目 录

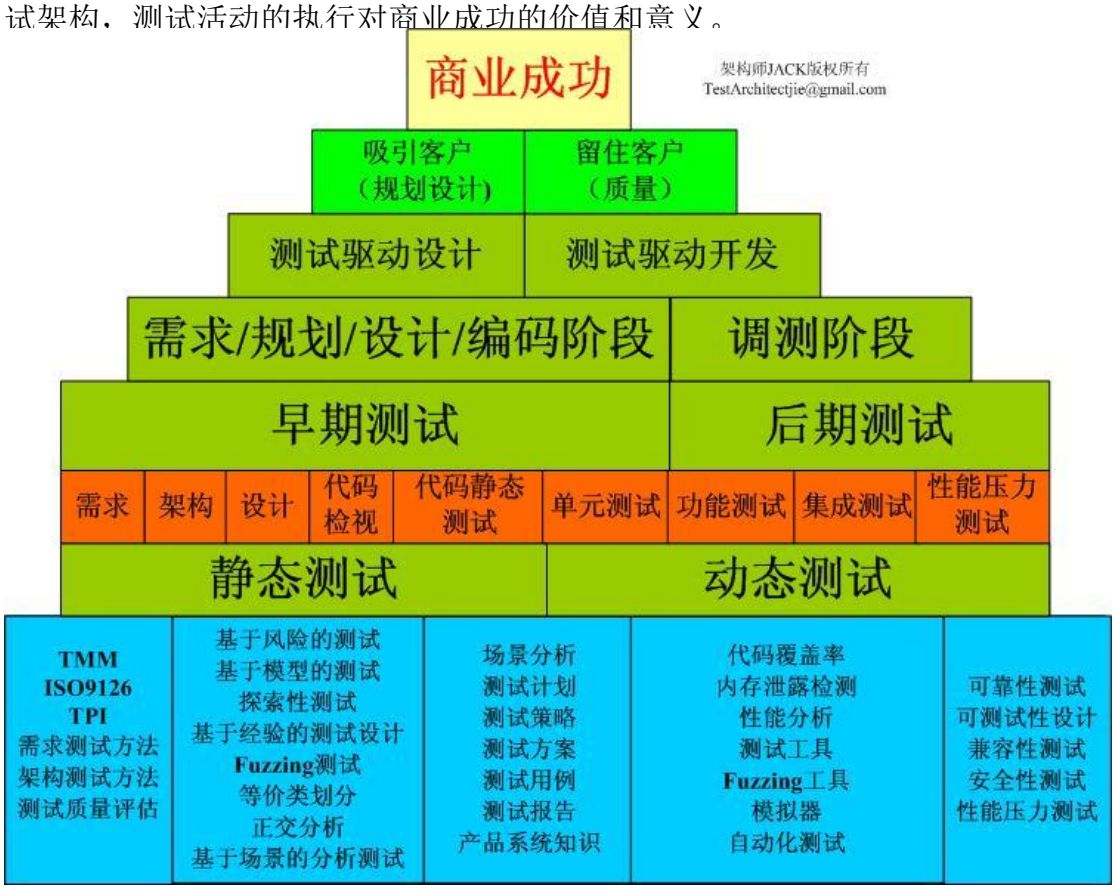
测试架构支撑商业成功（第二部分）

作者：架构师 Jack

上期测试架构支撑商业成功（第一部分）回溯：

什么是测试架构？商业成功的关键是什么？测试仅仅是找bug吗？经过多年商业环境中测试工作的经验和实践，我将在本文分享心中的——测试架构支撑商业成功。【本文第一部分见《51测试天地》电子杂志第十六期】。

首先，请大家看图一，后续的内容将是对图一的一个全面阐述，告诉大家测试架构，测试活动的执行对商业成功的价值和意义。



图一

三、测试的输出

测试计划

测试计划与一个产品商业项目计划没有太大本质的区别。

测试计划主要包含内容有：

测试范围；测试入口和出口标准；测试用例集；测试用例开发；测试预算；测试进度；测试工具和其他测试资源；用于测试的风险管理；测试状态报告。

测试计划本身至少需要考虑包含如下核心要素：

测试范围——测试计划所要达到的商业目标的范围，也可以理解成需要为多少商业目标提供检测服务，这是决定项目成本，项目进度，项目质量的源头。如果测试范围的制定过于主观，脱离现有资源的现实或工程科学性，那么可以说这个测试计划就只会是一个“无限通用”但又没有实际指导意义的计划。而好的测试范围则能帮助阅读测试计划的项目经理，开发经理，测试工程师们清晰明白的知道在后续的测试活动中，我们测试团队将会对哪些商业目标进行检测活动，测试工程师明白测试的商业目的，项目经理和开发经理知晓是否本计划的测试活动最终与项目目标对齐，有无对商业需求的遗漏。

测试项目的大小和资源——有经验的测试经理会根据测试范围的内容，预估本测试计划项目的大小（项目需要多少测试人力来执行），需要多少相关测试工具等资源。

测试进度——依据测试范围和测试项目的规模（大小和资源）来制定，完成测试范围所需要的里程碑。

风险管理——在测试计划中，测试计划的制定者需要依据项目的情况预先进行项目的质量风险预测，并针对预先识别出来的质量风险进行风险管理，针对不同的风险值制定不同的测试策略。

