泽众软件测试网
电子期刊

SPASVO

第十六期
2013年4月

上海泽众软件电子期刊

2013 年 4 月 第十六期

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 4 月 第十六期 电子杂志期刊

软件测试与开发的未来

今天与何老大的一些交流，引发一些心中很久的感想表达一下，主要是针对我们开发过程一些幻想，最后给出实现的规化方式。

1、云开发平台

我们开发人员整天忙忙碌碌，重复最多的就是编写代码->编译->简单测试->改代码->编译...

云开发平台正是为解决这个问题而来，它是什么呢？

所谓云，就是对使用者透明，所谓云开发平台，是指对我们开发人员（测试人员）几乎透明的编译调试环境。

你要做什么？

告诉它你的项目地址，告诉它你的编译方式。

它帮你做什么？

1、监控你的项目，有提交时帮你编译，返回编译结果。

2、准备环境，提供一个云端返回的编译完成的主机（我们的测试机），可以登录 ssh 进行测试。

2、开发过程自动化测试

我们现在正在测试前移，甚至在需求阶段介入，我这里不关注需求的测试方法，只说测试前移怎么去做？ 我们现在在强调前期的代码审查测试，前期的逻辑检查，这些属于白盒但是静态检视，我以为这些可以去做好，但仅仅对前期测试来说能暴露的问题有限，更多的时候需要靠更多的编码经验。

而开发人员在编码时更多的时间花费在调试（大约80%不为过），这部分工作实际上可以减少很多，而且大家也知道，如果更多的时间用来设计与编写高质量代码，测试的工作量也会更少，能够有效提高整个研发效率，而现在的问题是，开发不知道如何利用工具改进开发过程。

开发过程自动化测试是指，提供一种易用性框架，利用自动化测试优势，将过程的重复工作实施自动化测试，将每次都需要验证的测试点实现自动化验证。

效果是：

开发设计完成，开发编码。

测试前移，准备测试点，编写自动化用例。

利用某一个统一的平台进行交付自动运行。

难点一：对测试人员要求较高，但我们可以培养。

难点二：对开发有一定惯例限制，但每一次的限制用的好可以带来更大的自由（好处）。 像如今满大街的智能机不是对键盘的限制使用吗？

最后一点，也是最宏大的。

3、研发管理平台：

越来越多的流程，越来越繁琐的文档，越来越混乱的 IT 系统，经常这个账号记不清另一个账号无法登录的。

申请序列号这种小事都需要助理来处理，试想我们如果有一套完善的认证系统不可以自动下发序列号吗？ 系统会记录的更清楚。使用的人也会得到最快速的响应。

开发改了需求没有通知我!!!

忘了 xx 文档的 svn 地址了!!! 找其他人问还十分不好意思，有时候还得不到立刻答复，又影响他人。

SQA 累死累活的跑路收集各种信息，但却依然可能受到大家的数据置疑。

一项流程更新，通报了全研发体系却大部分的人在真正执行时仍然遗忘。

。。。

问题已经比较突出了。我们应该怎么做？

研发管理平台，正是我们的需要。

它的核心功能：

1、统一接入认证体系。保证内网安全可靠，并提供完善的日志。

2、集成研发流程，提供从需求到发布的过程跟踪。 可强制限定开发经理与测试经理的活动。 并做到智能提醒。(试想，每天我作为开发经理只需要登录一下系统就知道接下来该干什么， 每天只需要一次提交每日进展即可，并可随时查询项目成员的代码提交质量； 而项目责任人无需过多信息，通过研发管理平台即可收集到项目的信息）

例如，需求过程可以简化， 只需要一次录入需求， 以后每次需求变更，所有相关人员自动接受邮件，任何需求过程均被记录，系统发布前提供需求完成情况，自动形成可发布文档。

3、集成自动化测试， 提供统一的静态扫描并与 ATM 做接口。提供数据仓库可以随机构建有效数据，提供虚拟化硬件平台，任何人在需要的时候一键获取测试主机进行快速接入验证。

4、具备数据分析甚至挖掘能力，提供一定的 SQA 职责，供决策使用。

我们来初步分析一下，

1、云开发平台

实现难度： 中（监控 svn 应该不成问题， 提供调试编译环境，这点可通过部门虚拟设备解决，我们自动化已经基本解决虚拟设备实现克隆，开启，关闭，甚至修改 ip，执行任何命令的操作，剩下的就是工作量与需求问题， 其他难度在于解决不同环境部署的约束）

实现工作量： 低

实现效果： 能够节省每个开发人员的重复操作并易出错的问题。

限制： 需要开发人员配合实现代码文件存放和命名约束，以及相关需求细化。

2、开发过程自动化测试

实现效果：大规模提高发包与测试回归速度

实现难度：中

实现工作量：中（在于如何设计一个简单易用的框架来快速编写和执行自动化，这里与 ATM 平台不同的在于它是轻量级，更易于完成非关键字级的验证。并支持更多的语言。）

3、研发管理平台

实现效果最佳。

这个就不分析了，可以通过分步去做。

为什么想到这些？

首先，测试与开发是分不开的。我们测试的目的，保证版本质量，另一个也十分重要的在于提高测试效率。

开发的目的，快速高质量发布新版本，高可维护。细致一想，提高测试效率不简单在于测试过程，而是整个开发过程； 而高开发快速发布一部分依赖于测试的快速测试，除了高可用的架构以外，依赖于快速有效的自动化，依赖于高效率的工具。

不然，开发每次迭代10%开发，测试验证110%的功能局面无法得到任何改变，我们又苦又累却得不到结果。

为什么单元测试在我们项目中实施失败？

1、没有好用的工具， 如果有一个只需要写业务测试代码的单元测试框架被牛人整合出来，何担心没人去用？

2、没有明确的目标，或对目标效果要求太紧。 我们缺乏十分有效的数据度量，缺乏有经验的人，仅仅靠人的自觉基本上很难推行这些项目走向成功。

关于开发语言，

大多数人就像大多数人一样倾向于选择大多数人使用的语言， 而谓之于“最佳实施”...

