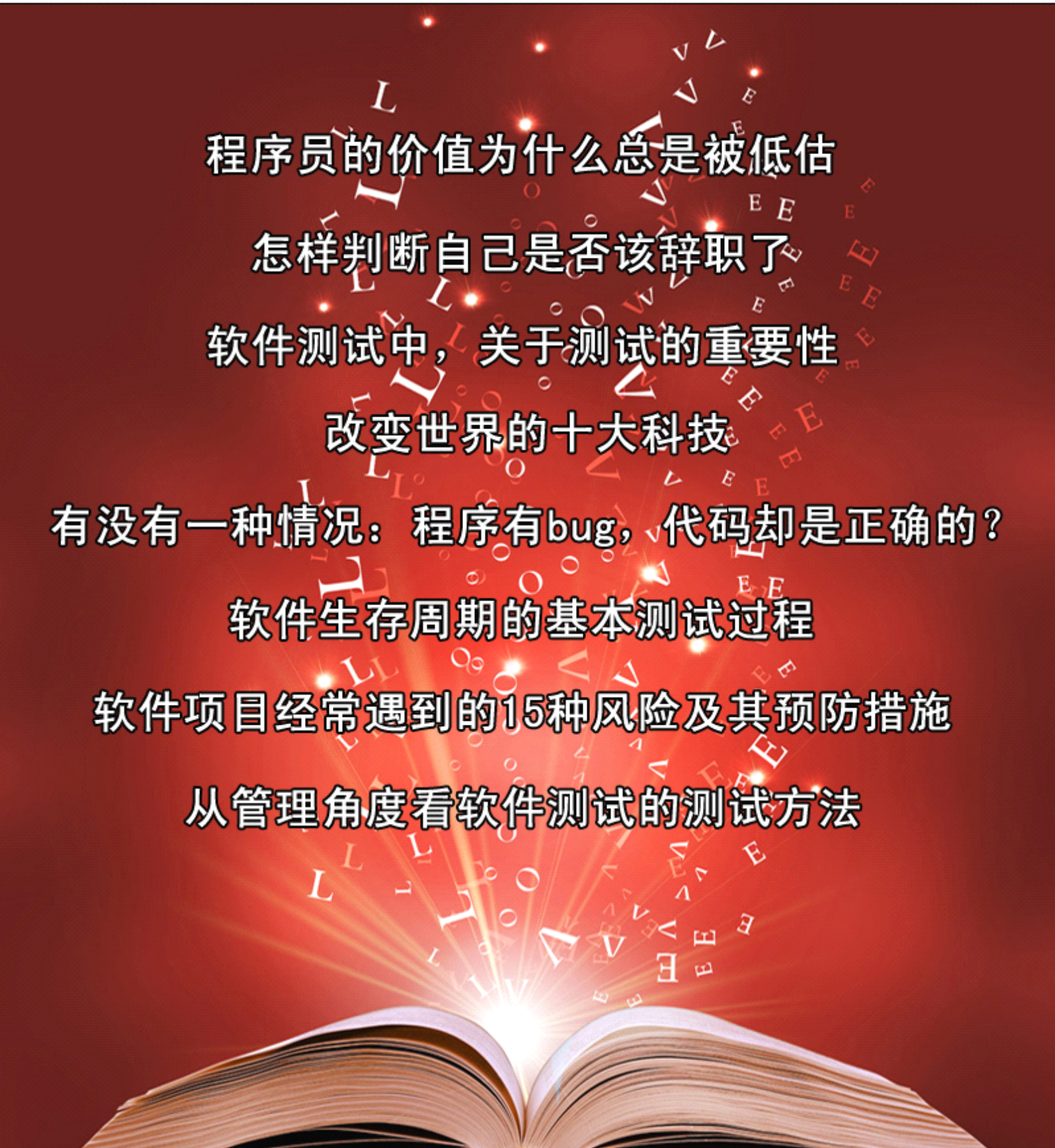
泽众软件测试网
电子期刊

SPASVO

第十三期

2013年1月



程序员的价值为什么总是被低估

怎样判断自己是否该辞职了

软件测试中，关于测试的重要性

改变世界的十大科技

有没有一种情况：程序有bug，代码却是正确的？

软件生存周期的基本测试过程

软件项目经常遇到的15种风险及其预防措施

从管理角度看软件测试的测试方法

上海泽众软件电子期刊

2013 年 1 月 第十三期

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

泽众软件——专业软件测试门户
2013 年 1 月 第十三期 电子杂志期刊

程序员的价值为什么总是被低估

在我任职于雅虎期间(大约 2001-2007)，我学会了做很多事情，但同等重要的，我还学会了如何避免做某些事情。对于后者，主要就是如何避免不公的对待技术人员。雅虎，尽管做出了很多善意的努力和明显的例外举措，仍然没有在公司内带来技术人员地位的提高。尽管我们这些技术人员创造了大量的价值，可管理层永远都是非技术人员。不可避免的，大量优秀的人才注意到了这些，忍无可忍，愤而离开。

在 2007 年离开雅虎后，我和别人合作创立的 Polyvore，从这时开始，我的一个人生主要目标就是，要建立一个高度重视技术人员、将他们作为一等公民对待的公司/社会环境。我毫不动摇的坚信，建立这样的环境能带来各种各样大量的好处。

此后我一直在思考一个问题，相对于很多的社会上能创造出巨大价值的传统职业角色，为什么软件技术人员的价值会被整体的低估?我总结出三个社会学上的原因：

一个有形，一个抽象。人们倾向于更认可他们能看得见摸得着的有形的东西。

一栋办公大楼，坐满了在办公桌前办公的人。人们能看见这些，很自然的会认识到大楼的价值。人们面对一栋高耸如云的建筑物，会欣赏它的规模和设计。人 们会习惯的认为，不管是谁在负责创造这些东西，他一定是极具价值的人。与之形成对应，人们却看不见和摸不着一个运行起来能等价于 50 个人工作的软件或一个 用来设计大厦的软件。软件和其它抽象的智力劳动是因为人们不可见，所以被社会性的低估。

缺乏测量方法。在缺乏好的方法来测量和评估一个东西的真正价值时，人们倾向于用他们的偏好来评判。人们倾向于认为大型团队=更重 要。对于我们这些曾经在“大企业”工作过的人来说，这是显而易见的，一些大型公司的管理者通过组建更大的团队来扩大他们的权利机构。更多的人=更多权力& amp;报酬。

要公平还是要平均?人们都渴望公平。但有趣的是，人们的这种渴望却无法应用到像软件工程师这样一个人产出会比另一个人高出几个数 量级的生产力异类上。一个技术人员能让一个操作过程自动化，于是 50 个人的活儿现在只需要 5 个人。人们很难接受这样的事实：一个人的报酬应该比其它人高出 几个数量级，这种悬殊看起来不公平。

还有一些实际操作上的原因。例如，用一个经理来管理一个大组织，这是必不可少的，因为少了他们事情会很快变得一团糟。这种必不可 少成了他们讨价还价的资本。而相对照的，优秀的程序员总是使得自己看起来是多余的，他们能让系统在缺少他们的情况下仍能不停的工作、生产有价值的东西。只有最开明的公司老板才能真正认识到这种人的真正价值。

如果你在做的工作正是这种创造可扩展且复杂、抽象的智力工作，很有可能你正是不被人看重并且因此被低估。

怎么办?

