泽众软件测试网
电子期刊

SPASVO

第二十三期
2013年11月

设计测试用例的四条原则

让程序员愤怒的10种事

让你的测试生活燃烧激情

测试工程师的十二最

不是技术牛人，如何拿到国内IT巨头的Offer

从软件实施看CRM：需求分析是关键

怎样从容应对客户的需求反复

WEB性能测试用例设计

上海泽众软件电子期刊

2013 年 11 月 第二十三期

主办单位：上海泽众软件科技有限公司

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

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

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

邮政编码：200063

公司主页：www.spasvo.com 论坛：bbs.spasvo.com

目录

设计测试用例的四条原则

这一年里来来去去，变化最大的就是很多一起工作了多年的同事离开了，很多都去了"更给力”的地方。公司里来来往往是很正常的，想想我最近一次换到“更给力”的地方，那都是5年前了。总之，现在的地方还是挺给力的，好好工作，争取有更大的进步，呱唧呱唧!

测试用例设计的最基本要求：覆盖住所要测试的功能。这是再基本不过的要求了，但别看只是简单的一句话，要能够达到切实覆盖全面，需要对被测试产品功能 的全面了解、明确测试范围(特别是要明确哪些是不需要测试的)、具备基本的测试技术(如：等价类划分等)等。那么满足了上述这条要求是不是设计出来的测试 用例就是好的测试用例了呢?答案：在理论上是，但在实际工程中还远远不是。之所以理论和实际会有这样的差别，是因为在理论上不要考虑的东东，而在实际工程 中是不得不考虑的 -成本。这里的成本包括：测试计划成本、测试执行成本、自动化测试用例、测试自动化成本，测试分析成本，以及测试实现技术局限、测试环境的 Bug、人为因素 和不可预测的随机因素等引入的附加成本等。

由于成本因素的介入，决定了工程中设计好的测试用例原则不只有“覆盖住所要测试的功能”这一条，下面是我根据自己的工作经验总结出的其它四条原则，在这里抛砖引玉，希望大家拍砖和指正。这些原则特别是针对那些需要被自动化，并且是要被经常执行的测试用例。

1. 单个用例覆盖最小化原则。

这条原则是所有这四条原则中的”老大“，也是在工程中最容易被忘记和忽略的，它或多或少的都影响到其它几条原则。下面举个例子来介绍，假如要测试一个功能 A，它有三个子功能点 A1，A2 和 A3，可以有下面两种方法来设计测试用例：

方法1 ：用一个测试用例覆盖三个子功能 -Test_A1_A2_A3，

方法2 ：用三个单独的用例分别来覆盖三个子功能 - Test_A1，Test_A2，Test_A3

方法1适用于规模较小的工程，但凡是稍微有点儿规模和质量要求的项目，方法2则是更好的选择，因为它具有如下的优点：

测试用例的覆盖边界定义更清晰

测试结果对产品问题的指向性更强

测试用例间的耦合度最低，彼此之间的干扰也就越低

上述这些优点所能带来直接好处是，测试用例的调试、分析和维护成本最低。每个测试用例应该尽可能的简单，只验证你所要验证的内容，不要“搂草打兔子” 捎带着把啥啥啥啥都带进来，这样只会增加测试执行阶段的负担和风险。David Astels 在他的著作《Test Driven Development：A Practical Guide》曾这样描述，最好一个测试用例只有一个 Assert 语句。此外，覆盖功能点简单明确的测试用例，也便于组合生成新的测试，在 Visual Studio 中就引入了 Ordered Test 的概念。

2. 测试用例替代产品文档功能原则。

通常我们会在开发的初期(Scrum 每个 Sprint 的头两天)用 Word 文档或者 OneNote 的记录产品的需求、功能描述、以及当前所能确定的任 何细节等信息，勾勒将要实现功能的样貌，便于团队进行交流和细化，并在团队内达成对产品功能共识。假设我们在此时达成共识后，描述出来的功能为 A，随着产品开发深入，团队会对产品的功能有更新的认识，产品功能也会被更具体细化，在一个迭代或者 Sprint 结束的时候最终实现的功能很可能是 A+。如此往复， 在不断倾听和吸收用户的反馈，修改产品功能，多个迭代过后，原本被描述为 A 的功能很可能最终变为了 Z。这是时候再去看曾经的 Word 文档和 OneNote 页面，却仍然记录的是 A。之所以会这样，是因为很少有人会去(以及能够去)不断更新那些文档，以准确反映出产品功能当前的准确状态。不是不想去做，而是实 在很难!这里需要注意：早期的 Word 或者 OneNote 的文档还是必要的，它至少能保证在迭代初期团队对要实现功能有一致和准确的认识。

就没有什么东西能够一直准确地描述产品的功能了吗?答案：当然有，那就是产品代码和测试用例。产品代码实现了产品功能，它一定是准确描述了产品的当前 功能，但是由于各种编程技术，如：面向对象、抽象、设计模式、资源文件等等，使得产品代码很难简单地就能读懂，往往是在知道产品功能的前提下去读代码，而 不是反过来看代码来了解功能。好的代码会有详细的注释，但这里的注释是对实现代码的解释和备注，并不是对产品功能的描述。这里有一篇很老的博客 Reading Code Is Hard，介绍了如何能够使代码更可读一些编写技巧。

那么就只有测试用例了，测试也应该忠实反映了产品功能的，否则的话测试用例就会执行失败。以往大家只是就把测试用例当作测试用例而已，其实对测试用例 的理解应该再上升到另一个高度，它应该是能够扮演产品描述文档的功能。这就要求我们编写的测试用例足够详细、测试用例的组织要有调理、分主次，单靠 Word、Excel 或者 OneNote 这样通用的工具是远远无法完成的，需要更多专用的测试用例管理工具来辅助，例如 Visual Studio 2010引入 Microsoft Test Manager。

此外，对于自动化测试用例(无论是 API 或者 UI 级别的)而言，代码在编写上也应该有别产品代码编写风格，可读性和描述性应该是重点考虑的内容。在测 试代码中，当然可以引入面向对象、设计模式等优秀的设计思想，但是一定要适度使用，往往面向过程的编码方式更利于组织、阅读和描述。

3. 单次投入成本和多次投入成本原则。

与其说这是一条评判测试用例的原则，不如说它是一条思考问题的思维角度和原则。成本永远是任何项目进行决策时所要考虑的首要因素，项目中的测试也是如 此，对成本的考虑也应该客观和全面的体现在测试的设计、执行和维护的整个阶段中。当你在测试中遇到一些左右为难的问题需要决策时，尝试着从成本角度去分析 一下，也许会对你的决策有所帮助。

测试中的成本按其时间跨度可以分为：单次投入成本和多次投入成本。例如：编写测试用例可以看作是单次投入成本，因为编写测试用例一般是在测试的计划阶 段进行(Scrum 每个 Sprint 的开始阶段)的，虽然后期会有小的改动，但绝大多数是在一开始的设计阶段就基本上成型了;自动化测试用例也是如此，它 也属于是一次性投入;测试用例(包括：手工和自动化测试用例)的执行则是多次投入成本，因为每出一个新版本 Build 时都要执行所有的测试用例(或者进行 BVT 测试仅执行高优先级的测试用例)、分析测试结果、调试失败测试用例、确定测试用例的失败原因(产品缺陷、测试用例缺陷、测试框架缺陷还是随机问题导 致了测试用例的失败)，以验证该版本整体质量是否达到了指定的标准。

之所有要引入单次和多次成本的思考，是希望能够通过区分测试中不同活动对测试成本的影响，从而进行帮助我们合理布局在不同阶段的投入和做出正确的决 策，以保证在有限可负担测试成本的前提下，最大限度地有效开展测试工作。例如，当我们意识到了，测试用例的设计和自动化属于是一次性投入，而测试用例的执 行则是反复多次的投入时，就应该积极思考如何能够提高需要反复投入的测试执行的效率，在一次投入和需要多次活动需要平衡时，优先考虑多次投入活动的效率， 其实这里是有很多工作可以做。

例如：第一条原则-单个用例覆盖最小化原则 - 就是一个很好的例子，测试 A 功能的3个功能点A1，A2和 A3，从表面上看用 Test_A1_A2_A3这一个用例在设计和自动化实现时最简单的，但它在反复执行阶段会带来很多的问题：

首先，这样的用例的失败分析相对复杂，你需要确认到底是哪一个功能点造成了测试失败;

其次，自动化用例的调试更为复杂，如果是 A3功能点的问题，你仍需要不断地走过 A1和 A2，然后才能到达 A3，这增加了调试时间和复杂度;

第三，步骤多的手工测试用例增加了手工执行的不确定性，步骤多的自动化用例增加了其自动执行的失败可能性，特别是那些基于 UI 自动化技术的用例;

第四，(Last but not least)将不相关功能点耦合到一起，降低了尽早发现产品回归缺陷的可能性，这是测试工作的大忌。例如：如果 Test_A1_A2_A3是一个自动测试 用例，并按照 A1->A2->A3的顺序来执行的，当 A1存在 Bug 时，整个测试用例就失败了，而 A2和 A3并未被测试执行到。如果此时 A1 的 Bug 由于某些原因需要很长时间才能修复，则 Test_A1_A2_A3始终被认为是因为 A1的 Bug 而失败的，而 A2和 A3则始终是没有被覆盖到，这 里存在潜在的危险和漏洞。当你在产品就要发布前终于修复了 A1的 Bug，并理所当然地认为 Test_A1_A2_A3应该通过时，A2和 A3的问题就会在 这时爆发出来，你不得不继续加班修复 A2和 A3的问题。不是危言耸听，当 A2/A3的代码与 A1的 Bug 修复相关时，当你

