。包含测试数据也可以让你更轻松地定义关于测试数据管理活动的恰当职责。组织框架放大了这些职责。框架内，开发出了组织测试数据管理职责的不同方法。有不同的组织类型，范围从每个负责自己测试数据的测试员到一个提供测试数据服务的中心测试数据单元。基础设施框架专注于你测试数据管理活动的技术方面。它概述了有哪些测试数据管理工具以及它们如何支持你。基础设施框架内，一个测试数据管理工具的定义被很广泛地使用。不仅专门的测试数据管理工具，用于测试数据管理设置的测试自动化工具还有其他可以有所帮助的工具在这儿都可以被考虑。

借用“销售渠道”说明

为了在恰当的时间准备测试环境，包含一个模板的测试策略中加了一段。测试经理不得不用他的特定测试数据需求来完成这个模块。DBA 团队全权负责创建子集并将它们操纵调动到如模块所示的环境中。此外，无论需求是否与已被呈现在环境和/或其他测试数据需求中的数据冲突，DBA 都会检查。增加了这一步，测试数据准备更高效了。

测试数据管理提升周期

测试数据管理提升周期子框架评估当前测试数据管理实践以基于线路图设计将来的实践。子框架定义一个包含留个六个连续步骤的重要周期。每一个提升周期都要过一遍这六个步骤且第一步都是“生成意识”。这是测试数据管理提升的一个通用方法，比如它只说采用什么步骤但实际上不告诉你要提升什么。

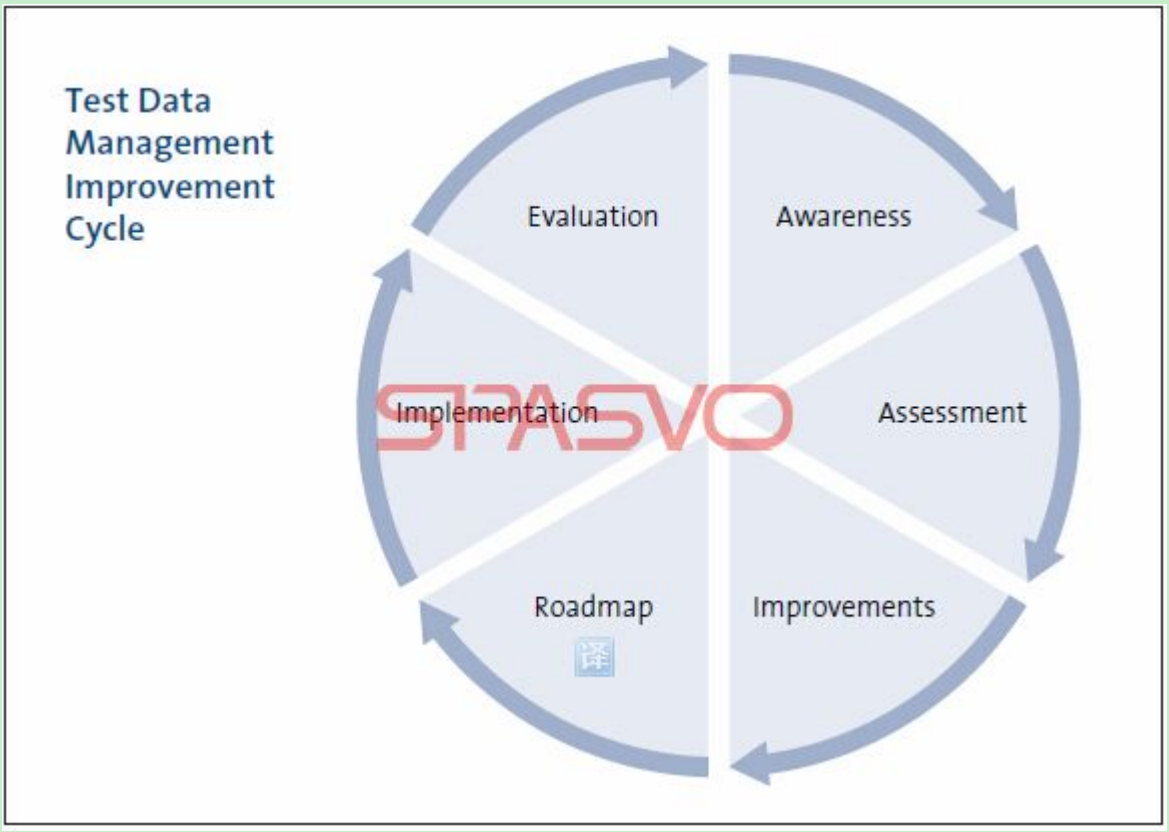


图5. 测试数据管理提升周期子框架

1.生成意识

这个步骤就是识别并定义测试数据管理提升的机会。提升测试数据管理实践的需求应该让所有的利益相关者一目了然。利益相关者也应该同意测试数据管理框架作为使用的参考模型。

2.评估当前测试数据管理实践

通过使用测试数据管理框架作为一个参考模型分析当前测试数据管理实践。这就决定了当前实践的优缺点，确定的缺点的原因是被研究了，并与测试流程提升中确定的机会相关联。

3.定义将来的测试数据管理提升

该步骤主要是决定测试数据管理实践将来想达到的的状态。提升目标被设定，实现提升目标的解决方案也被制定了。

4.定义提升线路图

实现这些目标并实施前一步中所确定的解决方案的一个策略已设计好了。资源定好了，行动计划也制定好了。

5.实施

现在提升线路图已被执行。确定的提升和解决方案已被实施。测试人员受过培训，试点项目已正式拉开序幕。简而言之，新测试数据管理实践已开始投入使用且固定在公司企业中。

6.评估

最后但并非最不重要，评估已实现的提升。目标是否被实现，实施的解决方案是否进行地不错都被证明了。如果利益相关者对结果很满意，那么测试数据管理提升在这里可以停止了，否则就可以在测试数据管理提升周期中制定一个新的途径。评估步骤的结果就被用来生成新想法。

借用“销售渠道”说明

多个测试人员在相同的环境中测试时，存在使用彼此测试数据的风险。也就是说，必须建能影响/推迟测试执行的额外测试数据。这点可以通过采用一些规则避免并改善。

▪添加测试员首字母作为测试数据的前缀（例：雇员），这样谁可以使用这个测试数据就一目了然。

▪分开不同测试员可使用的测试数据（例：所有英国顾客数据由测试员 A 使用，所有荷兰顾客数据由测试员 B 使用）

▪选择并提取所有所要求测试数据的额外份额（例：10%）以保存。

总结

测试数据管理框架给任一愿意处理测试数据管理问题的测试专家或项目经理提供帮助。不要死盯着一个（我们觉得无法发现的）通用的测试数据管理解决办法，测试数据管理框架提供方法理解一个组织的测试数据管理需求并设计和实施这些需求的正确解决方案。由于框架的目标不是找出一个通用解决方法，所以框架可以被广泛用于帮助参与测试数据管理的任何人。

