泽众软件测试网
电子期刊

SPASVO

第二十七期
2014年3月

2013年软件领域因缺陷导致的五大事件

软件测试工作中的不靠谱行为

测试工作的三个阶段

如何一步一步从 QA 到 EP

到底谁需要技术写手呢？

让工作更轻松的技巧

Gmail测试工程经理Ankit Mehta的访谈

女程序员的时间都去哪里了？

上海泽众软件电子期刊

2014 年 3 月 第二十七期

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
2014 年 3 月 第二十七期 电子杂志期刊

2013 年软件领域因缺陷导致的五大事件

摘要：软件正在吞噬世界，软件已遍布在我们生活的各个角落。话说软件除了给人们带来了便利之外，也会偶尔出现点小问题，比如系统瘫痪、用户无法注册网站、无法进行购票等问题。

时间飞逝，农历的2013年也快要跟大家说再见了。在软件领域里，2013年是个不平凡的一年，越来越多的科技产品走进了人们的生活，移动、大数据、云计算的快速发展给许多企业带来了各种机遇和挑战。作为程序员，除了要回顾自己一年所参与的项目之外，还得了解去年的业内动态、发生过哪些令人深思的互联网事件。

本文作者总结2013年软件领域因软件缺陷导致的五大事件，提醒所有的各位程序员在开发项目时，除了注重功能的实现，还得考虑一些其它因素，例如性能、代码规范等等。下面让我们一起来看下。

1.美国联航系统免费发放机票

2013年9月12日，美联航售票网站一度出现问题，售出票面价格为0-10美元的超低价机票，引发乘客抢购。大约15分钟后，美联航发现错误，关闭售票网站并声称正在进行维护。大约两个多小时后，该公司购票网站恢复正常，并且承认已卖出的票有效。

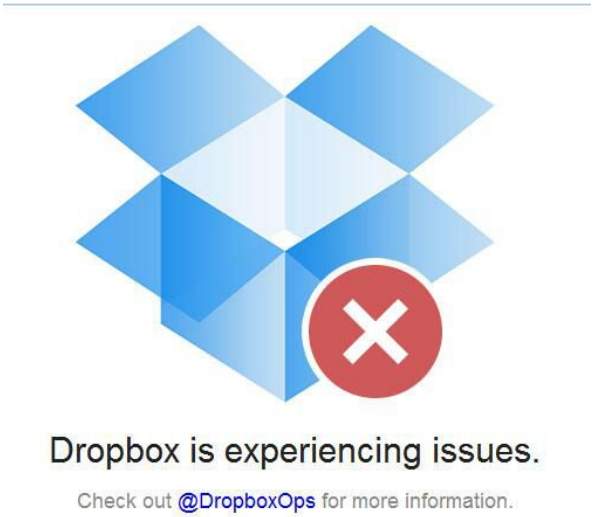
但是事情并没有结束，一个月后，注册常旅卡的用户在取消过程中，只需花几美元即可购买实际价值为几千美元的机票。美联航发飙了，指责发现该 bug 的用户，认为有人“有意”操作网站，因此不承认这些票。通常，软件公司会对发现重大 bug 的用户给予一定的奖励，但这样的事情并未在美联航身上发生。

@united 你不能指责用户在您的网站上的购票行为，即使是使用了“漏洞/缺陷”购票。测试并修复它吧。

— Gregory Mooney (@BearQuality)

2.Dropbox 宕机一小时

你在把数据上传到公共云时，你是否担心过数据会被黑客攻击，有一天你无法访问这些数据，虽然很恐怖，但噩梦还是变成了现实。



云端存储解决服务提供商 Dropbox 在2013年5月份发生了一次宕机事件，事件持续1小时，用户无法使用 Dropbox，在页面上显示无法链接服务器。而就在2013年，Dropbox 共发生过两次宕机事件，虽然官方回应并非遭到黑客攻击，但仍然引发不少用户的担心。

现在，越来越多的用户喜欢把数据、资料都上传到云端，因为它实在是备份文件或远程开发、测试软件的一个很好选择。尽管有这些事件发生，但事实上，把数据放在云中远比保存在家庭电脑上安全的多，所以选择一个优秀的云端存储网站/服务商也是非常重要的。

3.CBOE 事件

由于发布时间问题，程序员明明知道软件存在一些缺陷，但仍然会发布。对于轻微的软件错误，这倒没什么，但如果该缺陷影响到用户对产品的使用，那发布它，就相当于搬石头砸自己的脚。

CBOE（Chicago Board Options Exchange）是美国最大的期权交易所，在2013年4月，CBOE 因软件故障引起延迟开盘事件，事故从早上8:30开始，直到中午12点才全部开盘。

导致此事件的缺陷主要源于一个产品维护功能，是由于该功能中针对一个期权类进行标识符号改变而引起的。在事件结束后， CBOE 因监管失败被罚款600万美元。

4.FSSA 信息泄露事件

联邦条例规定，技术的加密和授权主要是用来安全地发送和接收保密信息。然而不幸的是，在2013年5月， 印第安纳州家庭和社会服务管理局（FSSA）泄露了用户的私人信息，其中包括社会安全号码以及错误的收件人信息，其中大约有188000人信息遭到公开。

FSSA 花了一个月的时间来修复这些错误，并且被公开信息的用户也的确因此受到了影响。现在，不仅用户在使用该系统时会陷入了困境，而且他们更担心个人信息会被陌生人利用或者操作来攻击他们。

其实，像这种错误应该在数据系统开发和测试过程中就应该快速发现，并且想办法解决，而不是事后才花那么长的时间解决。

5.Healthcare.gov 灾难

HealthCare.gov 是美国联邦健康保险交换系统的核心，该网站自2013年10月1日开通运营以来一直遭受各种问题的困扰，比如用户注册失败、浏览器崩溃、性能、数据问题等等。CSDN 研发频道曾对该网站架构进行过具体的分析，大家可以移步过去看看： 传说5亿行代码的 Healthcare.gov 网站的架构。

承包商表示，他们仅有两周的时间来测试该系统，实际上需要几个月的时间才可以完成，因此，网站崩溃的原因之一便是测试时间太短。更糟糕地是，共和党人试图利用这个原因来解释奥巴马医改系统为什么不能工作。

事实上，作为典型的政府项目，它失败的原因有很多，Ars 网站便列出了 HealthCare.gov 的七宗罪：

承包商太多，导致最终结构过度复杂；

整个项目都依赖于 Experian 提供的数据，而无论是政府还是承包商都对数据质量没有发言权，数据出现问题会导致整个网站遭遇问题；

从未经过时间检验的全新结构；

需求不断变化，设计不断改变；

由于需求直到上线最后一分钟还在改，导致网站根本无法进行全面的测试；

不是逐步增加新特性，网站将全部功能一下子推出，由于没有办法在高负荷下测试网站的性能，政府无法根据测试结果扩建基础设施；

网站没有有效的方法在多层组件中管理 bug 跟踪，没有方法识别问题的根源。

总结

大家每天都在与软件打交道，我们的世界已经整个运行在软件中。但随着市场对软件质量的要求不断提高，软件测试将变得越来越重要。

同样，从这些事件中我们也应该提高自己的上网安全意识，提高警惕，尤其是输入私人信息时。以上都是国外的一些互联网事件，那么在国内，2013年又有哪些引起大家关注的事件呢？比如最近讨论比较火的12306网站问题、全国 DNS 解析出现大规模故障等事件，大家不妨一起来讨论讨论。

软件测试工作中的不靠谱行为

昨天看了个电影《摇滚教室》，内容是一个不靠谱的摇滚乐狂热屌丝，偶然机会充当某精英小学的代课老师，他不教孩子科学文化知识，却教孩子们摇滚乐，组建 school of rock 乐队参加比赛。

大家都会觉得不靠谱吧，在学校的任务就是学习，考出好成绩，这就是我们从小接受的教育。延续到工作中也是如此，作为一个测试工程师，就要能够努力找出最多的 Bug，做出强大的自动化测试平台，构建完美的测试流程和质量保证体系…

下面我们来看下什么才是不靠谱的行为

Bug 数量衡量绩效

Bug 数量衡量绩效的方法五花八门，曾经所在公司的研发团队，在办公室的墙上订了一个板子，展示每个测试工程师每周提交的 Bug 数量和组内排名，还有开发工程师的修改 Bug 数量和组内排名；还有我的一个前同事所在团队，部门经理要求每个测试工程师在周报中列出本周提交的 Bug 数量，提交 Bug 少或者甚至没有提交的会被谈话。

我相信以 Bug 数量来衡量绩效的危害现在已经有越来越多的人体会到了，一味追求缺陷数量，必然会陷入发现的缺陷越来越多，产品质量并不能提高多少，反而进度无限延期的窘况。就不在这里做详细解析了。

要说这种行为也不是完全不靠谱，在抗日时期，用小米加步枪打飞机也是很有效的办法。那么在测试团队组建初期的摸索阶段，或者为了应付短期着急上线的项目或产品，靠这种土方法来激励士气是没有错的。如果把这种应急的手段作为长期的目标，那么就完全不靠谱了。

我见识过一些项目经理或测试经理，当初他们作为员工期间，曾经被组内公认是提交 Bug 最多的，或者是修改 Bug 最快最多的，因为工作绩效的突出升级做了管理。那么就是说有一些现在的测试经理甚至总监，很多是从当年 Bug 数量大战中拼出来的，没准在2007或2008年因为提交 Bug 最多获得过公司内最优测试工程师。可是请反思一下那时的生存法则还能适用于现在的环境吗？

构建强大的自动化测试平台

我想很多人都常常听到管理层的或者项目组里测试老人们，对于自动化测试提出各种各样的需求。比如“自动化用例覆盖率达到100%，能够适用于 Web 系统、PC 端程序、数据库、接口甚至移动应用的测试，能够自动生成测试用例，测试人员能够快速上手，提供界面化管理和操作平台，测试人员不需要懂代码就可以实现自动测试…”

如果你做的轻量级自动化测试框架无法实现这些需求，又需要测试人员具备代码能力才能使用，可能会引来管理层的质疑“那要你们有什么用？价值体现在哪里？不要给我说什么构建质量安全网这种防御性质的价值体现…”

我想追求高大上的自动化平台，没有错，如果有公司财力物力和人力的支持，这种造福人类的事情还是可以做的。

可是为什么会形成这种现状呢？我想是一些别有用心或者好大喜功的人，向喜欢数据说话的管理层

吹嘘太多了。如果一个公司开发流程还处于小作坊阶段，开发人员还是靠堆砌代码来开发项目，测试人员还是靠堆砌 bug 来实现价值。领导就要小心那些像你吹嘘什么高大上的自动化测试平台的人了。

构建完美的测试流程和质量保证体系

当初为了找工作为了应付笔试，看了很多关于测试流程的相关资料，来应付这些笔试题“V 模型包含哪些阶段？测试包含哪些阶段？测试应该从什么阶段介入项目？列举测试评审的重要性？…“。 工作多年才发现，实现 CMMI5的软件服务企业，项目还是小作坊流程的比比皆是。其实更痛苦的事是，突然公司要所有项目走 CMMI 的体系，PMO 来监控和考核项目的流程是否规范。

然后是写了一大堆文档用例，写了一大堆文档，进行了无数的评审；最后是文档用例太多太细，维护工作量大而放弃维护成了摆设；什么测试计划、测试规范、测试策略、问题记录列表，一个项目下来形成了太多碎片化的文档，没有时间去归纳整合，为了应付而浪费了真正思考和实践的时间；参加评审仍然是一言堂，管理层决策，发表尖锐的评论和建议常常被和谐，所以只好针对具体细节展开无休止的讨论，最终以和稀泥的方式完成评审。