有很多如此设计的测试用例时，问 题可能会更糟… …，真的!:(

综上所述，Test_A1_A2_A3这样的设计，减少地仅是一次性设计和自动化的投入，增加地却是需要多次投入的测试执行的负担和风险，所以需要决 策时(事实上这种决策是经常发生的，尤其是在设计测试用例时)选择 Test_A1_A2_A3还是 Test_A1、Test_A2和 Test_A3，请务 必要考虑投入的代价。

4. 使测试结果分析和调试最简单化原则。

这条原则是实际上是上一条- 单次投入成本和多次投入成本原则 - 针对自动化测试用例的扩展和延续。在编写自动化测试代码时，要重点考虑如何使得测试结果分析和测试调试更为简单，包括：用例日志、调试辅助信息输出等。因 为测试用例的执行属于多次投入，测试人员要经常地去分析测试结果、调试测试用例，在这部分活动上的投入是相当可观的。有时候，测试框架提功能的一些辅助 API 等就可以帮助很好实现这个原则。例如：Coded UI Test 就提供了类似的 API，详见 - VS 2010 测试功能学习(18) – Coded UI Test 三个必知的函数，来辅助基于 Coded UI 框架实现的自动化测试用例有更好的调试体验。

测试理论为测试工作指明了大的前进方向，在实际工程中还需要我们不断地“活化”这些理论，使理论和实践更好的契合在一起。在我看来，软件工程项目不论 成败和好坏，对我们每个参与者都是无比宝贵的。作为有心人，从中我们体会到很多书本上不曾提到过的东西，只要不断地去观察、体会和总结，你会有更多自己的 认识、理解和发现。有很多人写书称赞，代码之美、测试之美，其实工程项目也是很美，只是看你能不能更客观地去看待它。

让程序员愤怒的 10 种事

今天偶然看到的一篇文章，看了觉得很有意思，虽然自己不是做开发的，但是对于做测试的来说，同样有借鉴的意义！正所谓“知己知彼百战不殆”！

程序员是一个比较特殊的群体，他们因为长期和电脑打交道所养成的性格和脾气也是比较相近的。当然，既然是人，当然是会有性格的，也是会有脾气的。下 面，让 我来看看10种能把程序惹毛了的事情。一方面我们可以看看程序员的共性，另一方面我们也可以看看程序员的缺点。无论怎么样，我都希望他们对你的日常工作都 是一种帮助。

第十位 程序注释

程序注释本来是一些比较好的习惯，当程序员老手带新手的时候，总是会告诉新手，一定要写程序注释。于是，新手们当然会听从老手的吩咐。只不过，他们可能对程序注释有些误解，于是，我们经常在程序中看到一些如下的注释：

代码：

```
r = n/2;  //r是n的一半
 //循环，仅当r-  n/r不大于t
while ((r-n/r) <=t){
        … …
        r = 0.5 *  (r-n/r); // 设置r变量
}
```

每当看到这样的注释——只注释是什么，而不注释为什么，相信你一定会被惹火，这是谁写的程序注释啊?不找来骂一顿看来是不会解气了。程序注释应该是告 诉别人你的意图和想法，而不是告诉别人程序的语法，这是为了程序的易读性和可维护性，这样的为了注释而注释的注释，分明不是在注释，而是在挑衅，惹毛别人 当然毋庸置疑。

第九位 打断

正当程序沉浸于编程算法的思考，或是灵感突现正在书写程序的时候，但却遭到别人的打断，那是一件非常痛苦的事情，如果被持续打断，那可能会让人一下子 就烦躁起来。打断别人的人在这种情况下是非常不礼貌的。被打断的人就像函数调用一下，当其返回时，需要重新恢复断点时的现场，当然，人不是电脑，恢复现场 通常是一个很痛苦的过程，极端的情况下可能需要从头开始寻找思绪，然后一点一点地回到断点。

因此，我看到一些程序员在需要安静不被打扰的时候，要么会选择去一个没人找得到的地方，要么会在自己的桌子上方高挂一个条幅以示众人——“本人正执行 内核程序，无法中断，请勿骚扰，谢谢!”，可能正在沉浸于工作的程序被打断是多么大的开销。自然，被打断所惹毛了的人也不在少数了。

第八位 需求变化

这个事情估计不用多说了。只要是是程序员，面对需求变化的时候可能总是很无奈的。一次两次可能还要吧接受，但也顶不住经常变啊。据说敏捷开发中有一套 方法论可以让程序员们享受需求的变化，不知道是真是假。不过，今天让你做一个书桌，明天让你把书桌改成餐桌，后天让你把餐桌改成双人床，大后天让你把床改 成小木屋，然后把小木屋再改成高楼大厦。哎，是人都会被惹毛了的。那些人只用30分钟的会议就可以作出任何决定，但后面那几十个程序员需要搭上几百个小时 的辛苦工作。如果是我，可能我也需要神兽草泥/马帮助解解气了。

不过，这也正说明了，程序员并不懂得怎么和用户沟通，而用户也不懂得和程序员沟通，如果一个项目没有一个中间人(如：PM)在其中协调的话，那么整个 项目可能就是“鸡同鸭讲”，用户和程序员都会被对方所惹毛了。如果要例举几个用户被惹毛的事情，估计程序员的那种一根筋的只从技术实现上思考问题的方法应 该也能排进前5名。

第七位 经理不懂技术

外行领导内行的事例还少吗?领导一句话，无论对不对，都是对的，我们必需照做，那怕是多么愚蠢多么错误的决定，我们也得照做。程序员其实并不怕经理不 懂技术，最怕的就是不懂技术的经理装着很懂技术。最可气的是，当你据理力争的挑站领导权威的时候，领导还把你视为异类。哎，想起这样的领导别说是骂人了， 打人的冲动都有了。

其实，经理只不过是一个团队的支持者，他应该帮助团队，为团队排忧解难。而不是对团队发号施令。其实管理真的很简单，如果懂的话，就帮着做，如果不懂的话，就相信下属，放手让下属做。最怕的就是又不懂技术，还不信任下属的经理了。哎，这真是程序员的痛啊。

第六位 用户文档

用户文档本来不应该那么的令人害怕。这些文档记录了一切和我们所开发的软件有关的一些话题。因为我们并不知道我们所面对的用户的电脑操作基础是什么样 的，所以，在写下这样的文档的时候，我们必需假设这个用户什么也不懂。于是，需要用最清楚，最漂亮的语言写下一个最丰富的文档。那怕一个拷贝粘贴的操作， 可能我们都要分成五、六步来完成，那怕是一个配置 IP 地址的操作，我们也要从开始菜单开始一步一步的描述。对于程序员来说，他们在开发过程中几乎天天都在 使用自己开发的软件，到最后，可能都有得有点吐了，但还得从最简单的部份写这些文档，当然容易令他们烦燥，让程序员来完成这样的文档可能效果会非常不好。 所以，对于这样的用户文档，应该由专门的文档人员来完成和维护。

第五位 没有文档

正如上一条所说的，程序员本来就不喜欢写文档，而因为技术人员的表达能力和写作能力一般都不是太好，所以，文档写的也很烂。看看开源社区的文档可能就 知道了。但是，我们可爱的程序员另一方面最生气的却是因为没有文档。当然，让面说是的用户的文档，这里我们说的是开发方面的文档，比如设计文档，功能规 格，维护文档等等。不过，基本上都是一样的。反正，一方面，我们的程序员不喜欢写文档，另一方面，我们的程序又会被抱怨没有文档，文档太少，或者文档看不 懂。呵呵。原来在抱怨方面也有递归啊。据说，敏捷开发可以降低程序开发中的文档，据说他们可以把代码写得跟文档和示图似的，不知道是真是假。不过，我听过 太多太多的程序员抱怨没文档太少，文档太差了，这个方面要怪还是怪程序员自己。

第四位 部署环境

虽然，程序员们开发的是软件，但是我们并不知道我们的程序会被部署或安装在什么样的环境下，

比如，网络上的不同，RAID 上的不同，BIOS 上的不 同，操作系统的不同(WinXP 和 Win2003)，有没有杀毒软件，和其它程序是否兼容，系统中有流氓软件或病毒等等。当然，只要你的软件出现错误，无论是你的程序的问题，还是环境的问题，反正都是你的问题，你都得全部解决。所以，程序员们并不是简单地在编程，很多时候，还要当好一个不错的系统管理员。 每当最后确认问题的原因是环境问题的时候，可能程序员都是会心生怨气。

第三位 问题报告

“我的软件不工作了”，“程序出错了”，每当我们听到这样的问题报告的时候，程序员总是感到很痛苦，因为这样的问题报告等于什么也没有说，但还要程序 员去处理这种错误。没有明确的问题描述，没有说明如何重现问题，在感觉上，当然会显得有点被人质问的感觉，甚至，在某些时候还掺杂着看不起，训斥的语气， 当然，程序员基本上都是很有个性的，都是软硬不吃的主儿，所以，每当有这样的语气报告问题的时候，他们一般也会把话给顶回去，当然，后面自己然发生一些不 愉快的事情。所以，咱们还是需要一个客服部门来帮助我们的程序员和用户做好沟通。