总体测试策略——将为整个测试计划确立不同的测试阶段，为测试范围选择不同的测试技术和测试活动。

测试计划本身的内部冲突：

当公司所要求的测试进度与测试质量要求发生冲突时，如何处理呢？为了保障测试本身的质量水准，应该在制定测试计划时通过减少测试范围的内容来应对。这样既能保障公司所需要的进度，又不失质量。

测试计划本身的成本：

1年左右的项目，可考虑用几周时间来制定；

1个月左右的项目，可考虑用1-2天的时间来制定；

1周左右的项目，可考虑用几个小时制定；

1天左右的项目（敏捷story），在早上的站立会议上需要1个小时即可；

测试策略

测试策略的制定可大可小，依据测试策略制定者自己的经验和能力来把握。经验丰富的测试策略设计者可以制定整个项目的总体测试策略，此类测试策略甚至能指导和影响后续每个测试计划的制定，确立整个项目测试资源的投入重点，确立整个测试项目会拥有的各类测试阶段和测试活动，应该使用的测试技术，每个测试阶段的测试技术组合。经验稍少的测试策略设计者可以针对每个测试阶段来制定测试策略，确立在该测试活动中测试技术选取，测试资源投入的缘由。在测试策略设计时最重要的输入就是项目的质量风险预测，整个测试策略应该围绕如何充分利用现有测试资源来规避项目的重大质量风险来开展，可以这样来概括：质量风险是测试策略的指南针。而测试策略将会让项目经理明白项目的测试资源会被如何科学的利用，项目经理也可通过测试策略的清晰度，逻辑关联性，科学性了解测试团队是否为测试的明白人。

测试方案

常见的测试方案有性能测试方案，压力测试方案，自动化测试方案，安全性测试方案，XX场景测试方案，XX可靠性测试方案等。测试方案是测试计划和测试策略向测试用例转化过程中的重要测试设计中间件。一个测试方案至少应该包括：特定测试场景需求分析和场景风险分析，并从中提取出测试需求，在测试方案中考虑如何通过哪些测试工具或测试技术，测试用例来覆盖到这些测试需求。有时在时间非常紧急的情况下，甚至可以直接利用测试方案来进行测试需求的验证。

一个好的测试方案能很好的回答项目经理这样一个问题：“测试用例到底覆盖了多少测试需求？每条测试需求在哪个用例被覆盖了。”

测试用例

测试用例不仅仅只有功能测试用例，还有性能测试用例，压力测试用例，安全性测试用例，XX可靠性测试用例。测试用例是测试计划，测试策略，测试方案一系统测试分析和设计活动落地实施的最小测试设计单位。它们将是测试项目进度管理和资源管理的最小计算单元。好的测试用例集设计能够很好的回答哪些测试需求被覆盖到了。但是测试用例的设计不可能是一次性的，一次性设计的测试用例只能是用于基本verification的用例。如果希望能发现系统潜在的更多的深入缺陷，需要对测试用例进行多次补充，依据探索性测试活动或是随机测试活动来补充testing 测试用例。关于什么是好的测试用例，在互联网上已有足够多的资源了，大家尽可到网上获取。但在这里我仍要特别强调测试用例的设计一定要考虑避免出现用例理解的二义性，提高用例学习的易用性，这些将是保障后续测

试用例执行质量和测试执行进度的重要支撑。

测试报告

所有的测试活动最终可交付的最重要的成果就是测试报告，无论是一批功能测试用例的测试报告，全版本的最终测试报告还是性能测试报告，压力测试报告，安全性测试报告，XX专项测试报告等，它们的价值都不应该仅仅只是用例通过数，发现bug的直接描述，而更应该包含测试人员从产品质量风险和产品质量评估的角度的分析语言，从发现的bug中分析出系统潜在的质量风险和测试活动本身还待改进的地方，从测试通过的测试项中得到哪些质量风险得到了验证，给出产品质量的正向反馈意见。在我心目中，一份只有bug类别和bug描述列表的测试报告最多只是一个勉强及格的测试报告。而从bug现象和测试项通过的数据中，得出项目在质量领域的结论性语言和对后续测试，开发，设计，需求分析活动的改进建议，对市场活动的建议的测试报告才是有质量，更能体现测试价值的报告。

项目经理，开发经理，市场代表，需求分析师都可以从测试报告中直接得到能指导他们改进后续活动的有依据的建议，才是测试报告的最终价值体现。

产品系统知识

测试人员进行了一系列的测试活动后，对组织还会有这样一个额外收获，即测试团队还能为公司输送产品经理，需求分析，售前，售后的人才。因为，测试人员有着比程序员更能在最短时间内最全面的熟悉产品系统业务知识的机会，经过几轮测试后，测试人员对产品系统的知识积累很多时候比公司其他岗位的同事更宽厚，无论是对产品的功能，性能，亮点，短板都有直观的感受，而且对产品的总体架构和主要设计技术都有一定的认识和知识。所以，这时的测试人员成长为了产品经理不仅仅具备了产品经理，需求分析人员的能力素质，而且还能对产品质量的改进有着更有力的支撑。同时测试人员具有的客户意识结合积累的产品体系知识，又足以使测试人员比现有公司的其他售前和售后技术人员，更熟悉产品业务，能在客户处树立起公司的业务专家形象。

