探索式测试白皮书

@淘宝技术质量部-支撑平台测试部-共享中心测试组 季哥

目录

序

夜深人静的时候，有两个测试工程师在积极的探索测试的真谛。

Tester A: 哎，我们测试工作处在整个项目周期的下游，很多人认为我们为项目所做的贡献总是小于开发的，似乎有种被人遗忘的感觉呀。有一句是这么形容我们测试的工作：成功没我份，失败全我错。真郁闷啊!!

Tester B：不会吧，我前几天发现了一个非常严重的 bug，几乎把整个系统都搞崩溃了，PM 和开发们都很感谢我，非常认可我们测试的工作啊，开发老大还写表扬信说我们测试发现 bug 的手段非常新颖且有技术含量。

Tester A: 哦哦，这么厉害啊，估计那也是偶然吧，大部分的时候我们做手工测试的都感觉没啥技术含量，搞得像随便找几个人点点鼠标都会一样的。

Tester B：这也不能这么想的吧，你看那些做自动化测试也没多少自动化脚本能发现问题呢，大部分 bug 还是我们手工测试发现的呀，白盒测试其实也是同一个道理的，很多问题还是遗留到上层被手工测试点鼠标发现的啊。

Tester A：说的也对哦，但我觉得做手工功能测试比较单调且枯燥无味，而且多次重复进行测试，有没有快速进行测试的方法呢？

Tester B: 其实只要你用心去发现，手工功能测试也有她美的地方，也有她富有激情和创造力的地方，其实有个探索式测试也许能改变你现在的想法，而且也能让你能快速的进行测试，而不需走繁杂的流程，而且质量也不错哦。

Tester A：探索式测试，真的有这么神奇吗？好像很多人觉得这个不可靠啊。

Tester B：呀呀，这就像小马过河一样，你不去尝试你就怎么知道这个不可靠呢，何况未必别人就真的正确理解了探索式测试，其实我倒觉得她可以让你测试的价值得到大多数人的认可，也会让你真正的感受到测试像一门艺术，让你疯狂的爱上她，理解她，而不是抱怨它。

Tester A：呀呀，测试真的有这么大的魅力存在吗，我都不敢相信。

Tester B：还骗你不成，你如果相信我的话，跟我来，让我们开始测试的探索之旅吧，这里有业界测试大师们的集体智慧，你可要好好琢磨哦。

Tester A：一定，一定，很想知道啊，那我们开始吧!!

Date：2010/11/06

引入篇

1、测试手段的多样化

我们测试人员都知道整个 Test Cycle 的样子，也知道功能测试是其中很重要的一环。其采用的方法也是很通用的，根据需求写测试用例，站在功能是否被实现或被完美实现的角度去写测试用例，然后按照测试用例来执行所写的测试用例，发现了一定量的 bug。似乎很合理，似乎看来无懈可击。但平静的湖面下面是否存在那不为人所知的怪兽呢？

第一次听到测试手段的概念，无法理解，觉得测试手段和测试类型几乎差不多，估计是在炒概念，最近很流行。但了解了 James Bach 的思想后，感觉自己错了，测试手段使测试更加富有，更加活跃，更加专业；也使测试人员更能找到自我，更加富有创造性。

我们最熟悉的就是功能测试了，显然功能测试相对于性能测试，接口测试，安全测试，就是一个特别典型的测试类型，我们会对测试类型进行不同的测试策略。但往往越是我们最熟悉的东西就越存在非常隐患，这里我们从测试手段来考虑，功能测试只是一个测试手段而已，属于功能测试（测试类型）。但我们实际测试过程中，还可以把功能测试手段和兼容性测试类型结合起来，甚至还包括场景测试，这样做好吗？

测试手段关注于多个方面：**测试员，覆盖率，潜在问题，测试活动，评估。**

所以我们的测试都包含上面说到的五个要素，而测试手段就是将测试员的关注点集中在一个或几个要素上，把其他要素留给测试员自己判断。而且我们期望好的测试手段就是可以跨多个要素的更综合的测试手段。

那么我们所做的功能测试其实就是关注测试内容的基于覆盖率的测试手段，逐个测试每个功能，彻底测试每个功能，直到可以确信该功能没有问题。这里面包括白盒功能测试（单元测试）和黑盒功能测试。

另外还有其他的关注测试内容的基于覆盖率的测试手段：

(1) 特性与功能集成测试：一起测试多个功能，以 check 功能在一起执行的情况

(2) 菜单浏览：遍历 GUI 产品中的所有菜单和对话框，使用每个可以的选项

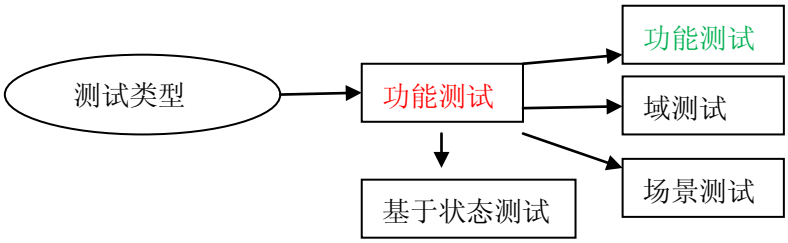
(3) 域测试: 使用等价类和边界值方法进行变量输入测试

(4) 等价类分析: 测试等价的一组变量的取值测试

还有很多这里就不说明了，说一个共同点：就是其实我们的其他很多测试手段都是在广义上的功能测试剥离出来的，也就是说，我们淘宝现在做的功能测试其实都或多或少的包括这些测试手段，只是做到的程度就不一样而已，也就是说这些测试手段发现的 bug 都被统称为功能测试这个测试类型发现了。

我们测试执行的时候：考虑说要站在用户使用的角度，要站在功能设计是否合理的角度，要站在破坏者的角度，要站在功能是否正确的角度，要站在市场的角度等等。不同的角度去测试，就会发现不同的 bug。我们实际做功能测试的时候，会全面考虑这些角度，但我们的比重是非常清楚的（不合理且很模糊），也就是我们更多的关注这个功能是否正确，是否符合需求。其最常用的手段就是上面说的彻底的测试每个功能，也就是功能测试。

下面的图可更易理解（结合了功能测试类型和功能测试手段）



红色的功能测试是从测试类型角度来分开的，测试类型是从产品或系统的质量特性角度来分开的。绿色的功能测试时从功能测试类型角度分开的一个关注与功能实现的测试手段。和绿色的功能测试同级的其他测试活动都是功能测试类型的测试手段。

那么如果我们看其他的手段，可以发现我们实际在做的功能测试都包含这些测试手段，但一个人的精力和时间投入是有限的，你把更多的权重放在这里，其他的地方的权重会相对减小。我们为啥会这样呢？ 首先我们没有深入的分析功能测试发现的 bug 和使用不同的手段去进行功能测试带来的好处。加上同一个角度的测试执行带来的浮躁和系统免疫现象，我们功能测试的手段的单一性带来的结果是值得怀疑的。我们使用不同的手段去进行类似于功能测试的测试执行，会发现很多 bug，这些 bug 表面上看像功能测试应该发现的 bug，像用户测试应该发现的 bug。一般情况下什么样的测试手段决定发现什么样的 bug。当然不同的手段之间也会存在交集的，也就是说使用探索式测试手段去测试，站在的角度也许会存在变化（在测试执行中），所以其发现的 bug 会很有可能是其他测试手段应该发现的 bug。

上面说的比较模糊，那么举个例子来说明下，比如就打印机而言，打印是其最主要的一个功能，那么功能测试就是测试这个打印机是否能正确打印出想要的文件。这里先说明如果是采用多样化的测试手段会发现哪些问题吧，其对应关系是：

(1) 正常情况下，打印机未打印出想要的文件——功能测试发现的问题；

(2) 打印机打印出了想要的文件，但文件上存在格式问题，不符合用户习惯的显示问题——用户测试发现的问题；

(3) 打印机要连续打印 100 张或打印 0 张时出现问题——域测试发现的问题；

(4) 打印机要打印其他操作系统存储的文件格式时出现问题——兼容性测试发现的问题；

(5) 打印机在连续打印 100 张时中途取消打印后出现问题——场景测试发现的问题

(6) 打印机在连续打印 100 张过程中，同步打印其他文件出现问题——基于状态的测试发现的问题

其实上面说到的那些问题，在我们目前的工作过程中，大部分都是在我们所谓的功能测试中可以发现的，在我们放大功能测试的范围后，要想覆盖更多的就需要在功能层面理解更深，更透彻，才能想到如上的测试用例，从而发现更多的问题。那么假设我们将功能测试分解成多个真正的测试手段，那么每个测试手段都可以发现其应该发现的问题。这样我们在写某个测试手段的测试用例的时候就是有针对性的写用例，去观察系统行为，从而更大的程度上去保证覆盖率, 保证产品质量。当然这里面是从功能测试类型去分解我们应关注的测试手段，其实我们对于其他测试类型也可以采用分解法，再与功能测试类型结合共同保证质量。

总之，国外已经有结果证明测试手段的多样性带来的成果是可观的。当然，成本也是需要考虑的。我们需要考虑存在哪些测试手段，需要考虑哪些测试手段适合在哪里实施，需要考虑每个测试手段对测试人员的能力的要求如何，有哪些能力或经验是可以传承的，等等。当然我们最新看到的就是这些测试手段是如何应用在我们的项目测试过程中，之间的应用顺序是串行的，还是并行的，等等。后面说下测试手段的组合成为新的测试模型。

2、测试手段分析之测试模型

测试手段的定义已经大概说明了，那我们想在测试的时候，其测试手段的多样性带来很多的好处肯定比较多，为啥我们可以这么说呢？

我们可以好好分析下我们现在所做的功能测试，测试设计和测试执行时间很长，看似花的时间很长，应该是对于功能的测试比较彻底了，这里说到的功能测试也就是我们目前所做的功能测试，包含真正的功能测试和一些与功能相关的测试手段。这样我们会出现很多现象：

(1) 我们在第三轮测试的时候发现的 bug 相对第一轮来说少很多

(2) 我们无法去验证功能测试是否已经完全的彻底

(3) 我们对于可靠性和场景测试关注不够

(4) 我们太过于依赖需求文档去做测试设计和执行

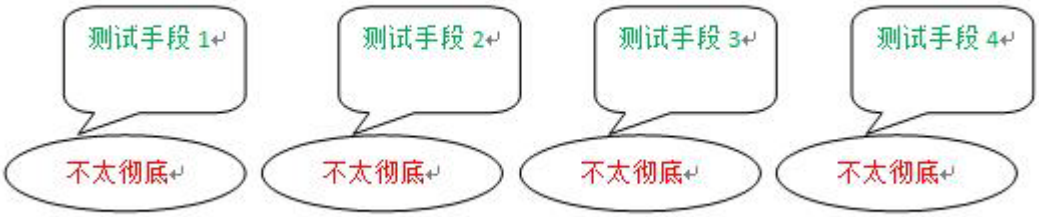
(5) 我们太过于压抑自己的创造力和探索精神（感觉测试就是简单重复的执行测试用例）

这些都可以说明，我们还有些地方做的不够深入。那我们这边有个信条：大多数情况下，使用多个不太彻底的测试手段的结果要优于单一的彻底的测试手段。

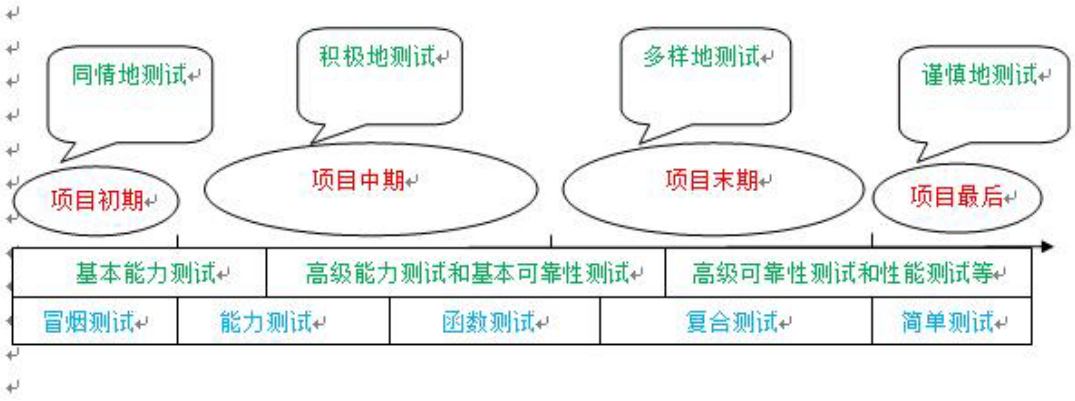
如下模型可易理解：

我们目前的测试模型：

而如果我们采用多样的测试手段则是如下的测试模型：



现在我们来分析下我们在做测试执行时的几个很关键的步骤：



先看下上面的模型图： 横轴是测试执行时间的走向，可以看到项目刚开始的时候，我们只需要用一些很基本的测试手段，很基本的能力去测试。一般是测试主要流程和重要功能。同样我们还可以看到如果使用一些高级测试手段去测试看似很稳定的系统时，会有一些意想不到的效果（更多 bug 的涌现）。就是说我们在系统越看似稳定的时候，越需要采用更高级的测试手段去测试，设计更复杂的场景去测试。这样做事基于如下理由：

(1) 系统软件的免疫性越来越强

(2) 遗漏的 bug 大部分是场景比较特殊的异常情况

(3) 发现高质量的 bug 更能体现我们测试人员的价值

(4) 找出目前功能测试的盲点

下面是从多个角度来说明测试手段多样性的必要性：

多样性地测试：项目末期采用更多其他的测试手段去探索已经熟知的产品

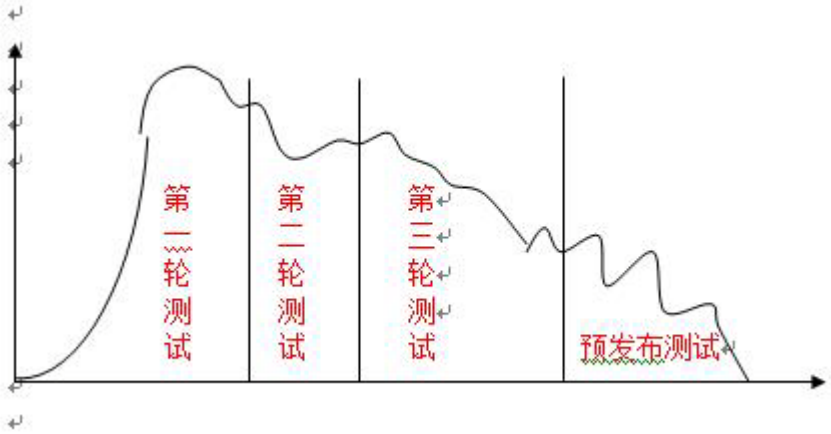
高级测试：由于系统免疫的问题，能发现的 bug 的手段往往是属于高级测试手段

项目周期：项目末期，越看似稳定，越需要高级的测试

复合测试：特性之间和测试手段的完美复合，让测试更具有特殊性和复杂性

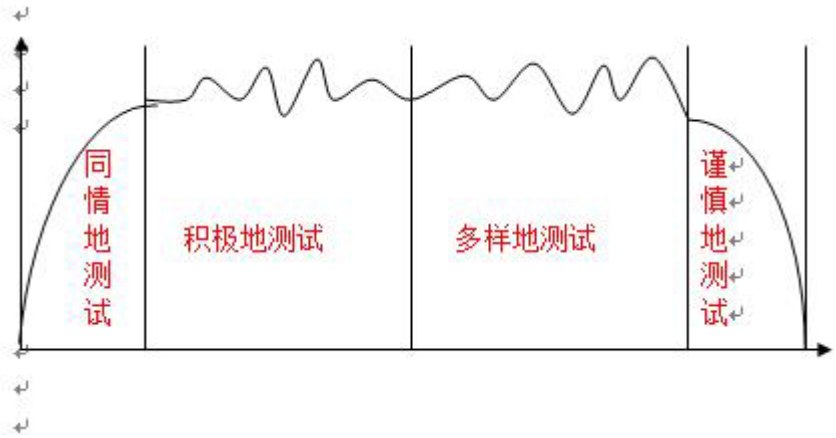
另外我们从 bug 缺陷图来看下如果使用测试手段多样性会带来什么样的 bug 分布图：

首先是我们目前的 bug 走势图：



这里可以看到我们第一轮过后其 bug 是几乎都是下降的，直到 **Zero Bug**。

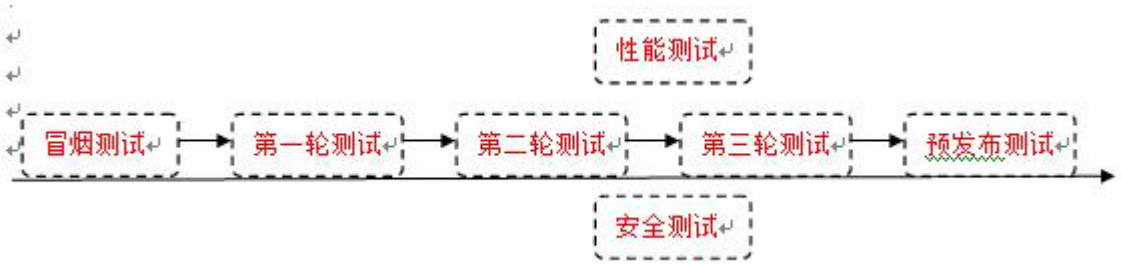
我们采用多样性测试后的 bug 走势图：



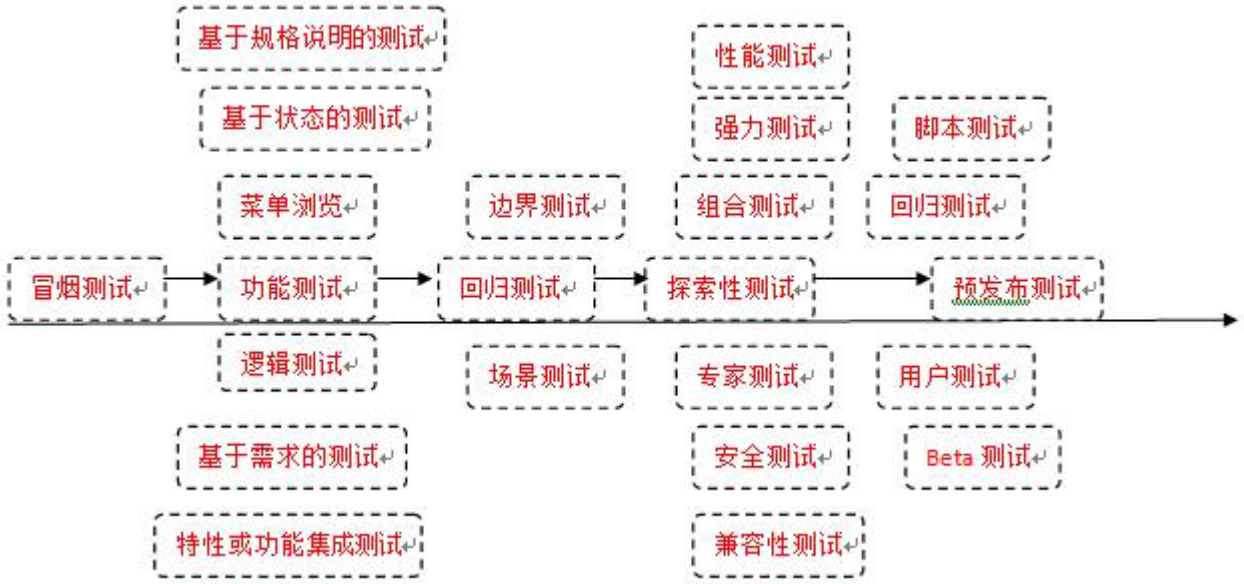
这里面可以看到无第二轮测试的概念，但 bug fix 的验证却无处不在，我们会采用回归测试这个手段，具体如何执行这个手段需要和开发沟通以及约定的策略(这里不详细说明)。我们可以明显的看到如果采用多样性的测试手段，其 bug 数量不会一直减少，在采用多样性测试阶段其发现的 bug 数量不会很低于积极地测试这个阶段。至少不会有较大的差别。后续是由于发布时间点的临近和系统逐渐稳定等各种因素，bug 缓慢的降低到零。