第二位 程序员自己

惹毛程序员的可能还是程序员自己，程序员是“相轻”的，他们基本上都是持才傲物的，总是觉得自己才是最牛的，在程序员间，他们几乎每天都要吵架，而且一吵就吵得脸红脖子粗。在他们之间，他们总是被自己惹毛。

技术上的不同见解。比如 Linux 和 Win，VC++和 VB，Vi 和 Emacus，Java 和 C++，PHP 和 Ruby 等等，等等。什么都要吵。

老手对新手的轻视。总是有一些程序员看不起另一些程序员，说话间都带着一种傲慢和训斥。当新手去问问题的时候，老手们总是爱搭不理。

在技术上不给对方留面子。不知道为什么，程序员总是不给对方留面子，每当听到有人错误理解某个技术的时候，他们总是喜欢当众大声指证，用别人的“错误”来表明自己的“博学”，并证明他人的“无知”。

喜好鄙视。他们喜好鄙视，其实，这个世界上没有一件事是完美的，有好就有不好，要挑毛病太容易了。程序员们特别喜欢鄙视别人，无论是什么的东西，他们总是喜欢看人短而不看人长。经常挂在他们嘴上的口头禅是“太差”、“不行”等等。

程序员，长期和电脑打交道，编写出的代码电脑总是认真的运行，长期养成了程序员们目空一切的性格，却不知，这个世界上很多东西并不是能像电脑一样，只要我们输入正确的指令它就正确地运行这么简单。程序员，什么时候才能变成成熟起来……

第一位 程序员的代码

无论你当时觉得自己的设计和写的代码如何的漂亮和经典，过上一段时间后，再回头看看，你必然会觉得自己的愚蠢。当然，当你需要去维护他人的代码的时 候，你一定要在一边维护中一边臭骂别人的代码。是否你还记得当初怎么怎么牛气地和别人讨论自己的设计和自己的代码如何如何完美的?可是，用不了两年，一刚 从学校毕业的学生在维护你的代码的过程当中就可以对你的代码指指点点，让你的颜面完全扫地。呵呵。当然，也有的人始终觉得自己的设计和代码就是最好的，不 过这是用一种比较静止的眼光来看问题。编程这个世界变化总是很快的的，很多事情，只有当我们做过，我们才熟悉他，熟悉了后才知道什么是更好的方法，这是循 序渐进的。所以，当你对事情越来越熟悉的时候，再回头

看自己以前做的设计和代码的时候，必然会觉得自己的肤浅和愚蠢，当然看别人的设计和代码时，可能也会 开始骂人了。

让你的测试生活燃烧激情

今天阅读了一篇文章“让你的 blog 重获青春”，里面介绍了几个方法。忽然觉得这些方法对于我们做其他工作也很有用。因为目前在做测试工作，所以仿写一下，让我们的测试迸发出激情。

很多做测试的朋友（特别是做较长时间的）都有一种感觉，没有了刚开始做测试的热情和积极学习的劲头，失去了对那些枯燥的程序的兴趣，不管如何改换花样 也找不到以往那些锲而不舍、认真负责的精神了。我个人也有这样的感觉，我想这与生活的压力，公司对自己工作的认可和测试工作的重视程度都有关系，不管怎么 样吧，我想这不过就是个理由罢了，这也是我特别想对自己说的话。

就象那篇文章所说的，我们需要返回菜鸟（新手）的情绪，我习惯用激情。让所有的事情看起来充满未知的可能，那么有希望而令人兴奋。

如何让我们的情绪激动起来，让事情变的富有希望呢？

1、了解新的东西。测试领域不过是一棵树。我们在其他方面的兴趣呢？时势、财经、体育、教育、医疗、衣着、饮食或者明星八卦，去尝试了解这些领域的内 容。看看那些不是 IT 圈子内的人，那些技术以外的五彩生活。我们可以成为另一棵大树上的树枝，另一个摘香蕉的猴子。我喜欢足球、篮球、下棋，也愿意看动画 片，研究一下 mm 呵呵，那么朋友你呢？

2、做点自己不擅长的事情。寻找学习一样新事物时的激动，重温菜鸟看问题的感受，把自己以前的学习体会翻出来看看，把新奇古怪的想法加到工作中。侍弄下鲜花、学习照相或是踢个足球都成。我个人习惯胡思乱想，记得曾经学过折叠纸、数独等等。

3、脚踏实地。无论是工作还是生活，都不会一日千里。你需要一步一步的做每件事情，哪怕事情很小，比如复印个文件，下楼取个东西。再写一个测试用例， 再追踪一个新问题，再看一篇测试文章。不断重复并坚持，总有一天，如同文章中所说，你会看到一些新鲜和令人激动的东西像幼苗一样慢慢发芽。虽然还没有在测 试领域崭露头角，但相信总会拥有自己的一片天空。

引用该文的一段话，“一定要记得，那些幼苗相对于那些已经完全成熟的同类来说，是青涩的、渺小的、可怜的。要知道哪一株才能健康的长大，并孕育出芳香的花朵。不管怎样，还是要尽力帮助这些幼苗的成长，并清除周围的杂草。”

4、时常转换到别人的角度看问题。想像自己在别人的位置上，透过别人的眼睛观察世界，感受别人的痛苦。遇到事情不要先抱怨，考虑如果你是当事人你会怎 么办．你有什么好的解决方法，怎么协调众多的关系．这能让你时刻感到新鲜，并让大脑获得更多的刺激，避免总是固执己见而忽略全方位真实的世界。时常想想假 如我是女生的情况，或者当左脚受伤变肿，反而说右脚减肥成功，是不是会很有意思呢？

5、不要轻易的下结论。如果我们是一个富有经验的测试者，时常不深入分析而草率的表现你的鄙夷，贬低他人所作的任何努力（即使这种努力并没有解决实际 问题）。当评论好坏的行为习惯成自然，固定的思维方式会让我们看不到很多东西，时常考虑新鲜的或激动人心的想法会有收获的。任何成熟的或不成熟的，完善的 或缺憾的想法都有可取之处，或引发你的思想或让我们引以为戒。幼儿园小朋友

的想法会让你大吃一惊的！

6、享受现在的一切。把我们所知的东西放在一边，注意力集中在现在。当看到一个新问题，也许会想起10个看起来差不多的问题，不要管这些。好好的感受 当前的这个新问题，专心的分析让它描述的更清晰，看起来更舒服。享受思绪流动的感觉。陶醉在畅想的空间里，我们越来越感到自信、变强，我们正在不断成长， 我们拥有激情，我们浑身都散发着活力。这都是真的存在.....

测试工程师的十二最

测试工程师最开心的事:发现了一个很严重的 bug，特别是那种隐藏很深，逻辑性的错误。偶第一次发现这种问题的时候，听到上司和开发人员的表扬时，高兴的就想扭 pp。不过现在慢慢矜持些了，呵呵。

测试工程师最提心吊胆的事:版本 release 出去后，客户发现了很多或很严重的 bug。经过紧张的系统测试之后，好不容易可以轻松一下了，却又陷入 了每天担心正在做验收或使用的客户一封邮件或一个电话说产品有问题。碰到好些的老板还会比较乐观的看这样的问题，最惨的就是有些人一顿臭骂，之前的辛苦， 加班全部都给抹杀了。

测试工程师最憎恨听到的话:"为什么这个 bug 没有在测试的发现呢?"这句话经常是客户发现 bug 后，老板对测试人员的质问。当然这里排除那种很明显 的错误。其实谁都知道 bug 是不可能全部发现的，这句话其实也是客户对大头，大头对小兵一级一级问下来的。除了希望测试人员警惕之外，还有更多的是一种" 踢猫"的行为。对于这句话，偶第一次听到这句话的反映是"我们怎么可能发现所有的 bug 呢"，后来变成"制造 bug 的人不是我们，是开发"，到现在的"让 我查查我的日志，问问开发这个 bug 的原因，为什么我们会没有找到，下次我们会怎样"的回复。

测试工程师最郁闷的事:"刚才那个版本打包打错了，你们要重测"。新版本来了，马上投入紧张的测试，希望能够多找些 bug。没想到辛苦了可能大半天， 开发人员说打包打错了，你重测吧。这种情况虽然可以通过规范流程之类的办法控制发生的机率，但人总会犯错，多少而已。碰到这样，你除了提醒开发部门下次注 意，你除了重测没有太多办法。

测试工程师最不想面对的事:在测试晚期或最新的版本里发现了以前一直存在的问题，特别是当问题很严重时决定到底报不报 bug。报吗，开发人员肯定会问 以前有没有这个问题，不报吗，客户发现更惨。毕竟客户或老板的责备比开发部门或主管的责备轻许多，最后还是会报到 bug 库里的。

测试工程师最不想做的事:申请版本推迟发布。由于在版本发现了太多的问题，觉得产品不能达到发布的，建议公司推迟发布产品。这时虽然大家都知道产品有问题，尽管你自己也不希望这样，但谁都觉得你是一个制造麻烦的人，毕竟市场的压力很大呀。

测试工程师最丢人的事:辛苦的发现了一个 bug，居然是该配的参数没有配等一些自己的失误造成的。有些该注意的地方居然测试时忘了，找出的问题给开发人员一顿臭扁，无比丢人啦。