四、测试工具

代码覆盖率

代码覆盖率最大的价值在于评估测试活动是否足够和充分，100%的代码覆盖率能证明测试活动基本验证了最初的产品设计，但是并不能证明产品不再有质量风险或没有bug了。因为代码覆盖的技术有多种：语句覆盖，分支覆盖，条件覆盖等。每种覆盖率达成目标的成本和技术难度都不一样，成本低和难度低的覆盖率，如语句覆盖率，只能说明测试验证了每行代码，能找到基本的编码错误，但是不能发现逻辑性设计错误。

虽然代码覆盖率不是一个可以100%衡量评估产品质量的工具，但是却是能衡量一定阶段测试活动质量的方法。如果产品连最基本的语句覆盖率都没有达到100%或90%，那就说明测试用例的设计还存在很基础的遗漏。

目前在互联网上已有一些代码覆盖率工具供大家自行下载，例如Gcov。代码覆盖率最好由测试人员通过黑盒测试用例结合部分单元测试来验证，而不应由开发人员全通过单元测试来验证，因为难免部分开发人员会为了达成代码覆盖率而刻意编写提高代码覆盖率的单元测试代码，“成为一个为了指标而做指标，忘了目标的活动”。

内存处理错误检测

内存处理错误检测工具的最大价值在于降低发现和定位代码内存处理错误bug的成本，尽早发现更多的此类bug。主要的内存处理错误检测工具有purify和valgrind，利用这些工具，我们甚至只需要执行一般的功能测试用例，就能发现代码中内存处理的错误，例如：未初始化，调用越界，开始出现未释放的错误。通过此类专项工具的告警信息，就能花很小的成本发现代码中潜在的内存处理错误的bug，而不用等此类问题积累到一定程度后，才能在产品层面暴露故障现象。

因此我建议：为了加快产品Time to Market的进度，减少产品研发成本，测试组无论是在功能测试阶段，性能测试阶段，压力测试阶段，自动化测试阶段，其他专项测试阶段，只要对系统进行任何黑盒测试，都在测试执行前先开启内存处理错误检测工具，在测试执行完成时，除了观察被测试目标是否达到预期目的，还要查看检测工具的测试告警报告，重视工具测试报告中的告警，并作为发现的系统bug一样进行跟踪定位。

Fuzzing工具

Fuzzing工具的最大价值是发现常规思维无法发现的一些系统边界值状态下的bug，其思想根源来自探索性测试。通过Fuzzing工具我们能在很短的时间内发现测试目标多个可靠性领域的bug。对于项目的商业目标，能提高系统的质量可靠性，同时降低发现此类可靠性bug的测试成本。但是Fuzzing工具最大的不足之处是无法对此测试活动进行定性的结论，不能证明系统不再有任何可靠性方面的缺陷，只能证明系统一直还存在Fuzzing工具所发现的这些bug。

而Fuzzing工具的最大缺陷是其经济性，它属于一次性使用就失效的产品，为什么呢？因为大多数Fuzzing工具只是异常穷举工具，当穷举的组合都在被测目标上验证完一遍后，该发现的bug都被发现了该工具的生命也就结束了。因此Fuzzing工具不像Loadrunner或winrunner之类的工具能有较广泛的应用范围和生命周期，所以，Fuzzing工具最好不要购买，要么自己开发一个简易版或租用商用工具，才是最佳的使用方式，也是降低研发成本，降低产品成本，提高商业

竞争力的做法。

模拟器

通常在压力测试活动中需要使用一些模拟器来替代大量的真实用户数或真实的测试资源，应用于压力测试的模拟器的最大价值在于降低真实测试资源构建的成本，同时也能简化测试难度，有时还能提高自动化测试率。

但是模拟器的应用场合不仅仅只有压力测试，某些模拟器的应用还能加快产品研发进度，降低bug修改成本。其应用方式是在项目中，使用模拟器部件代替项目的某些模块，使得产品研发并行开展，测试尽早启动，能取得bug提早发现，减少bug定位难度和修复成本的效果。例如：我们要开发一款设备，通常的做法在硬件板没有做好之前，是无法开展黑盒软件测试活动的。但是如果我们购买硬件模拟器——实现虚拟的硬件板，我们所开发的上层业务软件能在虚拟硬件板上运行，那么我们则可以提前进行业务软件的调测，当硬件板被开发完成后，只需要做集成后的验证。这样相比传统的开发模式，不但加快了研发速度，而且也降低了bug修改成本，同时在进度和成本领域支撑了商业竞争力，这就是使用科技技术化解了进度与质量，进度与成本的矛盾的价值。所以，进度和质量，成本之间不是永远无法解开的矛盾，只要我们善于创新，应用科技技术就能让三者都能获得满足。