最后根据上一篇说到的那些测试手段，结合我们的项目实际情况，我们可以采用多样性的测试手段来避免更多的盲点。这样我们可以对 SUT 说，你变强了，我会变的更强。那就是随着测试手段的高级性和多样性，会让 SUT 暴漏出更多的隐藏的 bug。

这里先看下我们的目前的测试执行流程（手工功能测试主导的）：



那我们再看看如果我们引用多样性的测试手段的话，会是什么样的测试执行流程呢，看如下：



可以看到，这里面的测试手段还是比较丰富的，但大家可能会存在如下疑问：

(1) 这样复杂的测试手段，是不是我们的测试执行时间会拉长？

(2) 怎样对各个关键的测试手段进行测试设计，甚至是测试执行呢？

(3) 运用这些多样性的测试手段是否真的能发现那隐藏多年的 bug 呢？

(4) 这些高能力的测试手段的掌握要经过什么样的培训和实践呢？

带着种种疑问，我们继续探索未来那美好的现实，继续从理论和实践中找出真理。我们会遇到很多质疑和困难，但这些都不可怕，可怕的是我们没有那种坚信于自己的信念和勇气能够给这个行业带来崭新的标准和前景。

笔者也将为了之前那些问题继续学习，继续讨论，继续实践，继续完善，直到我们可以做到那不一样的精彩。大家一起努力。

理论篇

1、ET 和 ST 的关系

上次说到了理想中的测试模型，有很多测试手段，细心的人可以发现那里面加了一个非常关键的测试手段就是探索式测试（最近越来越火了）。其实这里面把它叫为测试手段不是很科学，而且容易产生歧义。所以后面把它叫做一种测试方法。

笔者学习探索式测试都快一年了，看了很多国内和国外对于探索式测试的资料，对其有一定的了解，而且在国内也有相关的介绍，个人认为：

1. 大部分是正确的
2. 相当浅显的介绍探索式测试
3. 没有说明怎么来做探索式测试
4. 没有介绍探索式测试的实践情况

笔者看过了探索式测试（以下简称 ET）领域的 3 个核心人物的一些关于 ET 的 paper，也小小的实践了一个 ET 的一种形式，有很多感慨。但一直没有去写，有以下原因：

1. 不知道怎么表达出来比较好，一直都认为自己理解 ET 不深，怕说出来误导大家
2. 为了不让大家对 ET 存在误区，且自己对于 ET 某些的认识也是存在不同的看法（随着与那些大师们的沟通变得不一样）
3. 自己对于 ET 具体的测试方法的抽象还存在部分疑惑

最后发现了一点点的感觉了，就想把自己了解的一些情况记录下来 Share 给大家，希望能给大家带来不一样的认识。也希望不要给大家带来 ET 认识上的误区。

这里把 ET 叫做一种测试方法，而非测试技术，是有理由的（任何一个测试技术都可以应用到 ST 或 ET 方法中）。我们先可以分析下我们目前的测试模型，是集成在 spiral 或 waterfall 或类似的开发模型下，这就存在如下的几个特点：

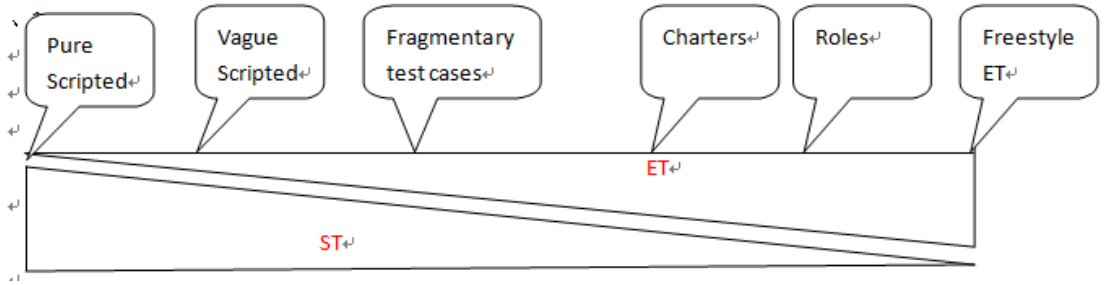
1. 测试文档（计划和设计和用例）必须非常详细和明确
2. 测试设计和测试用例对于开发的文档的依赖非常大
3. 测试执行的时候对于测试用例的依赖非常大
4. 测试执行的时候对于需求变更的应对力较差

我们可以把有以上特点的测试方法叫 Scripted based testing（简称 ST），显然我们目前大部分测试团队就是使用这种测试方法来组织整个产品的测试工作，由于这个方法有些缺点或不足，从而产生了一个新的测试方法：ET。

下面我们对于 ET 和 ST 进行了一些简单的比较：

类目	ST	ET
测试与测试用例的关系	测试用例在之前就设计和记录好，过后再测试执行或被其他测试人员执行	测试设计和执行时在同一时间完成，而且他们不是必须记录下来，但也有可能
与测试执行的关系	可以控制测试执行	可以加快测试设计和执行时的变化
过程的交互性	就像做个已准备好的演讲，由之前想好的想法引导着	就像一个对话，是自动向导的

在这里我们可以看到一点就是 ET 似乎就是弥补 ST 的一些缺点，理智的人就会想到能不能 ST 和 ET 结合起来。这样带来的效果是不是更好呢？请看下面的模型：



我们认为大部分的项目的最佳组合地方是在中间，也就是我们可以采用 ST 的优点和 ET 的优点进行组合。但这里要说明的是 ET 是 Context-Driven School 的代表作，其强调的就是没有最佳实践，任何项目都有自己的特点，根据特有的特点制定最佳的测试方法组合策略。

对于上面的模型图，可以看到有如下的特点：

最左边就是 **Pure Scripted**，也就是完全的按照测试用例来执行测试；而且测试用例非常详细。

最右边是 **Freestyle ET**，也就是自由式的 ET，在测试执行的时候没有任何测试文档的支持；不需要记录任何东西（bug 除外）；测试执行之前不需要任何特别的准备。

可以看到上面的两种方式都是不成熟的，而且都是不常见的，有点走极端。我们自己在做项目过程中不可能完全走 Pure Scripted 或 Freestyle ET。这里比较好的办法就是混合 ST 和 ET，并在不同的项目当中采取不同的混合策略来进行比较完善的测试方法的策略。在大部分的项目过程中，组合 ST 和 ET 会带来意想不到的效果。

目前我们所有的项目都是ST为主导的，偶尔会在测试执行的时候会去发散性的去测试一些东西，但这个完全建立在经验和时间和功能等众多因素上，其结果也不做期盼和追求。为了更好的混合 ST 和 ET，我们考虑尽量减少文档的编写时间，更多的提高在测试执行时创造性的发挥，也更加的体现在测试执行过程中持续性学习产品所带来的思维扩展性。

这里介绍怎么样混合 ST 和 ET 时，需要用到的几个关键性的因素(如上图)：

Vague Scripted：比较模糊的测试用例，也就是对 Scripted Test 的简化，这里可以理解为有测试用例（但没有一些比较详细的预期结果），也会有若干步骤（描述比较简单，其目的是留下更多的空间给测试者在测试执行的时候自由发挥）。

Fragmentary test cases：使用一句话或几个词语制定的测试用例，类似我们的用例标题。

Charters：ET 过程中使用到的一个非常清晰的任务列表，指出了要测试什么，怎么测试（强调策略，不是详细测试步骤），要寻找什么样的 bug，有哪些风险，要去检查什么文档等。

Roles：ET 过程中给测试人员一个独立的角色去测试产品的一部分。由他们自己掌控被分配的任务的测试的进度和质量。

这里主要说明了 ET 和 ST 的一些方式上的不同，至于 ET 的定义国内都有些介绍，但有些过于抽象，下面将从一个有趣的游戏来说明什么是 ET，她具有什么样的特点。

这个游戏是一个导师和学生们之间完成的一个问答游戏，首先是导师的说明：

导师：让我们玩一个叫”二十个问题”的游戏，我在想一个在宇宙中存在的一个东西，现在我让你们问我二十个问题看是否能够确定我在想的那个东西到底是什么。而且每个问题的方式必须要以是或不是来回答(因为我让你们问题的形式随意，那如果你们问”你正在想的那个东西是什么”，那么这个游戏就叫”一个问题”了)，准备好了吗？Brain，那我们从你开始吧，

Brain：这个东西是否与软件测试有关呢？

导师：无关，那样就会太简单了。

Michael：这个东西是否很大？

导师：不是，这个东西不大。

Rebecca: 是不是一个动物？

导师：不是。

Rayanne: 那是个植物吗？

导师：是的，它是个植物。

Henry: 是不是树？

导师：不是，它不是树。

Sree：它是不是有点大？

导师：不大，我已经说过了这个东西不大。

Eric：它是不是绿色的？

导师：是的，它就是绿色的。

Cheryl：那它是不是有叶子？

导师：是的，它有叶子。

Galina：是不是个室外的植物？

导师：是的，它一般长在室外

Jae：它是不是个带花的植物？

导师：不是，我认为应该不带，但我可不是个生物学家

Melanie：它是不是灌木？

导师：不是

Patrick：它是不是仙人掌？

导师：不是，它不是仙人掌。

Angel：它是不是黄瓜？

导师：不是，与其猜测具体是什么植物，不如确定这个植物的种类或范畴更有效

Sundari：它是不是种杂草？

导师：虽然问得好，可惜它不是。

Lynn：它是不是四季不断的，多年生的植物？

导师：不是，我认为应该不是的，我想它应该是每年都会别再植或改种。

Julie：它是不是从鳞茎(球形物)生长起来的？

导师：不是

Michelle：是不是每家的后花园(院子)都有这个植物？

导师：不是，至少我家的院子里面没有

Kristie：它是不是个非法的东西(同学们都笑了)？

导师：不是，它非常合法，好的，我们再来一次，Brain，让我们回到你这边重新开始

Brian：它是不是有毒的？

导师：不是，尽管我的孩子们认为它有毒

Michael：我们是不是可以吃它？

导师：是的，我们可以吃它

Rebecca：它是不是莴苣，生菜，苦菜类？

导师：不是

Rayanne：它是不是菠菜？

导师：是的，它就是菠菜，非常好。

如果我们提前写下这些问题，是不是可以更快的找到这个答案呢？当我们在玩这个游戏的时候，每一个问题都基于前面的一个问题和答案。对的，这就是探索式测试（ET）。每次测试产品都会提供我们相应的信息，我们可以看到产品是否运行正确的证据，我们就可以找到是否是产品缺陷的证据。我们也许会对某些东西非常好奇，也不确定它们到底什么意思，但这些会指引我们进一步去探索它们。因此，一旦我们在做 ET，我们同时进行学习产品，设计这些测试，执行这些测试。这也就是 ET 的最大特点。

上面我们看到了 ET 的简单的过程，而且可以看到 ET 还是非常有趣的，同时需要关注的是充分吸收 ET 和 ST 的优点，避免其缺点；下一篇介绍下 ET 在什么情况下适合做，ET 做的好不好与哪些因素有关系。

2、ET 的优势和缺点

上一篇我们说到了 ST 和 ET 的一些区别，大家对于 ST 都非常熟悉了，那么既然 ET 存在于测试领域，肯定有其应有的价值，下面说下这几个问题：ET 的优势和缺点和影响因素，最后说下一个优秀的 ET 测试人员应具备什么样的能力。

ET 作为一个比较现代的测试方法，肯定有其非常重要的**优势**：

1. 它可以鼓励测试人员的创造性且更有乐趣
2. 它增加了发现新的或者难以发现的 bug 的可能性(提高了一定的覆盖率)
3. 它允许我们有更多的时间去测试感兴趣的和比较复杂的用例
4. 它可以更有效率的驱使测试人员在一个很短的时间内找到更多的 bug 和对 SUT 做一个快速的评估
5. 它有助于有效的测试自动化（测试代码开发前利用 ET 找到潜在风险的/有价值的地方）
6. 它有助于设计更好的测试计划（利用 ET 找到潜在风险，合理安排资源和测试优先级）
7. 它能够在相同的时间内执行更多的测试，相比 ST 而言
8. 它可以促使测试人员快速的学习一个产品
9. 它可以 check 其他测试人员的测试工作
10. 它可以很好的应用在敏捷测试项目
11. 它允许我们不用花很多时间在编写那些简单和繁琐的测试用例

同样，相比于 ST 来说，ET 也有一些非常不好的**缺点**：

1. 它在测试管理上的局限性使得 ET 过程很难去协调和控制（ET 主导的测试模型）
2. 它在 bug 的重复利用或重现上提供有限的支持（可以做到，但相当于 ST 需要投入更多）
3. 它对于测试人员的测试技能和行业知识依赖比较大（ST 执行前也是有依赖的）
4. 当与 ST 进行组合时，会有重复测试的风险（漏测的风险比重复测试更有危害性）
5. 它不能完全保证最重要的 bug 已经被发现了（ET 能够尽早的发现重要的 bug）
6. 它只能在 SUT 完全可用的情况下才可以发挥大的作用（部分功能提交，也可以 ET）
7. 很难的去定义 ET 的生产率
8. 无法对测试思路进行自动化（事后可以考虑自动化）

9. 很难证明自己测试了哪些细节或无法重复使用（回归）测试思路（及时交流和总结，可复用部分 ET 测试思路）

我们知道了某些情况下 ET 比较合适，那么就像之前说过的一样，ET 没有最佳实践，ET 在项目中做的好与不好，都会受很多情况制约，这些情况都会或多或少影响着 ET 实践的数据产出，下面列出了比较重要的制约因素：

1. 这个项目的测试的具体任务（一般和测试类型和产品本身的特点有关）
2. 这个测试人员的角色（lead 或 SDET 或 STE）
3. 具体的测试人员（技能，天赋，擅长点）
4. 可用的测试工具和测试机器
5. 可用的时间
6. 可用的测试数据和文档
7. 从其他的人员那里获得的帮助
8. 当前的测试策略
9. 同一个产品已经经过测试后的状态

其实我们可以总结影响 ET 的基本因素为：**时间**，**测试人员**，**产品**，**任务**。我们还可以分析下 ET 过程中的几个关键的因素，其实也就是一个优秀的 ET 测试人员所具备的基本能力：

测试设计：一个优秀的测试设计师，一般有如下几个能力：首先是分析这个产品；评估产品的所有的风险；使用现有的工具去分析或记录；测试设计技术的熟练使用。

细心观察：一个优秀的 ET 测试人员必须比一般的人甚至是做 ST 的测试人员更具有细心观察细节的能力。ET 测试人员必须去观察一切看似不正常或有疑问的地方，他还要能仔细的在推论和其他一些的假设中辨别出真理何在。

批判性思考：一个优秀的 ET 测试人员能够快速的评审和解释他们的思考逻辑，并能在独立思考中寻找错误。这在重现 bug 的时候非常重要。

丰富的想法：一个优秀的 ET 测试人员能够在测试执行的时候比一般测试人员产生更多且更好的想法。但通过什么来产生这么多且好的 test idea 呢？这个也就是 ET 的核心机密了，目前 ET 的大师们创立了一个叫 Heuristics 的方法，这个方法比较抽象且实践过程在国内几乎空白，后续文章会讨论下。

丰富的资源：一个优秀的 ET 测试人员能够构建一个集测试工具，信息资源，测试数据，测试同仁的一个储存室。这样在测试的时候，可以很快的应用这些资源。

以上说明，ET tester 的能力要求还是比较高的，很多人觉得上面的能力都是软件实力，认为一

个非 ET tester 上面说到的能力也是很需要的，但这不需要说明的是 ET tester 更强调这些能力而已。下次说说 ET 怎么样在我们的项目中使用起来，什么时候使用 ET，还有如何更好的管理 ET 在项目中的投入，如何去衡量 ET 测试人员的工作量和产出。

3、ET 的管理手段

现在对于探索式测试，大家很多的看法都存在一点误区，都认为探索式测试很难控制和管理测试进度(原因据说可能是 ET 不需要写测试用例)，大家在网上可以找到很多关于探索式测试的介绍，很少介绍到探索式测试应用在项目过程中是如何管理的，没有涉及到核心，那么很多人都关心这个 ET 到底在项目中怎么来做呢？还有一个就是之前说了 ET(探索式测试，后续统称为 ET)很难去控制和管理整个进度，那到底有没有什么好的办法去管理 ET 呢？下面我们就去看看国外是怎么做的吧。

首先要说的是这些问题在 ET 发展过程中，ET 的那些大师们已经想好了怎么在项目实践过程中应用 ET，也有一些比较成熟的解决办法。

3.1 实践中 ST 和 ET 的使用模型

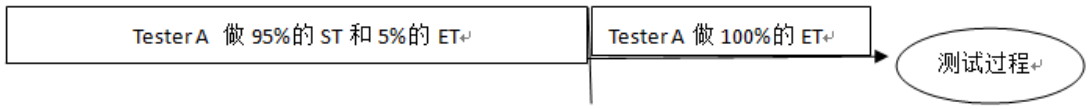
这里需要说明的是怎么应用 ET 在项目过程中，可以从不同的维度去考虑：

一个是 ET 和 ST 的结合方式，和测试人员具体做 ET 还是 ST 或都做有关；

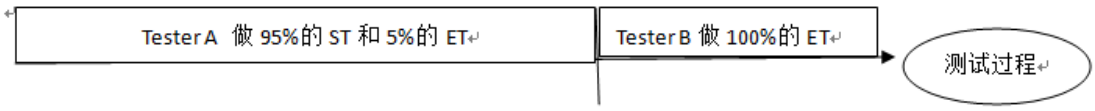
另一个是 team 的组成方式，从测试的专业性角度去分隔开 ET tester 和 ST tester；