测试工程师最怕的事:一天，甚至几天都没有发现一个 bug!经过一段时间的 bug 高峰期后，有段时间会发现 bug 数量的减少，最可怕的就是一天都没有 发现一个 bug。偶有时会难过的吃饭都没心情。搞得偶的开发朋友说了一句最让人吐血的话:"要不要我在代码里放几个 bug 给你呀，hoho"

测试工程师最伤心的事:每年的调薪，发 bonus 或发股票时，测试工程师总比开发工程师少。偶有一同事在调薪的第二天就申请转开发，说测试太没前途了。

测试工程师最有力的保护方法:把你认为是 bug 的问题都提交到一个正式的，可以追踪的地方(一般来说是 bug 库)。有时总会碰到一些很小的或是很难判 断的问题，犹豫一定是否要报，特别是一些 UI 的问题。有时问开发人员，他们可能会轻描淡写的回复你导致你没有 report 它。但多年的经验一定要报，了解 bug 流程走向的人都知道，后面还有人 verify，还有开发经理判断，如果不是 bug，自然他们会回

复，会写明原因。说白了，出了问题也不是你的事情。 当然一开始经验不足时会收到一些白眼球，但慢慢经验多了，对系统熟悉了，自然这种情况会少些。人也可以从一些问题中发现自己的弱点。但如果不报，那天客户 提出来，你除了懊悔还要面对指责，严重的炒鱿鱼。

测试工程师最任重道远的事:测试驱动开发。碰到这种开发模式的项目，既是测试扬眉吐气的机会，也是可能会陷你于深渊的恶潭。你就必须打起十二分的精神。等于你在引导开发，有什么问题一定要提出来，否则你就等着被盲目的牵着鼻子走了。

测试工程师最期待的事:测试能够越来越受重视，测试工程师的考核越来越合理。

不是技术牛人，如何拿到国内 IT 巨头的 Offer

不久前，byvoid 面阿里星计划的面试结果截图泄漏，引起无数 IT 屌丝的羡慕敬仰。看看这些牛人，NOI 金牌，开源社区名人，三年级开始写 Basic...在跪拜之余我们不禁要想，和这些牛人比，作为绝大部分技术屌丝的同学们，是否真的与国内 IT 巨头遥不可及呢？

当你打开这个帖子的时候，我已经默认你是此文的目标读者，也就是想进入国内一流互联网企业的非牛人应届生。

你不需要拿 NOI 的奖，无需是开源社区名人，也用不着发过牛逼的 SCI 论文。（没错，笔者就是这样的技术屌丝）

请记住，校园招聘，应聘的绝大部分人都只是才出象牙塔的毛头小子。企业需要的是你们的潜力与激情。牛人总是凤毛麟角的。

程序员笔试面试的经验贴、经验书不计其数。本文不会教你如何具体的解题，但是会告诉你，你距离你的梦想究竟有多远，以及如何去缩短这个距离。

笔者仅仅以自己的亲身经验为依据，将国内 IT 巨头按 Offer 到手难度降序排列，大致分为如下3个梯队：

T1：百度，阿里，腾讯，...

T2：网易，迅雷，完美时空，360，金山，...

T3：华为，中兴，联发科，...

Tx：垄断类 IT 国企。如中国移动，...

T1主要是 BAT 三巨头。他们对学生的技术能力与综合素质都要求较高。他们尤其喜欢寻找牛人。因此你必须有扎实的基础的同时还要有自己的技术个性和特 点，让他们欣赏你。这些公司无论是实力还是待遇都是一流的。但要注意这类公司太大，项目组太多，竞争也很激烈，因此要注意认真考虑你想去，而且对你而言有 优势的项目组。

T2都是其所在领域的领军企业。待遇会比 T1稍低。他们需要基础扎实的学生，如果你的项目或者技术方向符合他们所在的领域会很有优势。（如游戏领域偏爱图形学，安全领域偏向安全方向）

T3对学生的出生、资质最为看重（双211，四六级），甚至对性格有较为挑剔的考察（华为的性格测试反而刷掉很多技术较牛但是性格较怪癖的学生），专业考察的很基础，但考察面较广。

Tx 之所以给了个 x，是因为他们和其他的企业无法比较。因为这类企业笔试考行测+专业基础。面试考察综合素质、表达能力，尤其看重你的非技术方面的能力。因此不做本文的重点讨论。

无坚不摧——完整项目

“当他说他是 OpenCC 的作者的那一刻，哪个面试官不被秒杀。”

当然你不需要 NB 到这个程度。如果你能对面试官说：“我读书期间做的项目有 x 万行代码。Google 关键字 xxx 可搜到该项目的演示视频”，就足够了。

程序员的所有技术能力都能在一个完整的项目中得到淋漓尽致的体现，因此胜过千言万语的自我推销。所谓的完整项目应该满足以下条件：

1、完整性。具有一定的功能，或者解决了某个问题，具有一定意义。

2、难度。 使用或者研究了一些较新技术，或者有一定价值的技术含量或研究内容。

3、工作量。是一个需要浇筑一定心血的产出品。

因此，当你决心把一个项目写入你的简历中，你就一定要能回答出面试官的如下问题：

1、你负责了哪一块？

这个问题是想知道这个项目里究竟有哪些代码是你写的，尤其是多人合作的项目。你必须强调你所做的工作。

问题就出来了。很多时候我们参与的项目，他的架构、核心技术你并不熟悉，而仅仅是写了部分逻辑代码，那怎么办呢？

解决办法是，花时间去了解项目的核心，对项目的整体有清晰的认识，至少要达到能够表述的很清楚的程度（简单的说就是能吹的很有说服力。回想你答辩的情形）。

如果你做的那一块确实微不足道，而且你也无法表述项目全局，那就放弃提及这个项目吧，否则只能让面试官越看你越觉得锉。

2、你用到了哪些技术？

这是最好发挥的一个问题。你可以介绍项目用到的每个开源库，也可以介绍你用到的源代码管理工具（如 SVN、GitHub），调试工具（如 WinDbg）甚至项目管理工具（UML 工具、VS Project 等）。总之，这是一个很好表达你的项目综合能力的机会。

如果上面的都不出彩，那尝试从你的项目架构、设计模式、接口设计等方面入手。总之要站在一个较高的角度，空谈项目的业务需求和逻辑意义不大（当然，充满创意的项目除外）

3、你遇到的最大问题是什么？如何解决的？

这个问题是最重要的，也是最具有回答技巧的问题。你必须说出一个听起来确实很难解决，但你确实又解决（或者避开）的问题。

如果你实在没有头绪，或者你觉得项目确实太简单，没发现困难问题，不妨从这些方面思考：

有网络功能的项目，考虑网络传输效率和网络同步等问题；

有多线程、多进程的项目，考虑他们之间的同步/互斥、负载、调度问题；

需要处理大数据的项目，考虑数据预处理、数据调度等问题；

如果这个项目出了论文，那么尝试描述论文解决的问题；

你解决问题的渠道，如 MSDN、CSDN、开源社区的论坛、国外技术论坛、文档手册等。

面试官想要听到的，是你发现问题、分析问题、寻找解决方案、最终解决问题的思路与方法。细节并不重要，因为他也未必能完全弄懂每个技术细节。

总之，一个完整的项目能让你充分的表达你的技术能力。在项目这一块上，你需要下足功夫。

如果很不幸你没有，那么请往下看。

深厚内功——坚实基础

一般第一轮技术面都是来考察你最基本的技术功底。

招聘季节，随处可见抱着厚厚的《程序员面试宝典》啃的学生。偶尔也能看见《编程之美》《剑指 Offer》的神书。这些经验书确实有用。但是要想全面的掌握笔试面试的基础考点，还是需要完整的复习。

其实，笔试面试对计算机基础的考察是万变不离其宗的。其考点无非分为：

语言语法（以 C/C++为例）。

指针（数组），函数指针，操作符运算顺序，const（常指针与指向常量指针），static 四大用法，字符串（字符数组），字节对齐（sizeof），位运算。秒杀书籍：《C++ Primer》

面向对象。

构造与析构顺序、多态、重载、覆盖、C++对象模型等。秒杀书籍：《深入理解 C++对象模型》。

数据结构。

栈，队列，链表（双向、循环），树，堆，哈希表。

基本算法。

排序（最重要的是快速排序）、查找、图算法、贪心算法、动态规划。秒杀书籍：《算法导论》。

设计模式。

考察最多的就是单例模式。只因为他实在是太常见又太简单了。秒杀书籍：《设计模式》，《重构》。

数据库。

主要是 SQL 语句与存储过程。

操作系统。

进程与线程、互斥与同步、死锁、进程间通信，页表，虚存等。秒杀书籍：《Windows 核心编程》，《Unix 核心编程》。

计算机网络

ISO 七层架构，TCP，UDP，IP 地址等。

英语。

有些公司喜欢出一些用英文描述的问题，或者英文翻译题。看懂 IT 领域内的英文并不难，如果你平时使用 MSDN、Google、StackOverFlow 的话根本不是问题。