形成这种情况的原因我想和一部分人的狭隘思想有关：作为测试管理者，总要做一些能够实实在在体现测试部门价值的东西，可是光靠测试团队在短期内真正地提高项目或产品质量是不可能的，那么通过建立形式上的规范和体系便可以快速确立自身的地位。

反思

回到电影《摇滚教室》，他虽不经意，却带给孩子们真实的音乐体验旅程，孩子们收获的成长并不是考试分数的提高，却是心智发展历程中的一次飞跃。某些被大家认同的靠谱行为，我们是否应该真正反思下是否值得去做。

两年前关于是否需要专职的 QA 的热烈讨论，我作为一个测试工程师，一开始思想上是坚决站在维护测试正义的一方。我当时觉得你技术大牛也不能因为测试工程师不会测试某个随机算法程序，就否定我们测试工程师和测试工作的价值。

这两年经过互联网的发展，移动互联网的兴起，传统的研发测试方式都在受着挑战。也许正如有些人所说专职测试工程师这角色真的就会不存在了，或者更靠谱的说应该是原来的测试角色会被重新定义，执行测试工作的可能是开发工程师或者是计算机，也可能是用户。

那么对于目前的测试工程师，我们不应把自己局限于“Tester“这个角色本身，更不要为了指标和形式去做一些好看却不实用的事情，而应该去真正关注项目或产品这个交付物本身。不要总想着我测试工程师应该站在用户角度攻击项目或产品，甚至是攻击开发团队存在的问题，而是应该站在整个团队的角度，把产品或项目作为自己的孩子一样，从技术细节和实现细节上去分析考量，做一些真正对项目对团队有帮助的工作。

测试工作的三个阶段

上一篇里我们讨论了测试的必需性，如果大家目前还在公司里做着测试的工作，那就说明还是落在必需的范围里面，或者至少一段时间是吧。那接下来我们看下既然需要做测试，需要做哪些事情。

基于我自己的一些理解和观察，我试图把测试工作的层次分成三个阶段，越到后面涵盖的范围越广。这里讨论的一些做法可能更偏向于互联网方面的测试，特别是第三个阶段。

首先我想先从一个例子开始，一个现实生活中的例子。

对于一个城市，假设我们的工作目标是提升环境的质量，减少垃圾。那么我们可以做什么？

首先，我们可以请很多环卫工人，出去打扫各个街道，这个马上就有了效果，环境变得更干净了。但是还不够好的地方是明天还是有很多东西需要打扫，治标不治本，只要一停下来立马回到之前的状况。

接下来，我们往前面想一想，为什么有那么多垃圾呢？其中一个方面是很多人乱扔垃圾。所以更进步一点的方案是，对于乱扔垃圾的人有些约束或者惩罚，比如抓到了曝光或者罚钱，这样扔垃圾的人会变少。

再然后，我们发现即使做到了上面，还是有不少垃圾，而且上面强制的方案也带来不少的反感。我们需要更深层次的思考，为什么会有那么多垃圾？是因为垃圾桶太少？设计得不合理？如果是这样，就需要从其他公共设施方面做一些改进了。

对于我们的测试工作，也是有类似的思路，只不过细节上要考虑更多。

第一个阶段：发现和解决 bug 的阶段

这个阶段的思路基本上尽可能发现更多的 bug，见一个灭一个，来两个灭一双。

发现 bug，解决后验证 bug，没有任何根源性的推动，或者推动的效果不好。

这个阶段，测试工作主要集中在发现 bug，要做好这个，需要多个方面的努力，比如下面这些：

- 更高效的发现 bug，考验测试设计的能力。

这方面有非常多的方法和技巧，以及经验，这里不细说。

- 发现 bug 之后如何清晰的描述，定级，以及跟进和验证。

这个看似简单，但是你会发现很多测试工作做了几年的人这样的基本功还是不够扎实。也可能没有受到过很好的训练或者一直没有人指导。

- 对业务和架构的理解能力。

没有这方面的能力，很难发现一些深层次的 bug。而这样的能力对于快速学习和一些技术基础也有不低的要求。

- 发现 bug 之后如果举一反三的尽早发现更多类似的 bug。

大家看到的很多经典的测试书籍讲的基本都是这个阶段做的事情，比如 Software Testing，How We Test Software at Microsoft，以及探索性测试相关的书籍，都是专注在如何更高效的发现缺陷。

上面这些东西都是一个业务测试人员的基本功。看似简单，但是做好并不是一件容易的事情。也许这些事情一点都不 cool，不 sexy，甚至去做职级评审的时候不占优势，但是对于系统质量的提升，是切切实实带来很大帮助的，其工作的价值应该得到认可。但是如果一直停留在这个阶段，就陷入到上面例子中说的扫马路的阶段，因为如果没有其他方面的改变，每次都有那么多的 bug。

不过很多时候，我们的测试停留在这个阶段也是因为现状，考虑下这样的情况：

- 开发基本不自测，甚至没有自测的环境，特别是涉及多个系统的对接。
- 提测后很多基本的功能都不能正常使用
- 项目管理比较混乱，但是最终的发布日期又被老大们定死，所以测试时间常常被压缩

而且，而且没有对于开发人员的质量方面的考核，那么很长一段时间，我们的测试将处于这个初级阶段。

我相信目前还有不少的团队是处于这样疲于应对的情况下，不只是小公司，可能一些大公司的部分

项目也是如此。随着整个研发体系的发展和深入，我们应该有更高的追求。

第二个阶段：质量的管理

在第一个阶段中，可能有一些人会停下来想：我们一直这样下去也不是个办法？有没有更好的做法呢?

最直接会想到的就是，怎么让别人少丢垃圾，让本身的 bug 就更少一些。如果我们做的工作只是发现 bug 解决 bug，那么就是一个消耗战。不能形成一个良性的循环，就不能持续的优化，工作的长期累积价值就体现不出来。

这个阶段核心的思路是对缺陷做分析和考核，并做研发流程中主要问题的梳理和改善。

常常做的事情可以从下面几个方面来看：

1. 做质量数据的统计和分析

收集的数据很多，常见的有：

- 外网的 bug 情况，包括事故，及影响的程度
- 测试阶段的 bug 数量，分布（按系统，团队，开发个人），严重程度，bug 的类别等维度
- bug 的横向跨团队和系统的对比，纵向的和历史情况对比
- 版本发布的情况，代码变更行数的情况

从这些数据的收集中就能发现很多问题，比如问题集中在哪里，哪些模块，哪些人，哪些类别等等，以及有没有改善。

2. 问题的追溯和对于开发的考核

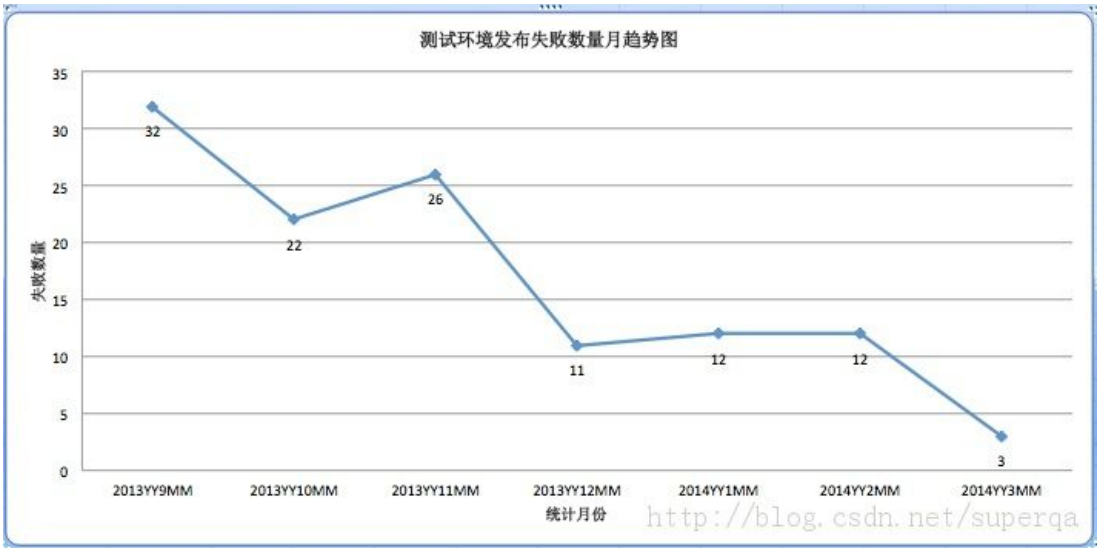
这个方面也许有一些争议，但是我还是觉得这个是一个很重要的方法。光靠观念和自觉是不够的，必需要有一定的反馈机制，就好比交规一定是配合着扣分和罚款等手段，否则记录闯红灯有什么意义呢？而且现实的来说，这些方法起到约束的作用，也是一种心理暗示，要做自己做的东西负责，也便于养成好的习惯。

通常的考核指标涉及这些方面：

- 编译失败次数的考核
- 外网事故和 bug 的数量
- 测试阶段的 bug，特别是基础功能 bug 和严重 bug

粗略的列了这么多，其实可以有很多，比如配置文件改错的情况，漏提测文件的次数等等。

这里也许有很多的讨论，但是让我们看看一个实际的例子。下图是某个系统的编译失败的情况，在 11 月份的时候提出要统计并公开（并无惩罚条款）编译失败的情况，包含到开发的团队和个人等明显，12 月份开始出现了明显的下降并稳定了。这个图隐藏了一些细节，如果剔除其他因素只看开发代码原因的编译失败则更明显，特别是后面有惩罚机制之后，进一步下降。



编译失败大幅的下降一方面是节省了大家的时间，另一方面其实也是提高了版本质量，想想如果有很多的编译失败，而且是到提交测试的阶段，这样的代码质量能好吗？是可能做过自测吗？有了这样的

机制，至少会更仔细一些。

对于 bug 方面其实也是一样，如果开发在乎（或者被迫在乎）外网 bug 或者被测试发现的 bug 数量，他写代码的时候一定会更仔细，也会做些简单的自测，让提测的质量更高，提高了整个研发系统的效率，同时也是提升了质量，因为 quality must be built in。

我个人的经验，作为测试人员几乎同时面对过两个开发团队，一个有上述的考核，一个没有。表现出来的版本质量和对质量的关注完全不一样，而且前者也并没有出现开发和测试的对立，以及测试不敢提 bug 等负面的情况。

3. 对于测试的考核

除了对于开发的考核，同样也有对于测试的考核，这样也更加的公平。

测试的考核通常考虑下面的指标：

- 漏测：绝对数量或者漏测率
- 版本的工作量和测试效率
- 发布延期的情况

如果测试有这样的压力，也需要不断努力去发现更多的 bug。

说起考核，总有人觉得这不符合智力劳动的情况，或者互联网的作风，其实不太理解为什么会这么觉得，放眼望去，有什么工作不被考核呢，sales 要背 quota，为什么软件开发和测试不能对自己的工作的质量负责呢？当然，具体的指标如何去定才更合理那是另一个要去考虑的。

换个角度来看，适当的压力（不应该导致焦虑和扭曲的做法），其实是让一个人表现最好的状态。

4. 推动开发的自测

这个问题一向是个老大难问题。愿意自测的开发团队你不用太多的推动，不愿意做的推动也很难，或者你无法判断他有没有做自测。而且这方面，通常取决于开发负责人的观念和态度。

如果是介于之间的，我们可以做一些事情，比如：

- 统计测试阶段的 bug 中，属于开发可自测发现的比例。通过这个可以看有多少 bug 是不应该到测试阶段的，以及横行纵向的对比。当然这个标准要自己拿捏。
- 给出一个自测的 checklist。开发在提交前要完成这个 list 并正式的给出报告。这个方式我们曾经在一个项目中用过，效果不错，基本功能都通过这个保证了，前提是开发负责人认可。
- 有一套自动化验收的用例，可以挂接到自动部署之后或者 daily build。前提是我们的自动化要足够的问题，效果才会好。