而如果我说，学一门脚本语言吧，你可能会说， 没听说过图灵等价吗？（ 意指 任何计算机语言的表达能力是等价的，一门语言可以完成的事件，理论说另一门语言肯定可以完成） 脚本语言啊，太弱了吧？ 不能开发大项目吧？

实际上，目前是 Lisp 类的语言的天下，从 perl 开始， python， ruby 已经不只是开发小型项目了。大家都在使用 vCenter 的时候，知道它是什么写的吗？ 实际上，它的 web 页到启动脚本均用的 python。javascript 已经火了 N 久了，最近的 Node。js 把它从前端发展到后端。

是什么原因？ 高效的开发效率， 强大的表达能力， 越少的代码往往意味越少的维护成本。有兴趣的同学可以关注下<<黑客与画家>> 作者是硅谷的投资之父，揭秘了 viaweb 快速开发的秘密。

我们可以尝试部分内部项目采用它。

关于开发效率， 最近关注：

github.com（一个 git 托管平台） 开发语言 rails，python，ruby 开发周期，3个人3个月上线（2008年），目前管理项目5000万

zhihu.org（一个知乎类似的问答） 开发语言 rails，开发周期1周2个人。

淘宝运维平台（内部）（一键发布平台，目标：关闭运维部门，说笑了，这个正是运维部门在做：） ）开发语言 rails， 开发周期6个月1个人，目前基本上线。

如何充分利用动态语言的开发效率可以在这些内部项目更快的发布与维护。

说这样好像与上面没关系啊， 就像老大说的，没有人限制你要做什么。能够达到目标的一切措施都可以试试。

以上一点想法，吐吐为快。

再谈团队，项目，产品

最近加入新团队，尝试新的项目类型，一段时间一下也感谢颇深，目前也算是深入了解了团队和部分项目，其实平时也经常习惯性的思考团队分工协作这些方面的东西，鉴于目前团队状态和先前已经有明显区别，自然也萌生不同的思考。

老实说本人是带着先入为主的想法而来的，一向推崇敏捷，推崇领域驱动设计（下文称 DDD），故而也无时不刻向周围的同事渗透这些思想和倾向，特别是目前是从事企业应用方面开发工作，业务导向，尤为适合。

针对目前的主要系统（下文以 wf 称，企业应用）和团队，且听笔者一一分析道来.

目前 wf 团队人员组成，与一般互联网项目相比优势之一就是需求分析师经验丰富，均可算是某一业务领域的专家，比起辛辛苦苦白手起家的 web2.0创业型项目来说，我们可说有着无比清晰的系统建设方向，靠谱的需求来源。需求的方向性是明确的，这点对于开发者来说是最好不过了，我们说“靠谱”很重要，企业应用应该不至于让你被需求雷死：)。既然如此，为什么不用好这些资源呢？因为是企业内部，我们总是能和需求方以及业务专家保持直接有效的沟通，他们能够且应该比开发人员更了解业务该如何设计和系统该如何被使用，这便是实施领域驱动设计最为重要的资源和前提，业务是最具有价值的，因而我们希望并且应该做到当业务在系统中被实现时，它是可重用，可理解，可验证的，而不是用一堆晦涩的脚本或者数据库表来描述牵强的描述它。往往我们在和业务专家沟通的时候，他们已经参与或者在帮助我们设计业务，他们期望系统中业务的运行/实现方式是如他们所描述的一致，这对于业务的深入和衍进具有很大的意义，业务在专家的脑子里，若我们的设计映射了这种业务描述，那么也就等于系统/业务设计也就在他们的脑海中。

当下，企业信息化阶段已经从早期的企业网站式向企业 IT 系统化转变，IT 系统将对企业发展提供全面支持和对业务的灵活响应，您无法难要求您的开发人员都是业务能手，但是您是否发现原来身边的业务部门人员都是天生的业务专家，如果让他们参与系统设计，让领域专家来支持您的企业信息化建设是不是能让系统更良好的支撑业务要求？

Ok，对于以上文字，笔者得出的结论就是：业务是最具有价值的，业务系统应该重视和维持价值而不是断章取义的代码设计。

接下来，笔者假设您同意了这个结论，往下看。

上文反复提到让业务专家来设计业务和系统，那么如何来具体实施？业务需要描述，同时也需要被

实现为可用代码，显然业务专家不认识 C#/Java，因而业界提出了所谓 DSL 这样的领域特定语言作为中间语言来连通业务和系统实现，DSL 最近很火热的，感兴趣的具体您可以去了解一下。笔者认为目前这个还是概念，理想化的东西，业界希望定义出诸如工业标准的东西，理智对待即可，取合适而用之，“软件开发没有银弹”，DDD 也是一种应对之道而已。

回归正题，如何实施。简单而言，我们需要和业务专家一起进行系统的设计，用双方均可理解的方式和语言来描述业务和设计，设计应和分析后达成一致的业务描述可一一映射，业务专家也需要能理解一定程度的传统系统设计思想（比如对象，关联，领域模型等概念），其间可进行领域建模，编写伪代码等来描述领域问题，具体设计对于开发人员有较高的要求，需要具备良好的 OOA/OOD 能力。

实施 DDD，很重要的一点，必须尽量真实到达业务和具体设计的真实映射，如果设计和领域模型不能较直观的描述业务，那就失去意义，为保证这点，需要依靠良好的面向对象设计来保证业务映射，对最终设计需要根据确定下来的业务设计描述来验收。

从代码层面来讲，需要保证业务可重用性，必须保证领域层完整性，可测试性，以及保证领域业务不外放。往往分层架构会导致有些业务处理出现在不合适的位置，比如向 UI 层渗透，或领域服务暴露不当等，理想状况应该做到领域层的平台无关性，当然这是理想化，笔者认为领域可测试性，业务可验收性是最为重要的。

具体框架支持方面，谈 DDD 估计必谈 ORM，领域层再优雅也需要持久化，一个映射能力优秀的持久化框架是必备的，目前我们使用的是 NHibernate，虽然有些地方实在不那么优雅，但它还是非常优秀的。

至此算是简单的把领域驱动引入的可行性阐述了一番，看了上述的种种文绉绉的方法之谈是否觉得还是缺少些什么？具体实施层面的有了依据，从项目和团队层面来说，是否需要一个更为有效的管理方式来支撑？需求/业务永远在变化，那么敏捷开发号称拥抱变化，是不是能引入而用之？

看一些实际状况片段：

需求分析一周，一份完备的需求送入开发，评估之后4周，继而投入4周的漫漫开发中，3周半的时候开始提交测试，上线前一两天，需求方开始体验/验证系统，“这个怎么是这样的啊”，“需求变更了，这个地方要这样”，“明天发布来不及改，下一期吧”，“测试来不及啊，这不是白测了吗”“排到后面…”。4周后发布了，变更被排入开发，接着，回到下一个漫漫的周期…

“这个样式怎么乱了？”，“底层接口不稳定，没办法搞”，“这个为什么要这么做，有价值吗”