如果你还有充分的时间，建议认真看上面推荐的秒杀书籍。如果时间不够，就有针对性的去掌握这些考点。

不得不吐槽的是，很多技术不错的朋友，有着很好的项目，反而挂在了笔试的基础知识考察上面。因此不要小看这些考点。该背的还是要死背的。

笔试或者面试如果让你在纸上写程序，会有2种情况：

1、写一个函数或算法。

不要因为题目简单就想在最短的时间写出来。请一定要注意，对所有参数做边界检测和有效检测。这才是考察的重点！

如果一个算法具体实现你记不清了，就写伪代码，在每行代码后加上详细注释。如果是面试，写完以后跟面试官解释说具体的代码你忘了，但是你记得算法思想，因此用了伪代码。

如果具体思想也忘了，就尝试用自己的思路解答问题。总之，尽量别交白卷。

2、设计一个软件或系统。

这种情况不要求你写详细代码。你需要在程序结构、框架、设计模式或者系统架构等方面进行设计。

这种框架性的东西最好先打草稿，想好了再重新画一遍，把每个模块的功能，模块之间的关系、各个模块的功能接口画出来，如果是面试，写完以后给面试官详细解释。

这里强调一点，是否懂得架构设计，是鉴别代码菜鸟和熟手的重要指标。T 级越高的公司，越偏向于考察架构层级的知识。比如百度笔试的最后一题经常是要求设计一个分布式服务器系统。

锦上添花——无限潜力

通常技术一面是面基础，二面更多的是双方的进一步了解。如技术方向，技术潜力等。

如果二面面试官不问你技术问题，那么你一定要积极主动的与他沟通，并表达你的意愿。尝试以下几个方向：

1、表达你的技术潜力与热情。

面试官可能会问你一些和技术看上去没有任何关系的问题，比如问你最近在看什么书，学习之余喜欢做什么，常去哪些网站之类的。

如果你说最近在看《诛仙》，平时喜欢玩 LOL，你就是在把自己往悬崖上推。实际上面试官希望听到的回答如下：

“我最近在看《C++ Primer 第5版》，因为我在项目中用的 C++11的特性越来越多了...”

“我业余时间喜欢看看 TED，上面总有很多让我激动的新技术出现...”

“前段时间比较闲的时候，和朋友参加了 xxx 组织的开发者大会...”

“虎嗅和猎云是我获取 IT 信息的常去地方...”

回答如此平凡的问题却能体现你的闪光点。你是技术人员，请记住，告诉面试官你时刻对技术保持着激情，时刻关心的 IT 动态，比你告诉他你是学生会某干部有用的多。（当然非技术人员，或者国企的面试除外！）

但是，一定要如实回答。面试官会针对你的回答进行紧逼追问。如果正好是他熟知的范畴，而你只是接触过而没有认真学习，就会陷入很尴尬的境地。

所以在回答这些问题的时候不用过于急着回答，不妨先想一想，要有能预测到面试官针对你的回答会问什么样的问题的能力。

就像上面的例子，面试官问你最近看的书，你未必要选择最近看的一本书，而是应该选择一本你吃的比较透的，最好还是面试官也会感兴趣的书，这样接下来的交流就能得心应手。

总之，平时的积累才是王道。

2、表达你的技术爱好。

进入正确公司的错误岗位，相当于考上了正确学校的错误专业。

所以请一定要记住，你的最终目的不是要进入该公司，而是要进入该公司你最想去的部门乃至项

目组。

所以，试探得知面试官来自哪个项目组也很关键，因为面试官可能跟你想去的项目组毫无关系。技术方向的不对口的面试官面试你，对你是不利的。

这个时候你要清楚的表达出你的技术方向，并注意考虑你想去的项目组收你的可能性。如果发现该组招的人少，或者加入难度大，你需要考虑是否表现出来你有同样的热情加入其他项目组。

不服从分配可能导致你一无所获。你之前面试的表现越优秀，在这一步能够选择的余地就越大。

3、态度和情商。

如果你面试次数多了，拿的 Offer 多了，通常到了最后一面，有多大的希望能拿到 Offer，你心里应该有所感觉。

如果感觉不好，最后一面你需要更努力表达你的优势。最后一面打动面试官的可能往往是你的真诚和热情。

当面试官问你有多少 Offer，不要惧怕回答。Offer 是企业对你能力的证明。有 Offer 的学生更容易被青睐。通常如果你有了该公司最大竞争对手的 Offer，你可以尝试追求更好的岗位和待遇。但切忌用这个来漫天要价，除非你牛到了他们非要你不可的程度。

T3，Tx 类的企业可能会问你一些很奇葩的问题。比如 A 公司问你他的竞争对手 B 公司怎么样。

很多计算机专业的学生思维过于死板，说了大实话，比如”B 公司是最大的 xxx 企业“，于是死的很彻底。

你可以说”A 公司的优势在与 xxx，而 B 公司的优势在于 xxx。不过在我看来，我更欣赏 A 公司的 xxx，因为 xxx“

这类企业就是这样。他们的面试和 T1，T2企业的面试差别很大，你要学会避重就轻，这不是谎言，而是策略。

写在最后

只要你资质不差，有针对性的进行努力，拿下国内 IT 巨头的 Offer 并不难。

最后，请记住，拿下 Offer，你的技术生涯仅仅是进入了下一轮新的迭代。

技术之路最公平也最残酷的原因是：没有捷径，需要日积月累的积累，以及对技术持久的热情。

从软件实施看 CRM：需求分析是关键

关于 CRM 的讨论在媒体上也越来越多，但许多讨论已越来越偏离应用软件系统，进入哲学范围，只谈理念与管理，忘记了 CRM 需要信息技术作为支撑。本文 总结了作者在 CRM 领域的多年从业经验，从软件实施角度重新审视 CRM 实施，为企业实施 CRM 与建设 CRM 系统，提供一点参考经验。

CRM 系统是管理信息系统

“没有软件也可以实施 CRM….”很多文章这样写道。大概现在已经没有人会质疑 ERP 实施一定要实施软件，但不知出于什么原因，很多作者提出了 CRM 实施可以不需要软件的观点。脱离软件的 ERP 实施不知道可以实施什么，但脱离软件的 CRM 咨询或实施业务好象还有许多培训与咨询机构在热烈鼓吹着。

追溯历史，CRM 系统与 ERP 系统都是作为管理信息系统被提出的，CRM 是先进的营销与服务管理理念与信息技术结合的产物，首先是因为有软件或信息系 统的支持，才提出 CRM 的概念。如果没有软件，在市场营销中的服务营销、客户细分等理论就可以涵盖，不需要标新立异，另立门户。

从管理信息系统角度看，CRM 系统是营销与服务业务的信息基础设施，实施过程中管理与软件两手都要抓:管理方面的实施与优化应以企业(甲方)为主，按 管理意图与企业客观情况实际进行调整，也可请咨询机构辅助开展;软件的实施与部署则应以软件公司为主，按管理信息系统的实施步骤进行。

应用需求分析是实施的关键

“你认为 CRM 系统实施过程中最重要的阶段是什么，如果只说一个阶段的话?” 有一个客户的高层领导曾问到。

“是需求分析阶段!”我很肯定地答复，“就象医生要为你提供保健或治疗方案一样，医生开出方子前，首先要搞清楚你的基本状况:你的体质哪里最弱?过去有什么病史?你的工作环境对你身体有无影响?……”。

调查与分析应用需求，是管理信息系统实施的基本步骤。在需求分析阶段，CRM 系统的实施人员通常要解决以下基本问题:

1、 明确管理目标:调查与分析管理上希望达到的目标或需要解决的问题，区分主次;

2、 优化管理流程:由于引入 CRM 系统，对原有的手工业务操作或审批流程必然会有调整，需要以客户为中心重新梳理流程，使流程顺畅、合理;

3、 明确应用权限与功能:根据岗位与业务角色，明确各角色在系统中的应用权限与详细应用功能;

4、 确定功能规格与应用界面:根据应用要求，确定应用界面与详细的信息格式与展现方式;

5、 确定与其他信息系统的接口;

6、 明确系统部署与应用模式;

7、 分析数据，确定数据导入与数据质量控制的方案;

8、 协商与明确系统应用培训的模式。

由于软件的特殊性，在应用需求分析阶段，实施人员的行业应用经验对确定应用需求较为关键，就象医生一样，个人的经验与水平在服务过程中，通常会直接影 响分析结果。为加速需求分析过程，准确把握需求，可采用行业化的 CRM 平台或相关应用进行示例与引导，将需求协商的结果直接反映到应用界面上，这样可大大 提高应用需求分析的质量与效率。

软件客户化能力对实施周期影响很大

CRM 系统的应用部门主要是营销与服务部门，人员的变动通常比较频繁，由于直接面对客户与市场，管理需求也会随着时间迁移、应用人员变化而变化，因 此，CRM 系统实施过程中应控制实施周期，以保障需求在现有管理框架内相对稳定，同时系统上线后，也可取得阶段性的应用成果，有利于 CRM 系统应用的进一 步深入与扩展。

从软件角度看，除实施人员经验、产品行业化程度等因素外，软件的客户化能力是项目周期的关键因素。通常，CRM 软件的客户化会涉及几方面内容:

1、 信息模型建立与数据格式定制

2、 流程定制与配置

3、 权限定制与配置

4、 应用功能扩展

5、 报表与分析定制

6、 外部数据接口

7、 历史数据导入与数据的初始配置

8、 应用界面的定制

从管理上看，各企业营销与服务模式差异性较大，再好的软件也不可能涵盖到所有应用需求，在企业应用中，CRM 软件的客户化工作是难以避免，也是为企业 管理量身定制系统的过程。通常，行业化越高、配置性越强的 CRM 软件在客户化方面的工作就越小，客户化的质量与进度也越容易保障。