当然模拟器的开发也是需要成本和时间的，根据观察大部分成功应用模拟器的公司都是采用商用模拟器的方式，来获得更好的投入产出比。

自动化测试

自动化测试工具主要有两大类，一类是自动化测试平台类，它可以集成各类自动化测试工具，和进行自动化测试任务调度和管理。另一类则是一些自动化测试工具，例如AutoIt这类工具，它能很好的针对客户端软件进行录制回放的自动化测试。对于一个测试团队来说，自动化测试最好能有一个自动化测试平台，方便进行自动化测试任务的可视化管理和调度，执行。自动化测试平台能大大帮助提升自动化测试管理的效率，降低自动化测试用例的重复执行概率，有利于根据测试环境和测试任务的突发变化，及时调整自动化测试策略，其价值也是降低研发时间，有利于Time to market。

对于自动化测试工具的选取，我的建议是如果你的测试团队在20人以下，公司研发总共100人不到时，最好使用商用的自动化测试工具或开源工具，没有必要去自主开发，自动化测试工具的开发和维护也是要成本的，小公司或刚启动的项目的时间紧急度是非常高，如果这时还自行研发自动化测试工具，反而会分散现有的研发资源，既影响研发进度，还影响测试质量。除非你的测试用例质量都不断改进到了很高的质量，确实很难再提升时，同时又有富余的测试人力时，

这时再投入自动化测试工具的开发或维护中也不晚。没有自研的自动化测试工具，我们的测试项目就一定不能前进了吗？在大多数的情况下——不是。现有的商用和开源自动化测试工具已非常丰富了，无论是基于命令行的，基于web页面的，基于windows客户端软件都有足够多的选择，你需要做的是先充分利用好现有的工具，我相信至少能满足你很大部分的应用。对于中国大部分的公司而言自研的自动化测试工具并不是你的核心竞争力，微软使用自研的自动化测试工具，是因为它的领先地位和丰富的研发资源，开发测试比1:2,1:3可以让他的测试人员来开发自研的工具。而中国基本是反过来的，更多地方的开发测试比为3:1，5:1,甚至10:1都有，这种情况下，你再学微软搞自研自动化测试工具，你牺牲的必是你在测试设计上的质量，因为你减少了测试设计的人力投入了。

所以，最后我提出一个独特的观点，对大多数中国的公司用好外部自动化测试工具来解决自己的问题，不求100％解决所有自动化测试的问题，能基本满足自动化测试工作需要足也。持续在测试设计上投入人力和时间和利用自动化解放的人力一起把测试质量做得更好，才是保障商业竞争力的最大价值，而不是让自研自动化测试工具消耗掉我们宝贵的测试人力资源。

五、专项测试技术

提出专项测试技术的目的是为了与常规的功能性测试进行区分，突出我们在特定测试阶段测试工作的重点。并且作为测试技术的分支之一，专项测试技术的掌握和积累，对于一个组织而言，其重要性不亚于自动化测试。因此，我有必要使用简单的篇幅描述专项测试技术对商业成功的重要意义，希望各测试组织像重视自动化测试一样重视专项测试技术的投入。本文仅提及：可靠性测试；可测试性设计；兼容性测试；安全性测试；性能压力测试。对于其他专项测试例如：UCD测试，白盒测试，可服务性测试等则暂不进行阐述。

可靠性测试

提起专项测试，大多数人都知道性能测试，压力测试，安全性测试，可是在大多数测试组织中并未有专人专项进行可靠性测试的技术研究和积累。虽然在日常的测试活动中通过压力测试和异常测试能进行一定的可靠性测试保障，但是有专人专项研究的可靠性测试肯定会更系统，可靠性测试覆盖更全面，可靠性测试的难题解决效率也会更高。那么可靠性测试专项活动应该如何开展呢？首先，要建立测试项目相关的可靠性故障模式库；然后，根据故障模式库中的可靠性错误类型进行分类，根据分类准备相应的可靠性专项测试用例和可靠性测试工具。最后，在测试计划和测试策略中，把专项的可靠性测试用例应用起来。有条件的测试团队，还可以根据可靠性测试用例的测试通过情况进行可靠性模型的评估打分。这样产品在可靠性领域的工作才能做到心中有数，而不是完全处于无法跟踪

和评判的状态。

可测试性设计

作为专项测试技术中，唯一1个对产品或项目进行正向构建的测试技术——可测试性设计技术，其重要性对于研发效率的提升，测试成本的降低，测试质量提升都非常直观。那么如何进行可测试性设计呢？大致思路可以这样参考，在项目需求分析阶段测试人员就参与分析，这时在可测试性领域测试人员可以做2件事，1、判断需求是否具有可测试性；2、在脑海中开始思考，如何才能用更低的成本来实现某个场景的测试。特别是第2件事，这时测试人员就开始进行可测试性设计的思考，通常情况下，可以考虑基于异常构造注入，内部压力构造，重要过程信息观察三个大的出发点来进行可测试性设计，输出可测试性设计的需求。需求分析结束后，测试人员和开发人员一起进行架构设计和系统设计，这时测试人员就可以把可测试性设计的需求在架构中进行落地，构建一个独立的可测试性模块来对所有可测试性设计进行统一的管理和调度。最后，在后续的测试活动中，包括测试设计阶段和测试执行阶段，想到任何有利于降低测试成本和难度的可测试性需求，都可以基于先前设计的可测试性架构进行添加补充。 因此可见，可测试性设计对测试人员的要求一点不比自动化测试的要求低，它是一种产品设计活动，而不是质量保障活动，但它却只有由测试人员来主导才能真正发挥好价值，它也是测试人员参与产品架构活动时的主要职责和架构工作分工。