是的，这是瀑布式开发吧，我们的迭代周期是多久，“上次是3周，有次是2周，有次是4周，哦，”。

敏捷可以让我们变成怎么样？用 Scunm 试试看？

组建2-3个 Scrum 团队，3-4人，每组一个 Master，ok，开工，初步定义 sprint 周期为2周，可由一位需求分析师暂代 ProductOwner 角色，来全程跟进，将项目需求进行分析后，对功能进行合理拆分为 backlog，进入迭代，每个 sprint 结束需交付可用产品/系统功能，交付需求方或测试验收验证，依据情况可让已交付功能上线。休整后进入下一个 sprint。依此，磨合一段时候后，确定一个稳定的迭代周期，快速响应需求，定期准时交付可用产出。迭代期间进行每日会议，及时沟通。由于是长期的内部系统，无需对较大项目或复杂需求做详尽估时（预估的几乎不可能准确，准确的代价是拒绝了变化，这是不符合期望的），只需总是按照迭代周期向客户进行交付，稳定的周期对用户进行反馈，我们将欢迎变化，你可以在任何时候提出变更，当然变更总是有代价，提出人心知肚明，而合理评估后欣然接受相信一定是客户期望的态度，稳定的交付周期也会让客户明确感知开发成效，藉此将树立团队形象。

当前很多时候是自己去领需求来开发，Scrum 团队将由 owner 来接受需求，配合 master 来分析具体开发任务直至具体开发人，Owner 精神是需要的，但是一个东西还没有分配到你的时候如何去 owner？松散管理对于团队成型未必总是有效，ProductOwner 必须充当需求认领人角色。

上面是一个团队改造的设想，几周前就有这个构想了，或者说遐想。接下来开始演进笔者的看法。

Scumn 在具体实施层面都给出了很好的实践，大师们勾画的蓝图确实很让人向往，然而脱离实际也是举步维艰的，笔者尝试将该方法论和现状结合，但还是出了问题。

显著问题之一，QA 测试的介入，一般测试思想认为一个迭代周期后交付的部分功能测试往往会导致最后产品完整完成需要重新测试，将浪费这期间的测试工作。笔者如下反驳：

QA 用于保障产品质量，参与的测试以黑盒，功能性，验收性测试为主。作为最终保障，更多的价值在于最终产品质量保障，对于产品代码状况，设计质量无法涉及，而代码作为最终产出，其质量直接关系产品质量，性能等，应该有充分的测试保障，核心代码的白盒/单元测试，系统集成测试，以及最重要的对于业务设计的测试都不可或缺，测试驱动是一个可参考的实践，业界对于 TDD 评价颇高，可惜笔者目前未有实践机会。若转型为敏捷团队提供测试支持，更多的测试目的在于帮助验证产品设计，监督和提高产出质量，而不仅仅是为测试而测试。

回过头来，一个现实问题需要考量，QA 部门往往独立于项目部门，编制不在一处，服务于此，现实利益相关的考核等约束使得测试人员不得不考量实际工作量和效益问题。凡事涉及利益，多半是无果…

问题之二，当前并无明确的项目概念，通常以零散需求为单位进行开发，多半是单个开发人员独立开发，从 SVN 不强制加锁即可窥见协作程度并不高，零散需求工作量长短不一，暂无一致的分配者来

调剂，由开发人员自行安排，缺乏统筹和资源安排合理性，且人员技能组成不够理想，强行合并人员组建团队，磨合期恐怕较长，新 scrum 团队内的分工较难定义。

问题之三，基础系统以及相关积累的成熟度不够高，拆分多个团队必然容易导致团队按各自风格积累，统一标准为定义前，将产生多个版本的局部标准。

综上，论述了可用实践和相关的考虑，然而实际状态并非理想环境，一蹴而就显然不可取，循序渐进为佳，如 DDD 全面实施成本较高，项目主体架构已基本成型，切换不易，笔者认为可取的架构目标应该是健壮底层和基础服务，提高自动化和规范化开发，降低开发和维护成本，应用层面的编码要求可适当放宽，以较可行原则来做合理约束即可，根据当前的技术能力组成，规避应用层面开发的技术风险或繁琐程度，企业应用求稳定之后再求体验和性能，稳定的响应在可接受范围即可，重点系统和项目实施 DDD，TDD 等，敏捷团队还是需要有意识的磨合和组建，思想转变后相信会带来不一样的成效。

这篇博文提笔之前，笔者认为写出来的想必是 XX 卫道者的思维去扯方法论，写完后，发现其实是希望记录下观点的转化过程。

鲜明观点到最后被模糊了，不过文尾还是想提一些关于产品方面的东西，毕竟本文标题里也忽悠了产品的字眼，

本人虽自称已经 UE 疲劳，不过这玩意即使疲了也得做。

用户体验不可能只靠 UED 的，况且他们并非我们的直接资源，哪天没排上计划了就不做体验了？部门通常需要制定一系列产品标准来引导和规范其产品的质量走向，这些标准落实到实际执行者，遵循优先，而不应只从具体人员角度去度量它是否有价值（往往是由于其实现成本而导致这个思想），如品牌形象一样，任何产品或部门必然需要树立起产品/职业形象，标准就是帮助你在没有该意识的情况下达到这个树立形象目的。而标准也如双刃剑，若当前整体产品状况或阶段并不适合一个庞大而复杂的标准的时候，尽管标准很优秀，也需要考量它是否合适于在当下实施，或者它的实施成本与实际价值的比例是否足够诱人，如果在稳定性和可用性都还未完美保障的时候就大谈 UE，结局惨淡笔者已见了不下少数。合适的阶段做合适的事。

最近脑子一直有很多需要记下的东西，可能是过多过乱而又忙的无暇静下思考，故而酝酿了这篇杂烩式作品，希望观点杂糅在一起也能碰出一点火花吧。

本文转载自：http://www.cnblogs.com/wsky/archive/2010/05/28/1746013.html

你了解 QA 工作吗----解雇之前的五分钟

“这个错误和我上个星期发现的类型相同。他们什么时候才可以认识到这种问题？什么时候我可以将精力放在预防 bug 上呢？那是一个更高尚的职业吗？”一个感到不满的测试员说道。你也许认为质量保证是你测试职业的下一个合理的台阶，但是我曾经走过了那条道路并且不赞成这个观点。在这篇文章中，我们可以发现作者并不是唯一一个有那样的感觉的人。

Fiona Charles 曾是个测试员。在她公司做了六年的测试以后，她走向了许多测试人员都渴望的道路－她毕业了，成为了一名质量保证人员。最后，她开始帮助预防 bug 而不仅仅是在错误产生以后指出它们。顺便说一下，我在这里提到的“QA”，是指评估流程并且影响人们改进他们的流程，而不是比较常见的用法，单单只是“测试”的同义词。