这个阶段除了业务测试的努力，也体现出了 QA 的价值。这里的 QA 是指质量管理，有的地方叫 SQA，专注在质量度量和研发流程的管理上。

到这个阶段，发现事情顺了很多，质量也有更大程度的提升，并有改善额趋势。

第三个阶段：推动全面的质量提升

到上面第二个阶段，我们发现质量有了一定的提升，但是还是有不少的问题，而且有些问题需要我们把思路和眼界拓宽来看。这里讨论的一些东西可能更适合互联网的产品。

这里列一些我们可以去做的事情，受限于个人的经验，可能还很片面。

1. 研发流程的梳理

其实在阶段 2 的时候也可能有些团队已经开始做这样的事情，因为在分析质量和效率问题的时候，我们发现很多问题不单纯是代码的问题，可能还涉及研发流程的很多方面，比如：

- 需求不清楚
- 跨团队的配合问题
- 代码版本管理
- 技术方面的评审和大家的理解

所以整个研发流程的规范和梳理，以及配合对应的需求和版本管理的系统也是非常的必要，实际中发现效果也是比较的明显。而且还有一点体会，在接手一个很混乱的状况时，这样角度的数量和调整比技术方案的引入更重要和切中要点，能从 40 分到 60 分，技术是往 80 分走的过程效果更明显。

2. 提交测试前后做的一些事情

- 代码的静态扫描

这个方法很多的团队都在做，但是实际的效果似乎差别很多，而且 ROI 也很难说，不过从方法本身来说还是值得去做的，对测试人员也提出来更高的要求。

- code review

这个开发应该要做，特别是开发间的交叉 review，非常的有帮助。不过这个也和自测一样，取决于开发负责人的态度。另外，测试也应该去做，特别是对于 diff 代码的 review，我们检查做了大概两个月的时间，发现还是非常的有收获。发现了一些黑盒难以发现的问题，以及开发的代码夹带，并且对于这个版本影响范围的评估也更准确。但问题是短期会花费测试更多时间，以及需要测试人员有一定的技术能力。

3. 测试能力的提升

测试阶段有很多的事情可以去做，觉得最主要的还是两个方面

- 自动化。 越来越觉得这个是绕不开的话题，要想尽办法去做，做得更高效更全面。前面有篇 blog 也提到了一些轻量级的做法，业务测试的团队可以参考
http://blog.csdn.net/superqa/article/details/20644285

- 辅助手段，比如代码覆盖率，特别是差异的覆盖率。这个大家都比较容易理解就不展开了。

- 拓展测试的类型

这个方面说起来有些泛，需要结合团队和业务的情况，比如安全测试，性能测试，兼容性测试等，去发现一些对于产品来说很重要的风险。

这方面有两个前提，一是我们的基本功能质量到了一个阶段，可以让大家腾出手去拓展测试的面，另一方面我们测试人员的能力要跟得上。

4. 发布环节的质量把控

这个方面和传统的测试不太一样，而且了解到不同的组织做法不同，执行发布的人员可能不同，有开发，运维，专职的版本管理或者测试来做。

在我们的实践中，发布后来都逐步收到测试这边，回头来看觉得还是有不少有帮助的地方。当然也不绝对的必须测试来做。

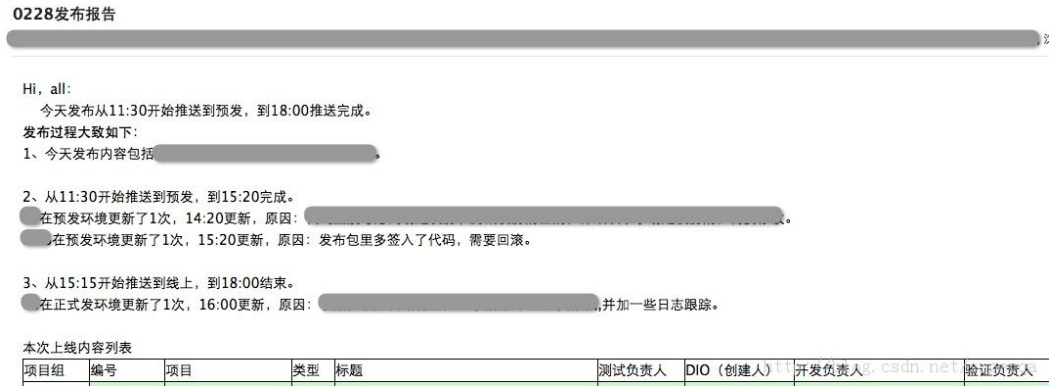
- DO 分离，避免了随意的发布，特别是在开发手上的时候。所有的 bugfix 都经过测试发布，可以更准确的度量质量（除非这个问题可以不修复，否则肯定要过发布环节）

- 知道最近发了什么，可能的影响是什么，需要线上关注什么。

- 灰度。 互联网产品常用的一个控制风险和节奏的手段。

- 扩容的快速自动化检查，这方面也依赖于自动化的建设。

- 发布过程支持灰度的控制，备份和快速的回滚。对发布系统有一定的要求，而且有可追溯性。



0228发布报告

Hi，all：

今天发布从11:30开始推送到预发，到18:00推送完成。

发布过程大致如下：

1、今天发布内容包括[redacted]。

2、从11:30开始推送到预发，到15:20完成。

[redacted]在预发环境更新了1次，14:20更新，原因：[redacted]。

[redacted]在预发环境更新了1次，15:20更新，原因：发布包里多签入了代码，需要回滚。

3、从15:15开始推送到线上，到18:00结束。

[redacted]在正式发环境更新了1次，16:00更新，原因：[redacted]，并加一些日志跟踪。

本次上线内容列表

项目组	编号	项目	类型	标题	测试负责人	DIO（创建人）	开发负责人	验证负责人

发布处在整个研发流程非常关键的节点，在这个点可以做很多的控制，也能发现很多的问题，对于测试团队来说，从这里可以发现很多的问题，做很多的提升，对自己和相关的合作团队。

5. 外网的监控

发现发布后的问题，持续运营过程中的问题，推动优化。

通常监控可以分几个层面，粗浅的可以分成几类：

- 运维层面的监控，比如机器，链路，资源使用，主要组件是否正常等。
- 业务指标的监控，比如来自点击率，BI 系统等。
- 集成在产品里面的监控代码，我们称之为模块调用监控。这个是全量的，有次数，成功率，响应时间等角度。
- 测试层面的自动化监控，关于在接口和功能层面。这个是采样的，但是从用户的角度来监控。

统一告警--自动化监控[redacted][总共收到告警1条]

已发送：2014年1月21日 星期二 下午4:58

收件人：[redacted]

此邮件的优先级为高。

统一告警--自动化监控[redacted][总共收到告警1条]

SN	告警源	发生时间	告警IP	告警内容	操作
196513544	[redacted]-[基础业务web]-[APP][web]	2014-01-21 16:40:00		[redacted]超时率100%&&超时数6	原因反馈

以上这些监控都有对应的告警机制，可以第一时间发现问题，避免造成更大的损失。为了实现上面的监控需要做大量的工作，但是这些对于整个外网运营的质量非常的重要。

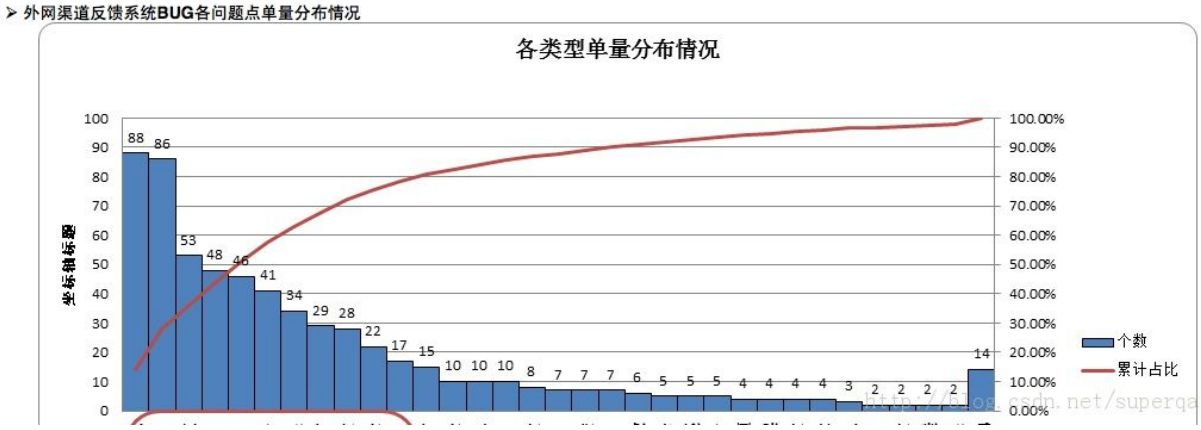
6. 外网事故和问题的收集，跟进和反向推动

和前面的思路一样，如果只是发现问题解决问题还是稍显被动，那么对于外网事故和问题的分析，还是有很多推动性的帮助。

7. 用户的问题反馈和满意度

进一步的质量不只是系统本身的质量，而是从用户角度看到的质量，有时候这个可能稍微超出一些系统层面的问题，但是因为最终的质量还是用户说了算，所以我们应该扩展下思路。收集这样的问题的渠道有很多

- 外网问题反馈，比如来自客服系统的，用户直接的反馈，现在很多 app 上都有反馈的功能。
- 论坛信息的统计收集。我了解的另一个测试团队，他们还专门开发了一个自动收集外部反馈，以及过滤分析的系统来帮助他们及时的了解外包的问题反馈。

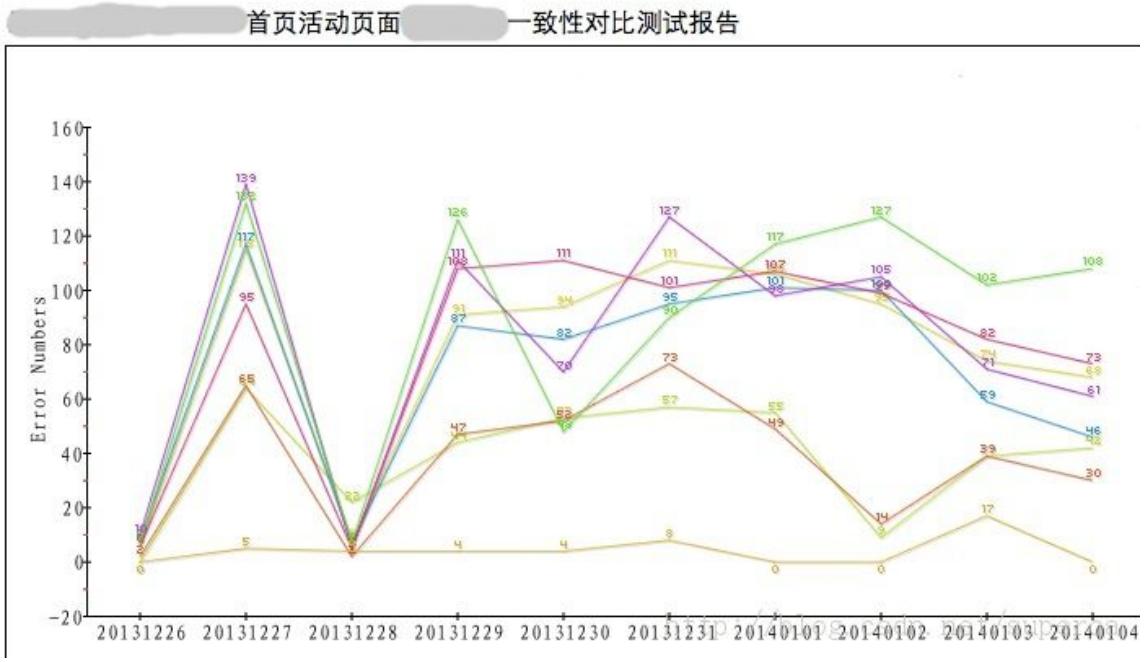


8. 运营层面的质量

更进一步，关注运营方面的质量，跳出传统意义的质量的范畴，关注我们的业务指标，不只是做一个高质量的产品，而是做一个业务上成功的产品。

比如下面这样的例子：

- 商品详情页的图片的质量
- 活动页面和详情页面价格不一致的情况
- 运营配置的错误导致的问题，哪些是可以监控发现，哪些是可以推动运营平台的规则检查?