注意点

[1]这个数据模式是基于 MS Access 2010中的样本模板。

[2]测试数据策略：你管理（加入测试数据到测试环境，整合测试数据以便下回使用，从测试环境中移除测试数据，维护测试数据）测试数据以便满足这些测试数据的需求。

移动 APP 性能——如何保证高质量

加速之必要：

不考虑技术，有一件事是肯定的——人们似乎总是希望可以更快。根据各种各样的研究，现在用户只愿意等待一个 web 应用程序加载三秒或更短的时间，超过的话，他们就会变得越来越不耐烦或者干脆换一个应用程序。这些高期待不断被压到移动 web 之上；现在还压到移动 App 上。就像 Web，现今的移动移动 app 都有它们自己的性能问题并需要做出一些微调。最新研究表明，过去，在手机上获取 app 时，47%的移动用户主要是抱怨速度慢且反应迟钝。App 在苹果的 app 商店上被谴责 “慢得可怕”。对于 Facebook 的 iPhone 应用程序，38，000 条评论中有超过 21，000 的用户只给 app 一星的评价。用户多数表示 app 慢，死机，“一直在加载”。

“移动 app 根据它们在 app 商店的排名而生存或死亡……排名不佳，用户采用率就降低” 佛里斯特研究公司的 Margo Visitacion 这么说道。这或许就是为什么 80%的品牌 iPhone，Android 和 Blackberry 应用程序无法达到 1，000 的下载量的原因。拙劣的移动 app 性能直接影响用户获取和用户维系。那么该做些什么以保证你的移动 app 性能尽可能的强大呢？

通过捕捉现实中移动 app 性能“获得真实信息”：

移动 app 性能首先，最重要的是：为了真正理解移动 app 性能，你必须衡量你的真正用户正在体验的性能。在数据中心的模拟机上进行测试可以有所帮助但是它基本和你的真实终端用户的真正体验无关。你的数据中心和你的终端用户间有许多影响性能的变量因素，包括云，第三方服务/集成，CDNs，移动浏览器和设备。衡量真实用户是在巨大的复杂物上精准评估性能并确定一个性能提升的基准线的唯一方法。衡量你的真实用户体验的性能可以让你就移动 app（关键参数方面的，如你客户使用的所有的地域，设备和网络）性能做出报告。

现在，移动 app 测试和使用 SDKs 监控以提交本地 app 可以让你快速轻松地鸟瞰你所有客户的移动 app 性能。

负载测试从终端用户角度看也很重要，尤其是在开始一个 app 前，综合测试网络可以让你在不同的条件下评估性能水平。

理解拙劣性能的商业影响：

确定移动 app 性能问题以及它们对转化的影响很重要：比如，你会注意到移动 app 的响应时间增加与转化的减少息息相关。这样你就可以进行分类，基于一些考虑（如：我的哪些客户，多少客户受到影响了）按轻重缓急解决问题。如果一个地区的流量份额很高但有问题，而另一个地区的份额较少，那你就知道该优先解决哪个性能问题了。

确保第三方集成有效：

就像 web 应用程序，许多移动 app 为了给终端用户提供更丰富更满意的体验吸收了大量第三方服务的内容。一个实例便是社交媒体集成，如 Twitter 就被集成到奥林匹克移动 app 中了。很不幸，如果你依赖第三方服务的话，你就会完全受限于他们的性能特点。在使用一个第三方集成的 app 前，你需要确保集成无缝顺利且可以提供你期待的性能。此外，你还要确保一直监控着第三方性能且你的 app 被设计得可以完好地降级以防第三方的问题。

让你的移动 APP 快起来：

在这个飞速运转的移动 app 世界有一句格言比任何时候都真——快比慢好。你可以使用一些特定工具和技术让你的移动 app 变得更快，包括以下：

▪优化缓存 – 让你的 app 数据完全脱离网络。对于内容多的 app，设备上的缓存内容可以通过避免移动网络和你的内容基础设施上的过多障碍以提升性能。

▪将往返时间最小化 – 考虑使用一个可以提供无数能够加快你的 app 服务的 CDN，包括减少网络延迟的边缘缓存，网络路由优化，内容预取，以及更多。

▪将有效荷载规模最小化 – 专注压缩，通过使用任意可用的压缩技术减少你的数据的规模。确保图像规模适合你最要的设备段。同样，确保你利用压缩。如果你有要花很长时间加载的内容，那么你可以一点一点儿的加载。你的 app 可以在加载时使用该内容而不是等整个加载完成后才使用它。零售 app 经常使用该技术。

▪优化你的本机代码 – 写得不好或全是 bug 的代码也会导致性能问题。在你的代码上运行软件或检查代码以找出潜在问题。

▪优化你的后端服务性能 – 如果对你的 app 进行了性能测试后你发现后端服务是性能削弱的罪魁祸首，那么你就不得不进行评估并决定该如何加快这些服务。

总结：

智能手机用户当然也是“快比慢好”，他们期待他们的 app 可以飞快。几乎每隔一段时间，移动运营商和智能手机制造商都要宣布更快的网和设备，但不幸的是，移动 app 本身的速度却跟不上。

最主要的原因是一组截然相反的目标使得实现飞速性能变得很困难。移动 app 开发者总希望提升速度的同时可以提供更丰富的体验。需要更多内容和特点能够快速地覆盖宽带，内存和计算机能力。

本文给出了一个简短的本地移动 app 的性能最佳实践的例子。性能调整的空间很大，但错误的空间

同样也很大。因此，早点测试你的 app，绝不要药听天由命。记住——快总比慢好。

了解测试用例设计的要领

显然，无论缺陷预防工作贯彻落实地多好，软件组件总有缺陷。这很明显，因为开发商无法阻止/消除软件开发周期的所有缺陷。因此，软件必须进行彻底的测试，然后才交付给最终用户。测试人员的责任是：设计既可以（ i ）找软件缺陷，又能（ii ）评估该软件的性能，可用性和可靠性等方面的测试。

现在，为了实现这些目标，测试人员必须（往往是从一个非常大的执行域中）选择和/或制定测试用例的有限数量。不幸的是，完整的测试通常不是在这个范围，预算和时间的约束内实现和/或执行的。重要的是，当测试开始失控且不按计划地运行时，由于预期无法实际，测试人员往往承受了来自管理层和利益相关者的巨大压力。

因此，测试人员必须有效地计划测试并制定正确的测试用例，选择并执行合适的用例，监控过程，以确保有效利用工作资源和时间。所以，要列出这些无疑是一项艰巨的任务；要有效地实施，测试人员需要受过适当的教育和培训并拥有赢得管理层支持的能力。

一般情况下，测试人员会用两种不同的测试方法，其中，使用常规方法，测试人员主要是尝试用所有可能的输入去测试一个模块或组件，用所有可能的软件结构去实践。尽管这种做法仍在使用，测试人员却在慢慢灌输推理软件的一切的价值，最终使他们能够检测出所有可能存在的缺陷。但见多识广且有学问的测试员们都明白，这在现实或经济上是不可行，不可实现的目标。