要杰出。并不是因为你是程序员，你就能创造出无数的价值。你能让你的团队的工作效率翻翻吗?你能让你开发的项目具有很好的扩展性吗?

会讲故事/沟通。让一个复杂抽象的题目变得易于理解和欣赏的好方法就是给人们讲故事。用一种其他人能听懂的方式解释为什么你正在研究的工作很重要。一旦人们理解了其中的原因，他们也就找到了欣赏你的工作的理由。

用合适的度量手段。采用人们易于接受的度量方法能让抽象的事物变得有具体，能够很好的让人们理解你创造的价值。如果你说你调整了 图像压缩比，使得每个图像的体积减少了 4.5KB，很少人能理解你说的是什么。而当你解释说这些压缩能使得用户界面的加载速度提高 15%，那大部分人都能 明白。如果你解释说页面加载延迟减少 10ms 意味着收入会增加 10%，这就更好了。每个人都理解收入是什么。

加入正确的公司团队。最后一点，你也许应该寻找一个开明的公司，一个能按人的实际贡献、而不是按老旧的社会标准来评价人的公司，加入他们。选择的方法可以看看这个公司是不是由技术人员创立/管理的，看看这个公司是否大量的使用各种方法评价人。去看看他们都有哪些评价标准!

你认为呢?还有什么其它原因导致程序员的价值被低估吗?你对如何选择一个正确的公司有什么好的建议?

怎样判断自己是否该辞职了

大约 6 个月前，我辞去了投资公司的工作，去和我的朋友 Adrian 一起做一个网站。我发现有必要把我的这点经验写一写，希望能给那些有跟我一样想法的人一点忠告和见解。

我想辞职。但不知道是否应该辞职?

这绝对是一个最私人的、最耗时间、最困难的你应该问自己的问题。也正因为这样，大多数人放弃回答这样的问题。

作为一个拿着工作签证的移民，我知道绝不会有一个辞职的“最佳时间”等着我，但一些已经出现的信号使得我知道给别人打工的日子将不会持续太久。不知道这些迹象是否在你身上也有?

1) 一直在做业余项目。如果你在白天工作了一整天，仍然在业余找一些项目做，很可能是：你从白天的工作中学不到足够的知识，或你对它不感兴趣。我总是在业余时间做一些小项目或经营一些小生意。最终，我认识到，我应该用全部时间来做它们。

2) 我对晋升不感兴趣，也不想去别的公司工作。在企业职场里，职位头衔和官级是让一个人走向事业成功最好的动 力。大多数的公司不会为你某个人创造一个新职位。如果你在想 5-7 年后你的职位，那你的老板的职位是离你最近的目标。而 7-15 年后，你很可能达到现在老 板的老板的地位。但那时你估计有 40 岁了。如果你对这些不感兴趣，你也许应该在另外一个“更好”的地方工作(比如大多数认为去雅虎或谷歌工作能更快的事业 有成)。但如果你已经做到了很高的位置可还是跟 CEO 或自己成立公司的距离很远，你应该认真的考虑一下在某些方面应该做巨大的调整。

3) 每月的固定工资不能激起我的激情。工资是个很奇怪的东西，它不会根据你的工作成绩而随时的上下变化，它们 只会一年变化一两次。当你刚从学校出来步入职场，你能在短时间内进步的很快，但工薪却几乎不会有多大变化。如果你拥有一个公司，或者是有一部分股份，你的 成功与否会紧密的跟你的表现和你的合伙人的表现直接相关(再加上一点运气)。

4) 你感觉自己在失去闯劲，虽然是缓慢的，但的确在发生。在大学生活和社会职场生活完全不同。如果你聪明且爱 学，你能迅速的完成学业，但工作却需要资历。如果你的工作不是你热心的，那它会慢慢磨损你的意志，虽然很慢，但却是真的。这是我的真切感受。作为一个创业 者，你有多聪明或多努力工作并不代表你能成就多少——重要的是你的斗志/锋芒/动力，这才是让你和周围其他人与众不同的地方。永远不允许自己往下沉。

5) 公司工作的机会成本 [opportunity cost，指为选择一项东西而放弃的其他机会]让你夜不能寝。这种情况，如果你还没辞职，但估计也快了。从这一点上看，你大概已经认识到了人生不是一场看看自己能往上流社会爬的多高的竞赛。你是想从无到有创造一些东西，你想试试能否做出这世界上最好的东西。

我决定辞职。可在我离职前应该做些什么?

1) 找个你能信任的合作伙伴。这几乎是可遇不可求的事。你也许有很多值得信赖的老朋友，但大多数在你想做的事情上没有相同的动力、热情、和信心。我很幸运，遇到了 Adrian(我和他的故事都在这里，你可以看一下)。

2) 选一个简单的创意开始做。我认为找一些你感兴趣的想法创意或问题，着手去做，这很重要。很难说你应该做什么，但我可以告诉你，我的 Backspaces 网站最初起源于分享一下我在纽约的所见所闻，最终发展成了更大的东西。

3) 尽可能多的还清助学贷款。如果你有一个报酬很高的工作，你可以很快的还清助学贷款。并不是因为你可以挣更多的钱，你就可以花更多的钱。不要把大把大把的票子都花在酒吧舞厅里，不要买房或买车。我在一年内偿还了所有的贷款。无债一身轻，没有经济负担，我感觉到生活的一种解脱。

4) 善待自己和家人。他们在你的整个生命中都在支持你。如果你取得了一些成就，向他们表示你对他们提供的支持的感激，因为这种机会很少，更多的时候是你对他们欠下责任和义务。

我辞职了。现在该怎么办?