看下面的模型可以更容易理解：

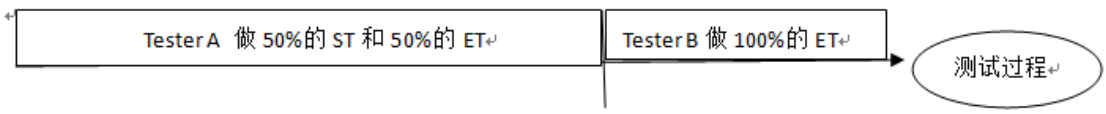
模型一：



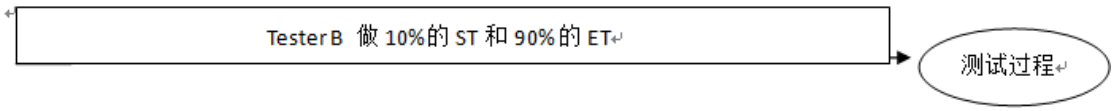
模型二：



模型三：



模型四：



备注：ST 就是 Scripted based Testing，就是我们目前所做的基于测试用例进行测试的测试方法；Tester A 是专门做 ST 的 Team 的 ST 测试人员，Tester B 是专门做 ET 的 Team 的 ET 测试人员，原则上不应该分开 ET tester 和 ST tester，此乃更好的让各位理解这四个模型。

从模型的演变过程可以看到，其实我们最终目标是没有 ST tester 和 ET tester 之分的，所有人都是标准的 ET tester。这里需要强调的是不同的模型对应着不同的测试设计和测试用例的深度和方式。也就是说采用不同的模型来应用 ET，其测试设计和测试用例的编写与不编写，或具体编写的方式都有很大的不同。国外 ET 的大师们确实带了好几个 ET team，在项目测试过程中已经达到了模型四的境界，可想 ET 是可以主导整个项目测试的，其进度控制和质量管理都在实践中有了自己特有的方法。

3.2 ET team 的管理方式

在 ET team 里面又存在两个不同的管理方式：Delegation 和 Participation，这个区分的角度是从 ET team lead 在整个项目 ET 过程中的作用来看的。

Delegation：

(1) Test lead 指定需要测试的 charters，不参与具体测试任务

(2) ET tester 完成这些 charters 并且 report back

(3) 对于一些问题和测试报告召开定期会议

Participation：

(1) Test lead 在项目测试过程中与 ET tester 一样，参与某些测试任务

(2) Lead 可以实时的根据测试质量情况制定最好的测试策略

(3) Lead 可以持续的了解他所想要的了解的 team 的任何情况

这里相比较单个人进行 ET 测试，我们还可以使一个 ET team 的几个人工作在一起，在同一时间对于同一个 SUT(Software Under Test)进行 ET，时常会出现更多更好的测试 idea。 所以还有一种组合 ET team 的方式是让测试人员组成一对且让他们在同一台计算机上进行测试，也就是结对测试(Pair Testing)，这里有几个好处：测试者之间可以互相学习，测试者可以相互启发有价值的测试思路，是培训测试新人，团队新人的好方法(国外很多 team 都采用这种方式)。

另一种就是其中一个测试人员进行测试，傍边有多个测试人员观察且做记录，通过问测试执行

人员不同的问题产出更多的测试 idea，这里有一个好处就是测试执行人员不必担心发现的 bug 难以重现，因为傍边的测试人员会做记录和分析，这样测试执行人员可以不必分心而继续执行自己的测试思路。还有一个好处就是如果一旦这个测试人员思维过于开阔，去测试很多非当前需要测试的模块时，傍边的测试人员可以给予及时提醒。但其缺点也很明显就是更多资源消耗在同一个功能模块上，成本上有待商榷。

3.3 ET 过程中的任务

下面我们说说具体的 ET tester 是怎么完成这些 Charters 的，还有怎么管理和控制他们完成的这些 Charters。Charter 的定义：ET 过程中使用到的一个非常清晰的任务列表，指出了要测试什么，怎么测试（强调策略，不是详细测试步骤），要寻找什么样的 bug，有哪些风险，要去检查什么文档等。

在测试执行之前，ET lead 和 Senior ET tester 参与制定所有的 charters，并根据实际情况从 charters 中分离出所有需要测试的 sessions（一个基本的测试工作单元，一般对应 1-2 个 UC）。这里我们可以看到 ET tester 所有进行的 ET 是基于 session 的，那么我们使用的管理方法就叫 Session-Based Test Management（Jon Bach 的首创）。

这里需要说下制定这些 session 的一些基本的原则：

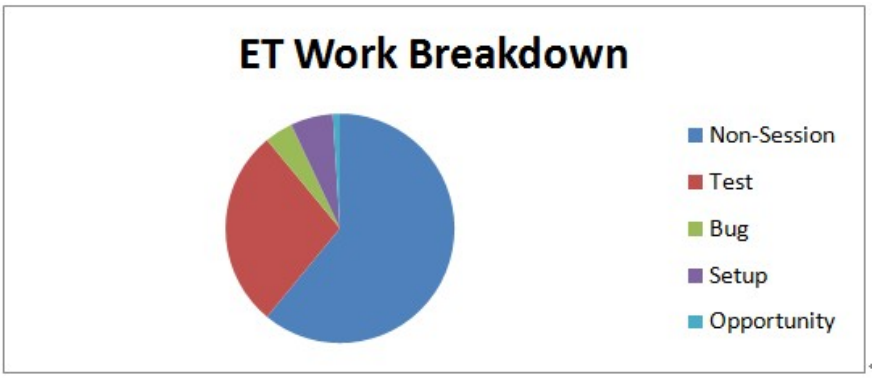
(1) 这些 Session 必须是 chartered（与测试任务是绑定的）

(2) 这些 Session 必须是 uninterrupted（独立的功能，且一般执行时不受外界的干扰）

(3) 这些 Session 必须是 reviewable（其报告产出 Session sheet 可以被第三方 review）

(4) 每个 Session 一般不超过 90 min，其大致范围从 45 min（short session）到 120 min（Long session）

首先我们来分解下 ET 过程中，大概花的时间的分配：



（Opportunity 指的是 Opportunity testing，就是测试执行了其他的 session 的功能）

从上图我们可以看出，测试执行的时间也就是我们确定了测试思路并对 SUT 进行测试的时间并不多，我们更多的时候是在观察我们能够观察的结果上面，再从中发现新的测试思路。

这里每个 ET tester 完成一个 Session 过后，必须记录报告 session sheet，下面是主要的因素：

(1) Session Charter (charter 名称，和 session 名称)

(2) Tester name(s)

(3) Data and time started

(4) TBS metrics (包括 3 个方面的 effort 统计：测试设计和执行，bug 分析和报告，session setup)

(5) Data files

(6) Test notes(ET 过程中的随时记录的一些东西，比如 test ideas, risk list 等)

(7) Issues(ET 过程中的问题和疑惑)

(8) Bugs

3.4 ET 中管理 Session

下面我们继续 ET 在项目中的管理（比较适应与 3.1 里面说到的模型三或四），大致过程如下：

1. Lead 根据时间任务和 SUT 本身制定出 Charters 和 Sessions
2. Lead 针对所有 sessions 和资源来做出 Test plan 和测试策略(包含对 SUT 的攻击策略)
3. 设计阶段 ET tester 对于自己负责的 sessions 进行 test idea development
4. 设计阶段 ET tester 还需要对自己负责的 session 所对应的 oracles(可以认为测试执行过程中发现的问题是个 bug 的所有信息,包括文档) 进行梳理
5. 测试执行的时候 ET tester 根据之前的 test idea list 来进行测试，根据 SUT 本身的 response 并采用 Heuristics 的方法产出更多的 test idea(这部分的 test idea 不是必要记录下来)
6. 测试执行完成的时候每个 ET 测试人员填写自己的 session sheet
7. Lead 每天汇总每个测试人员的 session sheet
8. 利用自己开发的工具支持分析所有的 sessions 的执行情况(国外已存在这个小工具)
9. 根据分析结果得到测试进度的具体执行情况和质量情况，并根据结果调整测试策略

这里的工具会提供足够多的数据来告诉我们的测试进度是否在合理的范围内，是否存在哪些风险，哪些测试人员在 Non-session 中花了时间过多等等(工具下载地址：http://sessiontester.openqa.org/download.html)。由于产出的 metrix 提供足够的信息去控制整个 ET 进度，当然我们还需要了解未来的计划安排，所以 metrix 里面会有个 item 叫 To DO Sessions 来表明我们未完成的 session 数量。 还可以帮我们更加的评估 ET tester 的生产率，同时也可以评估我们 ET

tester 的快速学习能力。

至于这个管理和进度控制的方式是否可行呢？笔者有幸采用了 Freestyle ET 的方式和这个管理方式来实践了 XX 项目，实践遇到的问题很多，总结也很多，其中最大的问题也就是我们需要花大力气解决的就是如何开展每个 session 的测试工作，也就是之前说的那个 Heuristics 的方法快速的产生更多的更好的 test idea。另外一个就是 Lead 如何更好的对于 SUT 进行 Charters 和 Sessions 的划分并安排合理的资源去跟踪和调整。

另外在 ET 主导的项目测试过程中，一轮测试完成后，测试团队应该总结本轮测试的情况：

1. 测试人员互通情报。这有助于避免重复测试，也为进一步的探索提供良好的基础
2. 总结缺陷模式(Bug Pattern),这对后续的测试有帮助：更好的理解 SUT，制定和调整更好的 charters 和 check list。典型的问题要反馈给开发团队，帮助开发者成长
3. 大家一起分享经验，一种集体学习行为，有助于构建有凝聚力的团队

这里大家感觉到的更多是在进度上是可以控制的，但在质量上不知道 ET 是怎么去把控的，管理层只是知道 tester 测试了哪些东西，由于没有细化到具体用例这一层，且不像 ST 那样可以 track 每天的 Run 的用例数，所以 ET 对于大部分人来说，就觉得这样进行测试执行不可靠且无法 track 结果。这里说下个人观点：一定程度上，发现的 bug 数就能很好的证明 ET 覆盖了哪些测试点，比较悲观的是，ET 目前无法很好的证明自己曾经测试过那些没有发现 bug 的测试点，这也许是管理层比较关心的问题。后面说下 ET 过程中测试思维的变化以及可依赖的资源。

《测试之美》第 12 章“软件以用为本”讲述了一个精彩的故事。虽然全文没有提到术语“探索式测试”，但是它展现了一个成功的 ET 团队的特征。

1. 测试团队测试医疗设备。 由于产品关乎病人的生命，所有成员的热情都被激发出来。一个被激励的团队的能量是无法估量的。
2. ET 团队工作在一个房间里工作。他们随时讨论，即时交流情报，用热情和技能相互支持。
3. ET 团队构造了非常接近真实环境的测试环境。他们在其中忘我工作。效率奇高。
4. 同时，有一个团队执行 ST，ET 团队提供情报给 ST 团队，使他们可以构造更有效的测试用例。

4、ET 的思维过程分析

之前说了些 ET 在项目时间过程中是如何来管理的，那么 ET tester 在拿到自己的任务的时候，自己是怎么来进行 ET 的呢？

这里我们先考虑两个比较基本的状态，一个是 ET tester 的状态，有如下几个具体状态：

(1) 这个 ET tester 的测试经验怎么样，丰富还是欠缺？

(2) 这个 ET tester 对 SUT 的行业经验怎么样，熟悉还是了解？

(3) 这个 ET tester 对 SUT 本身的功能需求了解怎么样，熟悉还是了解？

第二个是 SUT 产品本身的状态，有如下几个具体状态：

(1) 这个产品开发目前是处于什么阶段？

(2) 这个产品是否经过了测试，测试了哪些类型或大概内容？

(3) 这个产品目前的风险或潜在问题有哪些？

4.1 Heurisitcs 和问答模式

不管上面的状态怎么样，在压力(时间压力和业务质量压力)下，ET tester 采用正确的方法进行 ET，往往会取得较好的效果的。

首先这个 ET tester 要非常清楚自己的状态，也就是自己所拥有的 Knowledge，具体包括这些：

(1) 该产品的知识

(2) 测试技术相关的知识

(3) 该产品所在的行业知识

(4) 基本的计算机基本知识

为啥要去 check 这些知识呢？这样为了方便我们在做 ET 的时候，能够快速和有效的确定发现的问题是否是个 Bug。国外有很多这方面的总结，怎么去找到这些证据去确定发现的问题是不是 Bug，这些东西别名叫 **Oracle**，找 Oracle 可使用这个方法：**the HICCUPPS(F) heuristic**：

(1) History: 目前所做的产品的版本是否与过去的版本是一致的。

(2) Image：这个产品是否与项目组织所想要的 image 是一致的。

(3) Comparable Products:这个产品是否与相类似的产品是一致的。

(4) Claims：这个产品是否与重要的人所说的那样是一致的。

(5) User's Expectations: 这个产品是否与用户所想要的是一致的。

(6) Product:这个产品的每个元素是否与同个产品里面的可比较的元素一致。

(7) Purpose: 这个产品是否与其目的(明确的或含糊的)一致。

(8) Statues: 这个产品是否与可适用的法律一致。

(9) Familiarity: 这个产品与任何通用的问题的形式是不一致的。

尽管我们有很多方式去丰富我们的 oracles，但要想达到一定的程度还是需要很多时间成本的，所以我们在做 ET 过程中，会遇到：

(1) 没有 oracle 可以使我们提前确定这肯定是个正确或错误的结果。

(2) 没有一个单独的 oracle 可以说明某个功能在所有时间或所有情况下都是正确工作的。

(3) 有些功能看上去是正常工作的，但事实上在某些情况下会失败而且会使得所有的 oracles 都是不正确的。

可以看到我们在积累我们的 oracles 时，肯定会遇到很多 oracle 本身问题，我们该如何来解决呢？

(1) 忽略这个问题(也许这个信息的价值从成本角度考虑不值得)

(2) 简单化这个问题(需要可测试性，从需求源头开始追踪)

(3) 转移这个问题(并列测试，从类似问题下手)

这里面很多人都想知道我们到底怎么去做测试，怎么快速的产出 test idea。ET 的特点是我们做测试的时候，没有具体测试用例指导，学习 SUT，和测试设计和测试执行在同一段时间内完成。那怎么去做测试执行呢？我们拿到开发提交的系统后，Lead 分配给我们的任务后，该从哪里下手呢？

ET 大师 James Bach 说过，执行 ET 就像对一个人进行面试一样，那这样就少不了要问问题，该怎么问这些问题(一个问题就类似于一条测试思路)？通过面试者的回答怎样快速提出更好的问题？显然这里面也需要一定的能力和技巧，包括如下：

(1) 提出有用的问题(目的驱动的问题)

(2) 观察什么事情正在发生

(3) 描述自己能够感觉到或看到的东西

(4) 对于自己的所知进行批判性的思考

(5) 组织和管理业务上的规则

(6) 能够设计假设和进行试验

(7) 尽管已经知道了仍然进行思考

(8) 分析其他人的思考方式

(9) 根据因果关系进行推导

现在 ET tester 都对自己的 knowledge 和自己的 skill 有所了解了，接下来就是确定自己的任务，怎样对哪个 UC,哪个模块进行分析了，怎么分析，分析啥呢？

(1) 风险

(2) 覆盖率

(3) Oracles

(4) 资源或限制

(5) 价值或成本

(6) 已存在的 Bugs

上面这些分析都做好了，那我们就可以开始进行测试了，也就是对我们的 SUT 进行试验，就像“**question<--->answer**”的循环一样不断地进行试验。

那我们是怎么来进行试验的，具体包括如下几个过程：

(1) 配置

---安装产品

---为测试执行准备测试数据和工具

---确认产品是个足够干净的起始状态

---根据自己的任务准备一个有激发性的问题

(2) 操作

---通过问题对产品进行试探性的输入来使用这个产品

---使用正确的数据和正确的业务顺序来完成正确功能的练习

(3) 观察

---收集关于这个产品是如何工作(正确或错误的输入)的信息来评价产品是否如此工作

(4) 评估

---应用之前得到的 oracles 来发现 bugs

之前也提到了，我们的 ET 是学习和测试设计和测试执行是同一时间完成的，那么我们的测试执行也就是学习，类似于 Testing to learn。那么学习也有几个步骤：

(1) 去形成一个关于产品功能的模型

(2) 去了解这个产品是想实现什么样的功能

(3) 列出那些你需要测试的产品的因素

(4) 查看那些一致性的关系且尝试多个不一样的 oracle

(5) 产生测试数据

(6) 考虑其可测试性和尝试不同的有用的工具

(7) 尝试多种不同的方法去试验

(8) 报告你所发现的 bug

ET tester 在试验的过程中，如果遇到比较困惑的问题该怎么呢?