如果一个产品的可测试性设计做得好，不但测试构造测试环境的成本和难度大大下降，而且产品一些瞬息万变的故障隐患也能在可测试性设计的可观下被发现，从而提升产品质量，同时开发和测试定位问题的时间也会大大下降。甚至还可以把可测试性模块打包成产品的故障自诊断模块，仅可以提高产品在运行过程中的可靠性，而且也能提高产品的可维护性，例如：Windows的任务管理器就是一个优秀的可测试性设计模块。所以，可测试性设计对于商业成功的价值，是全方位的提升竞争力，不但降低产品成本（研发成本，减少研发周期），提高产品质量，而且还能降低产品的维护成本。

兼容性测试

兼容性测试有必要进行技术积累吗？不就是让我的软件在各种其他软件的平台上运行看看是否会影响基本功能吗？这样的观点应该只是兼容性测试认识的最初阶段，兼容性测试不光是功能互通的验证，事实上某些性能需求也需要进行兼容性测试，甚至通过压力测试还能发现兼容性的缺陷。我记得有个案例是某个产品A的系统中集成了另一家公司的模块B，在平时的互通测试验证中所有功能都正常。可是在进入压力测试阶段后，虽然开始也没有出现任何兼容性的问题，但是随着压力测试的次数增加，在压力测试过程中开始出现让我们无法解释的奇

异故障，最后才定位到模块B在整系统有压力情况下，会与我们系统中的自研模块产生兼容性的问题。从这个案例中获得的经验教训是兼容性测试验证不仅仅是互通了就认为没有问题，还需要在压力测试情况下进行验证。

兼容性测试中最大的挑战除了发现兼容性bug外，就是如何降低兼容性测试的成本。第一个问题是如果每次都人工进行互联互通兼容性验证，那么工作量和成本非常大；第二个问题是如果需要兼容性验证的第三方产品全购买的话，则不是一般的企业所能承受的成本。对于第一个问题，可以采用兼容性自动化测试的方式来解决，而兼容性自动化测试技术则比全自研产品的自动化测试技术要复杂的多，因为它需要应对更多种的第三方产品的特性环境，所以每个公司依据自己产品的应用场景需要积累和构建特有的兼容性自动化测试技术。对于第二个问题，则是完全需要创造性的思考解决方案来解决了，这类解决方案也是需要不断积累，不断改进，才能持续降低兼容性的物料成本，满足兼容性覆盖面的测试要求，难道解决兼容性物料的工作不属于有挑战的技术性工作吗？

所以为了减少公司产品的研发成本，加快研发进度，减少流向市场的兼容性bug，从而提高商业竞争力，我们需要抛弃那种在功能测试工作中顺带验证兼容性的不严谨的测试方式，而应采用专人专项投入的方式，来持续研究如何提高兼容性测试的质量，降低兼容性测试的成本。

安全性测试

安全性测试这些年很火，特别是搞互联网的项目的测试人员基本都知道，甚至成为了互联网产品的测试必经环节。回想起5，6年前，还很少听说有测试人员进行安全性测试，那时测试人员的关注点都是功能测试，性能压力测试，自动化测试。因此，过去认为只有黑客才具备的安全性知识，现在也成为了测试人员需要具备的知识。毕竟知识是需要系统积累才能发挥更好的效果，特别是安全性的知识面很广，所以，安全性测试安排专人专项准备和研究是必须的，否则安全性测试的质量会受到很大影响。不过，值得庆幸的是现在互联网上各种集成的安全性测试工具还是比较多，对于非专业安全性测试公司的测试人员，先学会如何直接使用现有的集成化安全性测试工具，比自己去学会某个安全攻击的原理，然后自己编写测试脚本或编写测试代码的方式，要更高效，产出更大。有所得就得有所失，为了组织在最短时间内完成安全性评估，只有先把安全性测试工具当成传统测试工具来应用，待测试任务完成后，再利用空闲时间来逐个了解学习每个安全性测试用例的原理。

对于有条件的组织，可以在安全性测试工具进行应用的基础上，从多个安全性测试工具的测试范围中抽取出安全性测试的共性进行分类，然后为每类进行专项的安全性测试用例开发，以后安全性专项测试的管理和工作规划就能基于自己

的专项测试用例来开展，无论安全性测试工具如何变化，如何推陈出新，我们都是基于自己的测试用例来更新和维护，使得自己产品的安全性评估是基于系统化的测试用例，而不仅仅是1－2款工具。

性能压力测试

这个专项测试技术的资料是非常多的，大家可以自己google学习。我只从商业角度分享一些自己的特有观点稍做补充：性能测试与压力测试的测试报告需要分开来编写，正如性能测试和压力测试不能混为一谈一样，性能测试与压力测试选择的测试数据范围是不一样的，测试报告的关注点也是不同的。性能测试报告的商业价值更在于“务虚”，大多数时候都是最佳最优值，或有利于市场宣传，或为了内部汇报，但我们都知道最佳值不可能是平常值。而压力测试报告的商业价值更在于“务实”，是属于关上门在自己家里开展自我批判的材料，压力测试的结果是实实在在的发现产品中存在的可靠性问题，是证明我们的系统是否是可用的系统，压力测试报告在早期“越丑越难看”越是好事，才是对我们自己说实话，让我们知道还有哪些工作没做好。所以，为了支撑好商业成功，我们在做好性能测试获得漂亮的性能数据之后，还需要在压力测试领域持续投入，持续提升压力测试的质量，这才是我们真正能站稳市场长期获得商业成功的根基。