现在，另一种方法可能就是让测试员们随机选择测试输入，希望这些测试能将大的缺陷找出来。不过，测试专家认为，随机生成的测试输入在评估系统的质量属性方面表现纪录欠佳。所以，从测试的角度来看，这是一个无休止的争论和悬而未决的问题。尽管如此，我们还是认为，测试员的最终目标是了解测试的功能、输入/输出域和使用环境，等等。

同样，对于某些特定类型的测试，测试人员也需要详细地了解代码是如何构造的。此外，测试人员也需要利用关于常在软件开发或维护过程中生成的特定缺陷的知识。有了这些信息，测试者就必须明智地选择测试输入的子集，以及被认为最有可能找测试过程中条件和限制内的缺陷的测试输入组合。然而，这个过程需要时间和精力。所以，测试人员知道且赞同：只有开发出基于执行的测试的有效测试用例，才能最大化和/或优化对时间和资源的利用。

“有效测试用例”我们是指：“一个很可能找出缺陷的测试用例” 。因此，制定有效测试用例的能力对于一个组织迈向一个更高质量的测试过程来说是非常重要的；反过来， 一个组织迈向一个更高质量的测试过程对制定有效测试用例的能力也有许多积极影响。

例如，如果测试用例是有效的，那么：

检测缺陷的概率更大。

更有效地利用组织资源。

测试再用的可能性更高。

更符合测试、项目进度、预算，更重要地，提供更高质量的软件产品的可能性。

测试用例设计方法 - 概念化

上面介绍了有效测试用例的种种好处，但缜密考虑测试人员用来设计这些有效测试用例的方法也同样重要。为了回答这个问题，有必要把软件作为一个精心设计的产品来查看和/或检查。现在，有了这个观念，就有两个基本方法可以用来设计测试用例：

黑盒（有时也称为功能或规格）测试方法。

白盒（有时也称为 clear 或透明盒 ）测试方法。

使用黑盒测试方法，测试人员把 SUT （测试中的软件）当作一个不知道其内部结构（即如何运作）的不透明盒子，测试人员只知道它的作用。使用这种方法的 SUT 的大小可以是一个简单的模块、成员函数、对象群、一个子系统、或一个完整的软件系统。此外， SUT 的基础行为或功能的描述可以由正

式规格，输入/处理/输出图（ IPO），或一套定义明确的先决、后置条件来提供；重要的是，另一个值得一提的信息来源是：需求规格说明文档，通常描述 SUT 的功能，输入及预期输出。现在，鉴于上述来源，测试员提供指定输入到 SUT，进行测试运行，然后确定所产生的输出是否与说明文档中提供的一致。因为黑盒测试方法只考虑了软件的行为和功能，它通常被称为功能测试，或基于规范的测试。这种方法特别有用，极有助于找到要求和规格中的缺陷。

与此相反，白盒测试方法关注将被测试的软件的内部结构。因此，使用白盒测试方法来设计测试用例，测试人员应该先了解结构，且为了实现这一目标，必须可随时参考和理解代码或适当的类伪代码的要求。一旦对结构有了必要的了解，测试者就可以选择合适的测试用例去实践特定的内部结构要素，并确定它们是否正常工作。例如，测试用例通常被设计来实践所有语句或发生在一个模块或成员函数中的真/假分支。但是，由于白盒测试的设计，执行和结果分析非常耗时，这种方法被限制和/或通常只适用于软件的小部分，如模块或成员函数。然而，白盒测试方法对于找出设计和基于代码的控件的逻辑缺陷和顺序缺陷，初始化缺陷和数据流缺陷等特别有用。

然而，从测试员的角度来看，要实现向用户提供低缺陷高质量的软件的目标，必须把这两种方法都用来设计测试用例。另外，这两种方法都支持测试员选择有限数量的将被用于测试的测试用例。这两种方法可以相互补充，因为每个都或许有助于找到某些特定类型的缺陷。重要的是，有了使用这两种方法设计出的一组测试用例，测试员找到 SUT 中各种不同类型缺陷的机会就增加了。

测试员还有一套有效的可再用的用来进行回归测试（更改后的重新测试），以及软件测试的新版本。

Test Strategy	Tester's View	Knowledge Sources	Methods
Black Box		1. Requirements document 2. Specifications document 3. Domain knowledge	1. Equivalence class partitioning 2. Boundary value analysis 3. State transition testing 4. Cause and effect graphing 5. Error guessing etc.
White Box		1. High-level design document 2. Detailed design document 3. Control Flow graphs	1. Statement testing 2. Branch testing 3. Path testing 4. Data flow testing 5. Mutation testing 6. Loop testing

上面是一份概要：使用任一设计方法制定测试用例的各种可用方法。

但是，在使用任一设计方法准备测试用例前有一些因素需要考虑清楚。它们分别是：

测试相关风险。

预期缺陷类型。

测试员的知识和经验。

测试水平和必须进行分组和管理的小组活动。

用于执行测试用例的工具。

应用程序，软件和问题域的类型。

客户要求，等等。

现在除了上述因素，以下几个要点和/或问题在选择正确的测试用例设计技术中发挥了至关重要的作用：

Sr. No	Questions	Suitable Technique
1	If the testing process need to go along with the V-Model or when it is a test-driven design?	Black box
2	If there are some unclear requirement specifications?	Black box
3	If the application/SUT is large or encompasses several inter-connected applications/sub-systems to be tested in parallel?	Black box
4	To achieve cost-effective testing?	Black box
5	When low-level testing need to be done or system's internal working/behavior need to be assessed ?	White box
6	When complete test coverage has to be achieved?	White box
7	Likewise, if is it necessary to achieve complete test efficiency and effectiveness?	White box
8	To optimize testing effort by way of designing test cases and executing them with limited resources?	White box
9	Availability of supporting tools to formulate and/or execute test cases readily?	White box
10	To achieve overall understanding about the application/SUT during the testing phase alone?	White box

基于“经验”的测试用例设计

在基于经验的技术中，是人们的知识，技能和专业知识（关于域，技术等）构成了测试条件和测试用例的基础，且对制定测试条件和测试用例很重要。

在这儿，人们技术和业务两方面的经验都是绝对必需的，必要的，因为这给测试分析和设计过程提供了不同的角度。

重要的是，有了他们使用类似系统工作的丰富（前）的经验，他们或许对什么会出错，什么有助于测试有了想法和/或深入的理解。

因此，基于经验的技术与基于规范既与基于结构的技术偕行，又可用于没有规格，或者规格不足或过时的时候。

这可能是用于设计测试低风险系统的测试用例的唯一技术，但是这种方法可能在非常紧急的情况下特别有用，事实上，这是导致探索性测试的一个因素。

“随机”方式—考虑了吗？

通常，任何软件模块或系统都有输入域，从这个域里选择并使用测试输入数据建和/或执行测试用例。

现在，如果一个测试人员从必要输入域中随机选择输入，准备测试用例，并用它们来测试应用程序，这种方法被称为“随机测试”。

例如，如果一个模块的有效输入域是 1 到 100 之间所有的正整数，然后用这种方法测试人员会随机或胡乱地从该领域内选择值，如，选 15 ， 27 和 33。

但是，使用这种方法，也有一些一直无解的问题：

值（上面的例子中三个值）足以表明，执行测试用或运行例测试时，模块符合其规格吗？

是否有其他输入值，比那些（在本例中）被选中的值，更能找缺陷？

抑或有效输入域外的任何值应该作为执行测试用例的测试输入？

这就是说，测试数据应包括大于 100 的浮点值，负值或整数值？

因此，上述问题可以立即通过更加结构化的黑盒测试设计方法解决，尽管使用随机测试输入可以节省一些时间和精力，其他测试输入选择方法要求。

但是，根据许多测试专家，随机选择测试输入会产生一个有效的用于执行测试用例的测试数据集的机会非常小，并且对于一个更结构化的方法，随机方法生成测试输入的相对有效性总成为自省和/或研究的课题。