(1) 简化自己的测试思路

(2) 保存当时的状态

(3) 不断重复执行自己的 action

(4) 返回到一个已确认过的状态

(5) 优先使用 OFAT 方式(一次只考虑一个因素)

(6) 做出精确的观察

之前也说到了我们需要用 Heurisitcs 来产生更多更好的 test idea。其实对于 Heurisitcs 本身来说，有如下几种类型：

(1) Guideword Heurisitcs：一些词语或标签能使 ET tester 看到自己的 knowledge 并且根据自己的经验分析并产出一些新的思路

(2) Trigger Heurisitcs: 一些存在于事件或条件中的想法能帮助 ET tester 认为现在可以采用另外一种方式来进行试验，就像思维的闹钟提醒一样

(3) Subtitle Heurisitcs: 能帮助 ET tester 重构想法并想到更多的选择点，或在一个谈话中找到其中的假设和论点

(4) Heurisitcs Model: 能帮助 ET tester 控制和管理和挖掘更多的想法和实体

我们做 ET 测试过程中，下一个 test idea 是不可预知的，完全依赖本次测试用例的执行情况来判定，那么就存在一个小的疑惑(是继续进行深入挖掘还是跳出其循环呢)，这种过程在 ET 过程中很常见，业界称之为 Plunge in and Quit Heurisitc，详细解释为：一旦我们决定去测试一些看似比较复杂或有困难的地方，就直接 Plunge in；但如果我们非常困惑或感觉自己完全被 Block 了，就 Quit。这样就意味着我们可以开始任何测试却没有要求一定要成功完成并得到结果(是一把双刃剑，有管理风险)，也就是我们不需要一个完整的计划，当然我们循环着 Plunge in and Quit，就会产出一个新的计划。

同样的可以解释我们在 ET 测试过程中，存在分支是非常好的，也就是说不停地产生新的分支(test

idea)，而不是一条直线走到底。所以在 ET 测试过程中，尽量的让自己走更多的分支，因为我们根本不知道走另外一个分支后，会发生什么事情，但凡事都有个度，我们需要定期的去 check 我们现在测试的模块是否与我们被分配到的任务是否一致。防止过多的时间花在之前说过的 Opportunity testing 上，要不然就是得不偿失了。

4.2 ET 总体过程和覆盖率

回到之前所说的，当我们拿到自己的任务的时候，知道了自己需要测试哪些模块，就需要有一个策略去进行 ET 测试，我们可以考虑如下：

首先，学习这个产品基本知识；

其次，思考这个产品存在一些重要的潜在的问题；

最后，思考怎么去探索这个产品才能找到这些问题；

一般情况下，思考怎么去探索这个产品，思考的方式有这些：

充分利用自己现有的资源

----包括：用户，文档信息，开发人员关系，同事，设备和工具，计划等。

使用一些不同的技术的混合体

----包括：功能测试，域测试，压力测试，流程测试，场景测试，用户测试，风险测试，自动化测试，声明测试。

利用自己实际能够做的东西

----包括：自己擅长的，最容易暴露问题的，自己的时间和精力。

在做 ET 过程中，需要不断的变化我们的测试思路去攻击 SUT，但一般我们会采用哪些方法或思路去变化我们的 test idea 呢？

微小的行为：其他的人或不用心的人改变了一点点思路，可能会产生新的 test idea。

随机性：能够让你从那些有选择性的偏见中走出来

数据交换：同样的操作，在不同的数据库中，或不同的输入，会得到很不同的结果。

平台替换：推测类似的平台，但却不支持某种操作

时间/并发变化：同样的操作，改变依赖其发生的时间周期或并发的事件时，会得到很不同的结果。

场景变化：同样的功能，当我们在一个不同的流程或上下文环境下，会出现截然不同的操作方式。

状态污染：一个复杂的系统，一般会有很多隐藏的变量，通过改变某些操作的顺序，级数，类型，

会加快程序状态污染，从而发现隐藏问题。

敏感度和期望值：不同的测试人员可能对不同的因素，会有不同的敏感，且有看到不同的区域。同一个测试人员在不同的时间可看到不同的东西。

为了在复杂的系统中快速发现未期望的问题或持续使用才会出现难懂的问题，更多的问题，我们必须 De-Focus：

1.从一个不同的状态开始（没有必要清除状态）

2.优先进行复杂且有条件的 action

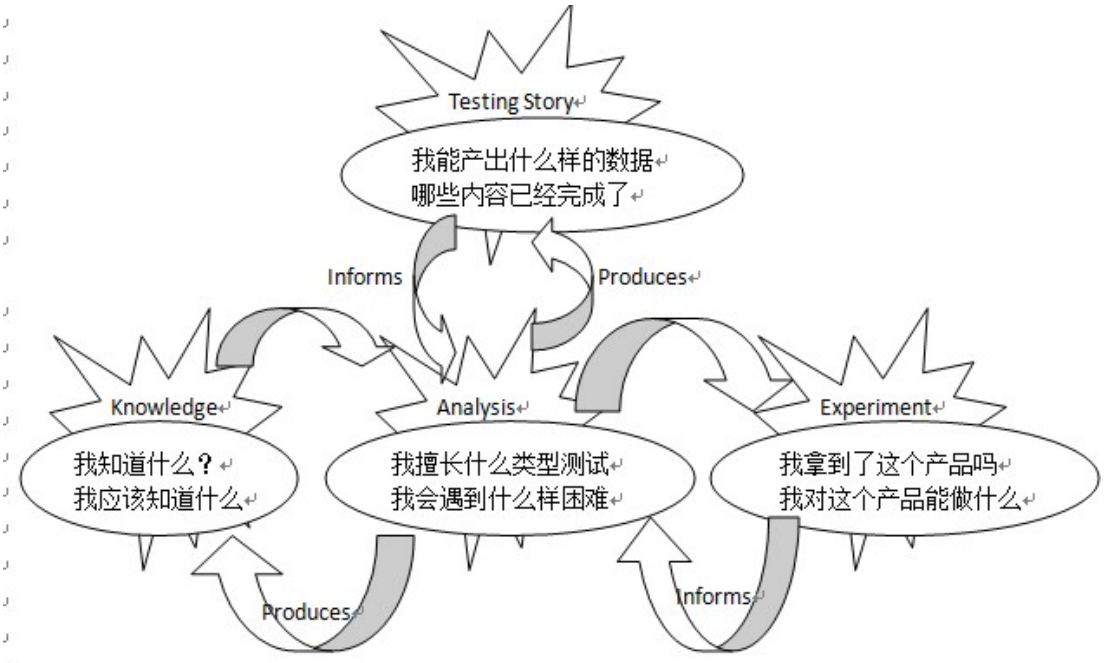
3.从模型的变化中产生 test idea

4.质疑我们的试验流程和工具

5.试着去观察每个东西

6.宁愿使我们的测试很难通过，也不能太顾忌问题的重现难易程度

下面从一个模型中来说明怎么开始 ET 测试，选择一个**有用的，有趣的，容易开始**的点：



这里还是需要详细说明下：Knowledge，Analysis，Experiment，Testing Story.

Knowledge: Product Story; Technical Knowledge; Domain Knowledge; General knowledge.

Analysis: Risk; Coverage; Oracles; Resources/Constraints; Value/Cost; Bugs.

Experiment: Configure; Operate; Observe; Evaluate.

Testing Story: Test Plan/Report; Work Products; Testing Status.

上面说了很多，但细心的人就可以发现，我们还没讨论 Coverage 这个关键东西。在 ET 实践过程中，同样在理论形成过程中，Coverage 永远是一个很大的问题，很多人都说 ET 最不能保证的就是 Coverage，所以不敢轻易去尝试。那么 ET 在 Coverage 是不是真是这样不靠谱呢？事实上，不管你采用 ST 还是 ET，对于 Coverage 都是很难完全保证的，但我们可以在 Coverage 上做一些关键性的努力，也许结果就不一样了。

接下来说下 ET 在 Coverage 上是怎么考虑的。我们说的 Coverage 一般就是 Product coverage，同样也是这个被测产品的一部分。那么对于 Product coverage 又包括哪些方面的 coverage 呢？

第一个就是 **Structure**，也就是产品的一个因素，对于这个 Structural Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品是怎么构成的，我们要 cover 的就是构成这个产品的部分。下面以打印机产品为例，看看 Structural Coverage 到底要考虑什么：

------打印需要用到的文件

------实现打印功能的代码模块

------在这个模块里面的代码语句

------在这个模块里面的代码分支

可以看到这个时候我们关注的是产品的内部结构。

第二个就是 **Function**，也是产品的一个因素，对于这个 Functional Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品能够做什么？我们要 cover 的就是这个产品做得什么样。同样以打印机产品为例，看看 Functional Coverage 到底要考虑什么：

------打印，页设置，打印预览

------打印 range，打印复制，zoom

------打印所有的，当前页，或指定的 range

可以看到这个时候我们关注的是产品的功能或特性。

第三个就是 **Data**，也是产品的一个因素，对于这个 Data Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品能够对数据方面有什么考虑？我们要 cover 的就是这个产品能够处理什么样的数据。同样以打印机产品为例，看看 Data Coverage 到底要考虑什么：

------打印文档的类型

------文档里面的元素，文档的大小和结构

------关于怎么打印的数据(比如 zoom factor; no. of copies)

可以看到这个时候我们关注的是产品使用过程中不同的数据处理。

第四个就是 **Platform**，也是产品的一个因素，对于这个 Platform Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品依赖什么才能使用？我们要 cover 的就是这个产品怎么处理不同的依赖的。同样以打印机产品为例，看看 Platform Coverage 到底要考虑什么：

------打印机，Spoolers，network behavior

------计算机

------操作系统

------打印机驱动程序/设备

可以看到这个时候我们关注的是产品使用过程中不同的环境和依赖。

第五个就是 **Operation**，也是产品的一个因素，对于这个 Operations Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品怎么使用的？我们要 cover 的就是这个产品使用的步骤是否合理/正确。同样以打印机产品为例，看看 Operations Coverage 到底要考虑什么：

------默认情况下使用

------真实环境下使用

------真实的场景下使用

------复杂的流程下使用

可以看到这个时候我们关注的是产品使用的场景(包括稳定性，可用性，安全性，可扩展性，性能，可安装性，兼容性，可测性，维护性，本地性等)。

第六个就是 **Time**，也是产品的一个因素，对于这个 Time Coverage，我们到底是测试什么呢？我们到底要 cover 什么呢？我们要测试的就是这个产品在什么时间情况下会受影响？我们要 cover 的就是这个产品在不同的时间下会表现什么样。同样以打印机产品为例，看看 Time Coverage 到底要考虑什么：

------尝试在不同的网络或端口的速度使用

------一个文档打印完，紧接着打印另一个文档，或隔很长时间再打印

------尝试与时间相关的限制，比如使用 spooling, buffering, timeouts

------尝试 hourly，daily，月底，或年底打印报告

------尝试从不同的 2 个工作站同时打印

可以看到这个时候我们关注的是产品使用的时候是否受时间影响。

上面我们可以看到 ET 在考虑覆盖率上还是有一点自己独特的角度，但感觉不是很具体，无细节，对于某一个类型的 Coverage 需要做出全面的分析，还有一个就是这些不同类型的 coverage 会经常组合在一起来使用的。至于组合的策略，需要在实际项目过程才能体会。

最后说下，ET 强调的是尽量的简化写文档的时间(不是说不要写测试设计或测试用例的文档)，但我们还是需要一些文档的支持的，以便管理层跟踪和做出正确的决定。ET 的文档包括如下 2 大块：

通用的文档：Testing Heuristics；Risk Category

项目相关的文档：Coverage Model；Risk Model；Test Strategy Reference；Schedule；Issues；Bugs；Status Dashboard

这里我们可以看到 ET 所做的文档确实不多，这就需要我们在 ET 过程中，还要准确实时的 Taking Notes，主要包括这些：

Coverage （测试了哪些，覆盖了哪些点）

Oracles（测试过程中，发现或依赖的 oracle）

Procedures（只要关键的流程就可以）

Test Ideas（关键的测试思路以及 check point）

Bugs/Risks（发现的问题以及风险）

Issues/Questions/Anomalies

看到现在，你也许很糊涂，很迷茫，不知道这里在说什么，其实 ET 测试过程是带有一些隐含的艺术在里面，思维的急剧转变以及思考问题的多变性，这些东西可以多了解下，这样才会明白后面的测试思路和抽象方法的来源，当然如果你只是做 ET 测试方法的应用者，你可以不去了解这些方法的来龙去脉，可以不去深入挖掘测试过程中思维的变化。但如果你想发掘和很好理解后续的 ET 测试方法，就需要全面了解这些考虑点。后面将说明如果一个项目要采用 ET 主导则其总体流程是个什么样的。会给项目测试管理带来什么样的变革。

5、ET 实践总体流程前奏

要推广和接受一个新的想法，大家都喜欢在实践中看看效果，那么现在大家已经理解了 ET 了，就很希望在一个项目过程中来实践 ET，实践的具体流程是什么呢？需要考虑什么异常情况呢？接下来就说说 ET 实践的总体流程(这个流程是非常官方且标准的 ET 实践流程，曾由 James Bach 写给微软做 windows 产品兼容性测试任务的官方证据)：

这个流程大体上包括 3 个部分：

----Working with Functions

----Testing Functionality and Stability

----Test Procedure

这里面就说下前面两个大点：

5.1 Working with Functions

这个流程是围绕着功能来组织的。所谓功能就是一个软件所要假定要做的事情。这就包括任何显示的，改变内部或外部数据的，或影响环境的任何结果。当然功能一般都包含子功能。比如：在微软 Word 产品里面“打印”功能就包含“复制几份”和“页面范围设置”2 个子功能。

由于我们需要测试所有东西，就必须通过制定基本风险的决策来简单化哪些功能需要花费多少精力。我们把所有的功能分为 2 大类：主要的，贡献性的。大部分情况，我们需要记录和测试主要的功能。至于这些功能该怎么样的分类和组合是根据情况来定的。你也许会认为一组贡献性的功能可以作为一个单独的主要功能，或一个单独的主要的功能能够分解为主要的和贡献性的子功能。

如果可能的话，我们想去测试所有的主要功能，但可能没有足够的时间去做。这种情况，虽然我们不希望会发生，但需要对你将测试的和不会测试的主要功能做一个文档的记录。

如果只看用户反映的话，就很难识别出一些功能。有些功能是与操作系统，其他程序，或修改文件进行直接交互，而这些在显示上并不能直接看到效果。特别需要重视那些重要的且有可能是部分隐藏的功能(用户不易观察到的，比如数据库数据文件修改，多个程序之间的修改等)。

如下定义功能的分类的方式：

定义	**说明**
主要的功能： 从一个普通用户看来，一些比较重要的功能，	一个功能是否是主要的与这个产品的目的以及对于这个目的来说，这个功能是否是必须

而且它的不易操作性和危害都使得这个产品的目的没有达到	的有关
贡献性的功能： 这些功能使得这个产品更加有实用性，但不是主要功能，能使用户更兴奋的功能	就像主要的功能的辅助功能，类似于增值服务类型的，从可用性角度会发现一些

5.2 Testing Functionality and Stability

我们测试目前大部分是需要测试功能性和稳定性的问题，那我们就必须有通过测试的标准，如下是定义的标准：

定义	**通过标准**	**失败标准**
功能性 (产品提供的功能的有效性)	每个主要的功能其操作的结果与其目的都是一致的,其输出的结果的正确性都经过测试的。	至少有一个主要功能与产品的目的不一致
	对于正常使用任何不正确的行为并没有严重损害用户的利益	对于正常使用任何不正确的行为严重损害用户的利益
稳定性 (持续性的提供功能的能力，且没有失败)	产品没有毁坏系统或平台	产品毁坏其系统或平台
	产品没有挂掉，毁坏，或数据丢失	产品挂掉，毁坏，或数据丢失
	测试过程中,没有主要的功能存在不易操作性或失效	测试过程中,存在主要的功能存在不易操作性或失效

Test Coverage

测试覆盖率就是将要测试什么？如下的测试覆盖率是需要的：

在时间允许下合理的测试所有的主要的功能。让 lead 知道有哪些主要的功能没有时间测试或没有能力测试？我们可以对一个有趣的贡献性的功能进行测试。也有可能在探索和测试主要功能时接触更多的贡献性的功能。

选择一些潜在的不稳定因素进行测试和选择一些有可能触发不稳定的功能的数据进行测试。一

般情况下，选择 5 到 10 个。Lead 会决定对于这个功能性和稳定性的测试需要多长时间，一般是花 80%的时间在主要的功能上，10%在贡献性的功能上，10%在不稳定的方面。

Sources and Oracles

我们在做 ET 过程中，是怎么知道这个产品就是应该这样做的呢？我们是怎么知道它什么时候不该这样处理的呢？这里如果我们需要回答很好的话，并使得 lead 满意，必须考虑下面 2 个方面：

Sources：就是做 ET 过程中需要的信息的来源。有时是你自己的灵感或经验。但更多的是我们已经了解了相关的产品文档。而且大部分情况，我们需要经常与相关人员确认该产品的目的和功能。

Oracles：就是做 ET 过程中确定所看到的产品的行为是否正确的策略。也就是回答：你怎么知道它应该是这样工作的？’

一般使用下面几个方式去确定 Oracles(ET 的过程分析 4.1 也讨论过):

与目的一致：功能的行为与产品的目的是一致的。

与产品本身一致：这个功能的行为与这个产品的相关的功能或功能实现方式是一致的。

与历史一致：现在的功能的行为与以前的功能行为是一致的。

与相比较的产品一致：这个功能的行为与相比较的产品的类似功能的行为是一致的。

Task Sheets

这个流程包括 5 个任务，而且每个任务都包含下面五个元素：

任务描述，Heuristics，结果，任务完成标准，FAQ

我们在做任何一个任务的时候，肯定会遇到很多问题，而且有些问题必须反馈给 lead，一般如下情况需要反馈给 lead，并咨询他如何处理这些问题：

当你遇到一个问题会阻碍你完成一个或多个测试任务的时候。

当你对于这个产品的复杂度感到困惑的时候。

当你在限定的时间内因为不能很好的了解系统而不能很好的进行测试的时候。

当你遇到一个问题有可能会严重影响到产品的功能性或稳定性的时候。

当你认为由于这个产品的复杂度而需要比原先分配的还要多的时间进行测试的时候。

参考自 James Bach 的《General Functionality and Stability Test Procedures》

6、ET 实践总体流程细节

这次我们说下第 3 大块的流程，那就是 Test Procedure，这里面有 5 大任务：

(1) 确定产品的目的

(2) 确定产品功能

(3) 确定潜在不稳定的地方

(4) 测试每个功能且记录问题

(5) 设计和记录一个持续性的验证测试

产出	**退出标准**
目的陈述 功能列表 潜在的不稳定和有挑战性数据的列表 产品错误和注意点 持续性验证测试	每个任务都完成了 每个问题或疑惑都被测试经理解决了或接受了 每个人员的产出都被测试经理接受了 有足够的信息可证明这个产品在功能性和稳定性上通过验收

6.1 Identify the purpose of the product

评审这个产品并确定这个产品需要提供的基本服务，如果可以的话，定义这个产品的使用者。写一段话简单说明下这个产品的目的和目标用户。

一些在目的陈述中用到的潜在的目的动词

创建，编辑

查看，分析，报告

打印

解决，计算

管理，控制

通信，交互

提供数据，提供介入，搜索

支持，保护，维护

清除，解决，使其完善

读，过滤，转移，转变

一些在目的陈述中对用户的能力的讨论

特殊技能，知识，能力，或无能

解决问题的能力

期望或需要

限制 (谁不会是这个产品的使用者)

产出	**退出标准**
目的陈述 问题/争论点	像上述那样完成任务 这个”目的陈述”必须都是经过需求方的确认 从这个产品的目的中选择对于用户来说的最重要的方面

常见的问题:

1. 为啥这个任务很重要？

如果没有对这个产品的目的有所了解，就不能很好的区分主要的和贡献性的功能。由于测试的大部分精力用在主要的功能上，所以这些区别是非常关键的。而我们不需要长篇大论，只要这个目的陈述里包含足够的信息来让我们跟踪重要的可作为主要的功能就可以了。

2. 该怎样来写这个目的陈述呢？

如果需求方提供了这个产品说明包含了用户的调研，那我们可以从这个开始先过一遍，写这个过程中，可以使用动词+名称的形式，比如’编辑简单文本文件’ 或 ‘一个用户无合法性的授权产生合法性的文档’。而且如果这个产品有些目的需要一些特殊的属性(相对于一般用户)，一定要在目的陈述中写清楚。而这些目的动词可以从之前说过的目的动词库中取(也可以丰富动词库)，这也可以帮助我们注意到一些容易忘记的一些产品目的。

3. 怎样区别目的和功能呢？

目的是和用户的需求相关的。功能是和这个产品所提供或产生的一些具体的东西相关的。有时候一个功能的目的和这个功能的名称是一样的，比如’打印’:打印就是这个打印功能的目的。大多数

时候，一个功能都提供了一个可以确定的更通用的目标。比如：一个文字处理器的目的就不是查找和替代文本；其实查找和替代就是编辑文本的一部分而已。编辑才是真正的目的。另外一方面，如果一个产品具有“超级搜索和替代”，那搜索和替代功能就可以是这个产品的目的。

6.2 Identify functions

1. 首先走遍整个产品并发现产品在做什么。
2. 对于所有的主要的功能做一个概要。
3. 记录一些有趣的或次要的贡献性的功能。
4. 对于我们不知道怎么去分类的或者觉得自己不能测试的任何功能，把它告诉测试经理。

一些查找出功能的途径：

查看下在线帮助

查看下需求方的调查问卷

查看组成这个产品的所有程序

查看产品所有菜单

查看所有的窗口

查看工具栏

查看所有对话框和小工具

右击查看所有数据对象，接口定义，窗口方框

双击查看所有数据对象，接口定义，窗口方框

查看产品所有为功能状态转换的选项设置

查看只有特殊的输入才能触发的功能

查看浸透在其他功能中的错误处理和恢复功能

功能分类:

主要的功能

贡献性的功能

产出	**退出标准**

功能列表 问题/争论点	像上述那样完成任务 每个定义的主要的功能对于产品目的完成都是必须的 已经合理的包含了贡献性的功能

常见的问题

1. 为啥这个任务重要呢？

对这些功能的罗列，我们可以对将要测试的功能做一个概要。当完成测试的时候，这个概要可以作为我们理解这个产品和测试范围的一个标记。这个功能概要对于测试经理或需求方来说都是一个很重要的记录，并且可以作为参考，防止他们(包括将来被其他的测试人员咨询)问我们做了什么，没有做什么。

2. 如果完全不知道哪些是主要的功能该怎么办？

报告给测试经理，不要随意的选择。测试经理会和需求方沟通，确定相关文档，或者会建议你去做什么。

3. 该以什么样的格式来记录这些功能呢？

要简单。使用 2-3 级的概要记录。有时一个功能并没有官方的名称或标记。也可以列出一些原始功能组。

如果要定义贡献性的功能，将它和主要的功能做清晰的区别。

例如：如下是一个关于微软书签的功能概要：

笔记.....