每次我们的思路跳出一些框框，都会有不同的领域。有点点哲学上的意味，很多领域做到后面，其实会超出那个领域本身的范畴。就好比高性能的汽车，到后面就不得不研究空气动力学这个原本是和航空有关的东西。但是，这是否超出了本意，如果去看待，又是另一个问题。

其实这样的三个阶段也是一个粗略的划分，并不一定要逐步的来发展，其实都是一些具体的做法和实践。以我目前经历过的实践只想到这样的层次，应该还有更高级的阶段。

我们越到后面我们发现进一步的努力带来的提升幅度其实不大。但是很多事情也是一样，从 85 分到 90 分付出的努力可能比 50 到 80 分的努力还要大。另一个更有趣的是汽车的极速和马力的关系，家用车 100 马力开到 180km/h 是能做到的，但是超过时速 300，每提升一点需要增加的马力要大得多，到

400 以上，车时速每再增加一公里，功率需要提升八马力。这篇文章读起来非常有意思，
http://blog.sina.com.cn/s/blog_4d0109a301000ajz.html

写到这里，我们可以跳到整个公司或者业务的层面，来思考一些对于测试更深层次的问题:

测试团队存在的价值和意义是什么？

只有对业务有明确的价值，业务测试，或者说整个测试团队才有存在的意义。只要业务 OK，砍掉测试团队也不是不可能。我们必须时不时的跳出我们自己的思维的圈子，站在整个事业部老大的角度来思考下测试的价值和意义。

在下一篇关于测试组织方面我们可以再讨论下这方面的内容。

还有一个体会：测试的水平反应整个研发体系的能力和水平。

如果我们的测试还专注在第一阶段，那说明整个研发还比较初级，开发和测试都是温饱的阶段。当我们的测试人员不再趴在地上盯着最基本的功能质量的时候，才有可能抬起来看看更多有价值，有帮助和有长远意义的工作，慢慢形成一个良性的循环。

如何一步一步从 QA 到 EP

两三年以前，和友人谈到 QA（软件质量保证） 这个行业，还有 QA 这个团队的未来，就有了一丝忧虑。而现在，终于有机会实践一下自己之前的想法，在这里分享给大家。

从我有限的从业经验到现在，经历了很多次软件开发模式的变化，这些变化，或因为跟风，或因为有切实的问题要解决，总之始终处于各种不同的尝试的路上。QA 团队从最早的强调流程，到后来强调开发技术，搞自动化测试，再后来又开始做敏捷和持续集成，这条发展的路上，对自己的要求不断变高的同时，也伴随着一个组织和团队发展的魔咒。

组织发展魔咒

这个发展的魔咒更像是一个循环，可能开始于任何一个环节。

例如，公司负责技术的高层，没来由的认为，测试和质量保证并不重要。这个判断会慢慢渗透到技术团队的各个角落，导致测试工程师，乃至测试团队的其他角色，例如 SQA，未来发展的空间会被压缩，而压缩发展空间的结果就是留不住人、招不到人。一方面相关工作的经验技能要求越来越高，一方面可见的天花板又摆在那里。于是整个 QA 团队都成了别人眼中的 「低技术」团队，不论真的低技术还是别人以为的低技术，这种印象都很要命，为了摆脱这种印象，大家需要做点东西来证明自己，于是各种自动化测试框架、平台、系统，纷纷出现，殊不知此时，QA 团队和整个公司的价值已经慢慢的不一致了，自己关起门跟自己玩，成了普遍现象之后，在公司高层看来，他会觉得自己的 「QA 团队并不重要」的判断被证明了，因为没有任何明确的证据表明，QA 团队与公司愿景和计划之间的直接联系。

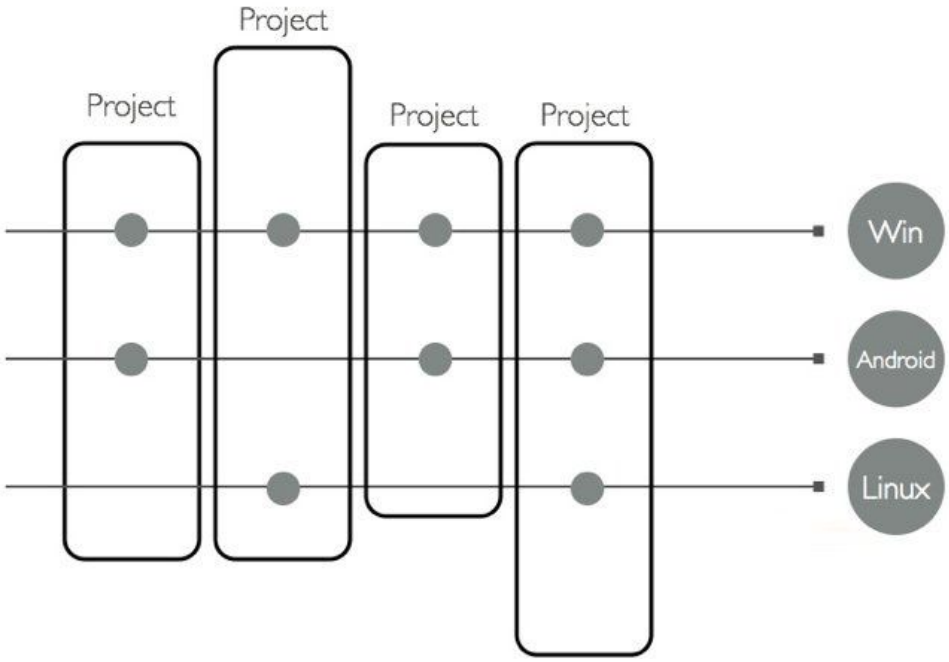
可怕么？在很多软件研发组织中，这是现实存在的循环。

说起来我们的实践，确实打破了这个循环，说起来好笑，我们解救 QA 团队的方式，就是彻底取消这个团队。但是反过来讲，只有杀死「QA 团队」，才能真正的解放「QA 工程师们」，真正解放整个软件研发过程。

一些基本的价值观

这个事情，就要从一些最基本的价值观说起。

比如，人总要对自己做的事负责。当然做了漂亮的事情，谁都希望头上有光环，但是做了丑事，也要能忍受得了羞辱。之为 「吃自己的狗食」，而老式的软件开发分段流程，等于鼓励上游做的错事，下游来擦屁股，于是上游颐指气使，下游低三下四，这种颐指气使和低三下四还能传导，于是的于是，最下游的一个环节，就是公认的受气包了。暂不说效率和质量，从最基本的做事方法的角度，似乎也有些欠妥。我们这一条怎么落地呢，就是改组研发团队，建立 Owner 制度。一个项目的 Owner，就像一个项目的 CEO，大事小情都要关注，从产品到开发，从测试到运维，总之一个项目的成败，都需要 Owner 来操心，项目外的人，都是他的资源。相应的，项目也变得和平台无关，而与特性有关，每个项目组都会涉及到几个平台的设计、开发工作。



还比如，给质量一个明确的标准。做质量工作或者测试的人，都会有强迫症，总觉得哪里不对劲，还得狠狠的回归一遍，又一遍。可其实大家都知道质量是没有个确定的标准说好还是坏的，那怎么确定质量呢？我们称作 「质量体现在造成实际的影响上」，也就是说，一个严重的问题，如果没造成影响，或很轻微，那就不严重。而一个轻微问题，如果影响面很广，例如有 1000 万用户都看见了，那就不轻微。

又比如，交付一个完美产品还是建立一个快速召回的机制？我们确实真的想每时每刻都能交付一个完美无暇的产品，但那不可能。特别是在互联网行业，跟传统的电信、医疗、航空航天的产品迭代有天壤之别，一个完美产品用一年做出来，市场可能早就变了天了。但不完美就有质量风险和代价，为了平衡这一点，我们必须建立一个快速召回缺陷产品的机制，甚至能让用户在发现缺陷之前，就用上了新版本。

有了这几条价值观，我们就大概知道开发过程改进的方向，以及做事情的原则了。那我们做了什么呢？我们组建了 EP 团队。

EP 是什么

说到这里，EP 这个词才第一次出现，这个词的内涵之丰富，可能需要仔细说说。

我最早看到 EP 这个词，是在当时还是 Google EP 团队成员的 James Wittaker 写的那一个有名的「How Google Test」的系列博客中，内容我就不转述了，很多人都读过。

EP 是 Engineering Productivity 的缩写，工程生产力的意思，这个团队，就是给整个大技术团队，甚至整个公司提高工作效率的。通俗直白的说，就是一个工具团队。因为工欲善其事、必先利其器，不要小看工具团队，某些程度上来讲，一个产品随着市场的变化可能很快凋亡，而一个好的工程工具，生命力要强得多，比如，开发语言其实就是最基本的工程工具了。那么，对一个公司，或者说交付团队来讲，怎么衡量工程生产力的高低呢？这个衡量方式其实就决定了「EP 团队」的工作方向。我们自己定义的工程生产力从低到高的定义是这样的：1）质量，这是最基础的指标，什么都不行，也要保证质量过关，否则一个产品连生存的可能都没有。2）同等质量水平下，追求速度。质量过关了，就要看迭代速度了，你比竞争对手快，你就能活下来。3）同等质量和速度下，工程师的幸福感。如果质量也过关

了，速度也快，但是大家过得很苦，天天加班，重复劳动，看不到未来，这也不行。幸福是什么？对我们来说，就是不被反复的简单问题所困扰，该自动的都自动，自动不是说一定快，但是一定省心，这里的幸福就是省心，有精力去关注更多的有意义的事情，而不是每天处理简单重复的问题。4）同等质量和速度，也有幸福感，再看成长。工程师们有没有感受到成长？不断的解决问题或开发产品，感受到的是重复劳动还是成长？其实前三点都做到了，第四点一定是有的。

EP 做什么

我们回头说 EP 团队，EP 团队也有自己的人生理想，那就是一个三部曲：替、教、独立。

第一个是替的阶段，其实就是比较老式的开发过程，我替你测试、替你上线、替你运维。

这个阶段，完全不符合我们「吃狗食」那一条价值观，按照狗食法则，一个人自己设计开发编码，当然要自己测试，自己写的代码 bug 多要一遍遍回归，这个苦果自己不吃谁来吃？但没办法，大多数程序员在如何测试自己的程序方面没有受过什么训练，为了尽快发布产品，只能替，但这个替的时间要越短越好，尽快进入下一个阶段，教。

第二个是教，就是教技术团队的其他成员，如何测试自己的程序，如何构造环境、构造数据，如何部署和运维自己的产品。这里的自己做，并不是回到蛮荒时期，例如创业初期只有一个程序员的时候，他当然是自己开发自己测试自己部署，但我们到了第二阶段的自己做，是自己规范的做，通过我们提供的相对完善和规范的工具做。我们就处于这个阶段的初期。