由于许多通用型 CRM 产品在设计上未考虑客户化问题，仅能进行简单的信息录入与展现，或实现固定的简单流程，在客户化上缺乏产品平台与工具，因此，面 对客户化需求时，只能通过人工方式直接修改代码。此外，在客户化实施过程中，由于没有应用系统的原型进行引导，CRM 应用需求容易反复，代码也经常被迫重 复修改，导致实施周期无法控制，甚至导致系统失败。

企业级的 CRM 平台通常在配置性上考虑较多(特别值得一提的微软 CRM 3.0的产品)，配置灵活与现有办公系统易于集成，可大大降低实施中软件开发的风险，并易于扩展，可随企业管理变化而调整配置进行适应，适合企业长期应用与部署。

数据质量是对应用至关重要

“数为什么老不准呢?报表怎么对不上?……”在上线应用过程中，许多客户都会遇到类似的问题。这里涉及的原因很多，但最关键的是数据质量控制问题。

常言道“垃圾进、垃圾出”，错误的数据或不正确的信息只能导出不正确的结果，数据输入的质量很大，决定了系统的应用效果。在 CRM 系统中，由于营销人 员管理难度较大，信息经常无法按时保质提供，数据质量需要从管理与系统两个方面进行控制:管理上应明确信息录入的时间与信息项要求，最好在绩效考核上有配 套的措施，以确保信息采集的准确性，此外，对外购信息或批量处理的信息应有熟悉业务的专人进行过滤与确认;在系统上，应根据业务经验，提供信息校验、排 重、修整与批量处理等机制，实施人员还要与业务部门确认数据模型与相关计算算法，保障数据计算与业务一致。

怎样从容应对客户的需求反复

在软件项目的研发过程中，需求变更贯穿了软件项目的整个生命周期，从软件的项目立项，研发，维护，用户的经验在增加，对使用软件的感受有变化，以及整 个行业的新动态，都为软件带来不断完善功能，优化性能，提高用户友好性的要求。我在自己的软件项目管理职业过程中，几乎天天面对用户的需求变更，切身感受 到，如果不能有效处理这些需求变更，项目计划会一再调整，软件交付日期一再拖延，用户的耐性渐渐消逝，研发人员的士气也越来越低落，最后所有的人都在等待 一个结果：项目最好马上结束。所幸，在不断的学习和实践中，我总结了几点比较有效的方法，在软件研发阶段能够较好地解决这方面的问题。

1．需求分析阶段采用原型方法明确用户需求。

在软件项目的需求分析阶段，有大量需求信息需要收集、筛选、加工，这是需求管理的开始。客户和研发两方面的人员对需求的理解呈现“大体上共识多，细节 上差异多”的特点。即使通过反复沟通，最终在时间表限制之内也能拿出一份“用户需求说明书”，但是以实践经验，用户需求的描述永远是“不够清晰”、“不够 明确”的。这主要是因为在这个阶段，所谓的产品都在大家的大脑中构思，正如100个人读《射雕英雄传》，就有100个郭靖的形象一样，每个人的想法都是大 致轮廓相同，而细节差异很大。在此阶段，原型开发是一个较好的辅助手段，它将存在于大家头脑中的虚境实实在在地表达出来，一个界面，几个控件，外观形式固 定了，功能描述明确了，这就是研发部门对用户的需求理解。此时与用户再次沟通，用户基本上可以说出来：“这是我想要的”，或者“不，这不是我想要的，我要 的是……”。一般情况下，原型之后的需求沟通就实际得多，双方的理解迅速向一个折衷方案靠拢，一个可以指导研发过程的需求说明书正式诞生了。

2．需求分析之后的研发过程采用严格的需求变更管理流程。

一旦需求分析阶段结束，此后如果用户要求有新的需求加入交付的软件系统中，需要走需求变更管理流程。这个流程必须在软件项目成立之初与用户约定好，一 般的软件企业内部有需求变更的管理流程，可以向用户解释这种管理的必要性，直至与用户就此问题达成共识为止。不必担心用户不会接受，有过多次成功研发软件 项目经验的需求变更管理流程，有着它不容置疑的合理性，这正是软件企业的经验和价值所在，用户最终会理解和同意的。

需求变更管理流程各家企业有各家的做法，借用 CMM 的需求管理 KPA 来讲，需要两级需求变更管理委员会（以下简称 CCB）来处理，即产品 CCB 和项目 CCB。产品 CCB 处理涉及到产品一级的需求变化，主要体现在需要多个职能部门，多个软件项目，以及与其他产品线的协调等问题；项目 CCB 处理本项目内部 的需求变更，如不同小组之间的协调，接口变化等等。每一个需求都要经过 CCB 的审批，决定这个需求的各种属性描述是否合适，如时间紧迫性，采用的技术是否 有风险，对系统的重要程度，需求变更的波动分析，需求实现的资源状况。在与会人员对需求属性取得共识之后，规划需求实现的版本，确定时间计划。

在此提醒大家，切忌对用户提出的需求拍胸脯，在此之前可以扪心自问：“如果拍了胸脯，以后不能按时完成，我能不能负担全部责任？”这样冷静一下就不会 胡乱应承了。有一个比较好的方式减少这样的麻烦，就是在需求分析阶段之后，与用户不要亲密接触，而是按照软件项目的周期，或者双方在初期的约定，定时通报 软件研发的进展。如果软件研发采用迭代式开发，就可以在每一期交付产品发布时做这个事情，征询到的用户需求将纳入以后某期的软件版本中。

3．为将要频繁改动需求的软件项目进行版本规划。

如果考虑到软件项目比较大，周期比较长，如超过一年，其间的需求变化必然多得不可胜数，建议采用迭代式开发，为每个阶段的产品进行版本规划。第一个版 本一般是包括了软件系统最基本的功能，用户最关心的功能，它的研发过程实际上还为后续版本提供了系统构架和新技术探索。一个按时交付、质量较好的版本可以 让用户保持对项目成功的信心，并给了用户在最终产品未出来之前逐渐接近最终产品的机会，这个过程将使用户需求更加明确和完善，以提高最终产品的一次成功的 几率。所以第一个版本的完成是项目的重要里程碑，建议项目组在此时举行庆祝会来提高凝聚力，鼓舞员工士气。后续的版本规划，一般是需求的分期完善，对系统 的缺陷做持续改进。这个过程将一直持续到软件的生命周期结束。

需求管理是 CMM 二级就开始关注的 KPA，可见其重要性。关于这方面的书籍多种多样，不过最好的还是行之有效的实践经验。我在自己的项目管理中，充分应用上述经验，可以从容面对我的客户，希望它对您的项目成功有所帮助。

WEB 性能测试用例设计

性能测试用例主要分为预期目标用户测试，用户并发测试，疲劳强度与大数据量测试，网络性能测试，服务器性能测试五大部分，具体编写测试用例时要根据实际情况进行裁减，在项目应用中遵守低成本，策略为中心，裁减，完善模型，具体化等原则；

一、WEB 全面性能测试模型

Web 性能测试模型提出的主要依据是：一种类型的性能测试可以在某些条件下转化成为另外一种类型的性能测试，这些类型的性能测试的实施是有着相似之处的；

1. 预期指标的性能测试

系统在需求分析和设计阶段都会提出一些性能指标，完成这些指标的相关的测试是性能测试的首要工作之一，这些指标主要诸于“系统可以支持并发用户200个；”系统响应时间不得超过20秒等，对这种预先承诺的性能要求，需要首先进行测试验证；

2. 独立业务性能测试

独立业务实际是指一些核心业务模块对应的业务，这些模块通常具有功能比较复杂，使用比较频繁，属于核心业务等特点。

用户并发测试是核心业务模块的重点测试内容，并发的主要内容是指模拟一定数量的用户同时使用某一核心的相同或者不同的功能，并且持续一段时间。对相同的功能进行并发测试分为两种类型，一类是在同一时刻进行完全一样的操作。另外一类是在同一时刻使用完全一样的功能。

3. 组合业务性能测试

通常不会所有的用户只使用一个或者几个核心业务模块，一个应用系统的每个功能模块都可能被使用到；所以 WEB 性能测试既要模拟多用户的相同操作，又要 模拟多用户的不同操作；组合业务性能测试是最接近用户实际使用情况的测试，也是性能测试的核心内容。通常按照用户的实际使用人数比例来模拟各个模版的组合 并发情况；组合性能测试是最能反映用户使用情况的测试往往和服务器性能测试结合起来，在通过工具模拟用户操作的同时，还通过测试工具的监控功能采集服务器 的计数器信息进而全面分析系统瓶颈。

用户并发测试是组合业务性能测试的核心内容。组合并发的突出特点是根据用户使用系统的情况分成不同的用户组进行并发，每组的用户比例要根据实际情况来匹配；

4. 疲劳强度性能测试

疲劳强度测试是指在系统稳定运行的情况下，以一定的负载压力来长时间运行系统的测试，其主要目的是确定系统长时间处理较大业务量时的性能，通过疲劳强度测试基本可以判定系统运行一段时间后是否稳定；