但是 Fiona 的故事并没有到此结束。在做了几年的 QA 工作后，支持她工作的高级经理离开了公司，新的经理并没有给 QA 分享他们的感激。进度表的压力导致了以前所有人都同意的流程被扔出到窗外。人们说 QA 正在妨碍着他们 。Fiona 说她可能会在公司解雇她之前的5分钟离开公司。现在她尽可能地避免 QA 这一角色，这是一个非常痛苦的经历。

当 Fiona 给我讲述她的故事时，我感到奇怪，因为它和我自己的 QA 故事是如此得相似。当我做测试员的时候，我印象中测试人员在测试方面做得非常好之后应该自然而然的转到 QA 的角色。因此当我在发现 bug 的过程中受挫时，我就变得相信预防 bug 是一个更高的职业，我坚持要创建一个 QA 的团队。我绘制了一些流程图并且开始执行一个正式的检查流程。我攻读质量保证运动的硕士。但是在一年以后不知什么原因一切都塌陷在我身上。我真的不知道为什么我开始收到来自管理层关于我没有做好工作的投诉，但是我有一种和 Fiona 相同的怀疑，我可能会在公司解雇我之前的5分钟离开公司。

测试和 QA 之间的相似点

让我们来探究一下 QA 对于测试人员而言是不是一个更高的职业。首先查找两种角色的相似点。有一点是相当明显的－测试和 QA 人员都需要有对质量的热情。两者必须有锻炼附带一份实用性的热情的能力。当你不能直接控制产品的时候，软技巧（Soft skills）对于影响人们是很重要的。并且由于测试人员经常做一些 QA 相关的工作，他们可能已经有些部分工作的经验。

每个测试人员好像在职业生涯中遇到了一些问题是最大的挫折。他们不能忍受相同的错误一再的发生，是因为他们知道有更好的构建产品的方法。 因此 QA 小组好象是达到那个目标的最合适的地方。

但是它们之间的区别是什么呢？

如何区分测试和 QA 呢？测试集中在产品的质量，QA 趋向于从产品上移开，集中在流程的质量，尽管 QA 也要查看产品相关的度量指标。QA 需要更多的关于整个开发生命周期的知识。QA 人员必须得到来自经理，需求人员，设计人员，程序开发人员，配置管理人员，当然还有测试人员的尊重。多年测试员的经验给予了一些帮助，但是不可能让测试员成为在所有这些方面的专家。QA 甚至需要比测试更多的软技巧，因为他们需要获得组织中人们的信任并且使人们信服地更改他们工作中一些核心的方法。

测试人员有切实具体的交付物，例如测试用例，bug 报告等。QA 工作就比较无形，例如观察，提倡，推动和影响。当 QA 分析人员感到比起做测试人员时，甚至更远地远离了构建一个高品质产品的切实的方面，它就可能感到灰心了。

怎样会误入岐途？

以下是一些作为一个 QA 可能会遇到的问题。

- 为一个不是很成熟的组织做流程工作，可能不会理解工作的价值。最需要 QA 的组织是最不可能接受它的。因此你必须查找一个经历了一番磨难后才获得足够改进的组织以认识到可能有更好的方法。不同的人在对他们正在工作的公司的成熟度和程序有不同的偏好。
- 为最佳实践本身，作为一个最佳实践的倡导者，而不是帮助人们从他们的角度解决问题。你不可能强推一个流程到一些人的喉咙里并且期望它一直呆在那里。接近改变的最好方法是帮助他人发现他们相信存在的问题的解决办法。那也意味着你可能甚至不能够获得对于你工作的信任。
- QA 会碍事。这是真的－在短期内。从长远的观点看，较好的流程将给予回报，但是只有当你通过了由于推行一个新的流程而延迟了一些人喜爱的项目所引起的对你的怨恨。
- 成功地制造大块地流程变更是格外困难的。最可靠的方法是随着时间做许多小的改进，这样获得了许多的耐心。如果你真的试图快速的改变世界，要先准备面临彻底的失败。
- 当时间被缩短时，或是高层管理层的态度改变时，QA 通常是第一个被放在砧板上的。不要自寻烦恼地在你的办公室里挂很多的图片。

把你从 QA 队伍中吓跑了吗？

我相信除了 QA 还有很多好的测试人员可以从事的职业道路。如下：

- 管理层。获得一些作为测试领导的实践，试着做测试经理，并且寻找侧向的或向上的管理职位。
- 工具专家（Toolsmith）。如果你想保持技术并且仍然想留在测试领域，就成为一个测试工具的专家吧。
- 顾问。比较陌生，但是这是真的，在这个行业里许多非常有经验的测试人员都是顾问。
- 其他很多的事情。一些你在做测试员时也用到的技术－想编程，技术文档的编写，技术支持，市场等等。

QA 工作可能适合你。我曾经听说过一些真的非常擅长 QA 工作的人。如果你选择了这条道路，希望你从现在开始就好好的准备，否则，我希望你有一个关于从事 QA 工作的全新的评价。

我的质量保证工作体会

记得非常清楚，2010年9月27日接到新浪质量保证的面试通知，听着名字叫质量保证，而不叫测试部，首先名字让我多了一种好奇和憧憬，更想了解新浪质量保证的工作是怎样开展的，工作流程和规范是什么样的，跟别的公司比较是否有自己的独特管理之处。带着这些疑问来到新浪，经过笔试和面试，顺利的成为了新浪质量保证的一员。

最初我对这个职位的理解不是很透彻，以为这只是单纯的执行测试，但接手几个项目后，我越来越感受到这个职位带给我的喜悦与成长。我们的工作从拿到合作方需求文档开始，随后就是从需求中挖掘出有用的信息，并对有疑问的地方进行深入讨论及分析，对合作方需求的准确把握是异常重要的，深入分析需求，能帮助我们制定测试方案，更是保障测试案例覆盖率的前提条件。在制定方案的过程中，要充分考虑项目风险，这样才能最大程度保证产品质量，制定项目测试周期，最后进入下一步执行环节，在执行环节作为项目负责人不仅要承担具体的执行工作，还要对项目内测试成员进行工作检查，及时反馈项目组相关人员项目进度并对风险进行预警，尽己所能与项目组所有成员一起最大限度的提升产品质量。

经过这3个月的项目实践，我已熟悉部门的工作流程和规范，也有了很多感想。下面将入职以来的感受与大家分享。

规范，这是我来质量保证的第一感受。整个项目的生命周期，项目立项阶段，项目测试开始阶段，项目测试准备阶段，测试执行阶段，测试完成阶段都按照统一规范来执行，这使得项目可以有条不紊的进行，更好的评估各个环节可能存在的风险，提早采取措施，保证项目的进度和质量。只有规范的流程，才能明确每个人的职责，才能更好的解决项目中各种遇到的问题，才能更好的管理项目。虽然之前在书本中了解过项目流程，但是理解始终没有那么透彻，现在终于清晰的明白规范的项目流程，也很庆幸成为质量保证的一员。