第三个阶段是独立，独立是说 EP 团队从一个替人做事的下游团队，到一个教人做事的教练团队，真正进化为一个提供技术服务的产品团队。这个产品团队的产品，大多数应该是以一个标准化的、健壮的服务的形式，而不是人力资源的形式，提供给其他团队的。当然这是我们的理想，能否达到或者是否切合实际，还需要时间来观察。

EP 团队和整个技术团队的关系

从这三个阶段的描述中，多多少少都提到一些 EP 团队和整个技术团队或整个公司的关系，那这个关系是什么呢？

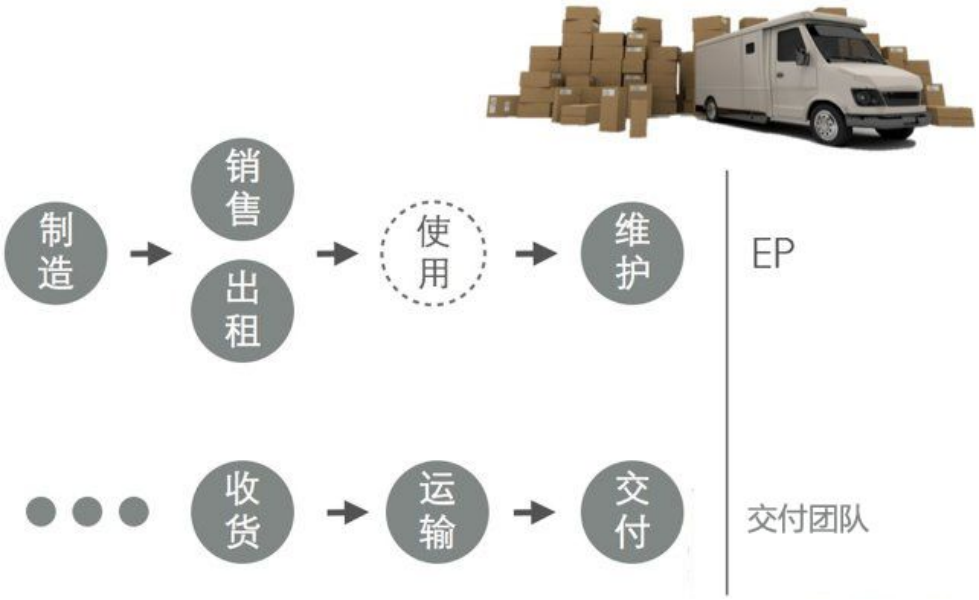
前面提过，我们不希望是个下游替人收尾的团队，我们也有「吃狗食」这样的原则，所以我想到一个比喻，来说明 EP 团队和其他技术团队的关系。

我们都知道有一个行业叫做汽车制造业，他们遵循一定的规范和标准，还能巧妙的将创意和标准结合在一起，制造出一些工业奇迹，这些汽车被卖给各式各样的人，汽车企业并不关心他们的产品用在什么地方，不过他们又发明了一种叫做 4S 店的东西，用来给那些工业奇迹定期维护、修理、还可以收集市场反馈以便改进产品。

还有另一个行业叫做物流业，他们的目的就是将货物从一个地方运到另一个地方，这种空间的转移，就是他们为客户创造的价值。在这过程中，他们会利用到汽车制造业产出的汽车，但他们不太关心汽车本身的标准、设计细节，他们只是使用，他们关心的是使用成本、维修方便。出问题，他们会找 4S 店。

这两个行业之间的交集是汽车，汽车制造业的价值是制造出好的汽车，物流业的价值是货物的到达，汽车制造业不关心你的货物的目的地，物流业不关心他的汽车的制造工艺。但汽车制造业会很关心你怎么用这个汽车，以及积极的帮助你保养，而物流业也会很关心这个车费不费油，好不好开。

说到这里，你可能已经看明白 EP 团队和其他技术团队的关系了：EP 团队就像汽车制造业，提供高效、低耗的工具；产品技术团队就像物流业，使用工具，快速前进，创造用户价值。他们之间互相依赖，却又彼此独立。



EP 都有谁

了解了 EP 和周围团队的关系之后，来看看我们的 EP 团队的角色和成员。

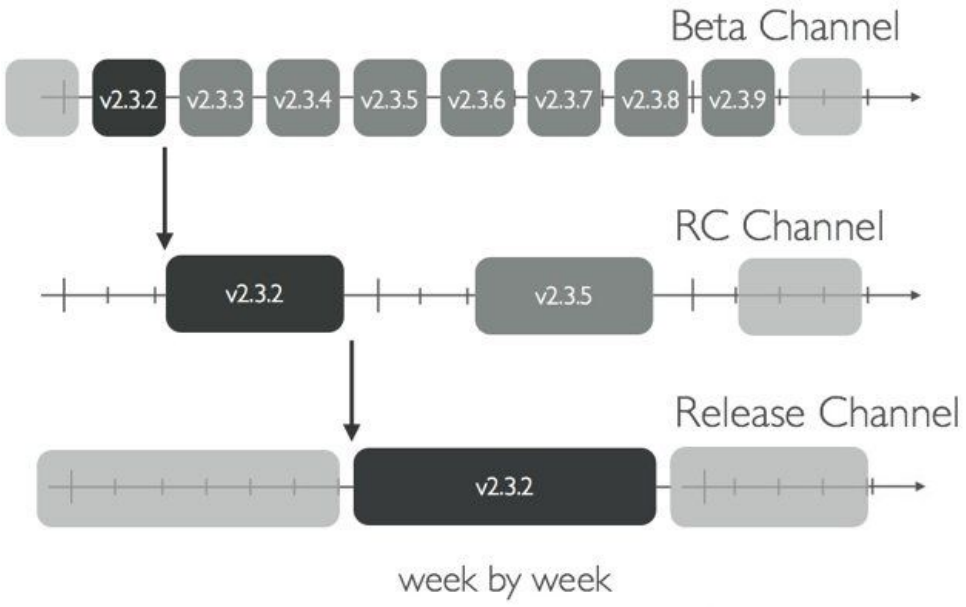
我们的 EP 团队，大致分成如下几个角色（而实际上的工作是混合的，之所以要分开成角色，主要是从招聘的角度出发）：

SED，Software Engineer in DevOps。顾名思义，这个角色首先是个软件开发工程师，其次，面向的领域是 DevOps，DevOps 的概念我们就不必多讲了，在实际工作中，SED 工程师是个真正的多面手，他们可能今天在开发一个 Linux server 的自动化上线和回滚的工具，明天就要设计或优化 CDN 的部署，后天又要解决一个 Windows 平台编译加速问题，还有还有一个自动性能 benchmark 工具等着他来开发。这个角色目前我们只有两位，而且这个角色的工程师是最难招聘的，因为新人，或者很小的公司出来的人，很少有受过系统的训练或有比较先进的软件工程思想，而从大公司出来的人，已经被大公司条块分割的工作方式同化，一般只擅长一个领域，而对跨界的或者不懂，或者没兴趣。所以这个岗位的工程师，都是有成熟公司工作经验的 Geek 型的人。

SA，System Admin。系统工程师，和很多公司的运维工程师很想像，实际上我们现在的状态，做的事情也和大多数公司的运维工程师一样，处理监控，优化服务部署等等，但不一样的是我们的目标是将绝大多数应用层面的运维工作交还给开发团队，所以我们在不断的将监控系统改造为友好的，自助的，也不断的将各位上线部署类的工作做成自动的，现在已经有了很多成果，我们的 SA 主要精力可以放在系统以及更底层的部分了。

TE，Testing Engineer。测试工程师，其实这个称呼有点名不符实，我们的唯一一位测试工程师，主要的工作其实是发布和迭代控制，要保证整个交付团队的迭代节奏，例如在代码上拉发布分支、触发发布事件、监控数据等等工作，这个工作要求非常精确，又很繁琐，因此和 SED 工程师有非常多的交互，他们负责将这个过程自动化。这里插入介绍一下我们的发布过程，可能大家会更理解为什么还有个「发布工程师」：

我们有三个发布 Channel：Beta、RC、Release，作用各有不同。例如 Beta Channel，主要用于一些新特性的提前发布，这里面可能会多少有点缺陷，所以一定要控制人数，并且是那些喜欢尝鲜的用户，他们会用的比较彻底。而 Beta Channel，可能每天都有版本更新，会有一些用户喜欢跟着 Beta 版。而这些新的特性如果用户反馈不错，并且没有什么严重的问题，就会进入最近一次 RC（Release Candidate），这个量就很大了，大概能占到我们每日活跃用户的十分之一到五分之一，这里面的功能在没有意外的话，就是正式发布的功能了，需要注意的是，不是每个 Beta 都会变成 RC。而 RC 在发布几天之后，如果一切正常，就会切换为 Release，Release Channel 一般会在一天之内，让绝大多数活跃用户升级完毕，这个时候，如果程序有 bug，影响就非常大了。



Venders，外包测试团队。我们有大约六七个人的外包测试团队（on-site），主要负责我们主要产品的人工验收测试。我们对外包测试团队的工作方式也有一个设想，就是一个项目刚开始的时候，外包测试团队应当是先上很多人，然后随着 SED 的介入，让自动化程度加强，慢慢人少下来，直到下一个新项目开始。但这个设想在国内想实现，却没那么容易，主要有几个原因：1）国内的外包测试的工程师，通常是技术和经验都比较初级的人来做，外包测试成了一个门槛低天花板也低的行业，技术和经验缺乏，导致进入新项目以后没办法非常快的上手，而有经验有能力的人，很快就会脱离外包行业；2）外包测试的公司，人才储备不足，很少有人力资源池，都是有需求，现从市场上招，或从竞争对手那里挖，有的人都没见过，就派到客户那边来面试，这也导致了没办法几个月就撤下来，因为他没办法跟候选人签合同。这两个客观原因，我们也比较无奈，所以我们的外包测试团队基本上还是长期 on-site。

UOE，user operation engineer。用户运营工程师，这个岗位很多人不太容易理解，一般用户运营人员都是跟内容啊、用户打交道的，就像贴吧管理员就是典型的用户运营人员，那为什么要有个运营工程师呢？这个我们是跟硅谷的 Dropbox 学习的。因为在日常工作中，我们发现有想当一部分用户的反馈，不论是新功能的需求还是缺陷，都是技术性很强的，如果你能做到第一时间和用户做深入的，技术含量较高的沟通，从解决问题的成功率上会高很多，而如果你反馈一个技术问题，总是过了几天才有技术人员跟你联系的话，你可能配合排查问题的愿望会小很多。基于这个思路，我们增加了这个角色，同时他们还负责一些运营过程中使用的工具和平台类的研发。可能会有人问这个角色为什么会在 EP 团队？其实仔细分析一下用户运营的工作，会发现他们处理的对象是一个个用户提交的 ticket，这非常像 test case，不同之处是一个是用户事后提交，一个是事先设计，分别保证了优先级和完备性，因此结合起来，对提高质量是非常有益的事情。

EP 团队的工作方式与面临的挑战

上面这几个角色，就组成了我们的 EP 团队，这样的一个团队，这样的一个能力构成，就有了一些鲜明的特点，例如：

1）没人管的事情我们管，支持所有团队。公司内部虽然分成了很多个团队，但是很多技术问题是找不到人负责的，例如，一个简单的内部数据统计脚本，或者一个发布内容到 CDN 的 CMS，等等。这些事情基本都会由 EP 团队接过来。