很好!你已经迈出了第一步。你面前的是一条开放的，没有阻碍的道路——但这次是你坐在驾驶座上。我毫不夸张的说，在开发我的 Backspaces 的头两个月里，我干的活儿比在其它公司里干两年的都要多。你正在启程开启一段非常特别的经历，但同时也是脆弱易折的。

在下一篇文章里，我将总结一下 Adrian， Wylie 和我在过去的 6 个月里开发 Backspaces 的情况。

重点包括：3 天时间里把想法变成原型，第一周做出 beta 测试版，我们的神奇的“两周”迭代开发，开源 Instagram，我们的 3 个月的苹果应用商店计划等。

软件测试中，关于测试的重要性

通过自己的实战经验和阅读别人经验体会，我感觉我们可以总结出在生产高质量和高可靠的软硬件产品过程中，大都会走如下流程：

1）尽快制出原型产品（prototype）；

2）尽快把原型产品拿到生产环境中做检测和获得反馈；

3）尽快把在生产中得到的反馈数据和教训融入到下一步的故障修复和性能提升；

4）把改进后的产品再次尽快拿到生产环境中做检测和反馈；

5）循环以上步骤。

在这样的流程里，有以下几个方面特别值得注意：

1、尽快做出原型产品并拿到实战中（或近可能类似于生产环境中）检验。

和用户互动，了解实战应用场景，检索、阅读、和了解业界同仁的文章和经验积累，搜索、阅读、分析网上网下的关于各种语言、解决方案的文档等都是很重要的过程。但把想法和了解到的信息付之与实践之前，那些理论、分析、和研究到头来只是纸上谈兵。把东西付诸与实践后，你会加深和巩固自己对已有知识的理解，碰到和解决自己和别人没有想象或碰到的问题，把学到的理论和步骤加以调整后运用到自己的环境中去。这是 Kent Beck 在极限编程/eXtreme Programming/XP 中的一大重要、也是在实战中极端有效的一个原则。这一点和陆游在 1199 年写的《冬夜读书示子聿》里说得是同一个道理：

古人学问无遗力，少壮工夫老始成。

纸上得来终觉浅，绝知此事要躬行。

2、快速的检测和反馈需要可靠、方便、和尽可能自动化的测试机制。

原型产品出来后肯定会有这样或那样的问题：有的地方需要改进和提高，有的地方需要添加新功能等等。我们在做改进和提高的同时还要做到已有功能不退化（regression bugs）。要能够实现这一目标，唯一有效的办法是建立起有效、系统、和自动化的测试脚手架。有了这种测试机制，并配有有效的源代码管理系统，我们就敢于去做代码改进和重构，就有足够的信心来保证代码的质量和可靠性。

前一段时间读 Kent Beck 的 TDD（Test-Driven Development）书，他举了这样一个例子，我觉得挺贴切的。我们都知道用轱辘打水比从井里直接往上提省力，是打水的有效方法。假设水提到一半后你累啦或有别的情况而 hold 不住，你的努力就会前功尽弃。但如果你在轱辘上安装个齿轮机制来咬住你的进程而不至于让提上来的绳子后退，那你就会轻松、自信地打很多水。有效、自动化的测试机制就是你用轱辘打水的那个齿轮，它给你信心和回旋余地，让你大胆、放心地去修复、改进、重构、和添加新功能，让你有信心和实效来生产和提高产品，在保留既有功效的前提下扩大战果，打一场干脆利索、无后顾之忧（或担忧最小化）、不拖泥带水的漂亮仗！

在编程中，如果我们把每一个类/class、每一个函数/function 都当作一个产品的话，根据上述“尽快做出原型产品并拿到实战中检验”的法则，我们就可以很容易地理解 TDD 的原理：

a、我们对某类或某函数的界面有初步打算，即脑中已经设计出原型；

b、在实施该类或函数之前，我们写出一个单元测试案例来在实战中应用该函数/类的原型；

c、我们运行该单元测试，那么这第一个测试应当失败；

d、好，接下来我们用代码来实施该函数或类的原型，直到该单元测试通过；

e、我们接着写更多的测试来强化该函数/类的代码，直至新的单元测试通过；

f、循环以上步骤，来搭建其它函数/类。

按照这种方式来写代码，这些单元测试就构建成咬定既有成果的齿轮和脚手架。它们给我们提供质量保证和信心，给整个产品的稳定、可靠、和高质量打下了坚实的基础。

先写到这儿，该系列的下一篇，我写一下单元测试。

后记：这些生产原型产品、实战测试和反馈、之后提高的良性循环经验过程，不仅仅可以应用到软件开发方面，个人以为在硬件、工业制造、能源和环保产品的设计和研发等很多方面也能派上用场。比方说关于节能型汽车的研发、改进和提高，关于高铁、大飞机各部件的研发和应用，关于能源产品如藻类/海藻产生能源和风能、地热能、太阳能方面的可行性研究和应用，关于很多产业向创新、高附加值型转型等等，这些对于中国和很多后发国家都有很现实的指导意义。有的东西你可以在网上、书里、和各种会议上学，有的东西别人会藏着、掖着不想让你学到，但你若不自己去应用和尝试，那永远都不会成为你自己的东西。不要自高自大，也不要妄自菲薄，一步步建立起完善的研发、测试、反馈、和提高机制，有勇气和实际行动去大胆尝试，踏踏实实地走稳每一步，那么最后领先的会是你。

改变世界的十大科技

时值 IBM PC 机 30 周年纪念日，该产品的问世引发了主流个人计算机的变革，但同时也有另外一些产品对信息技术的进步功不可没，以下罗列了排名前十的改变世界的科技产品。

10.康柏便携式电脑 Portable

重 12.5 千克的便携式电脑更像是一个手提箱，还应该额外配备小拖车和提手。但是它代表着观念上的突破，实现了电脑的移动携带。虽然康柏不是当时首个打着便携式电脑的旗号进入市场的同类产品，但是这款 Portable 却在业界具有里程碑似的的意义。它使得 IBM 的 BIOS 芯片首个用于移动箱子，同时又将自己的产品同 IBM 加以区别。Portable 自身是成功的，将康柏推入了主流计算机业务，同时超过了惠普和戴尔。

9.苹果 LaserWriter

苹果的 LaserWriter 并非当时市面上首款喷墨打印机，也不是质量最高的，但是它最大的优势在于支持苹果和 Adobe。当时的苹果提供了支持图形软件和打印机的计算机，而 Adobe 则生成最终希望阅读的东西。

对大多数人而言，电脑打印机噪音大，外形笨重，而镭射打印机则相对安静、准确且打印效果美观。可以说 Macintosh 电脑、Adobe Pagemaker 和 LaserWriter 一起引领了桌面出版的革命。