测试用例必不可少的部分—概念化

首先，设计一个测试用例用来回答这个问题：“我要测试什么？ ” 。因此，对于测试人员来说，

开发测试用例时周到地考虑很重要，这能够明确界定和/或提供需被验证以确保系统如期运行且能反映出它是用最高质量创建的信心的项目（模块，应用程序，子系统，或 SUT ）的完整概述。

现在，无论开发测试用例时用了什么设计技术/战略，测试人员都必须确保基本涵盖以下主要内容：

摘要 -应该反映实际的主题，类别和功能特性，使测试人员可以轻易地组织测试用例成逻辑组，并相应地对它们进行分类。

这部分可能具有关于基于测试时间，工作单元，和优先级等的执行工作的细节。它经常被称为测试用例的权重。

测试用例设计 - 这部分反映了测试用例的整体设计，其中可能包括一些高层次的描述。

正式审查 - 包含了关于必须审查或批准测试用例、并定义审批流程的团队清单的详情。

这部分主要是用来建立一个正式的审查程序，以确保业务流程符合标准。

此外，它可能包括关于测试用例所有人，工作项目，通知和成果总结等的细节

要求 - 本部分旨在：当要求被添加到测试计划中时，联系要求与一个特定的测试用例。

因此，一旦需求和测试用例间的联系被建立，测试人员就可以继续创建覆盖报告来了解和确定被测试用例覆盖的要求的比例有多大。重要的是，通过保持这种关联，有助于设置和检查整个项目的可追溯性。

先决条件 - 描述了形成前提的或必须在测试人员可以真正开始运行/执行测试用例之前发生的事物。

后置条件 - 不像先决条件，后置条件说明了需在测试用例运行/执行完成之后发生的事物。通常是产生适当的确认，如发送电子邮件通知等。

预期结果 - 本部分详细介绍了必须在测试员认为测试运行已取得成功前获得的结果列表。它可能包含了结果代码的文件或图像。

测试脚本 - 本部分概述了与特定的测试用例相关的测试脚本。通常，测试脚本有几种类型，包括手动测试脚本，关键字启用测试脚本，及其中每个测试脚本都包含用来实现一个测试用例的指示的自动化功能测试脚本。

在执行过程中，不像使用工具自动运行的自动化测试脚本，手工测试脚本是用语句处理语句。

测试执行记录 - 通常测试执行记录包含测试用例的详细信息，及从测试用例执行产生的高层次结果的细节。

重要的是，它们提供测试执行所需的相关硬件和软件环境的细节。例如，如果当运行在两个不同的操作系统和两个不同的硬件平台上，且使用了不同的浏览器的测试用例通过了，那么测试员可以为这些组合中的每一个创建测试执行记录。

测试执行记录还包含与该测试用例运行，测试运行的详细记录，以及所有执行结果的详细历史相关的的整体结果。

附件 – 本部分通常包含了支持测试用例的所有文档和文件。

风险评估表 - 本部分意在列出与某个特定的测试用例相关的风险。

所以，当上述所有部分都与测试用例相关，且如果这样的测试例被执行，那么就是一个好的迹象：关于实现完整的测试覆盖率，效率等方面的标准已达到。

使用元数据设计测试用例

正是因为业务需求推动应用软件的创建，所以应用程序的设计必须万无一失且通过质量保证认证。质量保证的一个重要方面是：设计出能确保所有设计场景已在测试中被抓取的测试用例。测试用例是一组条件或变量，在其中，测试员将决定被测系统是否满足设计的要求和功能。开发测试用例的过程也有助于发现应用程序的要求或设计中的问题。一个测试用例与一些元素指示（如测试集 ID ，测试用例 ID，测试总结和测试描述）有关。

测试用例设计有两个主要任务：

▪测试设计是所有逻辑测试用例的注意要求的草案。如果有效地设计，这就是一个能在测试执行时

节省相当大精力及成本的关键部分。

▪规格包含被转化为将要进行的物理测试指令的完整描述的草稿。

我们使用一个基于元数据的方法来设计测试用例。这种方法对于将要跨多个应用程序进行统一测试时以可重复的方式设计测试用例来说是很有用的。示例场景是涉及数据迁移或企业数据屏蔽的项目。基于元数据的测试用例设计和通用测试用例设计的主要区别是：前者没有在从需求去推导测试用例上花时间，因为通过元数据直接使用数据或前期数据的数据或属性是有可能的。

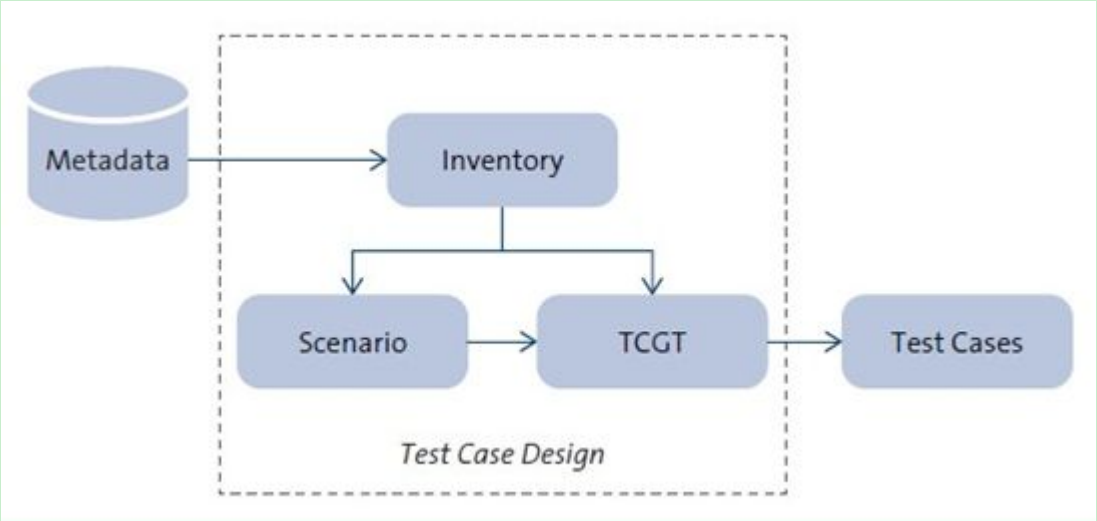


图 1．使用测试用例生成工具设计测试用例

用基于元数据的方法，我们可以着手处理库存要求；反过来，着手处理库存要求也可以获取元数据存储库中的数据属性。基于库存，就能准备高层次的场景，然后支持测试用例的开发。为了加快测试用例的准备过程，我们设计了可以用任意基本脚本语言（如 VB 脚本，UNIX 或 Perl）实现的方法，以可重复的方式高效地生成测试用例。