2）做事情没有计划。这个特点可能很多人会觉得匪夷所思，甚至不能接受，但实际上这跟 EP 团队的工作有关系，比如汽车 4S 店，有多少车祸的汽车要修理，多少人为损坏的车要修理，怎么做计划？实际上是遇山开道、遇水搭桥。外部的市场的变化、内部的技术人员的变化，都会有不断的瓶颈出现，而 EP 就要快速发现并解决这些瓶颈，直到发现下一个瓶颈，这个过程没办法有详尽的长期的计划。而替代的是使用目标管理的方式，我们公司内部所有团队都会用一种叫做 OKR（Objective and Key Result）的方式来做管理，简单的说，就是设定目标，然后评估完成度。EP 团队的目标大致有两个方向，一个我们叫做 「Smoothly & Fast」，就是让一切跑通做顺的能力，还有一个就是「实现自助」，能让其他团队的成员，不管是技术还是非技术背景，都能自己通过我们的工具完成任务。

这些特点看起来很不错，但是实际上带来的问题也非常多，例如：

没有成就感。因为所有人都是你的需求方，这个感觉实在是不太好，另一个角度讲，很多研发工程师会觉得开发一个对外的产品比较有成就感，对内的总觉得没意思。这个问题要解决，其实就要靠所谓的「工程师文化」来解决，国内长期以来在职业化上有一些不好的习惯，其实能发明工具的人都是大师，开发语言就是工具，操作系统也是工具，真正的牛人，都在做各式各样的工具。能帮助别人成功的人，是最成功的。

还有，就是脱离实际。很多人做工具，怎么炫怎么做，流行什么做什么，要么就大而全，这还是好的，更多的时候是想的大而全，半年做不出来。整个公司的价值取向是一致的，特别是小公司，容不得这种炫技一般的工作方式。所以我有一句话，叫做「自 high 无价值」，什么叫「自 high」？就是自己跟自己玩，玩的很高兴。

还有一个问题，就是招聘困难。这个在 SED 的工作职责部分提过，就不展开了。因为招聘困难，我们就要考虑怎么培养这样的人才，所以我们有一个方法论，叫做「要改进，先体验」，因为很多 EP 的成员是要改进工作过程的，但是怎么改，不是所有人都能搞定，这依赖于大量的经验积累，对经验不足的人，很简单，就是让他去做。要提高研发效率，找到痛点，那就先去做一个月研发；要去改进测试过程，提高效率，就去做一个月测试。一个技术和思维方式都很不错，只是经验少的人，经过一个月的体验，能提出非常多的、而其他人已经麻木了的改进点，并推动实施。

再有，依赖整个团队的成熟度。不是说有了 EP 这样一个团队，整个公司的效率和工作模式就会有大幅度提升，因为一个汽车再好，你开的方向不对，也到不了目的地。现实中存在着非常多缺乏责任感的 Owner，很多人觉得，我写完代码编译通过了，丢给测试组就行了，没我的事了，这样的想法大有人在，所以从成立 EP 团队，到整个公司的生产力真正被提高，中间不只是提供工具这么简单，还有一系列的指导和训练的工作。

Why we can & why you can

最后，关于我们为什么能做这个事情，我们也有一些总结：

1）创业团队。创业团队就是短小精悍，精力集中，没有太多无谓的纷扰，快速交付产品快速迭代是主要的工作方式。

2）从第一天开始坚持自由和责任的工程文化。从创始人开始，很坚持这种工程文化，有什么样的 leader 就有什么样的团队，所以大家接收和拥抱 EP 的工作模式，也非常快。

虽然上述这两条很多公司没有，但不代表做不成这个事情，在我看来，只要具备下面几条，想做成 EP 的工作，就并不难。

1）互联网行业。互联网行业有一个非常好的，区别于以往其他行业的特点，就是你的产品在物理上是自己控制的，提供的只是服务，这非常有利于快速迭代，因为传统行业不可能做到。

2）快速变化的业务模式。这不是说我们自己要快速变化，而是业务模式和市场不断变化，来推着我们前进。只有业务模式的快速发展，才对生产力有不断更高的要求。

3）有改变的决心。这个说起来有点虚了，但也很重要，因为 EP 这样的工作模式会有阵痛，例如团队的重组、转型，都会影响到一部分人的利益，特别是团队的管理者，而这些中高层管理者，也确实有能力阻止变革。但坦白的说，很多时候你不主动改变，到了客观环境推动你不得不变的时候，到最后就成了被淘汰的人了。我还有一句话，叫做「组织结构决定工作模式」，意思是说，很多工作模式的出现，是因为组织结构的需要。反过来说，在你的组织里很多很好的工作模式推动不下去，或者效果很差，你就要看看你的组织结构是不是有问题。比如两个本来应该紧密合作的团队，一直合作不好，互相鄙视，你想简化流程，最后流程越做越多，大家都在垒墙，那你就要看看两个团队共同的老板，是不是级别太高了。

4）对团队成员的高标准。前面我提过，我们 EP 团队的大部分是 Geek 型的人，Geek 这个词在英语里是一种很高的评价。只有一个技术和经验都非常丰富的人，才能胜任 EP 这样的工作，所以要坚持不懈的雇佣一流的人才，人不够，可以抓大放小，但绝不能有二流、三流的人在团队里。

到底谁需要技术写手呢？

技术写手的工作任何一个人都可以做。这份工作并不需要任何真正的技巧，因为它仅仅是写文本。更何况，通常情况下技术写手往往不能牢固掌握他们正在记录着的技术。

他们只是问一些，什么样的功能是应该做的，然后他们把开发的话写下来，也许增加一个章节和一些修饰的文字。

有没有想过这样一个问题？

事实上，人们有时候也会用这种思维，类比测试人员。

任何人都可以做测试。做测试只是需要有一个人站在产品面前，点击一下鼠标，或者给出一些测试案例而已。任何一个人都可以做到的。

为什么人们习惯在别人的工作上出现这种狭隘的思想呢？除了摇滚明星“The Developer”（注：People love to say that a rock star can do the work of 10 regular engineers.）以外，还有很多其他软件行业的角色面临着同样类似的偏见。

例如，配置管理工程师（或实施工程师）。

任何一个人都可以做到这一点。到头来，那仅仅是一个人肉文件处理器？

我们存在这些偏见的原因，是由于缺乏洞察别人的工作，和用什么标准来区分：一个好的测试人员/好的配置管理工程师/和一个平庸的，甚至是糟糕的技术写手。

也或许我们缺乏与真正优秀的人接触，而仅仅接触一些糟糕的人罢了。

例如我有幸与有可能成为最优秀的瑞典配置管理工程师(看您怎么定义)合作，在 Softhouse 的 Christian Pendleton (@chop_se)

与他近距离合作，给了我一些感受就是如何做“优秀的配置管理工程师”，看他作为一个配置管理人员对哪些方面感兴趣，以及他是如何保持对最新的技能的把握。

到目前为止，我还没有与技术写手近距离合作，所以我在努力理解用什么标准去区分一个好的和一个平庸的技术写手。

但是错误在于，我的工作结束了，相反，我从来没有做任何技术方面的写作，除了情不得已要为我自己的代码编写。

“这些角色已经存在了很长时间”，可能是至今的最好的一个理由。

虽然我相信有经验的老读者能够指出软件行业的某些角色已经存在过几十年的，但此后消失或过时了，比如：相当于 Copy boys 和 Elevator operators 的岗位。

在上一年，Scrum 和 Agile 正一直在处于上升期，而且构建跨职能团队的误解已经更多的在行业中

传播开来。

现在别误会我，其实我喜爱跨职能团队的理念和这个理念所表达出的东西，这些年来我都大力倡导它。

但是跨职能团队背后的最初想法是：你可以提高你的团队成员的综合能力水平，从而成为专家团队，这样他们可以独自完成一些较简单跨职能的任务。

但我遇到太多的管理者太计较，他们认为跨职能意味着每个人都应该能够做所有的事。

我有见过一些组织，“去掉头衔/职位”，如开发人员，测试人员等，并用一些含有“designer”的标题进行替换，并期望一个 designer 应该能够做测试，配置管理，技术支持等。当然，最重要的是编码，因为这是关系到推动更多的迭代版本产出的关键。

这既是悲剧又是错误，因为它剥夺了不同岗位的技能特质，而且还传送出一个“任何人都可以做你的工作”的信号。你再也不用指望在某个职业领域能够做得更好了。

这些公司要么外包一些可以所有事情都能做，但结果表现都很一般的 super-generalists，要么通过某些渠道发现一些凡是所能做，且都做得很好的 super-generalists(super-heroes)。这意味着，如果他们真的能找的这样的 super-heroes 话，也不得不通过猎头去雇佣这样的人。多数情况下，人们都会选择第一种情况，结果当然是不言而喻的。

我们需要意识到，长久以来已经存在的角色可能就是它得以存在的一个很好的理由。

还需记得，他们也可以随时间而变化。

例如 Elisabeth Hendrickson(@testobsessed)最近在她的 CAST2012主题演讲中给出了一个美好的引述:

"Testing is only dead if you're trying to find something that looked like testing in 1998"

“在当下，如果你仍旧尝试按照1998年的测试套路来做测试，那测试岗位是时候 dead 了”

最后，所有这一切都归结到这一点。我们对别人的工作了解得越少，我们就越倾向于认为，他的工作是简单的和可替换的。

所以当下次你在享受着咖啡和你的技术写手，或配置管理、或测试工程师，在一起聊天的时候，不妨对他们的工作提起兴趣，并试图找出区分优秀和平庸的标准。还有了解他们是如何在他们的职业上与时俱进。

因为老实说，比起仅仅了解自己的技能，我们为何要吝惜，通过了解更多岗位的技能而从中受益呢？

GNU Emacs 有很多“神奇”的功能。常言说“每一个 hacker 都有一个自己的 GNU Emacs”。这个事实在很大程度上得益于人们能够按照完全自我的方式去使用 GNU Emacs。将 Shell 运行在 GNU Emacs 里面就是众多的用法之一。 在 GNU Emacs 里面运行 Shell 有很多种不同的方法。包括各种各样的终端模拟。但是在笔者的工作当中更多的使用的是 Shell-mode 的方式。在这种方式下，可以最大限度的利用 GNU Emacs 所具有的各种神奇能力，让日常工作变得前所未有的轻松、有趣。

第一回 引子

GNU Emacs 是一个非常强大的编辑器，这个编辑器不仅可以用来写文章，写程序，更重要的是， 他可以和一些原本看似没有明显关系的应用程序在一起，合作创造出一些新的“不可思议”的应用。比如说可以在 GNU Emacs 里面运行你的 Shell。

通常来说人们在 Linux 或者 Unix 上面工作的时候，不论是在本机工作，还是登录到地球另一头的远端机器，都是使用各种各样的终端或者终端模拟器来运行 Shell。最常见的例如 xterm，rxvt，以及 Putty 之类的终端模拟器。与此对应，GNU Emacs 也有自己的终端模拟器，例如 ansi-term，multi-term 等等。这些终端模式，使得你可以像在在其他终端当中一样工作，甚至可以在 Emacs 的终端里面运行 Vim。

但是，今天要和大家分享的是另外一种使用方式—— Shell mode。这是一种完全不同的工作方式。这种方式和大家常用的工作方式最大的一个区别，就是在这里完全没有任何 terminal 的存在。用户实际上是工作在一个 Emacs 的文本缓冲区里面，并不直接和 Shell 进行交互。一切的命令输入都是写入到这个文本缓冲区当中，经由 comint.el 从缓冲区中读取，然后转交给后台的 Shell 进程。Shell 产生的输出再由 comint.el 进行收集，然后写入到用户所用的这个缓冲区当中来。这个缓冲区在 Emacs 当中叫做 Shell 缓冲区 （Shell buffer）。