8.Xerox 914 复印机

它 1959 年初一问世就被看作是变革性的设备，取代了冗重的碳纸技术，可以对文件进行多次复制并发布，同时 Xerox 公司也有次声名大噪。其复印质量也不错，价位适中，品质值得信赖，在成功的同时也使得 Xerox 公司得以支持 PARC，从而在 IT 界占据更为重要的地位。

7.苹果 iOS

这一理念初提出时，引发了很多话题，但是从硬件角度讲，iPod、iPhone 和 iPad 都不属于变革性的产品。iPod 只是数码化的索尼 Walkman，iPhone 则是标准的手机，iPad 也并非平板电脑中最棒的，支持他们快速发展的就是背后强大的软件。OS X 操作系统可谓最近几年最棒的移动操作系统，微软一直关注苹果系统开发团队的运作，但最终耗时多年，耗资上亿美元，也没能将其打败。

这场价格战中，苹果采取了铁拳政策，任何令公司不满的开发者都会被雪藏。不过 iOS 系统的优越性确实是不容置喙的。除了产品自身的简洁设计和功能强大外，滑动屏幕优越的导航特性也是值得称道的，其外观大方简洁，适合大多数消费者，整体上看是新一代设备理想的平台。

6.摩托罗拉 DynaTAC

DynaTAC 砖头式手机被看作是手机发展史上的杰出代表，在当时被认为是新奇的玩艺，售价也不低要 4000 美元，在 80 年代足够买一辆不错的汽车了。电池寿命很短，每个月要支付的通话费相对高昂。即便如此，它还是成为了当时必不可缺的昂贵玩具。如今，这一现象有所改变，iPhone 取代了它在消费者心目中的地位。回过头来看，DynaTEC 确实已经没有用武之地了，它只能持续通话 1 小时，充电就

要花去 10 小时;屏幕小也不美观，同时还很笨重。不过，如今的摩托罗拉公司也不断推陈出新，推出了一系列在市场上占据重要地位的手机产品。

5.苹果 Macintosh

乔布斯在 1984 年推出了 Macintosh，短短 7 年的时间，苹果从销售电路板和图解表发展成销售人人向往购买的电脑，发展确实令人咂舌。苹果、Adobe 和 Quark 等公司也在此产品基础上搭建了整个产业发展的雏形。同期的 Powerbook 和 iMac 也对产业形成起到了重要的作用。

4.DOS

人们谈论起早期的个人电脑都会想起台式机的箱式设计和闪烁的命令行界面，也就是所谓的磁盘操作系统 DOS。DOS 主要用于 IBM 家用 PC 机，也是 80 年代和 90 年代个人电脑普遍使用的操作系统。不过现在有了 Windows，它也退出了历史舞台，但它留下了普遍为大家使用的，且稳定的平台，后续的开发者利用它进行软件开发等，并为微软的振兴出了很大的力。

3.Linux

Linux 对开源的发展居功甚伟，并推动了一批 Linux 开发人员的成长。目前红帽将突破 10 亿美元的年收益成为第一位的纯开源公司，而 Linux 也毫无疑问将成为具有广泛影响的产品。就其表现而言，Linux 可谓是最有效和可靠的平台了，开发者们数十年的共同努力打造了这一款稳定且高效的产品。现在我们几乎可以在所有新的电脑、服务器、手机和平板电脑里发现开源软件的身影，这也将成为今后软件产品发展的主流和未来平台的基础。

2.Mosaic

Mosaic 推广了网页的概念，从而促成了因特网的产生。浏览器作为多款网络服务程序的平台，承载了更多的功能。在现代手机和平板电脑里，浏览器几乎就算是操作系统了，Mosaic 在推广使用网络的概念方面可谓是革命性的。

1.IBM PC

在电脑和商务发展史上，IBM 曾经辉煌了一代人的时间。IBM 不光适用于公司，也适用于个人，并为开发者提供理想平台。IBM 虽然并非首家面向个人用户推出产品的公司，但是它却是首个做到了行业内广泛推广的公司，并且其消费者范围之广是不可比拟的。它所推崇的理念就是办公室里的任何人，上至 CEO 下到接待员都可以从其系统中获益。

有没有一种情况：程序有 bug，代码却是正确的？

摘要：相信每个程序员都遇到过“不可能的 bug”，代码没有任何问题却出错了！问题肯定是出在操作系统上，或者是工具，甚至是因为计算机硬件的问题？!？当然，魔兽之父也不例外，他在本文中分享了多个处理异常 bug 的经验。

今天要分享的故事关于一些我职业生涯中真正遇到的 bug。

这个 Bug 是 Microsoft 的错，还是……？

Diablo 发布后几个月，StarCraft 团队开始加班来保证游戏的按时完工。那时“距离游戏发布只剩两个月了”，所以每天多加几个小时的班完全是正常的（有时候周末也得加班），有很多工作要完成，因为 Warcraft II 的游戏引擎基本上得从系统层面返工。大家故意不按日程办事（包括我自己），所以最后游戏延期了超过一年。（不清楚的可以看参考之前的文章。）

最开始的时候，我并不是 StarCraft 开发团队的一部分，但在 Diablo 发布后，StarCraft 获得了更多的人力资源，于是我加入了进来。但由于没给我安排固定的任务，我只有自己“使用武力”来驱动项目进展。

我打算实现一些有意思的功能，比如 AI，但 AI 主要还是 Bob Fitch 在做。其中一个功能是系统需要判定哪里是最适合聚集武装的地方，AI 部队会在那里集结并防守或者准备区域进攻。幸运的是，已经有成熟的 API 供我调用了，我可以直接使用路径寻找算法查询哪块地图区域是结合在一起的，以及敌人会在哪里集结重兵、准备进攻，以及加强易被突破区域的布兵情况。

我重新实现了某些组件，包括之前 Craft 系列延续的“战争迷雾”系统。StarCraft 需要拥有比 Warcraft II 更好的战争迷雾系统，因为地图的分辨率更高了。所以我们打算实现视线计算，位置更高的单位将会获得更好的使用，同时也增加了游戏战术的复杂度：如果你不知道对手在做什么，想要赢就变得更加困难。同样，躲在角落里的单位也将不会被外面的人看见。

新的战争迷雾系统是 StarCraft 项目中最令我感兴趣的地方，我需要做一些快速学习来保证系统功能实现和快速运行。上一个程序员的成果让我很不开心，运行起来非常之慢导致游戏几乎无法运行。我学习了纹理滤波算法和 Gouraud 描影，最终写出了我职业生涯中最好的 x386 汇编程序——几乎是现代游戏开发必备的技术。和大家一样，我也希望 StarCraft 最终能够开源，这样我就能看到自己最喜欢的编码成果，不过我记忆中的代码也许要更好！