【淘测试专栏】探索性测试进度控制的方法

作者：高翔

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

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

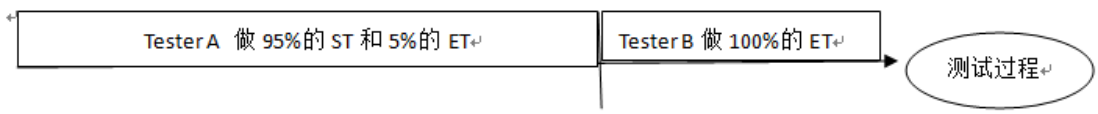
1.1 实践中ST和ET的使用模型

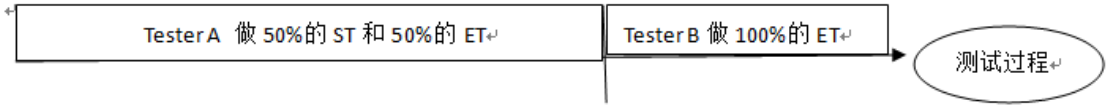
这里需要说明的是怎么应用ET在项目过程中，可以从不同的维度去考虑：

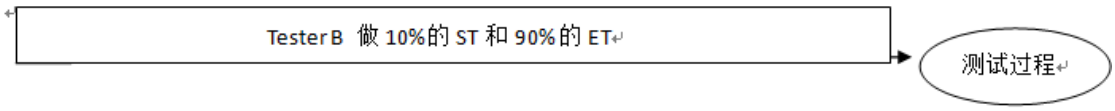
一个是ET和ST的结合方式，和测试人员具体做ET还是ST或都做无关

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

看下面的模型更易理解：







（备注：ST就是Scripted based Testing, 就是我们目前所做的基于测试用例进行测试的测试方法；Tester A是专门做ST Team的ST测试人员，Tester B是专门做ET Team的ET测试人员。）

从模型的演变过程可以看到，其实我们最终目标是没有ST tester和ET tester之分的，所有人都是标准的ET tester。国外ET的大师们确实带了好几个ET team，在项目测试过程中已经达到了模型四的境界，可想ET是可以主导整个项目测试的，其进度控制和质量管理都在实践中有了自己方法。

1.2 ET team的管理方式

在ET team里面又存在2个不同的管理方式：Delegation和Participation，这个区分的角度是从ET team lead在整个项目ET过程中的作用来看的。

Delegation：

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

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

(3) 对于一些问题和测试报告召开定期会议。

Participation：

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

(2) Lead可以实时的根据测试质量情况制定最后的测试策略；

(3) Lead可以持续的了解他所想要的了解的team的任何情况。

这里还可以让大家知道的是相比较单个人进行ET测试，在一个ET team中大家工作在一起，在同一时间对于同一个SUT（Software Under Test）进行ET，经常会出现很多更好的测试idea。 所有还有一种组合ET team的方式是让测试人员组成一对且让他们在同一台计算机上进行测试。另一种就是其中一个测试人员进行测试，傍边有多个测试人员观察且做记录，通过问测试执行人员不同的问题产出更多的测试idea，这里有一个好处就是测试执行人员不必担心发现的bug难以重现，因为傍边的测试人员会做记录和分析，这样测试执行人员可以不必分心去继续自己的测试。还有一个好处就是如果一旦这个测试人员思维过于开阔，去测试很多非当前需要测试的模块时，傍边的测试人员可以给予及时提醒。但其缺点就是更多资源消耗在同一个功能模块上，成本上有待商榷。