启动一个 Shell 缓冲区并且进入 shell mode 的过程非常简单。只需要在 Emacs 当中按下 Meta-x 组合键(在现在的键盘上通常是 Alt-x 组合键),然后输入命令 shell 并回车,Emacs 就会启动一个 Shell 进程并且打开一个与之关联的 Shell 缓冲区。Shell 缓冲区 的名字通常会是 *shell*。具体启动什么样的 Shell 进程 通过 Emacs 配置文件里的 shell-file-name 变量指定，或者由用户的环境变量 SHELL 或 EMACSSHEL 来指定。通常的写法是

（setq shell-file-name "/bin/bash"）

或者

export EMACSSHELL=/usr/bin/zsh

另外如果你希望使用一个支持 ANSI color 的 Shell 进程，那么最好在你的 Emacs 配置文件里面加入下面两行，以便在执行 ls － color=auto 命令的时候输出的色彩信息能够被 Emacs 正确解析。

（autoload ""ansi-color-for-comint-mode-on "ansi-color" nil t）
（add-hook ""shell-mode-hook ""ansi-color-for-comint-mode-on t）

说了这么多了，这种工作方式究竟能有什么好处呢？我为什么要离开熟悉的 Xterm，把我的 Shell 搬到 Emacs 当中来呢？

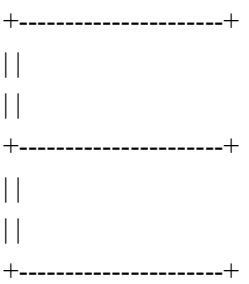
第二回 初识 Shell mode -- 窗口篇

下面我们就来谈谈好处。事实上不仅仅是好处，在相当程度上甚至是不可替代性。

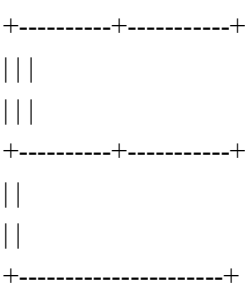
第一个明显的好处就是多窗口的工作模式。

通常在人们的工作当中都会打开多个终端，同时进行几份工作。在这个时候就需要对这些终端窗口进行排列和管理（在这里假设你工作在图形化环境之下）。而且通常需要频繁的使用鼠标在不同的窗口之间切换焦点。为了避免窗口之间相互遮盖，你也许会通过精心编辑的 .Xdefaults 文件使得两个或四个终端窗口恰到好处的平铺在整个屏幕当中。但是仍然需要使用鼠标在不同的窗口进行切换，在不同的窗口之间复制粘贴信息……这些窗口维护的工作在任务繁忙的时候会很繁重。并且如果这时候需要的不止 4 个窗口，或者你还需要进行额外的文字编辑的工作……最终窗口还是会要么被覆盖起来，要么被挤到别的虚拟桌面。

在这种时候最好来试试 GNU Emacs。GNU Emacs 天生具有完善的窗口管理功能，并且完全不依赖于 X Window。这是因为 GNU Emacs 的诞生要远远早于 X Window 的历史。在 GNU Emacs 里面你只需要按下 Ctrl-x 2 组合键就可以把当前窗口切分成上下两个等分的窗口，



按下 Ctrl-x 3 组合键又可以把当前窗口切分成左右两个等分的窗口。这些切分可以一直进行下去。



Gmail 测试工程经理 Ankit Mehta 的访谈

Ankit Mehta 在成为测试工程经理之前是一名测试工程师(TE)。在最初的几年，Ankit Mehta 一直在和测试自动化代码打交道。他作为技术经理的第一个大项目正是 Gmail。

Gmail 是个巨大挑战。它非常庞大，涉及很多快速发展的部分。Gmail 整合了很多 Google 的产品，如 Buzz、Docs、Calendar 等。它需要处理那些已经站稳脚跟的竞争对手所支持的邮件格式。Gmail 有非常庞大的后台系统。要知道 Gmail 是一个云服务，用户可以通过任意一种主流浏览器进行访问。有数亿用户在使用 Gmail，他们希望打开浏览器后 Gmail 就能工作，这从某种意义上也增加了复杂性。用户需要快速、可靠、安全的服务，并且还能包括自动处理垃圾邮件。增加新特性必须保证之前的功能持续可用，这使得测试任务变得非常复杂。一旦 Gmail 出现问题，全世界的人就会在第一时间发现。因此，测试工程经理责任重大。

我们对 Ankit 进行了采访，了解 Gmail 是如何测试的。

HGTS：告诉我们你是怎么接手一个新测试项目的吧。你首先会做什么事，问哪些问题?

Ankit：加入一个新项目的头几个星期，我主要用来倾听而不是发表意见。深入理解团队非常重要，要学习产品的架构，了解团队的最新动态。我不能接受一位医生在观察我不到五分钟的时间就给我开具抗生素类的药品。同样的，我也不期望一个测试团队可以接受我一开始就提出的什么解决方案。在进行诊断之前你必须先要学习。

HGTS：我们和你一起工作过，你可不是那种安静的类型啊。我估计你是不开口则已，一开口就会滔滔不绝，如黄河泛滥般一发而不可收拾!

Ankit：噢，是的!不过我也不会什么都说。多年来，通过不断地聆听，我发现最有力的问题就是“为什么”。为什么你会进行这些测试?为什么你会想到这个用例?为什么你选择把这个任务自动化而不是那个任务?为什么我们要投入做这个工具?

我感觉人们有时候做事只是因为看到别人这么做，或者他们测试某个特性的时候只是做那些他们知道怎么做的东西。如果你不问他们为什么，他们自己也不会费心思考这事儿，因为他们已经把那些作为了一种习惯。

HGTS：那什么样的答案算好答案呢?

Ankit：第一，因为它能够提高产品的质量;第二，因为它能提高工程师开发产品的效率。其他答案都没这些重要。

HGTS：Gmail 团队注重生产效率是出了名的，所以我理解你会这么说。不过除了质量和效率之外，你对测试工程经理还有什么建议来建立一个健康的工作氛围呢?

Ankit：团队的气氛非常重要。我深信优秀的产品和优秀的测试团队紧密相关。你必须要有拥有合适技能的人，正确的工作态度，并做正确的事情。特别是团队中资深的人，因为团队的文化和氛围很大程度上来源于这些人。拿 Gmail 来说，我花了三到六个月来建立团队，让团队具有凝聚力，每个人都能理解其他人的角色。当你有了一个好团队，就不会由于一两个人的不适应而出现问题。测试团队和开发团队的关系也是一种非常重要的气氛。当我刚加入的时候，这种气氛并不好。测试团队自顾自的工作，

而开发团队也不认可测试团队，这是非常不好的。

HGTS：你肯定把这个问题解决了，能具体谈谈你是怎么处理的吗?

Ankit：我刚加入 Gmail 的时候，测试团队只是专注于执行一系列 WebDriver 的测试，每个版本执行一次。每次执行测试结果都会由绿(成功)变红(失败)，然后再花大力气修复这些测试，让他们能够再变绿。开发团队没有过多质疑这种做法，由于这些测试通常还是能发现一些重要问题的，因此这种做法就一直延续下来了。但是曾经有好几回代码变化很大，测试代码根本来不及修改。整个过程非常脆弱，不能适应 Gmail 的变化。这是一种过度投入，因为要让它最终发挥作用所需的工作太多了。

可能是因为我新加入的这个项目，所以能发现一些其他人不能发现的事情。在我看来处理延迟是 Gmail 最大的问题。严格来说，从用户的角度来说，Gmail 最大的特性就是它的速度。我料想如果我们为开发团队解决了这个问题，我们就能赢得他们的尊重并开始建立平等的关系。

这是个难题。我们必须测试 Gmail 老版本和新版本速度上的差异，当新版本的速度下降时及时发现。然后我们需要检查所有新版本里改动的代码，并找到速度变慢的原因，从而修复这个问题。这是一个痛苦的过程，非常耗时，并伴随大量的尝试和失败。

我曾经和一位测试开发工程师一起想办法，想让 Gmail 的速度变慢，以便于我们能更好地观察前端和后台数据中心的通讯，从而发现造成性能下降的原因。我们最后到处找了些旧机器，弄了一大堆512M 内存、40GB 硬盘和低速 CPU 的机器。Gmail 在这些机器上运行速度慢了很多，我们可以把所需的信号分辨出来，然后开始运行长时间的压力测试。头几个月特别艰苦，我们有几次误报。我们花费了大量的精力搭建基础设施，可没有什么产出。但是后来，回归测试的需求滚滚而来。我们可以测量到毫秒级的性能损耗并把数据记录下来。开发工程师能在几小时内就发现产生延迟的问题，而不是以前的几个星期。这样就可以趁问题刚出现的时候就开始调试，而不像以前得在几个星期以后才能开始。这件事立即为测试团队赢得了尊重，以至于在我们着手开展接下来的重要任务(修复端到端的测试和搭建高效的负载测试平台)时，开发工程师实际上还自发地帮助我们。整个团队发现了高效测试带来的价值。Gmail 的发布周期从每三个月缩短到每周，再到每天都能向我们的部分用户发布新的版本。

HGTS：所以经验就是解决掉一些难题来赢得尊重。我喜欢这点。不过做完这些之后你还做了什么?

Ankit：其实，难题永远也解决不完!不过你说的对，基本思路就是关注最重要的事。我们确定 Gmail 最紧要的问题，然后一起解决它们。通过团队配合，你会发现这些问题并不那么困难。当然，我还是坚信只应该关注最重要的事情。每当我发现团队打算做太多的东西的时候，就好像你要同时做五件事情，但是每件只能完成80%的时候，我就会要求他们退回来重新安排优先级。把你需要做的事情减少到两到三件，但都能完成到100%。这样团队才能获得真正的成就感，而不是好多事情在他们手里没有完成。如果这些工作最后都能积极地影响到产品质量，那么我也会感到特别高兴。

HGTS：大家都知道 Google 的每个经理都有很多直接下属，而且经理自己还需要从技术上有所贡献。你怎么平衡这些事情?能告诉我们你自己是怎么完成那些技术工作的吗?

Ankit：管理下属和与其他人沟通确实是一种干扰。我其实总结了两个办法来让自己能保持技术敏锐度并像工程师一样参与其中。

第一，在与开发工程师和测试开发工程师团队沟通的过程中，有好多事情可以做，我可以选择留下一部分自己来完成。我在设计阶段会积极地参与，持续地跟进项目并且自己也编写测试。

第二，其实这才是关键的部分。如果你想做一些技术工作，就必须尽量排除管理方面带来的干扰。

起先，我每周都花一两天的时间做我自己的工作。我有一个项目是把 Google Feedback 整合到 Gmail 里，这个工作让我能从开发的角度来看待测试。当我碰到一个脆弱的测试，或者测试架构的某些部分拖慢了我的测试进度时，我就能够理解那些全职的开发工程师怎么看待我们的测试工作了。尽管如此，只要我在 Google 总部的办公室，人们总能想办法找到我，所以我就跑到苏黎世 Gmail 团队的办公室去。虽然在那儿有九个小时的时差，但是环境就安静多了，我在那里也不是谁的经理。我可以混进一个技术团队而不怎么引人注目。我在苏黎世干了好多活儿!

HGTS：你对测试项目的人员配备有什么建议吗?开发测试比是多少会比较好?SET 和 TE 的比例呢?

Ankit：人员的问题其实很简单，那就是绝不妥协。选用不合适的人来填充名额永远要比等待合适的人员要糟糕。只选用最好的人，不能动摇。Google 不让公布人员比例数据，不过以前我们团队中测试人员的比例比正常水平高很多。自从我们解决了很多最初的问题，并得到开发工程师的支持以后，我们的比例就降到和 Google 的标准水平差不多了。从技能分配的角度来说，Gmail 的经验是用20%的测试人员进行探索式测试。任何关注用户体验的产品都需要探索式测试。还有30%的测试工程师关注于产品的整体性测试，他们和测试开发工程师一起来保证测试的效果。另外50%的工作，是测试开发工程师开发相关的自动化测试和工具，以保持代码库的健壮和提高开发人员的工作效率。我不敢说我在下一个项目还会按照这样的比例分配，但是这个比例对 Gmail 来说是有效的。

HGTS：我知道你现在开始负责 Google+的测试了。在新项目中你发现哪些在 Gmail 的经验是最有价值的?