新增当前文章

删除

转到

评注

搜索所有.....

关于文章....

文章内容包括文字....

查找媒体

所有媒体

视频

图像

动漫

高级搜索

书籍

媒体

文章

6.3 Identify areas of potential instability

在探索这个产品的时候，注意到有些功能很有可能威胁这个产品的稳定标准。选择 5-10 个功能或功能组特别实施不稳定性测试。我们也许会选择贡献性的功能，如果它们特别容易失败，但主要的功能的不稳定才是我们最关注的。确认那些会引起产品不稳定的功能该如何测试。考虑超大的，复杂的或者有挑战性的输入。用列表将我们选择的那些不稳定的地方列出来，并且写出一些在测试他们的过程中使用到的数据或策略。

一些潜在的不稳定的因素:

与其他产品进行交互的功能

那些会消耗大量内存的功能

那些与操作系统交互特别紧密的功能

那些非一般复杂的功能

那些需要改变一些操作参数配置的功能

那些需要改变操作系统配置的功能

那些获取错误的和从错误中恢复的功能

那些替代了基本的操作系统功能的功能

那些涉及了多线程工作的功能

那些同时操作多个文件的功能

那些需要从网络上打开文件的功能

关于挑战性的数据的一些想法:

文档：大的文档；同时打开许多文档；或文档中包含许多不同的对象

记录：长的记录；很大数量的记录；或复杂的记录

列表：长的列表；空的列表；多列的列表

字段：输入大量字符，非常大的值

变化：新增和删除一些东西；编辑但没有保存或编辑

负载：使大量进程同时运行；大量进行批处理；在很短时间做很多事

无推断：在窗口随意点击；随意输入字符；输入无期望的值

异常和恢复：多次破坏进程；取消操作；使用错误数据触发错误处理

产出	**退出标准**
潜在的不稳定功能和挑战性数据列表 问题/争论点	像上述那样完成任务 这里每个定义的都是我们将要测试或已经测试的 对于定义好的不稳定功能，可以说明原因和来源

常见的问题：

1. 为啥这个任务重要呢？

我们可以关注最有可能出现不稳定的地方并进行测试，这是个不错的想法。而且有时候我们使用到的输入数据也很有可能触发不稳定。

2. 什么是不稳定性？

任何威胁稳定性标准的行为都是产品的不稳定性。常见的不稳定就是系统挂了。而功能失败和不稳定最基本的区别：就是后者功能有时候可以正常工作，但有时候又不能。这个功能是不可靠的，但又不是完全不可操作的。当一个功能在某些方式下可以正确工作，但又有不好的方面(破坏其他功能或产品)，这就叫不稳定。

3. 我们该怎么知道这里有潜在的不稳定呢？

我们不可能非常确定的知道。我们需要用到的是使用基本的线索去探索。我们在探索性的使用这个产品的时候，会有些感觉哪里会可能有潜在的不稳定。这是我们可以快速的测试来验证我们最初的猜疑. 同样我们也可以多去了解 SUT 的内部实现，甚至可以去看源代码，这样可以协助我们去

思考潜在的风险。假设我们怀疑某个特殊的功能存在不稳定的因素，因为这个功能非常复杂而且看起来会消耗大量的内存。 这时我们可以通过查看可见的输入和输出的复杂度和操作行为的变化来验证其功能的复杂度。 一旦我们确定了这个功能可能会不稳定，我们可以对这个功能进行覆盖和加压测试。一旦是测试稳定性，就可以不要使用普通的输入参数。

6.4 Test each function and record results

(1) 在时间允许内测试所有的主要的功能

(2) 测试所有已经确定的潜在的不稳定的功能

(3) 测试一个有兴趣的贡献性的功能

(4) 记录所有遇到的失败

(5) 记录所有遇到的产品的注意点。注意点就是指一些奇怪的，痛苦的，错误的行为，但从产品角度看又不是失败的。

产出	**退出标准**
产品的失败列表 产品的注意点列表 问题/争论点	像上述那样完成任务 任何的没有测试的主要的功能都已经告诉测试经理 对于所有的失败都有足够详细的信息来重现失败

常见的问题：

1. 为啥这个任务重要呢？

这个是整个流程的核心，也是实践性的测试。其他的任务都是帮助这个任务的完成。

2. 如果这是最后的一个任务，整个流程就更好吗？

仅仅在理论上。而实际上，测试本身总是就会透露出重要的信息，而这些信息是其他的任务不可能发现的。当我们完成了前面的所有任务，在我们实际的功能测试过程中，需要经常的重温以前的任务。

3. 为啥我们可以不需写出所有的测试设计和执行的用例呢？

尽管对于一个好的测试来说，把测试设计和用例都写出来是个好的基本点。问题是这里确实需要花太多的时间，而且会打断测试思路。如果我们停下来去写出每个测试的详细信息，我们就会写出很多，但实际执行的测试却很少。另外，只要我们能够给出已经测试的，怎样测试的大概情况，

这个详细信息写出来就没有那么必要了。请注意了，这里说的是测试的详细信息，如果可以快速的记录你的测试点或测试变量也是非常好的，可以后续做回归测试和测试经验交流分享之用(这个需要长久的经验的积累才能达到平衡，因为要记得简单和准确，并不是一件非常简单的事情)。在这个流程中，需要写出来的测试只有一个，那就是能够表现我们已经测试过的小集合，也就是持续性测试。

6.5 Design and record a consistency verification test

选择这个产品的最主要的功能去实践并确保这个功能行为在其他的平台和配置中都是一致性的。

持续性验证测试需求：

(1) 这个产品的测试必须经过不同的测试人员在同一个系统配置下进行，并且所有测试人员得到的结果都一样。

(2) 覆盖所有的重要的主要的功能

(3) 使用一些复杂的数据去测试，包括一些特殊的文件名称和文件路

产出	**退出标准**
持续性的验证测试 问题/争论点	像上述那样完成任务 测试已确定的所有的需求

持续性验证测试=兼容性测试+冒烟测试

常见的问题：

1. 为啥这个任务重要呢？

一旦我们完成了前面的那些基本功能和稳定性测试，在最后的关键阶段我们有时候需要对那些功能来一个比较简单的重(新)测(试)。这里面的持续性验证测试就是定义这个活动的。对于那些在不同的平台和配置下使用产品的(兼容性要求高的)的测试，早点定义这个活动是很重要的。

2. 这个活动是不是有点像冒烟测试呢？

持续性验证测试确实有点像冒烟测试。但有一点不一样的是持续性验证测试需要足够的精确度说明每次重复测试都是正确执行过了。

3. 这种测试需要深到什么程度呢？

一般情况下，对于每个主要的功能的一次正常使用上，我们至少测试一个案例。另外我们可以考虑去测试那些可以很快表明某个功能的质量是否正确的场景或业务流程。

参考自 James Bach 的《General Functionality and Stability Test Procedures》

实践篇

1、ET 的实践结果分析

ET 实践项目：XX1 项目

ET 实践时间段：09/12/15---09/12/21

ET 实践人：季哥

此 ET 实践结果分析包括如下几个部分：

1. 简要说明什么是 ET
2. ET 测试的范围
3. 为何要做 ET
4. 什么时候开始做 ET
5. 怎样做 ET
6. 做 ET 时注意什么
7. ET 产出了什么
8. ET 发现了什么样的 bug

1. What1：什么是 ET？

这里的 ET 定义就是实践与 XX1 项目的定义：就是在完全不熟悉项目业务需求的基础上，采用边学产品知识，边测试，通过一些手段来操作产品，使其暴漏出一些隐含的问题。其测试执行思路与测试设计思路是同时进行的。一个很明显的 Freestyle ET 方式。

2. What2: ET 测试了什么？

由于大部分项目存在一些共性，ET 测试的范围一般是主要的功能的实现，再加上主要的功能中隐含的一些潜在的风险，例如超长输入引出的系统错误等。具体可参见 ET 实践流程。

3. Why：为啥要做 ET？

至于做 ET 实践的原因多方面：

---------目前项目测试人员的功能测试手段太单一

---------目前第 3 轮测试发现的 bug 率以及投资回报率很低

---------为了质疑目前测试部三轮测试的流程规范

---------国外已经有了比较成熟的 ET 理论和实践经验

---------创新并实践前段时间 ET 的理论学习

4. When：什么时候开始做 ET？

根据 ET 测试的方式和目的以及时间安排，可看出 ET 并不是为了发现主要功能的流程问题。所以特别需要在相对稳定的系统上做 ET，这里有两个好处：一是由于 ET 测试人员没有项目测试人员对需求了解深入，对于主要功能的流程问题没有项目测试人员发现那么及时以及深入。二是在稳定的系统上做 ET，有益于发现项目测试人员的盲点，以及发挥测试的极限测试手段，同时也有益于 ET 测试产出的效果。所以在 XX1 项目 ET 实践过程中，是在第二轮测试的最后一天开始 ET。一般是在安全测试通过后。因为安全测试的 bug 修复后会引发比较多的页面 bug，此在一定程度上会影响 ET 发现较严重的 bug 数量。

5. How：怎么做 ET？

根据国外 ET 实践理念，采用 Session 来进行测试范围的确定(具体请看 ET 的管理)，下面是简单的一些说明：

第一步: 大概花 1-2 个小时时间看 PRD 和原型(了解目的和产品背景)。

第二步: 大概花 1-2 个小时时间确定下有哪些主要的功能模块和贡献性的功能模块。

第三步： 与项目组测试人员沟通哪个功能模块发现 bug 最多，哪个功能模块发现 bug 最少，哪个模块存在风险比较大。

第四步：根据前几步情况和参加 ET 的时间段来确定有多少个 Session，并指出每个 Session 大概花多长时间。一般是 1.5-2 个小时。就淘宝而言，一个 Session 大概是 2-3 个 UC 的情况。

第五步：制定 ET 测试计划，包含所有 Session 的名称和测试时间以及缓冲情况。

第六步：根据 ET 测试计划，边学习产品需求，边测试。发现问题立马记录问题描述。最后发送 ET 测试报告。

第七步：与项目组测试人员沟通 ET 的效果以及该产品存在的风险，从用户易用性角度给该产品总体评价，同时跟踪确认 bug 的 fix 情况。

6. Strategy：ET 测试的时候怎么考虑？

在做 ET 过程中，有一个基本的原则就是以最少的学习时间来获取最大的学习成果，也就是在测试执行时，由于系统是个相对稳定的系统，其主要功能的流程问题已经不大可能不存在了，这时 ET 测试人员需要以最少的时间去了解这个产品的某个需求，然后去发现这这个需求(一般对应的是某个

Session)的隐含的问题。这里一定要注意不要花很长时间去了解某个复杂业务的具体实现过程，然后去测试，这样时间投资回报率会比较低，违背了快速测试的特点。

要求 ET 测试人员在很短的时间内需要判断这个需求需要花多少时间去测试，大概会隐藏什么问题。然后发现一个可挖掘的需求，去多尝试操作去测试，直至发现问题。这些在 XX1 项目实践过。

ET 过程中，尽量去关注一些很细节的部分，多使用一些极限测试的手段，比如超长字符，非法字符，异步编辑等。

ET 过程中，如果被一个需求的特殊性 block 了，也就是 ET 测试人员尝试了很多次都没有成功进入下一个操作流程，则这时需要立马与项目组对应测试成员沟通，寻求帮助。也许该成员的一句话就可以搞定这个问题。在 XX1 项目实践过程中确实遇到过这种情况，要敢于提出问题。

ET 过程中，发现一个疑似问题，立马记录其问题描述，在每天的 ET 测试时间完成后，与该 Session 的项目组测试人员沟通这些疑似问题是否为 bug，并邮件报告每天测试的 Session 的 bug 描述以及优先级，特别是该功能在用户体验上的价值。

ET 过程中，需要 ET 测试人员**全神贯注**的进行边学习产品，边测试。在一个 Session 测试过程中，不能受到其他的干扰，完全沉浸在测试和破坏这个产品的紧张之中，由于需要不断的变化测试思路去挑战正在测试的功能，这时时间就是质量，时间就是生命。

7. Result：ET 产出了什么？

对于 ET 的核心价值就是花最少的时间得到最大的回报。该项目实践 ET 测试的数据如下：

A	B	C	D	E	F	G
时间跨度	**总ET时间**	**总bug数**	**P1bug数**	**P2bug数**	**P3bug数**	**P4bug数**
5天	17 hours	40	0	9	25	6

ET 测试发现 bug 类型分析：

A	B	C	D	E
页面显示错误	**字段校验错误**	**数据显示异常**	**页面提示信息错误**	**功能未实现**
7	10	6	6	11

这里功能未实现的 bug 类型里面一大部分是一些页面的按钮或链接功能失效。

8. Analyze：ET 发现什么样的 bug

对于 ET 测试发现的 bug 类型做一些个人分析，相信很多人对于这个比较感兴趣：

第一：一般情况下，ET 测试的目的不是为了发现正常流程下的主要功能 bug，特别是 Freestyle ET 方式，其实践的时间点在第二轮测试结束后

第二：ET 测试过程中，会变化测试手段去测试，更多的是用户测试和极限测试和交叉测试，也就会发现很多这些手段产生的 bug

第三：ET 测试过程中，使用的一些测试手段决定了发现的 bug 类型，常用的测试手段有：边界值测试，极限测试，用户测试，菜单浏览，域测试，组合测试。

第四：ET 测试过程中，需要在学习业务的同时，不断的变化上述的测试手段去测试(关键点)。

第五：从产出上来看，使用了极少时间去进行 ET 测试产生的 bug 率是比较高的(平均每个小时产生 3.5 个 bug)。对于项目组测试人员来说，其单位时间内产生的 bug 率应该是比较低的。因为其投入包括熟悉需求；测试设计，3 轮测试执行等。具体官方数据个人不是很清楚。可参照 XX1 项目某测试人员的数据：投入时间 267 个小时，发现 124 个 bug；则平均每个小时产生 0.47 个 bug。(注意：该数据不是直接判断标准，只做参考)

时隔 5 个月之久，笔者又实践了一个项目，根据之前的实践经验，对于 XX2 项目进行两天的 ET 测试，产出的结果也就是 bug 产出的类型和数量与第一次项目实践并无大区别，这里就不列出具体数据，但还是有一些其他的心得，也就是在做 ET 实践时印象较深的地方：

(1) 如果该产品的用户体验较差(或是使用过程中无很清楚的提示操作去引导用户去使用该功能或产品)，这样在时间非常有限的 ET 测试过程中，会遇到较多困惑。这种情况，强烈建议开发人员和测试人员和 PD 坐在一起，以便于交流，能够快速克服疑惑，另外如果有比较大的可用性差的设计，测试人员可以考虑记录为缺陷，并跟踪解决。

(2) 同样如果该产品流程非常复杂，且需要非常大的数据准备，这样在时间非常有限的 ET 测试过程中，测试内容比较有限，且大量的时间花在数据准备上，会减弱 ET 的创造性。

(3) ET 测试过程中，由于处在测试执行的后期阶段，对于主要功能的问题几乎不存在，这里使用 ET 发现的 bug 的严重程度和类型会有所变化，也就是说发现一些 bug(个人经验的不同会有所不同)不是需求的 bug，而是程序本身的 bug。比如某个查询项，正常情况下，用户输入 4 位数字进行查询，结果正确；但如果输入 20 个数字，进行查询，会出现系统错误。也就是我们测试不仅仅要保证正常输入得到的结果正确，更要保证非法或异常输入程序也能进行合理的处理。

(4) ET 测试处在项目测试后期，不是为了发现 bug 而去测试，而是更多的去探索和使用该产品，因为 ET tester 本身不知道接下来要去测试什么，更不知道这样测试能否发现 bug，同样也不会刻意去发现高难度或很严重的 bug。ET tester 只会根据自己看到的产品的 Response 来进行测试，至于能够发现什么类型的 bug 不必刻意追寻，否则会影响测试的高度注意力。这里不是说测试执行的时候不需要目标，目标是找到测试过程中提出的问题的答案，后面的漫游测试模型可以帮助大家确定自己测试的目标。

(5) ET 测试完成后，如果 Bug 较少的话(原因很多)，无法很全面的说明此次 ET 提高了测试覆盖率，

由于 test idea 的产生和执行都是同时执行的，而且并没有刻意去记录这些测试用例，但会记录一下 Notes 和疑惑。这方面对于 ET 测试的效率和知识沉淀的传承是很不利的，目前并没有特别好的解决办法。但测试结束后，还是建议大家及时开会总结，搞清楚这次 ET 做得好的地方，不好的地方，为以后的 ET 做出更好的指导和总结。