严谨，这是我的第二感受。不论写用例，写各种项目邮件，提交缺陷还是编写工作日报，都要求每个描述、每个数据统计的正确性，测试就是一个严谨的行业，我们首先要把自己的错误降到最低，不管错误或大或小，我们都要时刻保持严谨的工作态度。严谨也是质量保证的又一座右铭。

效率，这是我的第三个感受。一开始觉得每天的工作量太大，太饱和，这也使我开始思考自己的不足之处，为什么别的人可以执行那么多用例，而我觉得工作量大呢，经过尽快调整自己的工作方式，更好的利用时间，和其他同事的沟通，明显的提高了自己的效率并且能够保证质量。

沟通，这是我的第四个感受。有时项目涉及到的人员多，部门不单一，测试人员需要采用各种有效

的方法进行沟通确认，推进项目正常运行，使我深刻的感受到沟通是测试人员必备的能力。

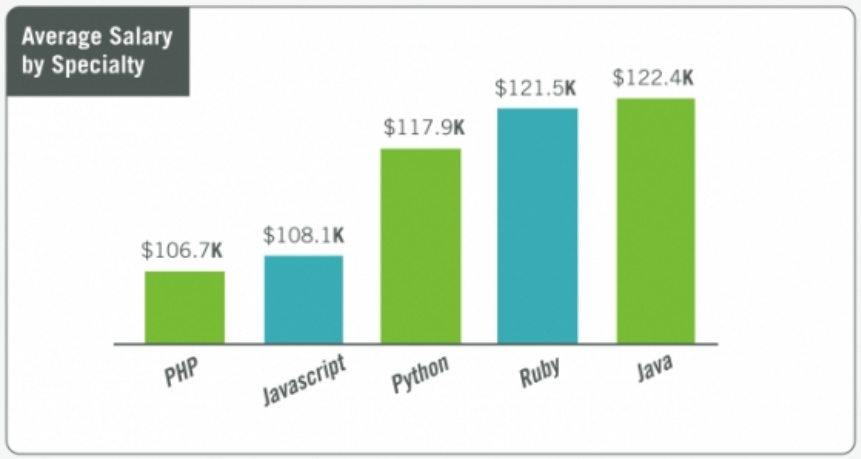
培训，这是我的第五感受。公司和部门各种技术领域的培训，不只是简单一次理论的讲解，而是深入浅出，注重培训效果，使我更能学到实际的知识，如自动化工具的使用、接口测试、性能测试等，深刻体会到团队浓厚的学习氛围。

你是一个工资太低的程序员吗？

我通过邮件和很多你们这些读者进行过交流，谈论程序员的生活状态。这些交流几乎都伴随着一个相同的主题。你是一个工资太低的程序员吗？如果你真打算问自己，那这答案几乎就是“是的”。

我们这里说的工资太低是考虑钱的方面。我们没有涉及到你从工作中获得了多少乐趣，或从工作中学到了多少知识。人们都想知道，跟其他程序员比起来自己的收入水平如何。然而，你总能找到一个比你挣得更多的人。事实上，几乎所有的程序员都是工资拿的偏低。

工资太低：普通公司程序员的情况



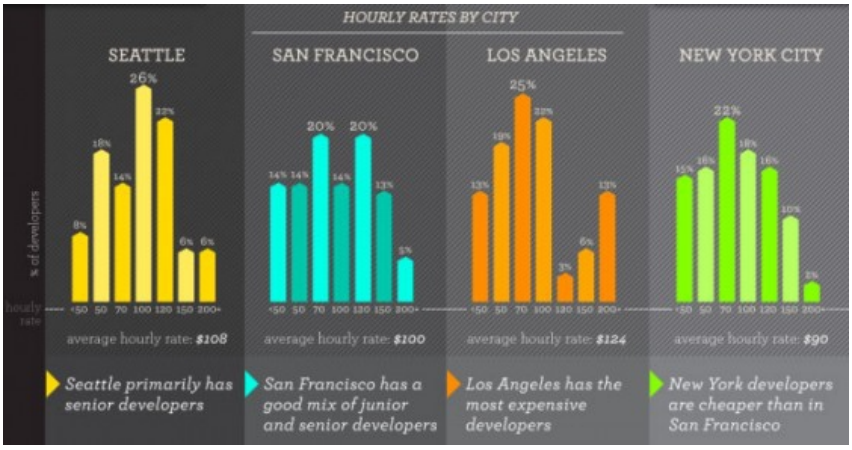
平均工资水平 信息来源: Riviera Partners

给公司打工，你永远都是工资太低。你应聘工作时他们永远都是低估你。他会想办法让你的讨价还价过程痛苦不堪。这是一种专门设计的社会体系，永远让你得到的报酬低得刚好不超过你的忍受极限。

出于一些目的，公司希望员工能对自己的工资水平保守秘密。所以，你不知道和你的技术水平差不多的人(或不如你的人)比你拿的工资多。总有这样的人。

甚至收入最高的程序员(年薪15-17万美元)也会大为吃惊于自由工作者的高收入。这会让你认识到你在公司的工作基本上就是做苦工。

工资太低：自由职业程序员的情况



各地程序员每小时报酬 信息来源: grouptalent.com

你就是一个被雇佣的枪手，别人给钱，你去干事。所以你可以为自己标上很高的价签。即使你的每小时的收费高达数百美元，仍然会有大批的程序员知道如何挣得比你还要多。他们的收入潜力不受他们工作工时的限制。

收入最高的自由程序员知道如何依据他们给公司生意上带来的价值来收取报酬。如果你能让客户相信他们诱人的前景，他们会毫不犹豫的答应你任何想要的价格。他们开发这个项目支付给你的钱，相比起他们能从这种业务上挣得的钱，九牛一毛。

我曾经看到过这样的程序员轻松的从自由职业上获得每年25万-50万美元的收入。

获得更高工资最大的障碍是你自己

一些程序员经过跟我的交谈认识到了他们的工资太低，想改变这种情况。而最常用的办法就是获得更多的编程技能——如果你是前端程序员，那就学习后端编程，反之亦然，或者学习一些新的技术，比如 Node.js。本质上讲，一个程序员永远都有可能通过进一步的学习挣到更多的钱。

然而，工资太低的感觉永远不会消失。编程是一种学无止境的工作。唯一你阻挡你挣更多钱的东西是你自己。行动起来，保持上进心，你就能得到你想要的。不要害怕，不要等待，不要以为你够资格后老板会主动给你涨工资。这一天你永远等不到。