5. 大数据量性能测试

一种是针对某些系统存储，传输，统计查询等业务进行大数据量时的性能测试，主要针对某些特殊的核心业务或者日常比较常用的组合业务的测试；

第二种是极限状态下的数据测试，主要是指系统数据量达到一定程度时，通过性能测试来评估系统的响应情况，测试的对象也是某些核心业务或者常用的组合业务。

第三种大数据量测试结合了前面两种的测试，两种测试同时运行产生较大数据量的系统性能测试；

大数据量测试通常在投产环境下进行，并独立出来和疲劳强度测试放在一起，在整个性能测试的后期进行；大数据量的测试可以理解为特定条件下的核心业务或者组合业务测试；

6. 网络性能测试

主要是为了准确展示带宽，延迟，负载和端口的变化是如何影响用户的响应时间的，在实际的软件项目中

主要是测试应用系统的用户数目与网络带宽的关系。网络测试的任务通常由系统集成人员完成；

7. 服务器（操作系统，WEB 服务器，数据库服务器）性能测试

初级服务器性能测试主要是指在业务系统工作或者进行前面其他种类性能测试的时候，监控服务器的一些计数器信息，通过这些计数器对服务器进行综合性能分析，为调优或提高系统性能提供依据；

高级服务器性能测试一般由专门的系统管理员来进行如数据库服务器由专门的 DBA 来进行测试和调优；

8. 一些特殊的测试

主要是指配置测试，内存泄露测试的一些特殊的 WEB 性能测试；

二、WEB 性能测试策略

性能测试策略一般从需求设计阶段开始讨论如何定制，它决定着性能测试工作要投入多少资源，什么时间开始实施等后续工作的安排；其制定的主要依据是软件自身的特点和用户对性能的关注程度，其中软件自身的特点起决定性的作用；

软件按照用途的不同可以分为两大类，系统类软件和应用类软件。系统类软件通常对性能要求较高，因此性能测试应该尽早介入；应用类软件分为特殊类应用和 一般类应用，特殊类应用主要有银行，电信，电力，保险，医疗，安全等领域软件，这类软件使用频繁，用户较多，也需要较早进行性能测试；一般类主要是指一些 普通类应用如 OA，MIS 等一般类软件根据实际情况制定性能测试策略，受用户因素影响较大；

1. 系统类软件

从设计阶段就开始针对系统架构，数据库设计等方面进行讨论，从根源来提高性能，系统类软件一般从单元测试阶段开始性能测试实施工作，主要是测试一些和性能相关的算法和模块；

2. 应用类软件

特殊应用：从设计阶段就开始针对系统架构，数据库设计等方面进行讨论，从根源来提高性能，系统类软件一般从单元测试阶段开始性能测试实施工作，主要是测试一些和性能相关的算法和模块；

一般应用：与使用用户的重视程度有关，用户高度重视时 ，设计阶段开始进行一些讨论工作，主要在系统测试阶段开始进行性能测试实施；用户一般重视时，可以在系统测试阶段的功能测试结束后进行性能测试；用户不怎 么重视时，可以在软件发布前进行性能测试，提交测试报告即可；

三、WEB 性能测试用例设计模型

性能测试用例设计通常不会一次设计到位，是一个不断迭代完善的过程，即使在使用过程中，也不是完全按照设计好的测试用例来执行，需要根据需求的变化进行调整和修改； WEB 性能测试用例设计模型是一个内容全面比较容易组织和调整的模型架构。

1. 预期性能指标测试用例

指一些十分明确的，在系统需求设计阶段预先提出的，期望系统达到的，或者向用户保证的性能指标，针对每个指标都要编写一个或者多个测试用例来验证系统 是否达到要求，预期性能指标测试用例主要参考需求和设计文档，把里面十分明确的性能要求提取出来，指标中通常以单用户为主；

如：对于普通的客户端，系统上传5MB 以内的文件，速度不低于2MB/S；

输入动作：选择1-5 MB 的文件并上传，用秒表计时；

期望的性能：上传的时间小于等于2.5S

实际性能：上传的时间2.29秒；

这类用例通常以手工的方式执行；

2. 用户并发性能测试用例

用户并发测试主要通过逐渐增加用户数量来加重系统负担，并通过测试工具对应用系统，各种服务器资源进

监控，用户并发测试可以是正常数量用户和特殊数量用户进行并发， 用户并发测试是系统性能测试的核心部分，涉及压力测试，负载测试，强度测试等多方面的内容.独立业务性能测试实际就是核心业务模块的某一业务的并发性能测 试，可以理解为单元性能测试；组合业务的性能测试是一个或者多个模块的多个业务同时进行并发性能测试，可以理解为集成性能测试，单元性能测试和集成性能测 试两者紧密相连合并称为用户并发性能测试；用户并发测试要求选择有代表性的关键的业务来设计测试用例，以便更有效的评测系统性能；其测试用例设计文档的基 本的编写思想是按照系统的体系结构进行

编写.

3. 独立核心模块用户并发性能的测试用例设计

完全一样功能的并发测试：主要检查系统的健壮性，从技术角度讲就是检查程序对同一时刻并发操作的处理.

完全一样操作的并发测试：基本要求是在同一时刻进行完全一样的操作，这类测试的目的是验证核心模块在

大量用户使用同一功能时是否正常工作；

相同/不同功能的子功能并发：每个不同的子功能都模拟一定的用户数量，通过工具来控制并发情况；

如发送与接收邮件模块的一个测试用例，

功能：当在线用户达到高峰时，发送和接收普通邮件正常，保证2000个以内用户可以同时访问邮件系统，能够正常发送和接收邮件；

目的：测试系统2000个以内的用户同时在线时能否正常发送邮件；

方法：采用 LOADRUNNER 的录制工具录制一个邮件发送过程测试，要监视数据库服务器和 WEB 服务器的性能，其中发送的邮件为普通邮件，附件大小不超过1MB.

并发用户数与事务执行情况：并发用户数，事务平均响应时间，事务最大响应时间，平均每秒处理事务数，事务成功率，每秒点击率，平均流量；

并发用户数与数据库主机：并发用户数，CPU 利用率，MEM 利用率，磁盘 I/O 参数，DB 参数；

并发用户数与应用服务器的关系表：并发用户数，CPU 利用率，MEM 利用率，磁盘 I/O 参数；

4. 组合模块用户并发性能测试的用例设计

组合模块的性能测试是最能反映用户实际使用情况的测试，它把前面系统中具有耦合关系的模块组合起来进行测试，可以理解为集成性能测试，组合模块并发测试可以真实反映用户使用系统的情况，可以从需求，设计文档；现场调查，系统采集数据获取用户场景；

具有耦合关系的核心模块进行组合并发测试：主要测试在多用户并发条件下，一些存在耦合关系或者数据接口的模块是否正常运行；

彼此独立的，内部具有耦合关系的核心模块组的并发测试：这类测试的对象是多个模块组，每个组相关的模块具有一定的耦合关系，组与组之间关系相互独立，主要站在用户的角度考虑问题；

基于用户场景的并发测试：选择用户的一些典型场景进行测试，测试对象不限制于核心模块或非核心模块；

组合模块用户并发性能测试的前两种类型仍然是针对核心模块的同时也关注用户场景，这样做的原因是大多数的性能问题都是由用户经常使用的核心模块一起 的；可以看出，组合模块的用户并发性能测试既关注功能测试，也关注性能测试，通过发现一些接口和综合性能方面的问题，使系统更加稳定的运行。

如下某 OA 系统组合模块的一个测试用例：

功能：在线用户数达到高峰时，用户可以正常使用系统，目标是满足500个以内用户同时在线使用系统；

目的：测试500个以内用户同时在线时能否使用比较常见的模块：公文系统，电子公告，网上论坛；

方法：采用 LOADRUNNER 的录制工具录制三项业务；业务1，在公文系统内进行打开，修改等操作；业务2，在电子公告系统内，察看发布公告； 业务3 ，在网上论坛系统内发布帖子，查看文章；每项业务分配一定数量的用户，利用 LOADRUNNER 来完成；

并发用户数与事务执行情况：业务1，业务2，业务3事务平均响应时间；业务1，业务2，业务3事务最大响应时间；业务1，业务2，业务3平均每秒事务数；业务1，业务2，业务3平均成功率；每秒点击率；平均流量；

并发用户数与数据库主机：CPU 利用率；MEM 利用率；磁盘 I/O 情况；DB 参数；

并发用户数与应用服务器的关系：CPU 利用率，MEM 利用率；磁盘 I/O 情况；

5. 疲劳强度与大数据量测试

疲劳强度测试：主要特点是长时间对目标测试系统加压，目的是测试系统的稳定性，持续时间一

般在1小时以上；疲劳强度测试属于用户并发测试的延续，因此 核心内容仍然是核心模块用户并发和组合模块用户并发，在编写测试用例时需要编写不同参数或者负载条件下的多个测试用例，可以参考用户并发性能测试用例的设 计内容，通常修改相应的参数就可实现所需要的测试场景；如下疲劳强度测试用例：