2、ET 和 ST 的生产率比较

之前我们说了很多 ET 的过程以及怎么去做 ET，也说过 ET 和 ST 的关系，是怎么来用在我们的项目过程中的，但很多人对于 ET 能发现的 bug 类型有很大的好奇，认为 ET 能发现什么样的 bug，是不是很严重，是不是很难发现，是不是效率很高，这次我们就来说说，ET 和 ST 之间的生产率比较(这个只能作为参考，不要纠结于这个具体数据)。

首先说明的是这个生产率比较是国外某个大学的研究成果，有一些基本的条件，比如实验者，选择的实验的产品，实验者的经验等等。这里不详细说明这个，也不说明过程（其中遇到很多的挑战），只说下试验结果，让大家了解下。

这里只说一个关键的条件（对于 bug 而言，研究方在试验之前就分开了已知和未知的 bug）

总体过程：

Phase	Group 1	Group 2
Preparation	对于 feature A 写测试用例	对于 feature B 写测试用例
Testing Session 1	对于 feature A 做 ST	对于 feature A 做 ET
Testing Session 2	对于 feature B 做 ET	对于 feature B 做 ST

对于 Testing Session 再说明下：

Phase	**时间**	**描述**
Session Setup	15 min	看产品的介绍和指导书或做些基本的测试准备工作
Functional Testing	90 min	专注于 ET 或 ST 和报 bug
Bug report	10 min	对于所报 bug 和 log 写总体 report

（1） 发现的 bug 总数

Testing approach	Feature set	Bug 总数
ET	A	44
	B	41
	Total	85
ST	A	43

	B	39
	Total	82

这里面的结论是：对于已知 bug（研究方故意注入在产品中的 bug）来说，使用 ET 或 ST 方法在发现 bug 总数上没有区别；但 ET 却可以发现更多的未知 bug。

（2） Bug 发现难度，类型和严重程度

Mode	ET	ST	ET/ST	Bug 总数
0=easiest	120	93	**129%**	213
1	327	320	102%	647
2	89	75	119%	164
3=hardest	20	15	**133%**	35
Total	556	503	111%	1059

这里的结论是：ET 发现更多的 bug 在各种发现难度上，相比较 ST 而言。

Bug 类型：

Type	ET	ST	ET/ST	Bug 总数
Documentation	8	4	**200%**	12
GUI	70	49	**143%**	119
Inconsistency	5	3	167%	8
Missing function	98	96	102%	194
Performance	39	41	95%	80
Technical defect	54	66	82%	120
Usability	19	5	**380%**	24
Wrong function	263	239	110%	502
Total	556	503	111%	1059

这里的结论是：ET 在 GUI 和 Usability 这 2 个类型上 ET 有比较大的优势，但在 Technical defect 上，ST 比 ET 要好一些。

Bug 严重程度：

Severity	ET	ST	ET/ST	Bug 总数
Negligible	23	14	**164%**	37
Minor	98	74	**132%**	172
Normal	231	203	**114%**	434
Severe	153	160	96%	313
Critical	51	52	98%	103
Total	556	503	111%	1059

这里的结论是：ET 在严重程度较小的上面有比较大的优势，其他无较大差别。

（3） 错误 bug 的报告

Testing approach	Feature set	平均 Bug 数
ET	A	1.396
	B	1.191
	Total	**1.291**
ST	A	1.564
	B	1.867
	Total	**1.767**

这里的结论是：相比较 ST 而言，ET 报告了较少的错误 bug。

看到了如上的结果，个人认为我们很多人都会有一些问题，研究者也想到了以下问题：

问题一：已经写好了的测试用例是怎样影响发现 bug 的数量呢？

之前说的那些是基本条件，其实还有个重要的条件，细心的人肯定可以看出，那就是做 ST 的那些实验者花了 7 个小时用在编写测试用例上面了(也就是上面说的 Preparation 时间)，而做 ET 的实验者根本没有花时间在这上面，这可以看出，当没有时间写测试用例的时候(也就是需要测试人员做快速测试的时候)，ET 明显更高效。

问题二：已经写好了的测试用例是怎样影响发现 bug 的类型呢？

我们已经从 3 个维度（严重程度，类型，发现难度）来说明了，认为测试人员不需要测试用例就可以发现很多 User interface 和 Usability 相关的问题，但考虑到严重程度，测试人员不需要测试用例会发现严重程度较低的问题。

尽管我们得到了这些结果，但仍然有其局限性，实验者的测试经验会影响其测试用例的质量和做 ET 测试执行时候的质量。

以上我们可以看到仅仅关注与测试设计技术并不能覆盖很多重要的方面，而这些方面只有在手工测试执行的时候发现的。还有一个方面就是 ET 方法更能提高测试人员在测试执行的时候的创造性和创新能力。

笔者使用 Freestyle ET 方法实践了两个项目，其产出数据还是比较匹配上面的研究结果的，只是个人认为在 Wrong function 的 bug 类型上面 ET 反而会发现较少的 bug，而不是更多。还有就是误报 bug 这方面，个人认为 ET 之后会产出很多疑问而并非是 bug，只是嫌疑 bug，而需要与相关的人进行确认，这样就会减少误报率（详见 ET 项目实践结果分析）。

参考 Juha Itkonen, Mika V. Mantyla and Casper Lassenius 《Defect Detection Efficiency: Test Case Based vs. Exploratory Testing》

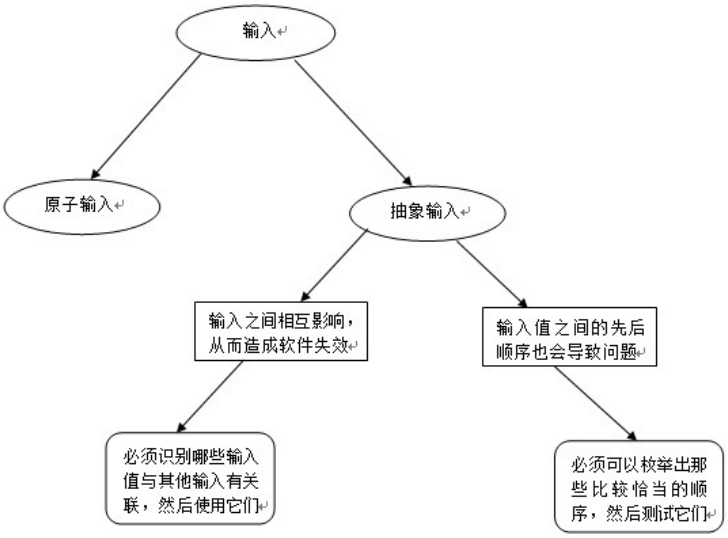
3、ET 方法的联想输入模型

我们前面说了很多 ET 怎么去测试的方法，ET(探索式测试)只是一种方法，是相对于 ST(基于脚本的测试)而言的，我们知道 ST 方法里面有个很重要的技术就是测试设计技术，使用到很多测试设计方法，比如使用等价类分析，边界值分析，因果图分析，判定表等等。这些测试设计方法可以在一定程度上可以保证测试的覆盖率。那 ET 要与 ST 去比较，也当然也有自己的测试方法去覆盖一些潜在的问题。下面将说说这些测试方法是如何应用在 ET 测试执行过程中。是如何让一个测试人员在一个完整的产品中实行 ET 测试的。

之前也说了 ET 和 ST 的关系，在一个项目实际测试过程中，是如何应用 ST 和 ET 的测试模型，不同的模型中怎么来做 ET，也就是 ET 中的测试方法也有不同。但其核心方法是一样的，如何快速的在 ET 测试中使用多种测试手段，如何根据产品的反馈快速的进行下一个用例的设计和执行，这些方法是通用的。为了让大家能够更好的理解一个优秀的 ET 测试人员是怎么对自己需要测试的模块进行测试的，其测试思路的变化是怎么形成的，特别举几个例子来详细说明下。

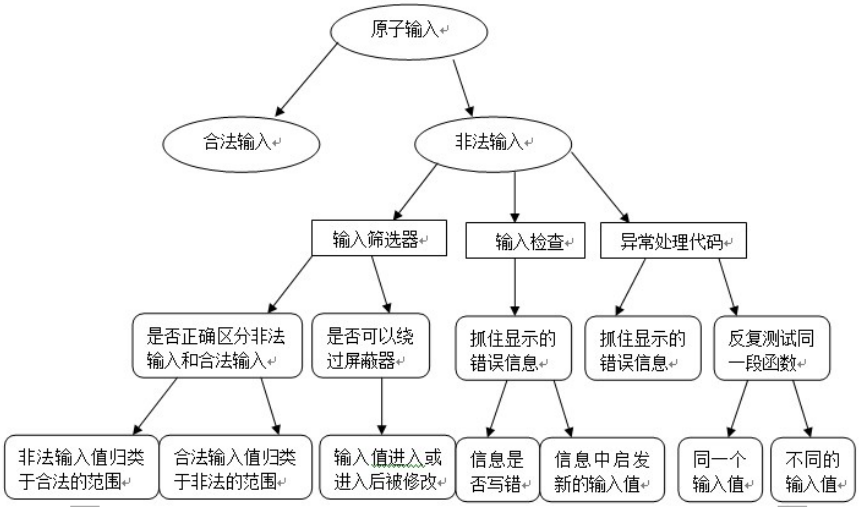
首先需要说明的是，这个快速测试的方法是基于如下一个观点：我们把测试工作简化为先在所有输入（或运行环境等）的全体集合中选择一个子集，然后在输入时使用选中的子集，最后通过推理认定是否这些输入已经足够多了。由于输入是无限多的，我们唯一的希望就寄托在正确决定我们需要变动的输入上。如果我们选择了正确的输入，就会发现有效的 bug，产品质量也就会提高。基于这些理解，这边整合了一些关于输入的测试方法，怎么在输入和输出之间发挥出好的效果，怎么更好的理解输入在测试中的作用。

请看如下的“联想输入”(LI)模型分类：



原子输入：简单不能再简单的输入，属于单个的事件，比如单击按钮，输入整数 4 等

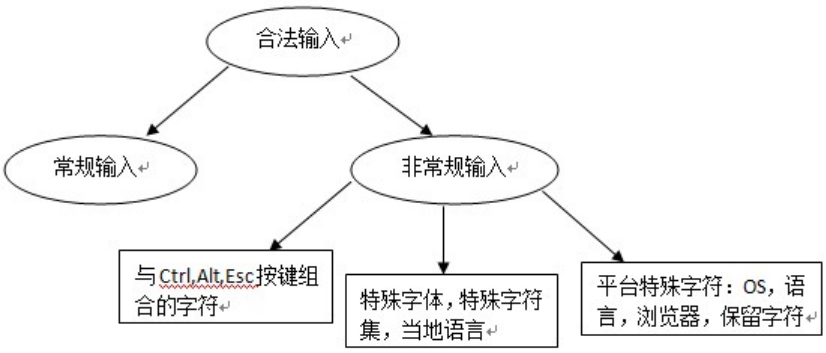
抽象输入：将有些相互关联的原子输入合并在一起输入，就成了抽象输入，比如输入正整数，这里面包含了原子输入 4 或 256 或 2048 或 32768 等



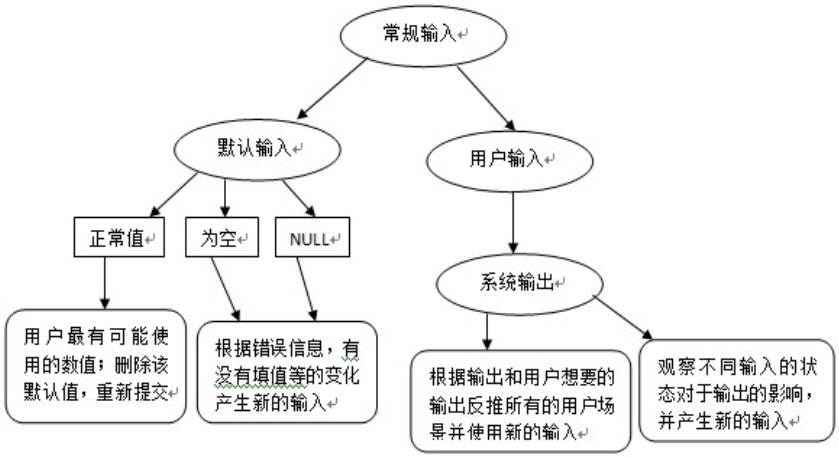
输入筛选器：用于防止非法的输入值被传递给应用软件的功能代码，不会产生错误信息

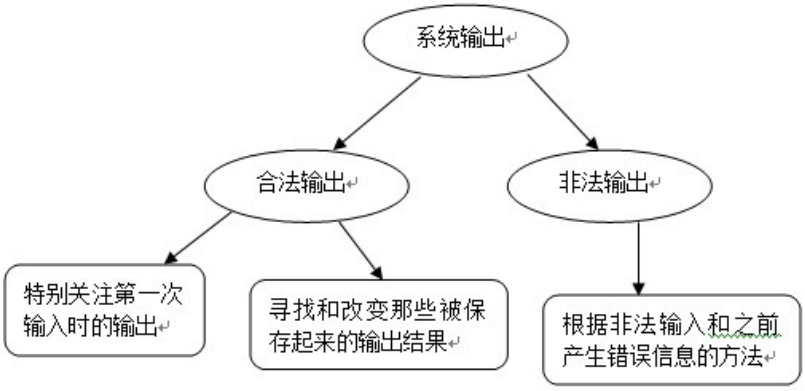
输入检查：属于功能代码的一部分，类似 IF,ELSE 语句来实现，产生特定的错误信息

异常处理代码：把整个例程(routine)当一个整体进行异常处理并产生通用的出错信息



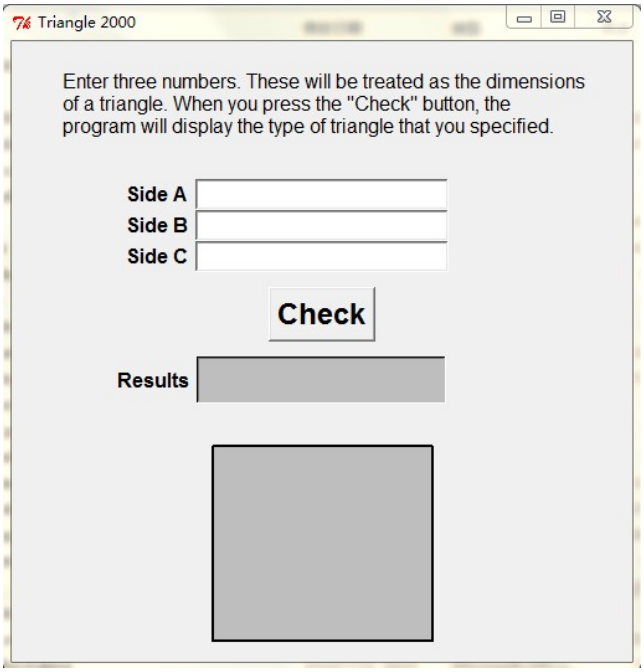
非常规输入只在那些比较特殊的情况下才会发生，这方面需要多了解些特殊字符。





如果大家都可以很好的理解了上面说到的“联想输入”模型，又了解我们在测试一个应用程序的目的时，我们就可以开始对于某个应用程序进行探索式测试了，这里再次强调，对于一个应用程序测试的目的不一样，其使用探索式测试的具体模型就有所不同。就拿功能测试来说，上一篇的文章也提到了探索式测试在发现业务功能性的问题上，没有占优势，ET 主要是靠使用应用程序的功能来理解和了解到隐含的业务功能或规则，并根据这些功能和规则设计下一步的测试，由于之前我们无法全面使用且推导出该应用程序的所能具备的所有业务功能或规则，所以我们要全神贯注的使用被测软件，并发现所能看到的或潜在的问题。

为了更好的应用上面说到的输入模型，也更好的体会 ET 的基本过程，我们来实战下吧：



这个是特别经典的一个应用程序(判断三角形)，这个本来应该在培训或现场来应用输入模型(特别的是测试人员在使用某个输入后，怎么采用新的测试用例来验证新的测试点的)，那在本文中，就只能看看各位怎么来初步应用该模型了，其实细心的人也应该看到，输入模型里面也用到了如果应用程序是这么反应的，那就要去更多的关注测试其他的什么。

现在给测试人员的任务就是：在 15 分钟内(期间可以问任何关于程序的问题，这个也是 ET 大师 training 我的时候用到的)，尽自己最大能力找到这个应用程序的所有 bug。这边就大概说下怎么来应用“联想输入”模型在这个 SUT 的 ET 中(重点是应用模型，而不是正式的 ET 实践流程)：

（1） 首先我们好好理解前面的一段文字，这个其实就是使用说明书，给用户看的，测试人员进行 ET 需要站在用户的角度去使用该应用程序(确定 knowledge)。

（2） 分析出原子输入：Side A，Side B，Side C，Check button

（3） 选择一个原子输入项：Side A，猜测其合法输入(如果无法得到这个 Oracles)是什么，暂时假设为大于 0 的数字。

（4） 在这些合法输入里面找出哪些是常规输入，非常规输入；对于 Side A 来说，就是大于 0 的整数，还有其默认值(为空)，还有非常规输入，考虑带有小数点的数字，是否支持 16 进制(带有 a-f)，是否支持+,-,*,/等特殊字符

（5） 在考虑非法字符输入的时候，通过 check 运行，来确定该应用程序对于非法字符是怎么处理的，在这里就是输入检查情况，提示信息为“Not a Triangle”;这里还需要考虑字符(a-z,A-Z)在数字前面，还是在数字后面

（6） 在对于所有原子输入项按照模型分解完后，就分析出抽象输入，在本例中，就是 Side A 和

Side B 和 Side C 之间的关系来判断是否是三角形了，如果是三角形，是什么类型的三角形了。首先找到这些关联关系，还要确定每个关联关系对应的正确输出是什么

（7） 识别出所有关联关系后，就使用抽象输入来验证这些关系后的结果，这里还可以多回忆下在选择某个原子输入的时候，其输出是什么；比如这里我们可以看到 Side A 输入为 f33 或 33f，输出结果是一样；这时如果我们 Side A，Side B，Side C 都输入 f33，发现输出是正确的；但这时如果我们把前面的 f 去掉，都是 33，发现输出也是正确的(等边三角形)，但如果我们在某一个原子输入项后面加个 f 呢，也就是 Side A= 33f，Side B，Side C 还是 33，这时就发现输出是错误的了(还是等边三角形)，还可以改变其他的原子输入项进一步验证

（8） 再通过不同的顺序来进行 Side A 和 Side B 和 Side C 的原子输入，判断之间是否存在先后顺序关系，从来得出不一样的结果，在本例中，不存在这个关系