测试用例生成工具（ TCGT ）是一个基于在矩阵上的信息的基础上生成测试用例的高度自动化工具。它生成的测试用例可以满足验收，确认，应用核实的目的。基于元数据的测试用例设计可以用于以下两种场景，在这两种场景中要求了基于工厂的测试用例设计和生成。

场景 1：数据迁移

数据迁移项目需要大量的数据库测试，以确保没有数据泄漏，且迁移后数据的完整性和质量得以保留。迁移过程是由一组作为映射规则和转换功能的规格决定的。例如，如果我们正在测试一个系统，把数据从 SQL Server 2005 迁移到 SQL Server 2008 中，我们就需要执行以下操作：

▪数据迁移的需求分析

▪规范化要求

▪元数据验证

▪数据验证

场景 2：数据屏蔽

基于元数据的测试用例的设计也可以在企业数据屏蔽中实现。数据屏蔽测试需要比较数据正确性和完整性的源头数据和目标数据。没有屏蔽或屏蔽后复制的表格应该测试其数据变化，屏蔽算法和业务规则。在大多数情况下，数据屏蔽场景需要可重复准备和执行的测试用例，这样测试用例设计中就可以使用元数据方法了。

测试我们的移动星球_移动自动化测试

移动设备已经改变了我们的世界。自 2010 年第四季度以来，智能手机和平板电脑销售已超过了个人电脑的销售。到 2012 年年底，全球有超过十亿智能手机用户[1]。根据 Canalys 公司的调研数据，2013

年第一季度 [2]全球前四大应用商店——Apple’s App Store， Google Play， the Windows Phone Store，and Blackberry World 的 app 下载总量已超 134 亿，总收入已达 22 亿美元。这些急剧的变化意味着软件测试工程师必须迅速适应现在的移动业的现状，这些都使得移动应用程序自动化测试重要到必不可少。

移动测试的挑战

在看移动测试的自动化工具前，你需要对与移动应用程序面临的挑战稍作了解。

设备

移动应用程序应该在你要求的设备上工作。

移动应用程序必须在每个设备上都正常工作。

移动设备必须为应用程序的运行时间进行测试。

移动设备处理能力不同，内存有限，还必须考虑通信协议。

应用

新的操作系统版本和功能意味着开发人员建立了必须被测试的新的，更复杂的程序。

多个构建常常时间很短，因此脚本执行往往不能完成。

网络

多种网络类型，必须进行测试，如 GSM，CDMA，GPRS 和 Wi-Fi。

不同的连接速度（包括 2G，3G 和 4G LTE）必须跨地点测试。

世界各地有超过 400 多的移动网络运营商，测试必须处理各种网络。

技术

必须考虑大量的测试用例。

必须处理手机特有的功能，包括触屏约定。

必须执行 API 级别的测试。

移动应用程序类型

当制定移动测试策略时，你必须清楚了解可能需要测试的应用程序。移动应用程序可分为本地应用程序（Native App）， 网络应用程序（Web App）和混合应用程序（Hybrid App）。

Native 应用程序

Native App 是专为移动操作系统所建，并直接安装到该设备上。

用户通常通过网上商店或市场(如 App Store)获取这些应用程序。

Native App 是用本地编程语言构建的。例如：iPhone 或 iPad apps 是用 ObjectiveC 构建的，Android apps 是用 Java 构建的。Native App 速度快，提供更好的用户体验和界面，并且通常可以获取目标设备的所有功能。

Native Apps 的功能包括：

存储。二进制“可执行映像”，被明确下载并存储到移动设备的文件系统中。安装过程可以由用户，或者在某些情况下，由企业的 IT 部门开启。

分布。获得 Native Apps 最常见的方法是去有相关特定设备的应用程序商店或市场（iTunes 有 iPhone 或 iPad 的 apps，安卓市场有 Android Apps），或者通过企业分配机制获取。

操作。程序直接在操作系统中运行：

由主屏幕开启。

不需要另一个存储器应用程序来运行它。

明确利用操作系统 APIs。

移动 Web 应用程序

移动 Web 应用程序是专门针对移动设备的网络驱动应用程序。

这些应用程序是通过移动设备的网页浏览器获取的（例如：iPhone 上的 Safari）。用户不需要在设备上直接下载和安装该应用程序。

	利	弊
手机模拟器	费用：手机模拟器是作为每个新的操作系统发布的软件开发工具包的一部分来免费提供的。 简单：模拟器下载和安装简单，即刻使用。许多模拟器能够以简单和直接的方式来运行。 快速：比起要连接到本地网络或云的真实设备，模拟器的等待时间更短。	硬件支持：仿真模拟里，完全的硬件支持是无法测试的。 计算资源：根据PC运行模拟器的处理能力和被用来测试的手机和智能机的类型，模拟器上的表现比起真实设备，可能是不切实际的好或坏。 网络互用性:使用模拟器不可能测试网络相关事件的影响（如：来电，短信），不同的关于移动应用程序行为的网络技术（如：HSPDA，WCDMA，UMTS和LTE）. 由于模拟器并没有连接到移动网络，它们不支持互操作性测试。
真实设备	可信赖的：在真实设备上进行的测试给出的是最精确的结果。 网络的互操作性：真实设备测试是在真实的网络中进行的。用户体验:通过使用真实设备，可以把特定设备的CPU，内存或屏幕大小等元素考虑在内，准确地看到用户体验。	物流和成本：购买不同版本操作系统的不同设备是相当昂贵的，还要浪费不少精力来购买和管理这些设备。

表 1：使用模拟器和真实设备的利弊

移动 Web 应用程序功能：

完全使用 Web 技术，如 HTML（尤其是 HTML5），CSS，Javascript 代码写的。

该代码是由浏览器执行，而不是由操作系统。

用户可以通过多种方式启动应用程序：输入网址，单击超链接，扫描 QR 码，或者单击主屏幕上的快捷方式。

安装是非强制性的。

支持多种操作系统。

混合应用程序

类似本地应用，混合应用程序是使用传统的 Web 技术开发的。

Hybrid applications 是在每台设备上的本地应用程序存储器中运行的，但却是集中部署和维护，是跨平台的性质的。通常情况下，他们是由云服务，所以地球上任何地方的终端用户体验是一致的，跨设备的。