Ankit：首先，不要把你所有的精力都放到前端。Gmail 拥有可能是最庞大的分布式后台系统，那里还有很多的测试问题我们尚未解决。除此之外，还有很多经验教训值得吸取：

— 使用与应用程序开发语言相同的编程语言来编写测试。

— 让负责开发新特性的人同时负责相应测试的执行，他需要对漏掉的测试负责。

— 关注测试基础设施的建设，让测试的编写和执行非常容易，甚至比忽略它们还要容易。

— 20%的用例覆盖了80%的使用场景(可能会有些出入)。把这20%自动化而别管剩下的。把那些测试通过手工完成。

— 这里是 Google，速度才是王道。如果用户只在乎一件事，那就是速度。确保我们的产品足够快。进行性能分析以便于可以证明给所有人看。

— 与开发团队的沟通至关重要。如果这点做的不好，你就会疲于应付，那可不是什么好事。

— Google 的 DNA 里富含着创新精神。测试团队也应该被看做创新者。发现重要的问题并能创造性地提出解决方案。

HGTS：你有发现技术团队可能遇到哪些陷阱吗?

Ankit：有的。假设我们知道用户的需求，然后进行了大规模的改动或编写了大量的代码提供新特性，却没有进行小规模的试验。如果用户不喜欢这些改动，麻烦就大了，而针对这些特性构造的测试框架再好也是浪费。因此，要先为少量用户放出一个版本，获得必要的反馈，然后再为大量的自动化测试进行投资。

另外，试图构造完美的解决方案可能花费太长的时间，到时候市场的发展早已超出你的想象了。应该快速迭代，展现阶段性成果。

最后，就像开车一样，你必须找到测试的离合点。过早编写测试，有可能由于架构的变化导致全部工作作废。若等待太久，则又可能错失测试良机而导致没有充分测试。测试驱动开发是不错的方法。

HGTS：对于个人来说有什么陷阱吗?年轻的测试工程师和测试开发工程师在新项目里会犯哪些错误?

Ankit：是的。他们可能一上来就开始干，不明所以。他们写了很多测试，但忘记思考为什么要写这些测试，怎么让这些测试为整体目标服务。编写测试的时候，他们往往没有意识到他们还要负责维护这些测试。测试开发工程师应该牢记测试应该是开发人员的工作而他们自己应该专心让测试成为开发人员工作中的一环。我们通过编写工具帮助开发人员做到这点，而且应该让开发人员在维护开发代码的同时也负责维护测试代码。这样一来，测试开发工程师才能集中精力让测试执行得更快，更容易分析。

测试工程师有时候会迷失方向，做起测试开发工程师的工作。我们希望测试工程师更全局地看待整个系统，全面地掌控整个产品。他们的重点应该是从最终用户角度考虑的测试，帮助测试开发工程师和开发工程师确保所有的测试和底层测试框架都被正确有效地使用。测试工程师编写的工具和对问题的诊断应该能够影响整个产品。

HGTS：除了你前面提到的性能方面的自动化测试以外，还有什么测试方面的工作让 Gmail 获得了巨大的收益吗?

Ankit：JavaScript 自动化测试。我们为 Gmail 本身加入了一个用于自动化测试的 servlet。通过它，开发人员就可以使用与前端开发一致的编程语言编写端到端的测试(译注：端到端的测试是指涉及整个应用系统环境，在现实世界使用时的情形模拟的测试。)。因为它使用很多相同的函数和程序库，开发人员对于如何编写测试代码很熟悉，没有学习曲线。他们可以很容易地写出一些测试，来检验他们的新代码是否影响了 Gmail 的正常功能，也能够更好地保护他们开发的特性不被其他开发人员破坏。现在，Gmail 的每个新特性都至少会有一个通过这个 servlet 编写的测试。最棒的是，在我现在负责的社交产品里面我也在用这个方法。我们已经有了大约两万个自动化测试!

还有压力测试。在 Google 你不做压力测试不可能蒙混过关，因为我们的所有应用都有大量的用户，后台数据中心的负载会非常大。我们基本上必须复制一份线上环境并引入真实用户流量。我们花费了几个月的时间分析线上系统的使用情况，构建了一个代表用户的使用模型。接下来，为了数据更为真实，我们使用和真实的 Gmail 数据中心一样的机器来运行我们的压力测试。然后，我们观察测试环境和被监控的真实环境上的结果差异。我们发现了很多性能退化的问题，并帮助开发人员细化和定位了这些问题。

最后，我们更专注于预防 bug 而不是检测 bug，这为我们带来了巨大收益。我们推动自动化测试在代码提交之前更早地执行，避免了大量质量不佳的代码污染项目。这让测试团队随时保持在最前沿，支持项目产出高质量的版本。这也给我们的探索式测试人员提出了更大的挑战。

HGTS：在选用人才方面你已经很有经验了。你现在转到社交产品项目上，你的测试团队需要找什么样的人呢?

Ankit：我需要寻找那些不会沉迷于系统的复杂性、遇到困难的问题时能够分解为可执行的步骤并能最终解决的人。我需要有执行力的人，他们会被紧迫感激发而不是吓跑。我需要能够在创新和质量中掌握平衡的人，他们不应该只满足于发现更多的 bug。但最重要的是，我需要能看到他们的激情。我需要那些真正想做测试的人。

HGTS：这也是我们最后一个问题。在测试领域什么东西会引发你的激情呢?

Ankit：我喜欢由快速迭代和高质量带来的挑战。这两者相互矛盾但又都很重要。这个经典的矛盾迫使我为这两个目标不断优化，而又不会伤害我自己或我的团队。创建一个产品不难，但要快速创建一个高质量的产品会有相当大的难度，而这正是使我的工作——富于挑战又充满乐趣。

女程序员的时间都去哪里了?

2月26日，北京雾霾已持续超过100小时，道路上的提示牌都在提醒人们尽量避免出行。上午10点，当记者到达雅虎北京全球研发中心的办公室时，茶水间阿姨正在收拾免费早餐的餐具，办公室里飘荡着早餐的气息，在这里做程序员，应该不是朝九晚五的节奏。

“我现在的工作量又回到08年那时候，每天工作时间都在14、5个小时，早上7点就和美国同事开会，晚上也在加班，好似二次创业。”

雅虎北京全球研发中心品质工程部总监潘敏肯定了记者的猜想，但结果出乎记者意料。

春节后，从9号开始上班，潘敏的时间表就安排的非常紧凑。因为雅虎 CEO 梅耶尔提出2014年一大目标是实现产品从代码设计开始到发布，必须在24小时内完成。潘敏领导的测试团队相应的也得在24小时内完成对产品品质保证的工作。这个目标不是轻易就能实现的。

潘敏说，雅虎产品原来的测试工作基本上是以一周为周期循环往复，每周都在出新产品，测试工程师就从产品定特性开始介入，然后写测试程序，最后产品代码完成，测试也自动完成。

测试工程师和项目经理的沟通是产品品质保证的第一步，产品特性应该是怎样的，要从哪些方面去测试，测试人员会和产品开发部门达成共识。这个周期细分下来就是：半天时间确定产品特性，设计需要一天时间，两天时间用来写代码，再用半天完成测试，第五天产品上线。

记者理解的产品测试就是给新产品挑毛病，潘敏纠正说：“我的工作不只是挑错，我做的是全面质量把控。”换句话说，产品测试人员比产品开发人员看问题更全面，这是职业带给潘敏的成就感。

当初选择做一名女程序员，对潘敏而言，是很自然的结果。

潘敏12岁随家人移民加拿大，高中时，因为学校不断地宣传计算专业如何好，潘敏尝试编程感觉不错，大学就选择了计算机专业和管理专业双学位，研究生阶段主攻计算机。毕业后第一份工作就是软件测试。

2007年潘敏加入雅虎美国，2008年到北京研发中心。

今年春节过后，潘敏的团队骨干成员被派到美国总部工作，潘敏的管理将从北京延伸到美国总部。

“在生活中，是否也是处处挑错、追求完美?”记者对潘敏的职业“副作用”很好奇。

“不会在生活中严苛，只是变得敏感，其实这个职业很适合女性。”潘敏说她总是能从家人寄来的照片上发现背景里不同装饰的改变，即时是很不起眼的一个小物件的摆放挪了位置。在北京家里，一点变化也会引起她的注意，而她的先生可能没有丝毫察觉。

潘敏说敏感一点没有什么不好，“做测试的人，性格都不错，善于与人打交道。”女性的细心和全面做这个工作就有优势。潘敏的团队，有1/3是女性，进入这个团队的都是有五年以上程序开发经验的员工，他们与产品设计研发人员沟通时，更能抓住要点和指出漏洞。

潘敏的时间大部分用在了工作上，女儿从出生到现在3岁的成长教育，更多是靠她先生和家人，为此，潘敏对家人充满感激。

高德女程序员于立娟的双喜临门

2月11日，春节后上班一周，阿里集团与高德软件联合宣布，阿里计划全资收购高德软件的消息， 高德软件运营商事业部高级工程师于立娟并没有比网友早知道。当天收到来自 CEO 成从武的邮件，于立娟才知晓公司的重大事件。

“这是一件好事啊，对公司发展有很大的想象空间。”2月26日，记者与于立娟相约在高德公司见面时，于立娟谈到高德员工对收购事件的看法。

于立娟说高德公司上下员工一致的梦想，是实现地图上的淘宝梦，让地图上每一个商家都成为淘宝店，其中支付是重要的环节，阿里的投入，将使这个梦想更快变成现实。

“2014年，你的工作有哪些变化?”

“除了原来做的天翼导航项目，会在移动互联网的生活服务上负责两项任务。”于立娟提到的生活服务项目是智能手表和智能相机，为此，近段时间她都在钻研 Tizen 系统平台，学习如何在 Tizen 上开发和应用工具。

于立娟是高德运营商事业部唯一一名做程序开发的女性，她认为做程序员对女生来说，压力太大。大学所学的计算机专业知识根本不能适应工作要求，所有的工作能力都是在实践中积累和学习的。

从2002年大学毕业从事软件开发，于立娟的工作相对稳定。

2011年，于立娟所在的协进科技被高德软件以830万美元收购，于立娟成为高德公司一员，继续做软件开发。2014年，阿里宣布全资收购高德，于立娟仍然是软件开发工程师，虽然几年间有来自百度等公司的跳槽诱惑，她还是坚定地选择留在高德软件，“可能去别的公司薪酬会暂时高一些，我还是喜欢和熟习的团队一起工作，这里的工作氛围一直很好”。

加班对互联网公司的程序员来说是再平常不过的事，于立娟认为自己没有后顾之忧，公司办公室从中关村搬到望京宝能中心以后，离家更近，家里还有父母支持，愿意接受这样的挑战。

生活中的于立娟自认为比较简单。没有逛街的爱好，网购就能满足要求，也节省时间，理财选用余额宝，“自家产品用起来放心”于立娟说如果去银行买理财产品还要看很多说明介绍，还得跑银行，用余额宝很简单，每天都能在手机上看到数字变化，很有满足感。

除了练练瑜伽或者打羽毛球保持身材的苗条，周末看电影算是于立娟休闲的一种方式。

对于立娟来说，2014年夏天她将升级做妈妈是今年最大的喜事。

“现在最不想做的事是拍照，长胖了”。

记者眼里的于立娟依然是年轻健康的正常体型，并没有胖的感觉。女性爱美的天性无论何时都不会改变。

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