1.3 ET过程中的任务

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

在测试执行之前，ET lead和Senior ET tester参与制定所有的charters，并

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

这里需要说下制定这些session的一些基本的原则：

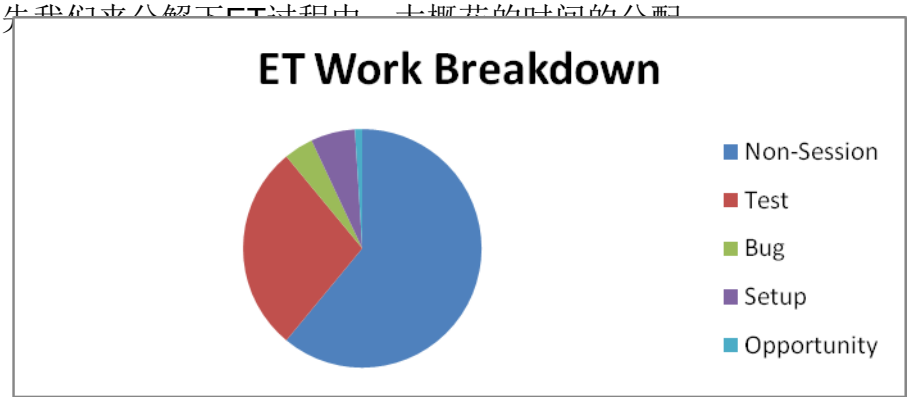
(1) 这些Session必须是chartered（与测试任务绑定的）

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

(3) 这些Session必须是reviewable（其Session sheet可以被第三方review）

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

首先我们来分解下ET过程中一大概花的时间的分配



（Opportunity 指的是Opportunity testing，就是执行其他的session的功能）

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

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

(2) Tester name（s）

(3) Data and time started

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

(5) Data files

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

(7) Issues（ET 过程中的问题和疑惑）

(8) Bugs

1.4 ET中管理Session

下面我们继续ET在项目中的管理（这里说的是上述的模型三或四），大致如下在：

(1) Lead针对SUT做出Charters和Sessions；

(2) Lead针对所有sessions和资源来做出Test plan和测试策略（包含对SUT的攻击策略）；

(3) 设计阶段ET tester对于自己负责的sessions进行test idea development；

(4) 设计阶段ET tester还需要对自己负责的session所对应的oracles（可以准确任务这个问题是个bug的所有信息,包括文档） 进行梳理；

(5) 测试执行的时候ET tester根据之前的test idea list来进行测试，根据SUT本身的response并采用Heuristics的方法产出更多的test idea（ 这部分的test idea 不是必要记录下来）；

(6) 测试执行的时候每个测试人员填写自己的session sheet；

(7) Lead每天汇总每个测试人员的session sheet；

(8) 利用自己开发的工具支持分析所有的sessions的执行情况；

(9) 根据分析结果得到测试进度的具体执行情况和质量情况。

这里的工具会提供足够多的数据来支撑我们的测试进度是否在合理的范围内，是否存在哪些风险，哪些测试人员在Non-session中花了时间过多等等。由于产出的metrix提供足够的信息去控制整个ET进度，我们当然还需要了解未来的计划安排，所以metrix里面会有个item叫To DO Sessions来表明我们未完成的session数量。 还可以帮我们更加的评估我们ET中的生存率，同时也可以评估我们ET tester的快速学习能力。

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

【本专栏文章由淘宝（中国）软件有限公司测试部独家提供。】

构建企业级自动化测试框架的一些思路

作者：邓强

关键词：自动化测试，测试框架

引子：目前国内软件企业对软件测试越来越重视，都有专门的测试部门，而且很多公司还有专门做自动化测试的。但是由于我国的测试行业起步比较晚，积累还不多，很多公司都在探讨适合自己的测试方式，包括手工测试和自动化测试，当然也包括性能测试。笔者长期从事自动化和性能测试的设计与培训工作，对如何构建一个企业级的自动化测试框架有一些心得体会，愿与所有测试人一起分享，如有纰漏欢迎指正。

声明：本文将不探讨自动化测试的可行性，也不探讨具体的自动化测试如何展开，而是站在一个更高的层面，探索如何构建一个企业级自动化测试框架，并使之提升企业的生产力。本文面向已经具有自动化测试经验，且认可自动化测试并愿意在自动化测试领域进行深入研究的企业和个人，另外，本文只探讨基于应用级的自动化测试，不包括基于代码级的单元测试和基于协议级的性能测试。

一、自动化测试概述

1、自动化测试原理

但凡提到“测试”，它一定与另外四个字密不可分，那就是“期望结果”，毫无疑问，我们要真正理解测试是什么，最重要的就是理解这一个基本原理。手工测试是这样，自动化测试也是这样，自动化测试工具是不知道任何产品的商业逻辑的，工具也不能像人一样具有智能。工具其实非常单纯，它只做一件事情，那就是把实际运行的结果和我们给它的期望结果进行比较，结果一致则测试通过，否则测试失败。