使用模拟器和实际设备进行自动化测试

模拟器是用来复制一个移动设备的内部工作的。它是用于开发和测试移动应用程序的强大工具，被用于手动和自动化测试中。

当然，移动 APP 是用在真实设备，而不是模拟器上的，所以测试必须在实际设备上进行，以确保应用质量的最高水平。

然而，让你们组织里每个移动测试团队都拥有一个实际设备是很烧钱的，所以使用模拟器是一个可以控制成本的有效方法。

在制定移动测试策略时，你们组织应该谨慎拳皇使用模拟器或实际设备的利弊。

移动设备自动化测试工具分类

有三种类型的工具可以支持移动设备的自动化测试。

本地平台工具

本地平台工具通常是由移动平台供应商提供的软件开发工具包的一部分。这些框架通常与用户界面对象级别的应用程序进行交互。

这些工具允许更复杂的基于对象的交互，十分成熟，还支持本地 UI 对象，因为它们是平台供应商支持的。

因为这些是操作系统级别的应用程序对象，你可以通过用测试中的应用程序编译的小数据库（也被称为 “instrumentation”）洞察他们。

基于视觉的多平台工具

基于视觉的多平台工具最常用在移动设备自动化测试里。

这些工具通过可视化手段与设备交互，并可以识别文本或图像，使测试人员构建基于这些认识和内置的手势的自动脚本。

基于视觉的多平台工具的优点是它们支持多个平台，并且可跨多个设备执行测试。

视觉对象由 OCR 引擎（基本上都是将扫描的手写、机打图像，或印刷文本转换成机器编码的文本智能软件引擎）识别。

基于对象的多平台工具

基于对象的多平台工具可以在应用程序内通过识别，拦截，并发送信息到对象，直接用和传统的测试工具一样的方式来与应用程序 UI 对象进行交互。

这些工具的优势是，他们支持多种平台，并且可以跨平台上执行测试。

对象级整合也对应用程序变化更加宽容，从而降低与自动化测试相关的整体维护成本。

移动自动化测试的方法

在规划您的移动自动化工作时，别忘了以下的工具评估和选择，对象技术的方法：

工具评估和选择

执行工具的可行性，以检查是否该工具可以在各种移动技术和平台使用。

选择一个同时支持真机和模拟器或仿真器的工具。

识别多种设备和版本支持。

用实用性和可重复使用功能增加自动化测试工作的价值。

了解如果选择的工具需要你破解或获取设备的根。

确保该工具支持操作系统的新版本。

对象识别

基于图像的对象识别：把每个测试对象记录为图像，在 GUI 中匹配对象和可用运行时间图像。

光学字符识别对象的对象识别：使用光学字符识别（OCR）功能获取屏幕上的控件的文本。该功能使用了通过字符读取字符文本的专门算法。

真实对象或本地的对象标识：标识唯一对象的属性，如“ ID”，“名称”和“类” 。

基于 DOM 的识别：利用 DOM 属性来识别 web 应用程序对象。

特点	图像识别	OCR对象	本地对象	DOM对象
对象识别的复杂度	容易	容易	中等	中等
对象维护工作	高	高	容易	中等
跨设备支持	高	中等	容易	容易
执行中的识别速度	中等	中等	中等	高

结论

通过在移动应用程序测试中使用自动化测试，测试团队可以在保持质量和减少将产品推向市场时间的同时降低成本。

许多工具可用来支持移动设备自动化测试。选择正确的工具需要理解业务需求和移动测试独有的因素。

权衡手机模拟器和真实设备的优劣，企业的最佳移动测试解决方案往往不是只选择其中一个，而是选择结合这两者。

更好的测试用例设计

很多测试用例设计很少或根本不费什么力（只需搜索我们社交媒体上的所有建议），但设计好的测试用例是复杂且极具挑战性的。为设计好的测试提供资源这一点往往很容易被对编写和执行测试不负责的人所忽视。更不要说为获得足够的使我们能够衡量我们测试工作成效的报告需要了解哪些信息了。然后还需要有可以定义（或有时甚至强制执行）所需测试的组织，开发团队，测试团队，流程及工具。搜索讨论板时，很多人在上面寻找通用解决办法的并不意外。

互联网上和书中有足够多的信息介绍：如何分析需求，应用测试设计技术，并将它们实施和执行。我建议去看看 Cem Kaner 写于 2003 年的题为《什么是好的测试用例？》的一份简短却详细的研究报告。它引用一大堆参考文献，提供了充足的信息让你能基本了解。

在这篇简短的文章中，我不会解释如何应用测试技术，而会分享信息作为需要考虑的因素的入门指南，使我们能够设计出好的测试，我希望我能为读者创造一个分享他们经验的平台。这里的内容是基于我的解决呈现在我面前的挑战的测试经验。我也是假设读者有一些正式的测试经验且已对测试设计技术有了理解才写的这篇文章。

在阅读这篇文章时，我不断查阅测试设计的高层次目标和因素：

1. 合并测试分析中定义的执行状态。
2. 建立特定输入。
3. 根据输入和规范定义预期结果。

和一些基本因素：

1. 覆盖范围。
2. 有效性。
3. 弹性和维护。
4. 执行者的技能。

“There’s a fine line between wrong and visionary. Unfortunately, you have to be a visionary to see it.”
– Sheldon Cooper, Big Bang Theory episode The Pirate Solutions

我们遇到的影响设计出好测试的第一个挑战是使每个人都了解测试原则。

ISEB / ISTQB

软件测试基础突出七项原则。这些原则提醒我们并降低（一些）成员对有时会超出我们控制的各种挑战的期望。解释并使参与的每个人都了解到这些原则将导致现实的结果和一个更愉快的合作环境。

1 .测试将显示出缺陷的存在。开始设计更多将显示缺陷的测试。 “幸福路径”应该会起作用，当然，除非你质疑交付测试的代码质量。

2 . 许多情况下都不可能测试一切。建议使用风险分析并优先关注测试工作的重点。接受一个事实：没有一个各种组合和排列的测试会降低压力水平和期望。有时候会错过一些事。如果你想要 20 个测试，最好设计 100 个测试，再从中挑选出最好的 20 个 。

3. 开发生命周期中尽可能早地开始测试活动。别以为测试只是一旦产品可执行时用来确认和验证产品的功能的。相比后来在 SDLC 这样做，在验收前正在审核要求时设计测试，比起在 SDLC 中晚点这么做，可以获得很高的投资回报率。

4.缺陷很集中。测试执行后不要停止设计新测试。如果时间够的话，审查被排除在（与在其中最新发现的缺陷被确定的组成部分相关的）最初范围外的测试集合中的一些测试。

5.在某些时候，系统会对被多次反复的测试“免疫”。再次强调，在设计好的测试停止寻找缺陷后不要停止设计新测试。审查被排除在最初范围外的测试集合，或者利用更多测试技巧拓宽缺陷类型。