自动化错误报告：通往高软件质量的大门

无知是福，但是在处理软件 bug 的时候，这句话并不适用。

软件质量专家会告诉我们，那些努力找出 bug 并且提高软件质量的公司可以得到客户更多的信任，得到更高的利润，降低开发和维护的费用，简化交付的流程，同时还可以避免客户的流失。

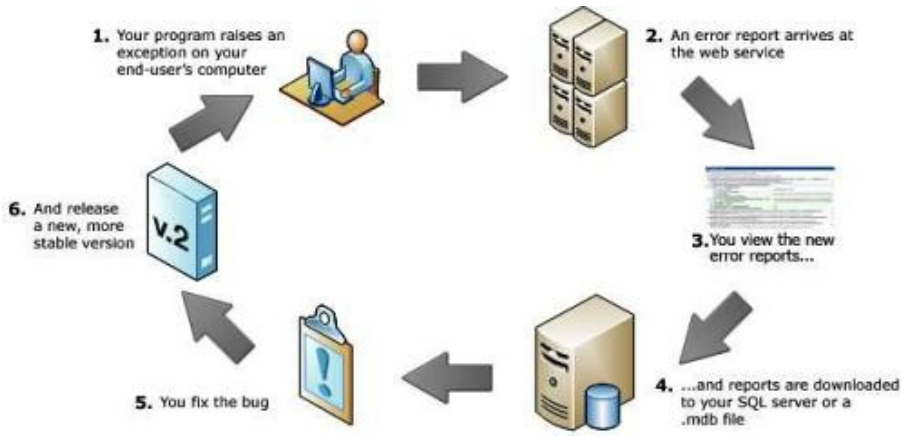
软件质量是个大工程

如果你觉得软件质量没有什么大不了的，那么就来看看 Capers Jones 在2011年6月软件质量杂志上发表的文章（这份杂志由美国质量协会主办）：

- 高质量软件的发布周期比低质量软件短15%。
- 高质量软件从第一个版本发布到之后五年的周期里花费的总费用比低质量软件低30%。
- 高质量软件每年的维护费用比低质量软件低40%。
- 规模越大的软件，质量越是一个重要的因素。

根据 Jones 的观点，软件测试在查找 bug 时的有效性有限，还不到35%。

考虑到软件质量对公司业务的影响，开发团队都在寻找新的提高软件质量的方法。其中最有前景的方法就是自动化错误报告（Automated Error Reporting），这种方法以用户为基础来确定软件的错误和异常。



我们的目标是让错误报告变得易用、完整和可自定义

有许多让用户参与到软件纠错当中的方法，比如邮件和论坛社区，自定义的软件功能，Windows Error Reporting（WER）和各种现成的解决方案。

但是这些方法都不一定比自动错误报告有效：有的方法会让用户承担过多的责任；有的方法不能收集到精确的信息；有的方法不能捕捉关键的特征，如变量运行时的赋值可能就是一种关键特征；有的方法不能把错误提交到通用的错误跟踪系统里面。

我们的目标是让错误报告变得易用、完整和可自定义，这就要求用户可以方便地提交错误报告，开发人员也可以方便地理解错误报告。这可不是一件简单的事情，每个错误报告都应该包括完整的堆栈追踪信息和有助于查找、修复 bug 的上下文信息。自动错误报告还应该可以和 bug 追踪工具整合在一起。

Andrew Neville 表示：“自动化错误报告让用户可以十分方便地提交错误报告。只要用户按发送键，我们就可以收到修正错误所需的所有信息。使用了自动化错误报告之后，我们让用户可以更方便地与我们沟通。”Andrew Neville 是 Neville&Rowe 的一名资深软件工程师，在 Red Gate 的 SmartAssembly 中使用了自动化错误报告，从而使商业智能分析软件 ImpactEdge 可以很好地追踪 bug。

认识自动化错误报告

自动化错误报告的价值在于让开发团队了解到未知异常的详细信息。它从五个方面为软件开发带来了好处：

1、它为用户提供了一种有效的反馈途径，增加了用户和开发团队之间的沟通效率。

2、它让开发团队了解到哪些 bug 是最容易发生的，让开发团队可以根据事实情况而不是猜测来修正 bug。

3、它让开发团队可以更快的修复 bug。

4、它可以以更低的代价确认和修复 bug。

5、它可以让开发团队在发布软件之前获得更多的前期用户反馈。如果开发团队正在使用敏捷开发，自动化错误报告可以让开发团队根据用户的反馈快速制定出迭代的计划，从而缩短迭代的周期。

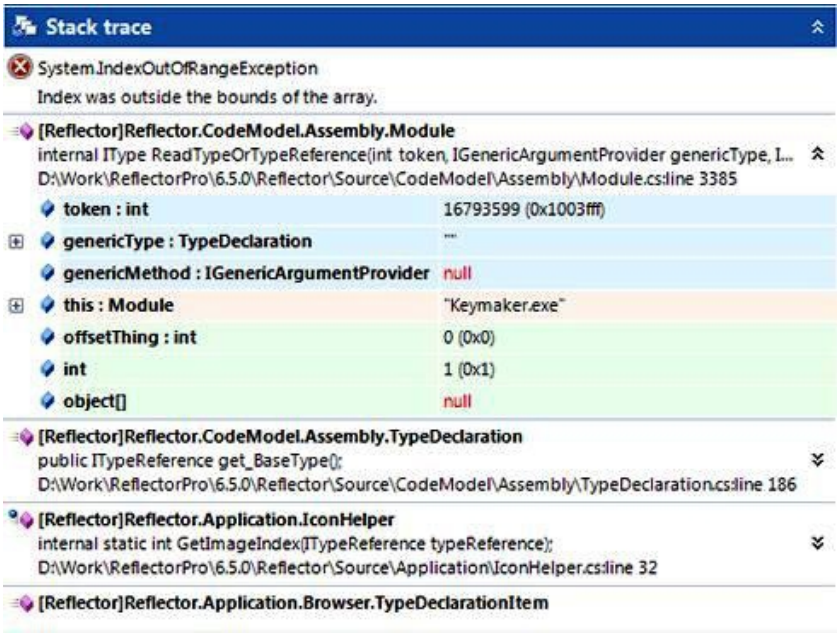
微软 MVP 和 IT 顾问 Ed Blankenship 解释说：“知道异常发生的频率对于修复异常来说是非常有帮助的。知道 bug 的详细信息对于修复 bug 也是必不可缺的。“

自动化错误报告是怎样工作的

一个理想的自动化错误报告系统可以让用户方便地告诉开发团队软件产品的问题在哪里。在理想状态下，当软件发生异常的时候，用户应该可以得到一个简单明了的错误提示并且可以通过用户界面发送错误报告。这可以大大节省用户花在描述和反馈错误上的时间。所有的必要信息都应该被自动收集，而不需要用户操心。