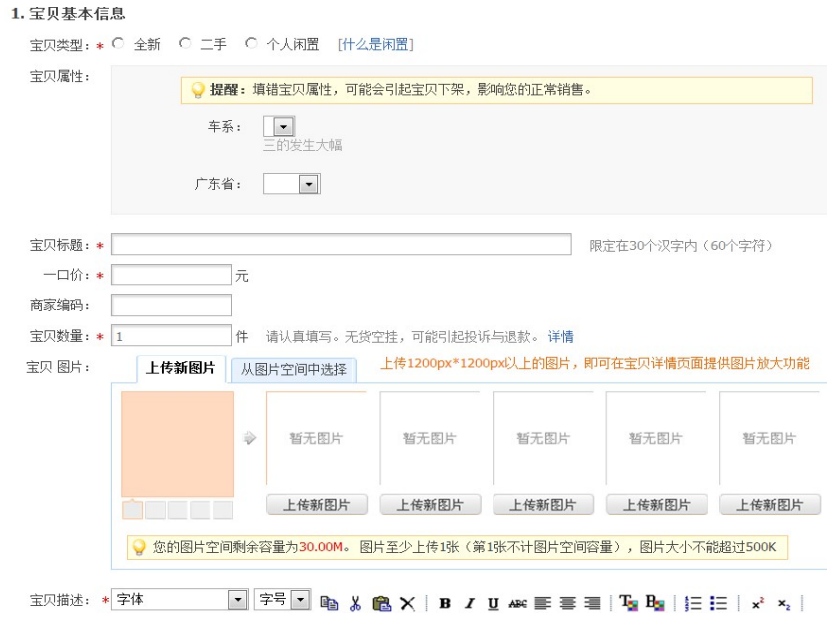
（9） 找到应用程序对应的所有输出，包括非法输出。本例中，有 2 个输出结果：Results 和下面的图显示。这 2 个结果之间的关系也要考虑，是否有必然的一致性，找到 Results 对应的所有结果：非三角形，一般三角形，等腰三角形，等边三角形。选中某一个输出，反推出应采用的输入，比如选择等边三角形，可以采用输入:1,1,1 或 222,222,222 或 1234567890,1234567890,1234567890 等; 还要考虑输出的持续性和间断性。就是多次输入都产生同一个输出，另外一个就是多次输入都产生不同的输出(这里主要检验是否保留中间结果)

（10） 考虑抽象输入的时候，需要不断的变化原子输入的值，这里强调多考虑最大值的表现形式，对于本例来说，你可以输入很多很多数字或字符，无法确定最大值，这时还需要更多输入以便可以探索到这个原子输入 Side A 的可接受的输入值，根据输入框的接受范围，当我们输入很大的值时或超长的字符时(大于 10W)，再次运行 check，发现该应用程序崩溃了

以上是粗略的介绍了应用“联想输入”模型来进行 ET 实战，由于不能演示操作该应用程序，无法非常完整的应用该模型测试这个应用程序，也不能很好的显示如果程序给出某个结果后，又怎么想出下一个测试思路的，但上面还是给出了一些想法。希望大家能很好的理解该模型，并真正的应用在项目的应用程序中。可以看到要能完全的应用 ET 在项目的测试过程中，我们需要更多的模型去支撑(还有场景/漫游模型等)，只要我们能够熟练的练习和应用这些模型，我们就不用花太多的时间去写测试用例了，完全在测试过程中，就能控制整个质量和进度，但从输出中总结出业务功能或规则并制定新的测试思路，确实需要某些经验的积累。

我们需要更多的练习，怎么去应用模型，我们可以经常找到这些案例去练习我们掌握模型的程度，比如淘宝网的邮箱注册页面(应用过程中会逐渐学会了测试设计方法)：

或者淘宝卖家的发布宝贝页面：



如果我们对于上面 2 个基础功能进行了很好的 ET 测试设计(ET 测试执行之前能想到的测试思路)，那么就完成了 ET 测试第一层次的阶段。这里需要说明的是在测试单个功能时，还需要考虑到运行环境的变化(驱动程序，代码，文件，路径，设置等)。我们把每个 Session 具体的测试模型建立后，ET 测试就是试图把制定计划，进行测试，重新制定计划等多个过程有机的结合起来，虽然每次只前进一小步，但这每一步都由本次软件运行状况，通过输入表现出来的各种行为和蛛丝马迹来确定的。

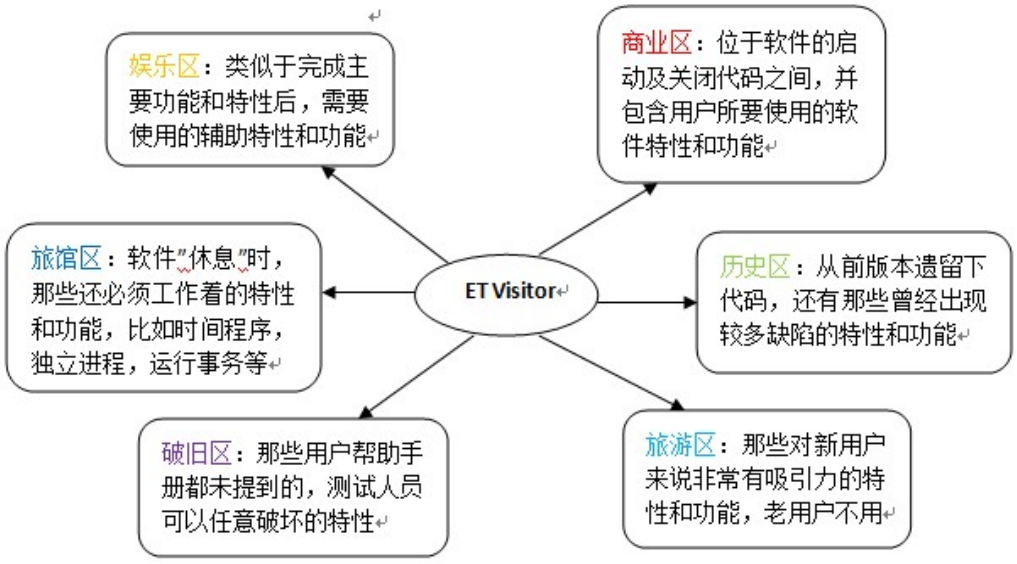
我们可以看到，每一个确定出来的 Session 都可以使用到该模型进行测试，但 Session 和 Session 之间的测试呢，整个 Charter 的测试策略呢？下面将会说另外一个模型来解决这个问题。

4、ET 方法的漫游测试模型

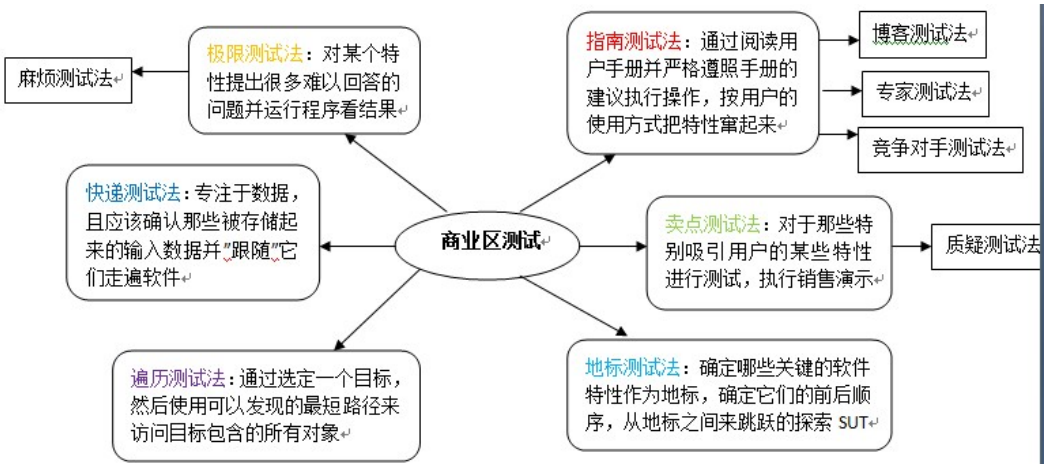
前面我们是对于某一个单个非常基础的功能进行 ET 测试给出了一个简单的模型(会潜移默化中学会我们 ST 时的测试设计方法)，但实际工作过程中，我们项目的功能复杂，模块多等多个特点，我们需要对于多个功能的之间的组合关系，主要流程进行探索式测试，这里又需要用到新的模型。这里会简单的说明下这个模型，但不能很好的举出例子进行说明。虽然之前也说过，主要的功能流程问题由 ST 来保证比较好一点，ET 也可以起到自己的作用。但这里并不是说 ET 不能在 SUT 上操作这些流程，使用这些主要的功能，也不是不能发现这些 bug。那 ET tester 是怎么来操作这些主要的功能和流程的呢？是怎么去抓住这些关系呢？

了解了之前说过的 Session 管理 ET 的话，这里可以想到该怎么去分解 Session，Session 的力度是怎样的。一个项目所有的功能也就是特性，我们完全可以整理出来，且一般情况下，一个特性很少能完全独立于其他特性。因此，单独测试某个特性(上一篇说到)可能无法发现那些只在特性互相集成时才会表现出来的 bug。

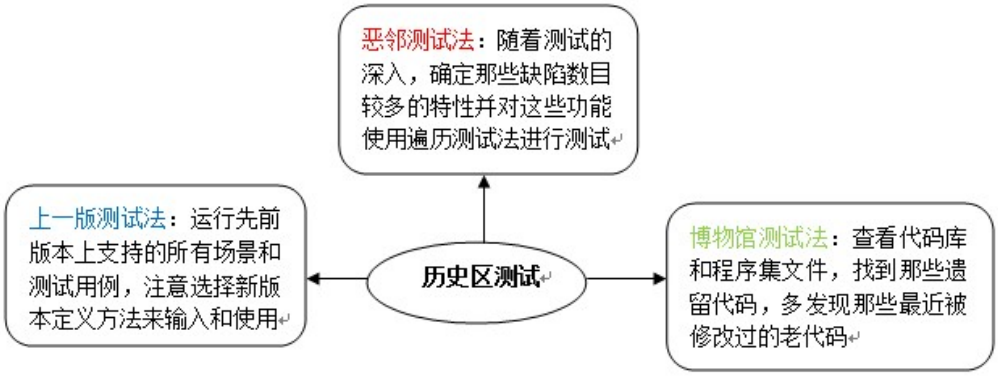
这里说到的模型是漫游模型(James A. Whittaker)的首创，是以一个游客的身份去测试 SUT，就像一个旅游者对于一个城市进行探索性的旅游一样。请看下面的模型：



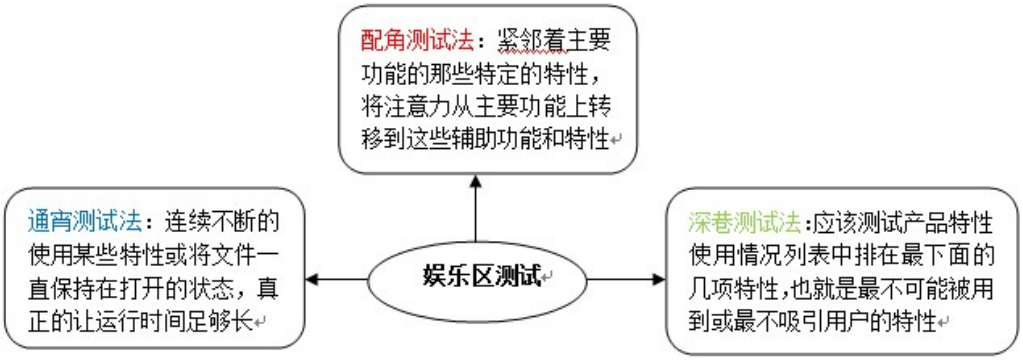
下面看看对于每一个区域，该使用什么方法来进行测试，下面是商业区测试：



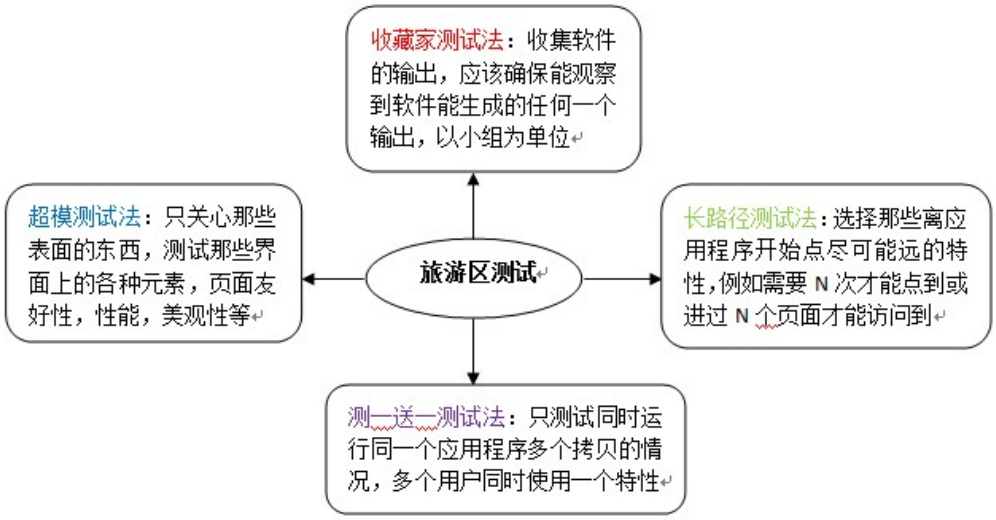
下面是历史区测试：



接下来是娱乐区测试：

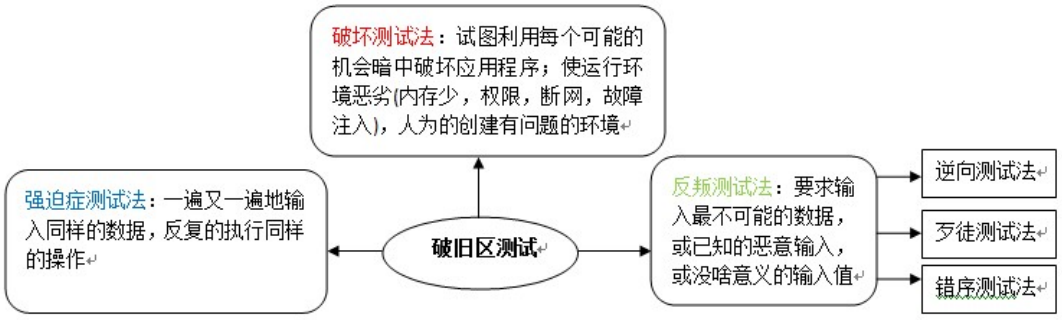


接下来是旅游区测试：



接下来是旅馆区测试：

接下来是破旧区测试：



现在我们六大区域的测试方法都已经说明了，细心地人可以看到，这里的方法很多都是基于我们发现的 bug 类型来反推出经常会出现问题的区域，这里的测试策略让大家在测试相关程序的时候，需要多多考虑重点测试什么，哪些方面需要花更多的精力。

如果我们觉得我们可以理解上面说到的作为一个旅游者来进行一些区域的访问，也就是理解测试人员在使用我们的 SUT 时，需要访问到那些地方一样。我们可以拿一些非常经典的一些软件做练习，怎么使用上述的那些方法，比如我们常用的 FreeMind, AMT，Rational，Amazon 等，这里需要说明的是，只有我们通过大量的练习我们应用模型的方式，多思考，多总结应用模型的方法，甚至可以修改和完善此漫游模型。我们可以在一个项目测试过程中，反复的运用上面说的那些测试方法，最关键的是我们需要在我们的测试场景中加上一些变化，一种途径就是通过如下几个大的方式（这

里简单说明下，在用户正常使用场景中）：

- 插入步骤（增加更多数据，使用附加插入，访问新的页面）
- 删除步骤
- 替换步骤
- 重复步骤
- 替换数据
- 替换环境（替换硬件，替换容器，替换版本，修改本地设置）

另外一种途径就是应用漫游模型中的测试方法在场景中，下一章详细说明。

我们现在需要的更多的是 practice，让每个测试人员都能发挥自己的主观能动性，从测试过程中产生更多的测试思路，不断的改变用户的使用场景，更多的覆盖 SUT，更全面的提升手工测试的兴趣，根本上传承这些测试经验，让产品质量得到真正的提升。

其实这个漫游模型还可以帮助我们在测试组成员直接分配测试任务。随着测试人员对上面说到的测试法逐渐熟悉，也应用的非常有效，比如针对这一类的 bug 哪种方法最有效，各类测试法之间有哪些相互兼任的特性。这样可以避免测试人员遗漏某种测试类型，还可以帮助测试人员吧应用程序的功能和适合这些功能的测试技术相匹配。

上面说到的测试方法很多，但在实际的 ET 过程中，我们需要结合 SUT 的特点，此次测试的目标，来选择合适的方法。我们可以考虑：SUT 的特征和属性，具备哪些可能存在的错误；选择何种测试方法有助于发现此类错误；本次测试的具体目标等等。这些都需要在 ET 的测试策略中考虑的，这些都是测试 SUT 的思考方法，这也再次说明了前面说过的，ET 只是一种测试方法，所有的测试技术都可以应用在此测试方法上，如果可以的话，我们可以在这个测试方法上创建一个新的测试技术都是有可能的。

这边多说几句关于这些漫游测试方法的来历：Bug -> Pattern -> Tour Testing。首先分析具体的缺陷（Bug），发掘出它们的共性（Pattern），然后提出若干种能够发现此类缺陷的“漫游测试”（Tour Testing）。大多数漫游测试方法都针对一类（或几类）典型的缺陷。要熟练使用漫游测试，要理解漫游测试方法针对的缺陷类型。

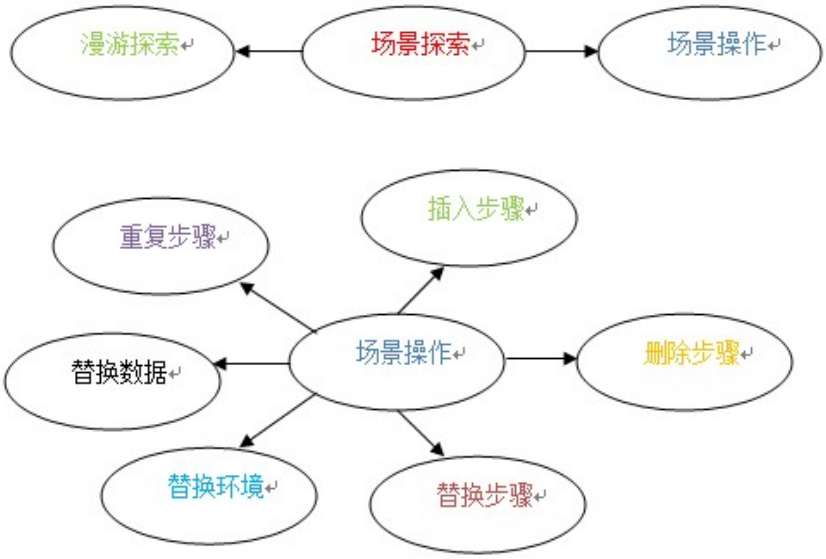
5、ET 方法的场景探索模型

上面说到的联想输入模型是在 Freestyle ET 的方式中较易适用，不知大家是否还记得之前说过的 ET 和 ST 的组合模型呢？而且现在业界大部分都认为 ET 是 ST 很好的补充，那么不管你怎么想，不可否认的是 ET 的价值所在，而且想最大化这个 ET 价值。那么这次我们要讨论的是在不写 TC 的情况下，我们是如何将场景测试加入探索性因素的。