6.你如何设计测试受到正在开发的系统的环境和业务领域的形式影响。不是每个人都在建桥梁。此外，根据测试水平去设计测试。

7.如果你曾经用代码设计了很多测试，后来实现了代码并不能解决用户的需求和期望，那一刻就像在经历一次史诗级的失败。如果规范欠缺，就根据用户期望系统如何工作以及它将如何满足他们的需要去考虑设计测试。不要依赖代码作为规范的来源。

> *“If my calculations are correct, when this baby hits eighty-eight miles per hour, you’re going to see some serious s***.”*
> – Dr. Emmett Brown, Back to the Future

第二个挑战是确定测试什么，为何要测试，以及如何测试。

在团队中或和不明白测试目的和目标的人一起工作时，这就变得非常具有挑战性的（有时候争权夺利的）。开发生命周期及利益相关者和团队的成熟度或许会成为在设计好测试时有创意的主要障碍。确定测试的内容就要在分析需求时提出正确问题。根据我的经验，将这些问题基于 ISO / IEC 25010:20113 标准文档中所提到作为一个起点的软件质量特点是正确方向上迈出的一步。我觉得把这些特性包括问题中是一种非常有效的收集信息并阐明规格（尤其是与敏捷团队合作时），因为它提供了一个机会来定义超出原所有者认为的需求细节。他们也许之前没有专家帮助确定可测试的细节。试着预测使用这些高水平质量特点时可能出现的客户需求和经验方面的问题：

1. 功能
2. 可靠性
3. 可用性
4. 效率
5. 可维护性
6. 便携性

这些高水平特点有子特点，如：精度，性能，可变性，可安装性等等。现在，你应该能够根据计划，预算和可用资源去选择最合适的测试技术了。

我认为软件开发中不常被提到的一个特点是可测试性。如果你曾经设计过电子电路和组件，那么你就会明白 DFT （为可测试性而设计）的惊人效益，或当工程师把 BIST （内建自测试）包含到他们的设计能力中时。许多软件产品是以一种非常耗时的，容易出错的且很难测试方式设计的。

理论上，DFT 将在设计阶段从测试专家考虑输入，以便早早地设计出更好地测试。可测试性的一个简单的例子是，如果配置设置被存储在数据库中，但它仅在系统启动时加载。在某些环境中执行重启可

能挺麻烦。可测试性将通过轻击开关或点击按钮来实现配置设置的动态重载。通过询问“我们如何测试，间隔多久需要回归一次呢？”开始测试性的讨论。团队成员更频繁地考虑简单性和自动化是非常令人兴奋的。研究为一般软件设计问题提供指导及技巧以帮助发现问题的测试模式是有益处的。“我们长期辛苦去打造一个天堂，结果却发现它填充了恐怖。 ”——选自 Alan Moore 的《守望者》

第三，测试内容需是明确且“可测量的”。

这可能并不占设计测试多大部分，测试过程中需被测量的内容及测试过程的测试控制活动有望被定义。测试就像团队的照明灯。测试活动应该指导团队使他们能够避开障碍。奇怪的是，这是最后一件需要考虑的事。所以，我只是想把它作为一个提醒。在某些时候，你会想提供反馈，例如：

1. 测试工作。
2. 改进测试重点。
3. 向别人证明一些事。
4. 提高覆盖率和技术。

看一个测试用例需要什么信息的良好的开端就是读取 IEEE 8292 标准文件。从这里你可以排除那些从长远看来无关紧要的项目，或根据需要添加更多项目。确定需要什么样的报告，然后寻找能够尽快产生这些报告的最佳工具。不幸的是，测试往往是非常依赖工具来处理大量信息，但工具也可能对测试形式有影响，并需要流程去确保信息准确。（你是否曾经被要求“调整”数据，掩盖报告中“无效”数据，或收到过了一份其结果不稳定到甚至斯蒂芬•霍金都站起来含泪走出房间的报告？）记得用和审查代码一样的方式经常检查测试。这样就能持续改进。我希望这篇短短的文章为提供了一些理论或实践指导并引起对设计好测试用例的讨论。

你要怎么切你的披萨?

关于统一等价类划分的术语和过程：

等价类划分是很重要的软件测试设计技术之一。重要到几乎每一个测试员都要用到这项技术，他们中的一些人甚至还没有意识到被这他们称为 “常识”的实际上是一项正式的技术。

但不知何故，作为一个测试团队，我们就等价类划分的过程这一点上似乎无法达成一致意见。甚至连它的术语都意见不一。划分和类是同一回事吗？等级划分的有效和无效意味着什么？当我们了解什么是类之后，又该如何把它运用到测试用例中呢？我们需要输出划分吗？在这篇文章里，我会提出一个关于统一等价类划分的术语的建议，并努力找出一个单一的方法来得到测试用例。我汇总了许多测试专家的知识见解并且找出其中的共同点，努力做到不遭到一丝质疑地去除这些不统一。

来源：

参考咨询了多方来源，我汇集了关于等价类划分技术的信息，大多数作者都偏向 Glenford Myers [1]和 Boris Beizer [2]的观点。并不是所有的关于等价类划分技术的信息来源都描述了整个过程，也不是所有的都描述了同一个过程。最实用的关于怎样运用这项技术的信息是由 Erik van Veenendaal [6]和 Edward Kit [3]所描述的。

术语：

零星碎片

在等价类划分里，我们取输入一个电脑程序的内容，把它切成零星碎片，这本应由该程序自己以同

样的方法处理的。

不同的来源里用不同的术语来描述这些碎片：

• Myers [1]认为: “等价类划分是通过考虑了每一种输入条件... 确认的，并把它分成两至更多份以上。

• Black [9]：“等价类，也叫做等价分区。”

• Van Veenendaal [6]：“..... 等价类或者划分区.....”

• De Grood [10]：“..... 有效和无效的等价类别.....”

让我们看一看关于“划分”的专业术语的准确的起源“set theory”。我是找不到我的旧教科书来看了，但是我可以参考维基百科，它是这样写的：“集合 S 的一个划分 P 是两两不相交的非空子集，使得 ∪P = S。”，“集合 S 的任意一个分区 P 给 S 引入了一个等价关系，其中每个 A∈P 是就一个等价类。同样地，给 S 引入一个等价关系，不同的等价类的集合就是 S 的分区。”

我提议回归最初，坚持在 set theory 中使用的准确的术语。

从上面的准确的术语，我们学到了以下（下面是用简单明了英语改写了）：

• 划分是把东西切碎。

• 某物被切成零星碎片的方法就被称为划分。

• 这些零星碎片被称作类

所以当我确定了整数1到10有两种划分时就意味着我可以用两种方法把它们切碎，也就是，奇数和偶数（这是第一种划分）或者质数和非质数（这是第二种）。把小于5的数和大于等于5的数分开会生成由两个类构成的第三种划分。

有效和无效

一旦一个程序的输入范围被划分成等价类，我们就不得不在我们可以把它们与测试用例结合起来之前决出这些类里面哪些是有效的哪些是无效的。但是我们怎么定义“有效”?

• Black [9]：“... 有效类... 描述有效的情况，系统需要正常处理...”