但我在 StarCraft 的开发中最大的贡献在于修补 bug。因为大家都在透支着自己的极限来编写代码，以至于整个开发过程都穿插着 bug：每向前两步都会倒退一步。大多数团队成员都在做功能开发，所以我不得不花费大量时间来解决 QA（Quality Assurance，质量保证）团队捕捉到的问题。

高效修复 bug 的诀窍在于探索可靠地重现这个问题的方法。一旦你知道如何重现一个 bug，就很容易分析 bug 出现的原因，通常离 bug 修复就不远了。不幸的是，重现“will o’ the wispbug”这样偶尔才出现一次的 bug 需要几天甚至几周的努力。更糟的是，因为很难甚至不能提前预估修复一个 bug 会花多长时间，这又会在会议日程上花费更多时间。我说得最多的一句话是“嗯，还在找”。通常我会从早晨开始办公，然后整天都在做 bug 修复，有时候一天能修复数百个，有时候一个都解决不了。

有一天我正在检查一段无法运行的代码：我们本希望它能按游戏单位类型选择行为（“采伐单位”、“飞行单位”、“地面单位”等等）和状态（“活动的”、“伤残的”、“受攻击”、“繁忙的”、“闲置的”）。因为时间太过久远，我记不清具体的细节了，有几行代码可能是这样的：

```
1.if (UnitIsHarvester(unit))
2.      return X;
3.if (UnitIsFlying(unit)) {
4.      if (UnitCannotAttack(unit))
5.            return Z;
6.      return Y;
7.}
8.9....
10.11.if (! UnitIsHarvester(unit))
12.      return Q;
13.return R;     <<< BUG：永远不会执行到这行代码
```

在观察这个问题几个小时后，我猜测可能是编译器 bug 引起的，于是我又开始查看汇编代码。

对于非程序员来说，编译器只是将程序员编写的代码转换成可以由 CPU 直接执行的机器语言的工具。

```
1.// Add two numbers in C，  C#，  C++ or Java
2.A = B + C
3.; Add two numbers in 80386 assembly
4.mov       eax，  [B]      ; move B into a register
5.add       eax，  [C]      ; add C to that register
6.mov       [A]，  eax      ; save results into A
```

在查看了汇编代码后，我确定是编译器导致了错误的结果，因此向 Microsoft 发出了一个 bug 报告——也是我提交的第一个编译器 bug 报告。很快我就得到了回应，回想起来还真是让人惊讶：Microsoft 的编译器在世界范围内是如此地流行，我的 bug 报告竟然得到了回应，而且非常之快！

或许你能猜到——这不是一个 bug，虽然我看了很久的代码，但是却还是忽略了一个小错误。我很疲惫——连续数周每天 12 小时以上的工作——所以没发现这是不可能工作的代码。一个单位不能既非“采伐者”又非“非采伐者”。Microsoft 的测试人员礼貌地回复了我的失误，但那时我却感到被羞辱了，但幸好 bug 可以解决了。

顺便说一下，压缩时间是一个失败的开发模式，我在博客上很多篇文章中都提到过，这里也一样：疲惫的开发者很容易犯一些低级错误。合理地安排工作时间才能得到更高的开发效率，所以，回家休息去吧，然后明天再以饱满的精神面来编写代码！当我和两个朋友开始创办 ArenaNet 时，“没有危机”正是我们开发的哲学基础，原因之一在于我们没有在办公室置办足球桌和街机。工作-回家休息-再工作！

这回 bug 真的出在 Microsoft 身上了！

几年后，在开发 Guild War 时，我们发现了一个灾难性的错误会导致游戏服务器在启动时崩溃。不幸的是，我们编程团队日常使用的“dev”（development）分支没有任何问题，测试团队最后验证用的“stage”（“staging”）分支也没有问题。唯一出现问题的地方在于“live”分支，也就是玩家使用的分支。我们把这个版本“推送”给了终端用户，于是他们都玩不了游戏了！WTF！

数千名愤怒玩家要求快点修复这个问题。幸运的是，我们可以把代码回滚到上一个版本，而这花不了多长时间，但仍然需要查清楚是哪里出了问题。最终我们发现是多个错误共同导致了这个问题，这在编程中很常见。

Microsoft Visual Studio 6（MSV6）中的有一个 bug，而我们正是用的 MSV6 编译的游戏。对！不是我们的问题！自然，我们的测试无法找出问题。Whoops。

在特定的情况下，该编译器会在处理模板时生成错误的结果。模板是什么？它们很有用，但是会让你很头痛；有胆量的话就看看这个。

C++是一个很复杂的编程语言，所以它的编译器有 bug 并不是什么奇怪的事情。实际上，C++比其它主流语言复杂得多，你可以看看 C++和 Ruby 复杂度对比图。Ruby 功能全面，所以很复杂，但如图所示，C++要复杂一倍，所以在其它一样的情况下，C++的 bug 也会多一倍。

在研究这个编译器的 bug 时，我们发现其实自己早就知道这个 bug，而且 Microsoft dev 团队已经在 MSVC6 Service Pack 5（SP5）中修复了这个问题，所有的程序员都已经升级到了 SP5。悲剧的是，我们忽略了构建服务器，而它是集合代码、插图、游戏地图、等组件，并最终组成游戏的地方。所以，虽然游戏在每个程序员的计算机上能够正常运行，却在构建服务器上出了巨大的问题，因此也只有 live 分支有问题。

为什么只有 live 版本？嗯，理论上所有分支（dev、stage、live）同样有机会消除这样的 bug，但实际上还是有区别的。首先，我们在 live 版本取消了很多编程和测试团队使用的调试功能，这样可以节省时间和金钱，但同样也会孕育出巨大的灾难，甚至导致游戏崩溃。

我们想确保 ArenaNet 和 NCsoft 的员工在游戏中没有作弊的机会，因为每个玩家都应该在一个公平的游戏平台上娱乐。很多 MMO 公司都曾有员工因使用“GM 特权”而被开除的情况，因此我们想通过删除该功能来解决这个问题。

另外就是我们清除了一些“sanity checking”代码，它们本是用于验证游戏是否在正常运行。这类代码被程序员称为断言（asserts or assertions），用来保证游戏状态在计算之后是合适并且正确的。断言会造成性能上的损失：每次例行检查都会花费时间；如果代码中嵌入了过多的断言，程序运行就会变得缓慢。我们在 live 版本中禁用了断言以降低游戏服务器的 CPU 利用率，但无意间导致 C++编译器生成了错误的结果，最终造成游戏崩溃。