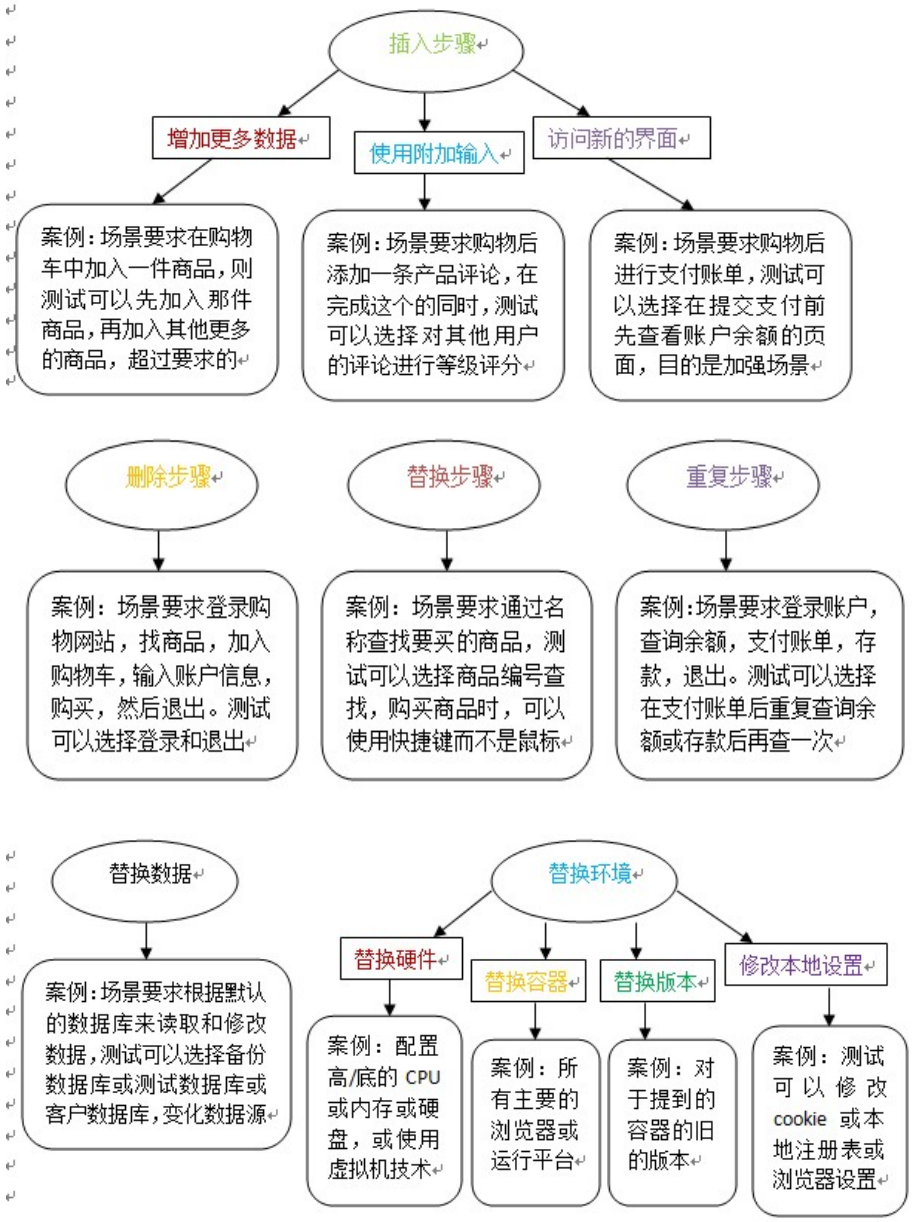
简单说下场景测试：测试人员设计真实的用户使用软件时的场景，可以包含多个测试用例，一般只描述了基本的功能，但有价值的场景可以包括如下几个方面：

1. 用户故事：通常记录使用软件的动机，目的和具体动作。比如“用户输入他的基本资料信息”，但不会具体到用户单击某个输入框或输入具体的值
2. 描述的需求：描述的是软件具体的功能，涉及到如何使用产品来执行这些功能
3. 集成场景：多个功能之间集成后是如何工作的
4. 设置和安装：初始安装过程，初始化信息，配置信息，定制化功能
5. 警告和出错：用户非正确执行软件时的错误信息和警告信息，一般 negative 测试

要想在有基本的场景的情况下，或者在我们的测试设计完成后，直接进行 ET 测试执行，而且还可以取得很好的产出，这边借鉴与 James A. Whittaker 的两种集成方法来达到这个目标，都是通过对于既定的场景加入变化来更多覆盖我们的 SUT。请看如下的场景探索模型：



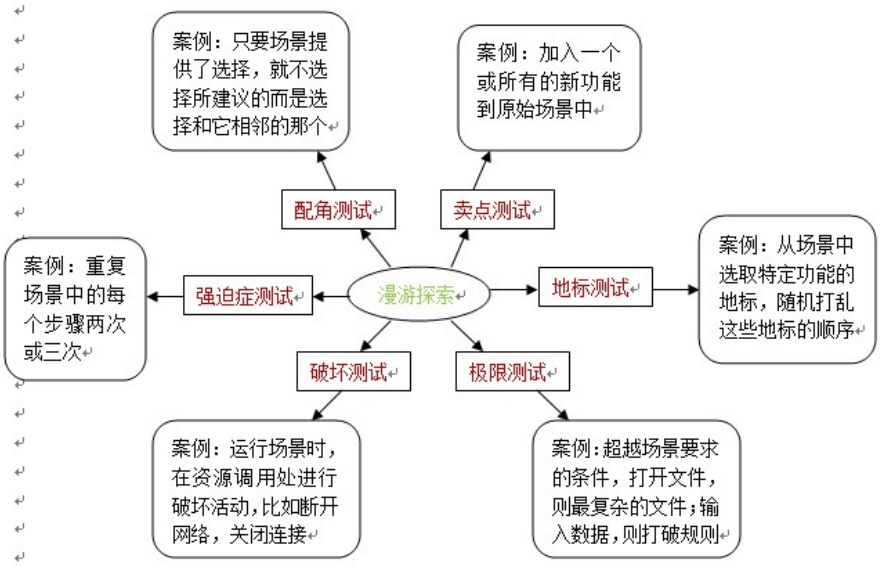
对于某个具体的方法，下面会有些更详细的介绍和例子：



所有上面用到的方法都是基于存在一些的基本场景的，这里可以理解为我们的测试设计，再给我们的场景的步骤加以操作，来给场景注入变化，就得到了一个新的场景，我们称之为衍生场景，我们甚至还可以对衍生场景进行场景操作，又可以再次生成新的场景。

前面我们讨论了漫游测试模型，那这里也可以将漫游测试模型里面好的方法应用到我们的场景里面，进一步的丰富我们的场景，一般情况下是在执行场景时，我们可以在任何一个步骤停下来，注入某种变化以创建新的场景，这就是场景操作模型。当如果我们要使用漫游测试的方法的话，我们就必须改变策略：查看基础的场景，找到需要测试人员做决定的地方或者找到可能产生逻辑分支的地方，先往不同的方向走，然后回到场景的主要路径里面来。

下面请看漫游探索模型是怎么使我们的场景更加富有的：



只要我们更多的在场景中加入这些方法去产生更多的衍生场景，更熟练的应用这些方法到实际的 ET 测试执行过程中，又何愁不能有很好的覆盖率去保证 ET 测试的整体质量呢。我们可以一开始在应用这些模型的时候，可以边测试设计边看模型；如果我们对模型非常熟悉的时候，我们完全可以不看模型，在自己的头脑中实现模型并设计出该需求的所有测试用例；最后我们就可以在 ET 测试执行的时候能完全的应用这些模型，那我们的 ET 的质量和覆盖率就能快速的保证了。

未来篇

1、ET 是否能够代替 ST

这个问题也许是大多数最关心的问题，其实前面几篇文章也表达了我的个人的观点，但我这边想说的更尖锐一点，讨论未来 ET 的发展和 ST 有什么关系，真正了解 ST 和 ET 的定义的人，应该可以感觉这个问题就像问自动化测试是否能代替手工测试一样，我想绝大部分人都可以说未来不管测试行业怎么发展，自动化测试不可能代替手工测试。但这样比喻，又有一点疑问，那就是自动化测试不可能代替手工测试的主要原因和 ET 不能代替 ST 是完全不一样的。 我们可以想到自动化所有的测试用例受到的制约非常复杂，有一个代价和成本太高，还有一个就是技术要求高。但 ET 如果在未来会代替 ST，并不会受到这方面的制约，可以看到如果我们可以多实践和掌握那两大模型，我们完全在 ET 中来代替 ST。但这里只能说存在一部分人的能力达到标准，这些人所做的项目完全可以使用 ET，但并非适用所有人。

这里面说 ET 代替 ST，并不是说不采用 ST 的某些方式在 ET 中，比如我们 ET 测试之前完全可以做 test idea list，类似于测试设计，这一点和完全意义上的 ET 似乎有点矛盾，下面说下如果想让 ET 来代替 ST，ET 的发展需要做些什么。

首先需要就是引用 ST 的测试设计技术，让 ET tester 真正的知道怎么去做测试设计，在测试执行之前做到心中有数，但 ET 真正的魅力是在测试执行时的表现。假设未来我们有一个非常炫的技术: 测试人员的专有提示显示(测试人员在测试执行的时候，光标移到什么位置，比如某个用户接口控件，会弹出个提示显示，里面有属性，源代码，代码改动量，输入数据的状态等，就像打魔兽世界游戏时的迷你世界地图显示一样)，那简直是酷极了的事情。

其次需要的是综合所有 ST 的经验，之前 ST 所写的那些测试用例，那些好的用例，好的测试思路，都是需要我们去积累的。假设未来我们有一个真正全面的测试百科(有大部分所有重用和共用的功能的所有的测试用例和测试思路，如果可以的话，还包括自动化测试用例)，但测试人员有了这些，我们越少的减少写用例的时间，就越来越远离 ST，真正的发挥在测试过程中 ET 的作用。

之前也提到过 ET 和 ST 组合的模型，现在 ET 业界的部分大师，大部分认为 ET 可以作为 ST 的一个有力的辅助，起到一个推波助澜的作用，有则好，无也 OK。 有的话，做得好则好，做得不好，一般也 OK。这边说下，似乎 ET 的大师们，在推 ET 的过程中，强烈限制自动化测试的作用，将人的思维以及创造性放大，还有就是强烈 BS 测试用例写的非常详细和维护。其实说白了，ET 的发展依靠于我们如何更多的抽象出方法，并能让新人很快的接受，真正的使其门槛降低，减少经验的限制。

畅想下未来，我们不需要做任何测试设计，也不需要写测试用例，我们完全在需求的本身质量

上，根据强大的测试百科，就可以自动生成适合项目需求的所有的测试用例和测试思路；且在 ET 测试执行的时候，能根据系统的反馈，执行的代码，数据的状态等多个维度下想到下一个更好的测试用例，不断的攻击 SUT。真正的把 ET 发挥到极致，在任何特定的情况下，都能找到对应的 ET 测试方法的指导，并自动产生测试用例。让测试变得越来越有趣，越来越有技术含量，越来越提高 SUT 的覆盖率。

ET+ST 的测试团队到底在项目测试过程中的效率到底怎么样呢，还有具体的关系。对于复杂的软件系统，真实的使用软件是高质量测试设计的根源，所以一旦我们由于 ET 而沉淀了很多优质的测试用例，这个时候再用 ST 来固化这些经验并传承给所有测试人员。

大家很多人都认为 ST 的做法，很容易看到我们测试的覆盖率，可以看到我们能够发现和即将测试的场景和思路，相对于 ET 来说，认为 ET 在随机性方面还是存在的，这样势必会带来用例的遗漏；对于这个问题，我想从几个方面来说明下：首先看我们应用 ET 在项目中的模型方式，笔者认为只要正确理解了 ET 和采用了正确的思路来执行，至少 80%的 ET 是带有非常明确的目的性的，而不是随机性的，这时存在 20%的随机性为何一定要说明不可接受呢？要知道我们很多严重的 bug 都是在偶然性的机会下发现的。另外一个层面，我们需要知道每个 ET Tester 每天所执行测试的模块和测试点是否符合当前项目的情况，我们需要定期的开会总结，并吸取优秀的 ET Tester 的测试思路和想法，积极的分享自己的收获，并提出自己在测试执行时遇到的问题。积极的发现问题，积极的解决问题，这里当然需要我们的 ET Lead 有足够的经验，有足够的敏感度，对于 ET tester 成员能力的了解，产品模块的特点，发现 bug 的缺陷模式等多个角度快速的做出新的判断，制定出新的 ET 执行策略，从而最大的避免过多的用例遗漏。

2、ET 是否能够自动化

这个问题提出来大家也许觉得不可思议吧，如果能够正确理解 ET 的概念，能够记得 ET 的核心的人，都认为 ET 几乎不可能做到自动化，ET 的特点是学习，和测试设计和测试执行同时 Action，那么这边说下国外是怎么考虑自动化的 ET 的。

自动化的 ET 就是利用把在 Session 中做测试时使用一种自动化的方法去进行 bug 重现，回归测试，bug 证据收集。一种方式就是使用第三方工具录制测试人员做测试执行且分析日志文件，甚至用于将来的回归测试。这种方式叫做 Passive EAT。另一种就是使用 KDT 的自动化工具去自动化某个 session 的测试执行。比较适于 pair testing，一个测试人员负责创建自动化测试脚本，另一个测试人员执行脚本，记住这些自动化脚本也是在测试执行和测试设计和学习产品的产物，这种方式叫做 Active EAT。

这边详细的说下 Passive EAT 的实现方式，之前使用工具录制测试人员做测试执行，在这个 Session 完成的时候，测试人员需要根据录制的 Session 测试设计新的测试用例和原来的测试结果。这个时候使用工具录制，一个好处就是测试人员完全可以自己做测试，且可以转移思维去报测试过程中发现的 bug，真正的做测试执行，节约了大量时间，同样也会减少不能重现的 bug 率，可以看到这种方式在重现，记录和报告 bug 方面有很大的作用。

关于 Active EAT 方式，对于一个 Session 的测试执行肯定就是执行所有的自动化测试脚本，同样的在这个 Session 完成之际需要做些事情: 第一个测试人员和第二个测试需要互换角色(这样可以最大程度保证 SUT 的健壮性和稳定性)；做出一个以后用于回归测试的范围出来(考虑覆盖率，唯一性，独特性)；对于每一个完成的自动化测试脚本做一些更高级的描述。

自动化的 ET 相对与手工的 ET 来说，它到底有哪些优势呢：

-----第一个就是更容易的分析测试用例；

-----重现 bug 方面更容易；

-----提高测试覆盖率(在 session 测试开始之前就可以 run 已经存在的自动化测试脚本)；

-----节省时间(当使用 Passive EAT 时，交叉测试方面可以减少很多时间)

-----提高更好的保证(让管理层或 check 方能够清楚的知道我们的测试活动)

这里面我们可以看到所谓的自动化的 ET，就是将 ET 测试过程中的一些关键点保存起来再事后进行新的优化和完善。现在说的 ET 大部分情况下是手工的 ET，充分发挥人在测试执行过程中的创造性，如果能让机器模拟人的思维过程，从系统给出的反应中又能快速想到的新的测试用例。

想象下未来的自动化的 ET，能够先写一部分的 ET 自动化测试脚本，在运行这些脚本的时候，

每个测试脚本中存在某个检查点能够时刻观察系统的输出，页面的显示情况，并记录下来，如果能够根据这些情况，自动产生新的测试脚本，那就更酷了。这样就让那些检查点模拟人的思维去思考这个输出是否合理，且在过程中的一些异常，某些数据的状态变化，并从中找到可疑点，并产生新的用例再次验证之前的假设或再次攻击最有可能出错的地方。这样我们测试工程师就可以 run 这些 ET 自动化测试脚本，然后等着接受大量 bug 的 report 了。如果真的可以实现这个，这个时候测试人员的价值又在哪里呢，我猜想应该是不断的完善和提供更多的高智能的检查点，让其具有更强的思维去产生更多和更好的测试脚本。

当然我们还可以从另外一个层面来考虑，也就是对与自动化的定义了，James 曾经说过一切能够 support 测试的工具都可以说是自动化测试的一种。对于这个，我们完全可以相信探索式测试也可以借用自动化的手段来进行测试执行，我们可以考虑在测试执行的时候，去写个小工具来满足我们测试思路的需求。打个比方：比如我们想验证某个输入框最大会接受多少个字符，手工层面我可以不停的输入 100 个，1000 个，甚至是 10000 个，不管你是否是单个 copy 还是多个 copy，总会觉得有点土，这时如果我们考虑做个工具来实现我们想要输入多少长度的字符都可以，还可以定制化到是字母还是数字等。利用这些实现好的或即将实现的工具来辅助测试执行，这时候也可以说将探索式测试进行了自动化。

References：

(1) Cem Kaner, James Bach, Bret Pettichord; 软件测试经验与教训
(2) James Bach; Exploratory Testing Explained
(3) Jonathan Bach; Session-Based Test Management
(4) James Bach; Michael Bolton Rapid Software Testing
(5) James Bach; Michael Bolton Rapid Software Testing Appendices
(6) James Bach; Get the Most Out of Exploratory Testing
(7) Juha ltkonen, Mika V. Mantyla and Casper Lassenius; Defect Detection Efficiency: Test Case Based vs. Exploratory Testing
(8) James Whittaker Exploratory Testing
(9) Sven Sambaer Exploratory Testing: Evolution or Revolution
(10) James Bach; General Functionality and Stability Test Procedure
(11) Alan Page; The Beauty of Testing

致谢

在本白皮书的评审过程中，非常感谢如下测试同仁给出的意见和建议:
王迪----------阿里云测试工程师
史亮----------微软中国测试开发工程师
夜雪----------淘宝网测试工程师
星璇----------淘宝网测试工程师

同时也感谢一直支持我的老大们: 郭芙，自在，云齐，侯风，皇析
如果各位有好的建议和意见，或者有兴趣交流的请联系笔者:
MSN: gaoxiang_0911@hotmail.com
Email: jige@taobao.com
旺旺: 季哥