一个错误报告应该包括了完整的堆栈信息和异常上下文信息，包括运行时变量的值。

另外一个特性是用户可以在异常报告中添加额外的自定义信息。这些信息包括日志文件，截图，或者是用户的联系信息。



隐私问题和不合理的错误报告

发送错误报告的时候经常会出现下面三个问题：

1、错误报告会附带用户的隐私信息。

2、用户反复遇到相同的错误，但是每次系统都要求用户重新发送一次错误报告。

3、开发团队不停地收到相同的错误报告。

第一个问题涉及到用户的信任程度。再好的错误报告系统也不可能保证绝对不会把用户的隐私信息传送给开发团队。好的错误报告系统应该考虑到敏感信息的问题，这样可以尽量避免发送敏感信息。如果开发团队没有能够控制敏感信息的传送与否，那么错误报告系统应该可以让用户选择是否信任开发团队并且把敏感信息发送给开发团队。

第二个问题的解决方法是用户应该可以自定义错误报告发送系统。

Red Gate 的程序员 Alex Davies 说他总是会在合适的时候才显示错误报告提示，并且给用户“不要再提醒了。总是发送报告。”的选项。

针对第三个问题，开发团队应该把错误报告进行分类。Davies 告诉我们应该把收到的错误报告与 bug 追踪系统进行同步以甄别是否是一个有效的错误报告。

使日常工作更加方便

随着时间的推移，自动化错误报告的好处就会显现出来。其中有一条是最重要的，那就是可以让用户和开发团队都可以更方便地工作。一个 IT 决策者讲的话很好的概括了这一点：

“用户只需要点一下按钮就可以把信息反馈给开发团队。这避免了不停地电话或者邮件以及不完整的错误报告。这的确可以让用户和开发团队都更方便地工作。”

软件测试总监的一封回信

在《软件测试团队新人，遇上这个情况该怎么办？》中提到了测试流程中关于 bug 定义，流程方面的一些争论。

我之所以选择这个话题作为 blog 的第一篇文章，因为我也有过类似的体会。

这里把这三个问题再列一下：

1、一般来说，在项目准备阶段，会树立一个缺陷等级（bug bar），定义缺陷的严重程度。随着项目的进行，这个缺陷等级应该发生变化呢，还是应该保持不变呢？

2、当发现一个 bug 后，会根据缺陷等级来定义这个 bug 的严重度，比如1级，2级或者3级。一旦一个 bug 被发现并且赋予了对应严重等级后，是否存在其他因素导致这个 bug 的现有等级发生变化呢？比如研究后发现，修复某一个 bug 可能需要花很多时间，这个发现会导致这个 bug 的严重度变化吗？

3、对于发现的 bug，修还是不修，取决于哪些因素？除了 bug 的严重程度和对用户的影响外，目前团队的进度和资源对做决定是否有影响呢？比如本来有些开始准备修的 bug，到了后来发现开发进度滞后了，会不会就决定不去修这些 bug 了呢？

回到上面的话题，我这里先把测试总监的回答帖下来：

1、缺陷等级（bug bar）是根据产品质量标准来定义的。在不同的产品周期（milestone），缺陷等级标杆可以是不同的。比如在临近项目结束，已经到了 BETA 的最后阶段，或者马上就要 RTM 发布了，这个时候的缺陷等级标杆就会非常的高。这是为了防止项目后期 regression 的风险。

2、bug 的严重程度，是把这个 bug 给客户带来的影响，同缺陷等级标杆比较得出来的。只要这两个因素没有发生变化，那 bug 的严重程度就不应该变化。对于修复 bug 所需要的开销，当前项目的进度等，都不应该对 bug 的严重程度计算有任何的影响。

3、一个 bug 修还是不修，同样是有当前产品周期的缺陷等级标杆所定义的。如果预先已经定义好了哪一个级别以上的 bug，必须在当前（milestone）修掉，那就一定要修。如果时间不够，那只有延长当前项目周期。或者极端的时候，会评量是否需要裁减功能。但总的来说，对当前产品周期定义好的质量标杆才是修或者不修的标准。当前 bug 数量，资源，进度什么的，都不是考虑的因素。

上面的回答，是小王和测试总监面谈后，测试总监专门总结下来通过电子邮件发给小王的。

小王非常满意测试总监明确利索的回答。但是，这并不等于小王心中的阴霾就以扫而光了。小王接下来又问了这样一个问题：

“现在项目比较被动，作为测试人员，我会按照上面的标准，尽量把产品缺陷提前找出来，并且坚持上诉原则，确保产品质量。但是这么多 bug 都一定要坚持修的话，看来推迟产品发布很难避免了。那到了最后作工作总结的话，作为测试人员，既然我做好了测试工作，也坚持了产品质量原则，那产品延期是不是我就不需要承担责任，反而应该得到奖励呢？”

请问，测试总监的第二封 email 应该怎么回答呢？

PS：

如同有朋友卡通一下提到的那样，这是没有正解的。看了大家对前一篇 blog 的回复，发现不同的人看待同样的问题，角度和思维都有不少差异。有的直接去考虑出现这种被动情况的根源在哪里，有的追求解决当前问题的现实可行办法，还有的认为测试人员的基本原则才是最重要的。我觉得这样的讨论非常有意思，同时也想听听各位对于软件测试，有哪些感兴趣的话题？

总结一下各位朋友的观点，和网上其它渠道收集到的一些看法：

看法1：

1、bug 等级应该分两种：严重程度和优先程度，严重程度是不会变的，优先程度在不同的阶段是不一样的。

2、严重程度和是不是有足够的时间修复是没有关系的，不能说这个人快死了，现在没时间医就说他还好着呢。

3、都有影响的，严重程度和进度都会影响到一个 bug 是不是应该修复，但是不是要修复 bug 不应该是开发人员自己决定的，也不应该是测试人员单独决定的，应该由开发，测试和 PM 共同决定，如果这个 bug 很严重，即使延期也必须修复，不那么严重，则可以推迟到下一个版本再做修复。

看法2：

如果我是小王，那么此时我到是觉得我必须要尽职尽责，对目前系统存在 bug 进行归类总结，分出 bug 的优先顺序以及 bug 出现所属 rootcase，并更具这些归类做一个整体的解决方案，然后尽可能的去和开发人员进行沟通，尽可能做到既不影响当前的开发进度又能解决这些 bug，当然解决 bug 的前提是以及你掌握的优先级的，比如决策性的 bug、逻辑混乱的 bug 等等那是必须需要当即进行修正的。