这个 bug 修复起来很简单，只需要升级下构建服务器就可以了，但最终我们决定保持断言是开启状态，即使在 live 版本中也是如此。为了保证不再出现这样的 bug，我们放弃了节省 CPU 利用率（或者更准确地说，未来需要的计算机数）。

经验总结：每个人，包括程序员和构建服务器，都应该使用同样的工具！

也可能是你的计算机坏了

鉴于之前的 bug 误报，我实在是不好意思再向 Microsoft 提交 bug 报告了，开始怀疑是不是我或者其他组员的代码有问题。

在 Guild Wars（GW）的开发期间，我接收到并且检查了很多玩家返回的 bug 信息。GW 的玩家可能会记得（最好不记得），当游戏崩溃时会提供向我们的“实验室”发送 bug 报告的信息供分析。收到这些信息后，我们会筛选 bug 并并决定由谁来处理。这些 bug 的原因、程度都各不相同，有的没有专人负责，而是我们轮流负责处理。

我们经常会遇到挑战信仰的 bug，总是让人抓狂。bug 的出现总是有原因的，我们首先可以假设可能的原因，并不涉及空间-时间统一性的重新定义。它看起来像是因为内存破坏或者线程竞争问题，但已知的信息告诉我们这不大可能。

Mike O’Brien，ArenaNet 的联合创始人之一，也是一名骇客，最终想到这可能是电脑硬件故障引起的，而不是编程问题。更重要的是，他还给出了测试这一假设的方法，简直是一个杰出的科学家。

他写了一个模块（“OsStress”），可以分配出一块内存，在那块内存中执行计算，然后和已知答案做比较。他把这块“压力测试”代码添加到主要的游戏循环中，这样每秒将执行 30-50 次这样的验证步骤。

在正常的计算机中，这样的压力测试不会出问题，但有大约 1%运行 GW 的计算机会出问题! 1%听起来不是个很大的数字，但当有 100 万玩家时，意味着每天会有至少 1 万个崩溃 bug，这样编程团队将需要几周来研究这一天的 bug！

压力测试失败时，GW 会关闭游戏并打开一个“硬件问题”的网页，以此提示用户哪些常见的原因会导致这样的错误：

Memory failure: in the early days of the IBM PC， when hardware failures were more common， computers used to have “RAM parity bits” so that in the event a portion of the memory failed the computer hardware would be able to detect the problem and halt computation， but parity RAM fell out of favor in the early ’90s. Some computers use “Error Correcting Code” (ECC) memory， but because of the additional cost it is more commonly found on servers rather than desktop computers. Related articles: Google: Computer memory flakier than expected and doctoral student unravels ‘tin whisker’ mystery.

Overclocking: while less common these days， many gamers used to buy lower clock rate — and hence less expensive — CPUs for their computers， and would then increase the clock frequency to improve performance. Overclocking a CPU from 1.8 GHz to 1.9 GHz might work for one particular chip but not another. I’ve overclocked computers myself without experiencing an increase in crash-rate， but some users ratchet up the clock frequency so high as to cause spectacular crashes as the signals bouncing around inside the CPU don’t show up at the right time or place.

Inadequate power supply: many gamers purchase new computers every few years， but purchase new graphics cards more frequently. Graphics cards are an inexpensive system upgrade which generate remarkable improvements in game graphics quality. During the era when Guild Wars was released many of these newer graphics cards had substantially higher power needs than their predecessors， and in some cases a computer power supply was unable to provide enough power when the computer was “under load”， as happens when playing games.

Overheating: Computers don’t much like to be hot and malfunction more frequently in those conditions，which is why computer datacenters are usually cooled to 68-72F (20-22C). Computer games try to maximize video frame-rate to create better visual fidelity; that increase in frame-rate can cause computer temperatures to spike beyond the tolerable range， causing game crashes.

在大学期间，我的 Mac 上有个扩展硬盘，经常会在春夏因为温度过高而出故障。因此我买了一个 4 英尺长的 SCSI 电缆，足够从我的计算机连到冰箱（我叫它 Julio）了，并且全年将它存放在冰箱里，后来就再也没出过问题！

于是每当 GW 支持团队收到过热问题的反馈，都会鼓励玩家去改善空气流动、增加散热风扇，或者清理一下计算机中的灰尘，这些做法通常都很奏效。

这个计算机压力测试不仅完成了它的使命，还获得了丰厚的回报：我们能够识别电脑产生虚假的 bug 报告并且忽视这些崩溃。一周内有数百万玩家在玩我们的游戏，即使很低的故障率也会产生很多 bug 报告，以至于超过编程团队的处理极限。通过这些减少 bug 反馈信息的措施，编程团队能够更专注于开发玩家想要的新功能而不是去给 bug 分类。

当然还有更多 bug

我认为现在还没有到计算机程序不会出现 bug 的阶段——用户期望的增长要比高级程序员的数量更快。Warcraft I 大约有 20 万行代码（包括内部工具），而 GW I 的代码量已经超过了 650 万行（也包括工具）。尽管可以降低每行代码中 bug 出现的几率，但代码行数的巨大增长仍然会导致问题数的剧增。但我们仍在努力。

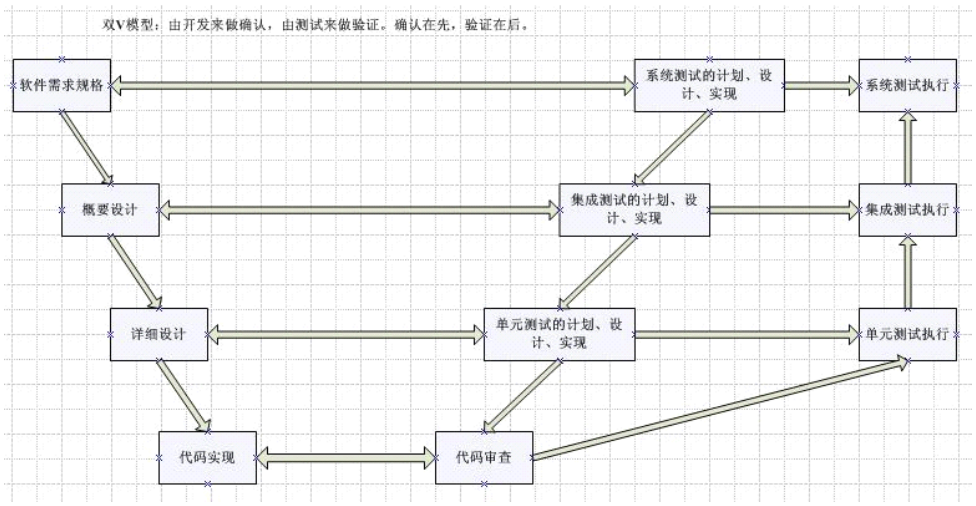
最后，我想分享一下在 Blizzard 时的同事——Bob Fitch 的一句玩笑话，他说道：“所有代码都可以优化，但所有程序都有 bug，因此所有程序都可以被优化为一行代码，只不过无法运行。”这就是为什么我们总有 bug。