• Van Veenendaal [6]：“等价类划分下的无效数据并不是说这个数据是错误的；而是指这个数据不在具体的划分范围之内。”

• Burnstein [7]：“…有效类…描述系统可以正常处理…的情况。”

• Van Veenendaal [6]：“无效类表示输入错误或异常”

别人告诉过我：一个类，当程序不指定输入来自该类时，它被认为是无效类。但是要是指定行为是一个错误信息，又该怎么办？D.J. de Grood [10] 说：“…并不是一个无效值，因为这次输入的错误处理已被具体…”。

我建议采纳和适应用于分类树方法中的定义。“有效类描述被测试对象有条不紊地处理的输入情况。无效类应该被测试对象引发错误处理反映。”这段引述的大部分都是和原文一样的，我加了一个词“应该”来覆盖错误处理机制（还）不到位的情况，我还用“类”替代了 Grochtmann [4]原话中的“测试用例”。

食谱：

切片和切块

现在大家都清楚了什么是划分（划分的过程），什么是类（划分结果）了，但还不明白输入域是如何被划分的。Van Veenendaal [6] 和 Kit [3]就所有输入和通常它们是如何被划分做出了详细准确的概括。但是划分是一个测试员的工作吗？划分的基础已经在测试员要求之中了。例：“6岁以下的儿童在至少一位家长的陪伴下可免费进入”。就是按儿童的年龄和陪同者的人数划分免费的部分人群。但是我们必须确保顾客即要求工程师和软件开发员都以同样的方式来理解它。

结合类

现在我们已经解释清楚了术语，我希望你们同意我做出的选择，我们来说说结合类吧。

等价类划分中所提到的来源中有许多的观点，所以再一次的，我们不得不作选择。

首先让我们来看看软件构件测试的标准 – BS 7925-2 [5]。这个标准给我们提供了自由， “在生成测试用例时，可以采取两种截然不同的方法。用第一种方法，一个测试用例在一对一的基础上生成每一个确定的划分区。用第二种方法，一组最小的测试用例生成并覆盖了所有确定的划分区。”请注意这句话中的“划分区”一词，我是提议用“类”的。

大多数来源生成了覆盖所有有效类的一组最小的测试用例。接下来看看 Myers 的方法：“写一个测试用例，这个用例只覆盖不被覆盖的无效等价类之中的一个。”直到所有无效类都被覆盖。不把多个无效输入结合到一个测试用例中的理由 Myers 清楚地说明了，Kit and Veenendaal 也这样解释到：“如果多个无效 EC 在同一个用例中被测试出来，一些测试可能无法执行，因为第一个测试可能会掩盖其他测试或者终止执行测试用例。”但是一些人（[6]， [9]）提出一个单一的，完全无效的测试用例可能是有用的。尤其是在 Web 应用程序中，输入数据在被发送到服务器前经常先进入一个表格检查其有效性。让我为您展示以下几种结合类的菜单：

选择1—低脂餐

- 开胃小吃：一组最小的覆盖所有有效类的测试用例
- 主菜：一组最小的覆盖所有无效类的测试用例
- 饭后甜点：抱歉，没有甜点

选择2—常规餐

- 开胃小吃：一组最小的覆盖所有有效类的测试用例
- 主菜：一组测试用例，其中每一个用例每次只覆盖一个单一的无效类，除非所有的无效类都被覆盖（提供）了

- 饭后甜点：抱歉，没有甜点

选择3—丰盛餐

- 开胃小吃：一组最小的覆盖所有有效类的测试用例

- 主菜：一组测试用例，其中每一个用例每次只覆盖一个单一的无效类，除非所有的无效类都被覆盖（提供）了

- 饭后甜点：一个只覆盖最好的无效类的单一测试用例

选择4—（几乎）所有你能吃的

- 开胃小吃：一组最小的覆盖所有有效类的测试用例

- 主菜：一组测试用例，其中每一个用例每次只覆盖一个单一的无效类，除非所有的无效类都被覆盖（提供）了

- 饭后甜点：一组最小的覆盖覆盖所有无效类的完全无效的测试用例

- 额外： 给极度饥饿的人，我们提供了精心挑选的覆盖所有输出类的额外的测试用例组合套餐

我们该如何选择呢？这个取决于我们对于测试之下的主题所知道什么。

在低风险的情况下，你不是很饿的话，一组最小的测试用例或许是最好的选择。像上面的网站的例子，如果我们知道该软件是用来在开始一次计算前检查在表单中的字段中所输入的每个值的话，那么明智的做法就是添加额外的完全无效的测试案例。

在高风险的情况下，我们想要确保每一个无效类被识别为无效并被视作无效来对待。所以我们至少需要一份不错的常规餐或者“所有你能吃的”。

但是额外呢？输出划分是有营养的一部分还是它只是让你发胖的部分呢？

另外，它还取决于实际情况和测试目标，当你在测试一个将输入转化为输出的模块（这经常发生），接着，该输出被用作另一个模块的输入时，知道是否所有的可能输出会因为正确的理由被接受或拒绝可能是相当有价值的。

事实上，我们在这儿讲的是集成测试。为了执行一个这样的测试，必须生成接受模块的所有的可能输入类。因此，我们需要全面了解调用模块的输出。调用模块的范围必须符合被调用模块的域。正如 Beizer 所说：“…调用者的范围就是调用者对被调例程域的概念…”我们想证明这个观念是正确的。

祝你们有好胃口！

我猜我有必要提前道个歉，我从未想过让你们下一次吃披萨的时候，再也不想去随意切它而是想把它划分成4类，每次都从每一个单一的类上咬一口，等价地品味。

电话：021-61079698

Email：sales@spasvo.com

QQ：1404189128

产品租用		
下载	在线申请	详细

AutoRunner 是一款自动化测试工具。AutoRunner 可以用来执行重复的手工测试。主要用于：功能测试、回归测试的自动化。它采用数据驱动和参数化的理念，通过录制用户对被测系统的操作，生成自动化脚本，然后让计算机执行自动化脚本，达到提高测试效率，降低人工测试成本。

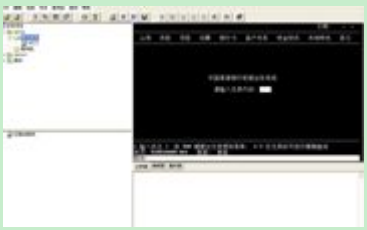
在线体验		产品租用	
企业版	免费版	在线申请	详情

TestCenter 是一款功能强大的测试管理工具，它实现了：测试需求管理、测试用例管理、测试业务组件管理、测试计划管理、测试执行、测试结果日志察看、测试结果分析、缺陷管理，并且支持测试需求和测试用例之间的关联关系，可以通过测试需求索引测试用例。

其他测试工具

Precise Project Management

Terminal AutoRunner



PerformanceRunner

有关培训、产品购买及试用授权方法等事宜

电话：021-61079698

Email： sales@spasvo.com

QQ：1404189128

SPASVO