同时了，此时和老板进行交流时，我想就不会是用一堆 bug 来和老板面谈了，而是说目前系统存在

了 bug，而现在我制定了解决 bug 的方案，我也尽力说服大家来对 bug 做修改了，同时也保证了开发进度。

看法3：

1、缺陷等级不一定等同于优先级。问题的严重程度并不一定取决于问题的技术性质。很多项目，问题是否能得到及时解决取决于该功能是否常用，影响面是否广大等因素。至于说到缺陷等级是否变更，我觉得要看你这个缺陷等级所要表述的是什么概念。

2、看了这个问题，前面的问题，即缺陷等级，可能偏向于缺陷本身的性质这样一个概念。那么，的确存在相关因素会导致这个 bug 的修改优先级发生变化。例如：形象因素等等。

3.1 Bug 修改与否，取决于较多因素，常见的是以下几种情况

3.1.1 bug 本身的性质

3.1.2 Bug 的影响面，是否会影响现行业务或对业务支撑造成不可弥补的错误

3.1.3 不修改会存在巨大隐患么？

3.1.4 客户关注

3.2 客观而言，不会。有影响的只是项目最终交付的时间和进度计划。当然，随着项目推进，之前的 bug 可能会被冲叠，造成隐匿，这种情况的隐患极大。所以，必须强调的是 Bug 修改优先于新功能开发。

3.3 这种情况最好不要出现，一旦发生，即意味着项目可能会出现较大的不可控因素，导致质量的急剧下降。除非，这块功能不再使用，或之前的需求存在漏洞需要重新考量。

看法4：

1、个人认为不应该变化。

2、严重等级一般不会发生变化，而发生变化的可能是优先等级，这是两个概念。

3、修还是不修，主要是看这个 BUG 是否在进度的关键路径上或核心功能模块，如果在那么就必须干掉，否则可以 postponed；进度和资源当然有影响，这些都是需要项目团队及早的做风险管理规划的，如果协调不好资源和进度，那么非关键路径上的问题可能会成为关键路径上的问题，那么这个时候就对团队的执行决策产生影响；发现的 BUG 应该都要修的，只不过会根据进度安排，把一些不重要的 BUG

推迟到下一个版本修复

看法5：

这件事说明一个问题！测试人员在公司的生存状态！换成我，我也不会妥协，Bug 可以 pending，但是不能一直拖下去，你们公司如果这样发展下去，迟早要出问题的！我现在会定期去查看 Bug 状态分析图，如果 open 的 bug 多的话，我会要求开发暂停新功能的开发，先处理 Bug，我想这样是对公司，对客户，对自己负责，测试有时候要强硬一点！

上述的看法我觉得没有谁对谁错。因为不同的公司和不同的项目特点，决定了解决问题的方法会各有不同。从国内的不少技术社区了解到，很多人觉得国内测试水平比较落后，其中例子就有比如没有统一的规范，对测试不够重视等。这里我会尽我所能对测试领域做一些分析和讨论，希望有所帮助。

欲速则不达，我们是否需要 放“慢”脚步

在软件开发时，经常会出现质量下降的时候，从我自己做过的项目来看，主要的原因是开发的太“快”了。这里的快不是真正的快，而是有问题的快。

这里可能有的人说了，我们要的是快速的反馈，我们要的是“敏捷”。这里我想说，如果只是给客户演示产品最终是什么样的（这只能是 Demo），那么这种快，也许可以接受，但是我们很多时候做的都不是 Demo 产品吧。

我说过这个问题，但有人反驳说：“我们不需要过度设计”。然而，我的经验是我们设计不足的情况比过度设计要常见的多，为何设计不足？大概有一下几个原因：

- 程序员“说”时间不够。
- 程序员设计方面的知识欠缺。
- 程序员被要求快速的完成功能。
- 程序员受到太多干扰。

上面有两种情况都是太快了（怎么解决上面的问题，这里就暂不提了），举个 Web 开发的例子（以下的三层开发，都是指单个功能的，而不是指整个层全部完成才进入下一层，因为常常是增量开发）：

- 代码全写到 Code Behind 里，然后就迅速的写界面，然后就不停的运行页面，发现问题，修改，最后完成功能。
- 典型的三层结构，写界面，写后台代码，写逻辑层，写数据层。由上到下。运行界面调试，修改。
- 三层结构，定义 Model 层，定义数据操作层，定义业务逻辑层，写界面。运行界面调试，修改。
- 三层结构，从底层到上层，每一层做充分的测试后再进入下一层，最后写界面。

上面的4种方式，第一种最先看到界面（但常常最后完成，因为代码有问题，调试修改很麻烦，应为代码多时很难定位错误），第二种次之，第四种最后。第一个迭代（或者说项目的早期）第一种方法最先完成，第四种方法最后完成。所以有不少人人习惯了前三种方式，然后被一种“高效率”的假象所欺骗，这往往会瞒过管理人员甚至是客户。但最终回过头来看，整个过程是“快、慢、更慢、项目停止”。我不是危言耸听，经常会出现这样：

（下面的四个迭代只是说明顺序关系）

- 第一个迭代递交功能了，但质量较差。
- 第二个迭代也递交了，但之前差的代码让我们慢了下来。
- 第三个迭代想递交时，发现差的代码越来越多，开发效率大大降低，客户开始受不了了。通过加班或者延迟提交，终于客户可以看到功能了。
- 第四个迭代刚准备开始时，发现客户反馈了大量的问题，当想修改代码时，发现代码已经一团乱麻，我们自己这个时候会说当时最么会这么些呢？由于什么什么原因太难改了，我们需要推倒重写。Oh My God，客户这个时候会说，见上帝吧，偶尔会有客户客气点说“下次再合作吧”。

通过上面的分析，有时侯我们太快了并不是好事，我们是否需要慢下来，为每一层的方法加上测试，

一步一个脚印，而不是急于展示自己的“成果”呢？ 我们每一步都走的很自信不是很好吗？

最后补充一下：

有以下几种慢，是不同于本主题说的慢，我一般认为是人品有问题。

- 一天应该做完的，三天都没做完的。
- 三天的活用一天做完，汇报说是干了三天的。
- 在客户的项目里实验各种新技术的。
- 晚上不知干啥，白天眯着眼睛干活的。
- 白天边上网，边聊天，边…， 边写代码的。

本文转载自：http://www.cnblogs.com/cnblogsfans/archive/2009/07/29/1534523.html

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。

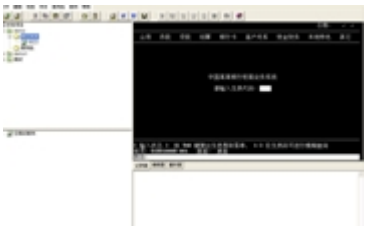
在线体验		产品租用	
企业版	免费版	在线申请	详情

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