软件生存周期的基本测试过程

一、测试过程

描述测试活动的过程，通常使用双 V 模型做为理论模型和依据。双 V 就是验证和确认。

验证是否做了正确的事情，确认做的事情是否正确。具体如下图：



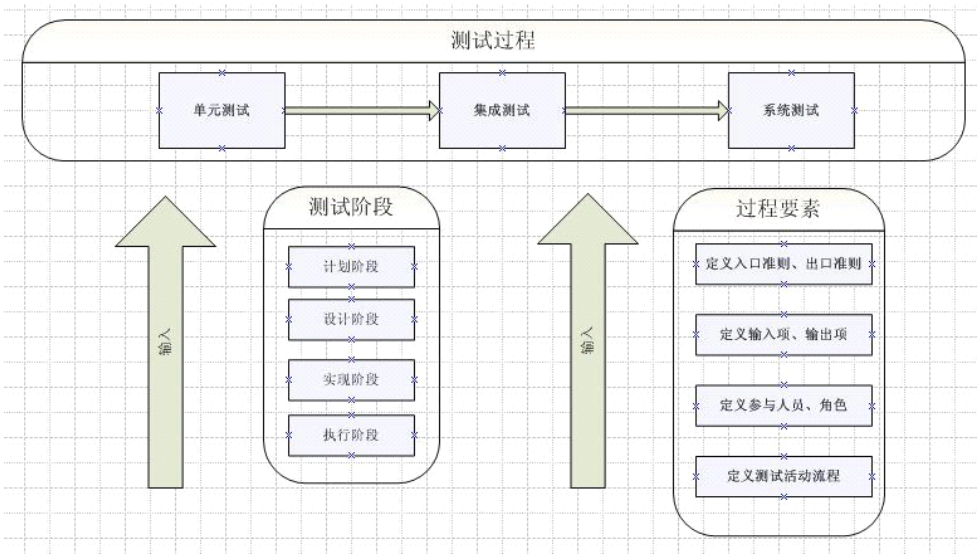
测试过程模型

二、测试过程要素

描述测试过程中的基本要素：入口准则：前提、出口准则：完成标准、输入：参考资料、输出：成果物、角色：人员责权、

活动：工作步骤、规程。以上六个要素是测试过程中最基本的、必须要有的。此外，包括评审、培训、工具使用等也是可选的要素。

具体如下图：



三、生存周期

软件的生命周期始于需求、终于产品或项目下线。包括基本过程、支持过程、组织过程。具体到软件开发的生命周期，

包括如下的阶段：计划、需求分析、设计、实现、测试以及运行维护。

1、软件生存周期基本过程

生存周期基本过程包括 5 个过程，这些过程共各主要参与方在软件生存周期期间使用。主要参与方是发起或完成软件产品开发、

运行或维护的组织。这些主要参与方有软件产品的需方、供方、开发方、操作方和维护方。

基本过程：

a）获取过程：为获取系统、软件产品或软件服务的组织即需方而定义的活动；
b）供应过程：为向需求提供系统、软件产品或软件服务的组织及供方而定义的活动；
c）开发过程：为定义并开发软件产品的组织即开发方面定义的活动；
d）运作过程：为在规定的环境中为其用户提供运行计算机系统服务的组织即操作方面定义的活动；
e）维护过程：为提供维护软件产品服务的组织即维护方面而定义的活动。即对软件的修改进行管理，使它保持合适的运行状态。