极限名称：200个用户同时使用系统的3个模块；

前提条件：测试客户端要有足够的资源；

运行时间：连续运行16小时；

测试方法：采用 LOADRUNNER 录制3个任务，然后开始对系统加压；

输入动作：任务1，任务2，任务3 ；持续时间， 任务20小时， 任务2，21小时，任务3，16小时；用户数量；现象；

大数据量测试：主要针对对数据库有特殊要求的系统进行的测试，如电信业务系统的手机短信业务；可以分为实时大数据量，主要目的是测试用户较多或者某些 业务产生较大数据量时，系统能否稳定运行；极限状态下的测试，测试系统使用一段时间即系统累计一点量的数据时能否正常的运行业务；前面两种的结合，测试系 统已经累计了较大数据量时，一些实时产生较大数据量的模块能否稳定工作；如下大数量测试用例：

功能：数据库中的短信息表可以保存所有不能及时发送的短信息，用户上线后又能及时发送已经保存的信息；

目的：

方法：

并发用户数与事务执行情况：输入说明； 事务平均响应时间；事务最大响应时间；平均每秒处理事务数，事务成功率；每秒点击率；平均流量；

6. 网络性能测试

基于硬件的测试：主要是通过各种软件工具，仪器等测试整个系统的网络运行环境，一般由系统集成人员负责 ；

基于应用系统的测试：主要测试用户数目与网络带宽的关系，通过测试工具准确展示带宽，延迟，负载和端口的变化是如何影响用户响应时间的；

网络性能测试的用例设计主要针对后一种类型，可以独立进行测试，也可以和用户并发性能测试，疲劳强度与大数据量测试结合起来，在原有的基础上采用工具来调整网络设置，从而达到监视网络性能的目的；如下网络性能测试用例；

目的： 测试系统运行在不同网络带宽条件下的性能情况，以及与并发用户数量的关系；

方法：在不同的广域网带宽下使用 LOADRUNNNER 录制邮件系统得相关事务操作脚本，然后以不同的带宽和并发用户数进行压力测试，并记录在各种用户条件下各种事务的响应情况，同时记录路由器端口的流量和其他数据；

运行时间：

并发用户数与事务响应时间：

7. 服务器性能测试

服务器性能测试主要是对数据库，WEB 服务器，操作系统的测试，目的是通过性能测试找出服务器的瓶颈，为系统扩展，优化提供相关的依据；分为：

高级服务器性能测试：在特定的硬件条件下，由数据库，WEB 服务器，操作系统相应领域的专家进行的性能测试；

初级服务器性能测试：在系统运行前面的性能测试时，通过测试工具对数据库，WEB 服务器，操作系统的使用情况进行监控，然后进行综合分析，找出系统瓶 颈；性能测试的主要目的是在软件功能良好的前提下，发现系统瓶颈并解决，而软件和服务器是产生瓶颈的两大来源，因此服务器测试一定要和前面的测试结合起来 进行；在进行用户并发性能测试，疲劳强度与大数据量性能测试时，可以完成对服务器的监控并对服务器性能进行评估；这类部分的测试用例一般不必单独编写。

四、WEB 性能测试用例设计

WEB 性能测试用例设计模型是设计性能测试用例的一个框架，在实际项目中，需要对其进行适当的剪裁，从而确定性能测试用例的范围和类别，裁减的依据是性能测试策略和测试范围；在测试用例主要框架确定后，接下来就要如何设计各类性能测试用例中具体数据。

基于用户的测试多在用户现场进行，而为了测试目的而进行的测试多在开发环境即开发团队的内部进行；为了测试目的而设计的测试用例场景主要根据测试设计 人员的经验来进行，但是仍要参考用户的实际场景，用户实际使用场景是设计所有测试用例的依据，性能测试用例设计首先要分析出用户现实中的典型场景，然后参 照典型场景进行设计。比较常见的用户场景有如下三种：一天内不同时段的使用场景；系统运行不同时期的场景；不同业务模式下的场景；各类测试用例设计的细 节：

1. 确定用户使用系统情况的方法

确定用户对系统的使用情况是设计用例具体数据的基础，后面并发用户数据设计，疲劳强度设计以及各种场景设计都要依赖对用户使用系统情况的分析，分析用户使用情况经常采用现场调查和分析系统日志两种方法；

用户现场调查：通过和用户进行沟通，可以确定用户的人员组成情况；这类方法适用于用户群体固定且目标测试系统没有投产前的情况；

分析系统日志：当用户比较分散，现场调查比较困难时，可以采用对系统日志进行分析的方法，作为对用户现场调查的补充；

2. 并发用户数量设计

设计并发用户数量前，首先要了解确定系统最大并发用户数量的方法；可以根据系统的最大使用人数或者最大在线数量来评估最大并发用户数量的方法；

极限法：取最大在线用户数作为最大并发数，这种方法适用于系统已经投产目标用户群体不确定的门户网站，可以通过分析日志来进行测试；也可以使用系统已经注册的用户数量作为系统的用户数量，按照经验公式来估算最大用户数量；

用户趋势分析：对软件生存周期内的用户未来走势进行分析，预测系统可能达到的最大使用用户数目，从而估算系统的最大并发用户数目，这种方法多用于用户数目逐渐增多的情况；

经验评估法：多用于系统的使用用户数目相对稳定而且比较明确的系统；

并发用户数量的设计基本是按照最大并发用户的数量的百分比来设计的，对于某一特定的用例，需要注意：

一按照各类用户同时递增的方式来设计用户数量，是为了按照由浅入深的方法来发现系统的瓶颈；二并发用户的最大值一般不会超过前面计算的最大并发用户数 量的 20% ，除非是为了测试系统能支持的最大并发用户数量；三设计用户数量时要考虑成本，因为每组用户数都意味着至少执行一次测试；

3. 系统不同时间段场景的设计

不同时间段的场景更接近用户使用情况，它也是设计核心模块和组合模块并发性能测试用例的基础，不同时间段场景分析的数据主要是前面的需求分析和日志分 析结果；不同时间段场景的设计基本原则有两个：一是选择典型的场景进行测试；尤其要选择场景中并发用户数目较大的场景；二是要覆盖全面，设计出的用例要覆 盖到压力可能较大的时间段；用户场景的设计一般与后面的业务模式结合起来进行；

4. 业务模式的设计

业务模式的设计是不同时间段场景设计的特例，也是设计核心模块和组合模块并发性能测试用例的基础，设计业务模式的目的是专注于某些功能模块的组合，按 时间段来设计场景通常会涉及很多模块，如果系统存在的由应用软件引起的瓶颈则很难定位，所以才抽象一些特定的业务模式来进行用例的设计；

按照业务模式和时间段的场景来设计性能测试用例时，会涉及到如何设计每个模块并发用户数目的问题，通常会取各个相关模块在24小时内最大的并发用户数目进行组合；

5. 大数据量测试用例的设计

历史数据相关的大数据量测试设计与并发用户的测试设计很类似，首先要确定系统数据的最长迁移周期，确定了系统的最大数据量后，接下来选择一些前面的核心模块或者组合模块的并发用户测试用例作为其主要内容即可；

运行时大数据量测试主要根据模拟系统运行时可能产生的大数据量来进行测试，这类测试用例通常根据实际情况去分析设计；

6. 一些特定测试用例的设计

疲劳强度测试，最大用户测试，容量测试等一些特殊的测试用例设计，根据用户的需求进行，这类用例的相关要求通常十分明确；

性能测试用例最重要的是注意用例间的关系，孤立的设计各类用例只能增加测试成本，浪费人力。性能测试用例设计人员应该追求设计既能覆盖性能测试需求，又能以较低的成本来执行测试用例；

五、WEB 性能测试用例设计总结

1. 测试用例可用性总结

对于一个比较完善的性能测试项目，经常会有一些测试用例不能执行，，因此测试完成后应该分析哪些用例不能执行以及不能执行的原因，这样可以为下次测试打好基础。

2. 用例执行效果分析

通过对用例执行效果进行分析，可以为升级或者开发新的性能测试用例提供有利的参考，不是所有的用例都能导致系统瓶颈的出现，因此应该分析哪些用例能够 发现系统问题，哪些用例执行时没有太大效果。分析那些设计好的用例不但有助于以后设计用例，还可以为再次执行提供参考：当下次测试进度压力较大时可以先执 行重要的用例，跳过那些尝试性的，不容易发现问题的用例；

3. 用例执行时间分析

分析用例的执行时间是为下次规划性能测试提供参考，由于很多用例执行时间不是特别确定，导致性能测试计划也具有一定的不确定性。通过分析用例的执行时间可以为以后的制定测试计划提供参考；

总之，性能测试用例的设计是需要通过不断分析总结才能做好，不但要分析性能测试用例的可用性，执行效果，执行时间，还应该分析用例的设计方法，设计思路等。

泽众软件工具使用技术支持

电话：021-61079698

Email：sales@spasvo.com

QQ：1404189128

MSN：spasvo_support@hotmail.com

产品租用		
下载	在线申请	详细

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

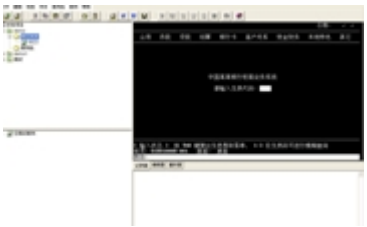
在线体验		产品租用	
企业版	免费版	在线申请	详情

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

其他测试工具

Precise Project Management

Terminal AutoRunner



PerformanceRunner

有关培训、产品购买及试用授权方法等事宜

电话：021-61079698

Email： sales@spasvo.com

QQ：1404189128

MSN： jennyding0829@hotmail.com

SPASVO