该过程包括软件产品的迁移和退役。

2、生存周期支持过程

a）文档编制过程
b）配置管理过程
c）质量保证过程
d）验证过程
e）确认过程

f）联合评审过程
g）审核过程
h）问题解决过程
i）易用性过程

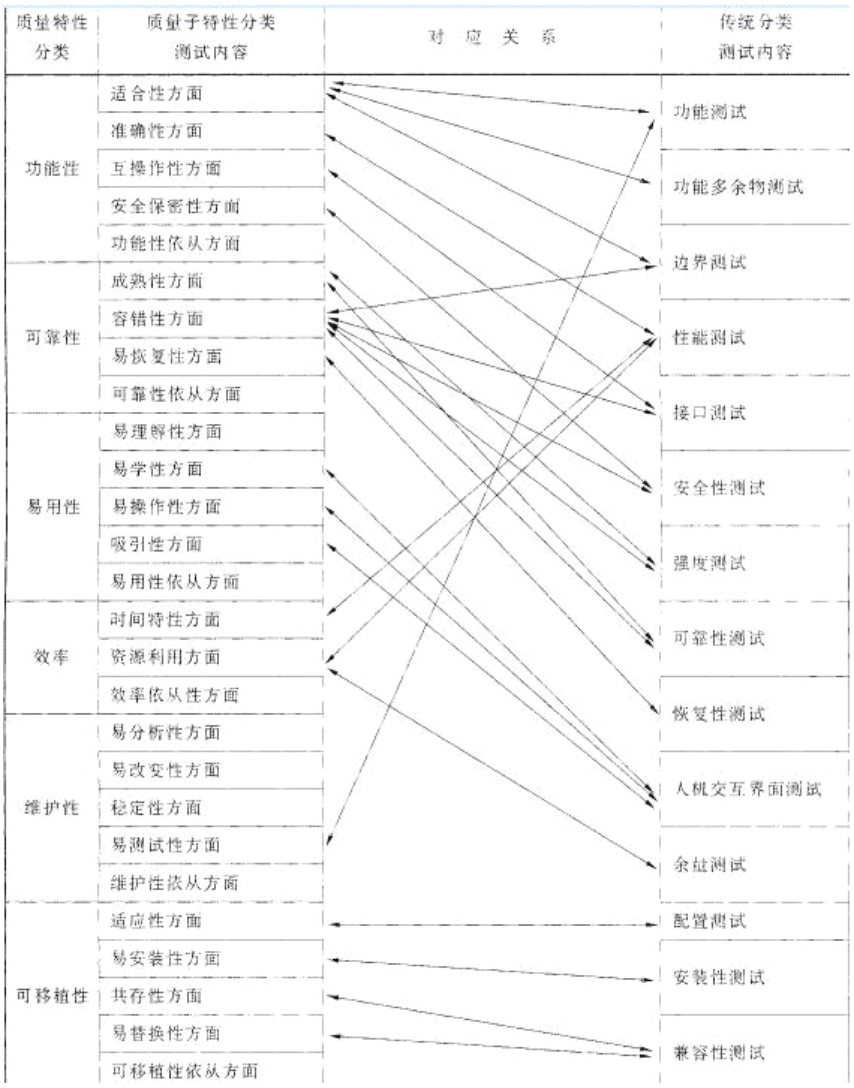
3、生存周期组织过程

a）管理过程
b）基础设施过程
c）改进过程
d）人力资源过程
e）资源管理过程
f）重要大纲管理过程
g）领域工程过程

参考：GBT 8566-2007　信息技术软件生存周期过程

四、测试方法与质量特性的对应关系

（也就是使用何种测试方法对那些内容进行测试的思路）



软件项目经常遇到的15种风险及其预防措施

(1)合同风险　　预防这种风险的办法是项目建设之初项目经理就需要全面准确地了解合同各条款的内容、尽早和合同各方就模糊或不明确的条款签订补充协议。

(2)需求变更风险　　预防这种风险的办法是项目建设之初就和用户书面约定好需求变更控制流程、记录并归档用户的需求变更申请。

(3)沟通不良风险　　预防这种风险的办法是项目建设之初就和项目各干系方约定好沟通的渠道和方式、项目建设过程中多和项目各干系方交流和沟通、注意培养和锻炼自身的沟通技巧。

(4)缺乏领导支持风险　　预防这种风险的办法是主动争取领导对项目的重视、确保和领导的沟通渠道畅通、经常向领导汇报工作进展。

(5)进度风险　　预防这种风险的办法是分阶段交付产品、增加项目监控的频度和力度、多运用可行的办法保证工作质量避免返工。

(6)质量风险　　预防这种风险的办法一般是经常和用户交流工作成果、采用符合要求的开发流程、认真组织对产出物的检查和评审、计划和组织严格的独立测试等。

(7)系统性能风险　　预防这种风险的办法一般是在进行项目开发之前先设计和搭建出系统的基础架构并进行性能测试，确保架构符合性能指标后再进行后续工作。

(8)工具风险　　预防这种风险的办法一般是在项目的启动阶段就落实好各项工具的来源或可能的替代工具，在这些工具需要使用之前(一般需要提前一个月左右)跟踪并落实工具的到位事宜。

(9)技术风险　　预防这种风险的办法是选用项目所必须的技术、在技术应用之前，针对相关人员开展好技术培训工作。

(10)团队成员能力和素质风险　　预防这种风险的办法是在用人之前先选对人、开展有针对性的培训、将合适的人安排到合适的岗位上。

(11)团队成员协作风险　　预防这种风险的办法是项目在建设之初项目经理就需要将项目目标、工作任务等和项目成员沟通清楚，采用公平、公正、公开的绩效考评制度，倡导团结互助的工作风尚等。

(12)人员流动风险　　预防这种风险的办法是尽可能将项目的核心工作分派给多人(而不要集中在个别人身上)、加强同类型人才的培养和储备。

(13)工作环境风险　　预防这种风险的办法是在项目建设之前就选择和建设好适合项目特点和满足项目成员期望的办公环境、在项目的建设过程中不断培育和调整出和谐的人文环境。

(14)系统运行环境风险　　预防这种风险的办法是和用户签定相关的协议、跟进系统集成部分的实施进度、及时提醒用户等。

(15)分包商风险　　预防这种风险的办法一般是指定分包经理全程监控分包商活动、让分包商采用经认

可的开发流程、督促分包商及时提交和汇报工作成果、及时审计分包商工作成果等。

从管理角度看软件测试的测试方法

很多公司都客观的讲究时间管理，人员管理，绩效管理。那么如何做好对于测试流程的管理呢?

随着敏捷模式的不断引入，测试的小迭代化更加明显，很多的情况下测试作为发布流程的最后一环，却是发布的关键一环，如何在短至几个小时内做好测试的管理呢?技术，管理，流程缺一不可。有好的技术人员，没有流程和管理的支持将会不断的流失。

切合 Test2.0 的理论测试更为关键的一点需要逐渐的向 SQA 的方向发展，测试更多的作为流程的监督者。

从项目管理的四要素 范围，时间，质量，成本 出发，下面简要介绍下对于测试管理的个人理解。

范围：测试范围决定了测试的时间，间接的决定了测试的质量

如何确定测试范围，这也是测试的一种艺术——精准测试

精准测试说白了就是针对代码，测试点关注于 release 间的 diff 而不是全局 case，更多关注增量代码和差异代码。

这样的流程需要管理流程上的配合，当前 release 只包含上线功能，不包含已废弃，开发 ing 的工程代码。

时间：一直是牵制测试人员比较大的一个因素
对于大多数的业务测试人员来说时间是最大的敌人
业务线人员如何能将时间利用到极致呢？——精准测试配合自动化测试

精准测试将本次测试的范围精确化，更好的确定关联模块。

自动化测试更多的是在持续集成代码 checkin 之后，自动触发将自动化测试通过作为迭代的代码准入原则。

质量：如何评估测试效率

非常难评估一个测试人员的贡献，测试漏测率，BUG 发现率，这些客观的衡量功能测试的指标也已经越来越不明确了。

测试人员有更多的思考和发展，测试都是以发现 BUG 为目的的，但是实现的途径多种多样，自动化已经成为互联网的趋势，但是自动化够了吗?平台化，工具化将自动化的工作移交给普通的 PD，PM，DE 等。

所以一个 SDET 开发一个工具用了一个月却简化了其他业务线 20%的工作时间，这样的评估需要很好的衡量。

我个人认为测试的质量分为两类，当前规避 BUG 有效率，未来规避 BUG 有效率。结合 Test2.0 的

理论，BUG 更多的不应该发现更应该规避?

从管理流程的角度，测试必须从产品 idea 形成初期进入测试，这样更好的为产品进行风险评估及自动化工具开发。担负客户成功的责任，测试的客户就是 PM，PD，DE，UED 等。更好的从客户的角度出发，换位思考。

真正测试的质量也应该由一组数据分析得出，可以是线上故障率，high level BUG 数，代码覆盖率等多元 metrics 综合计算得出。

成本：一般测试管理最关心的 hc，不是本次讨论的重点。

如何将测试从管理的角度执行完善，在各个环节都需要技术和流程的支持，好的流程是成功的一般，rules rather than codes。

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