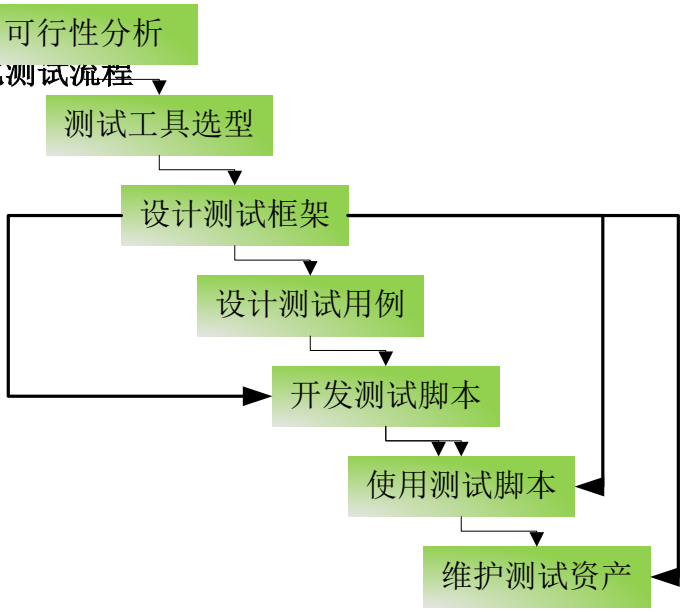
自动化测试工具要实现自动化从而代替人的手工操作，只需要实现如下三个功能即可：

对象识别：手工测试中点击鼠标和键盘是必须的操作，使用工具来做测试则需要首先找到它应该点哪个对象，应该在哪个文本框中输入值。

检查点：检查点就是期望结果，我们将期望结果写入检查点，运行过程中工具将实际结果与检查点进行对比来决定测试是否成功，代替人工判断。

参数化：为了实现代码的重用和不同的输入，我们使用参数化来完成并可提升测试效率。

2、自动化测试流程



从流程中我们可以看出，一个好的测试框架，将直接决定着测试脚本的开发，使用与维护，也就意味着：时间。

为何要做自动化？我想不会是为了找到更多缺陷(道理很简单：如果自动化测试能找到缺陷，那么在设计自动化测试脚本并调试时缺陷已经暴露)，而是为了将人从重复劳动中释放出来，一些重复的事情交给工具完成，这样测试人员就有更多的精力做更重要的事情(当然也包括寻找软件中的缺陷)。这是自动化测试的初衷，如果我们的自动化体系还需要过多的人为干预的话，则有违自动化测试的初衷，所以设计好一个自动化测试框架显得尤为重要，最好能全自动，免维护，不是吗？

3、自动化测试工具

自动化测试工具相对来说已经是一门比较成熟的技术，由于其实现原理相对简单，所以自动化测试的工具框架越来越多，给企业选择带来了不少的麻烦。本文并不探讨如何进行工具造型，但是考虑到后面章节会提到一些工具，所以将目前市面上一些较有名的工具或框架按功能列出，仅供参考：

测试Windows窗口程序：HP公司的QTP，IBM公司的RFT，免费的AutoIt，MS的UIAutomation。

测试Web应用程序：QTP，RFT，UIAutomation，开源的WatiR（及Java和.NET的实现版本WatiJ和WatiN)，Selenium等。

测试Java应用程序：QTP (插件方式)，RFT(原生支持)，Abbot(开源)。

测试.Net应用程序：QTP (插件方式)，RFT(原生支持)，UIAutomation。

某些工具同时提供对Flex，SilverLight，SAP等的支持，另外也包括自主开发的测试工具。

二、如何构建自动化测试框架

1、何为框架？

1) 框架不是一个可以直接使用的产品，最多算是一个半成品。

2) 框架定义了统一的规范，通用的接口，封装一切可以封装的功能，简化代码的开发量。

3) 框架的初衷是使一切变得简单，易用，否则，框架则失去了意义。

2、何为自动化测试框架？

1) 所谓自动化测试框架，当然就是基于自动化测试来构建的一个半成品，主要用于自动化测试。

2) 自动化测试框架应该使用一种简单的方法实现自动化测试的三大功能：对象识别，检查点和参数化(或数据驱动)。

3) 自动化测试框架应该更多的考虑易用性而非性能，易用优先，最好全自动，无人干预。

4) 测试框架应该让测试人员像写测试用例一样来完成测试代码的生成或维护，而不是让测试人员掌握艰深的编程知识(虽然笔者较强调测试人员对程序设计的理解)。

5) 测试框架还应该满足企业级的应用而非单个项目或产品，一个企业应该使用一套统一的测试框架。

3、应该满足哪些功能？

1) 良好的、层次分明的用例管理及目录规划。

2) 将对象识别变得简单，最好只提供一个对象的名称或ID号即可识别。

3) 最好不使用对象库存储对象信息，而是动态识别，减小对对象库维护的工作量。

4) 添加或删除任何用例均不会影响到已有的用例。

5) 能够自动安装部署，并自动运行自动化测试脚本。

6) 能够将测试结果以简单的界面呈现或者保存到数据库中以便于集中管理。

7) 如果程序运行出错最好能将当前屏幕截图以便查找问题，或将错误信息输出到测试结果中。

8) 根据企业中的不同产品来调用相应的测试脚本，在相应的平台进行测试。

4、理想中的自动化测试应该怎样进行？

FiveOne公司构建好了一个自动化测试框架，并开发好测试脚本，该自动化系统运行良好。 产品A在晚上11点构建了一个新的Build，产品B在凌晨1点构建了新的Build，当第二天早上项目组成员上班的时候，他们收到了一封邮件，列出了如下的数据：

被测产品	测试版本	测试平台	用例执行数	用例失败数	错误信息
A	1.1.1234	X+Y+Z	100	5	…….
A	1.1.1234	X+Y+A	100	4	…….
B	2.0.4321	B+C+D	200	10	…….
B	2.0.4321	F+C+E	200	6	…….

在新的版本上发现有执行失败的用例，相关模块的负责人立即运行自动部署的新版本，检查具体的错误原因，并与开发负责人讨论最后决定是否需要将此缺陷交由缺陷管理系统管理（当然，也可以让系统自动地在缺陷管理库中登记一个BUG，但是我们不推荐这样，BUG是否需要登记需要测试人员或开发人员的最终确认，而不应把这个严肃的工作交给工具完成）。

最后，经过确认后，发现有两个错误是由于新版本的界面变动导致测试工具不能正确处理导致，还有两个错误是因为测试脚本的期望结果不对引起。这时测试人员将测试脚本签出，对该测试用例进行了更新，并添加了两个新的用例，添加用例时使用类似如下的语句完成（以下伪代码与编程语言无关）：

```
system(A).page(B).button(C).click
system(A).page(B).text(D).settext(E)
system(A).page(B).button(F).click
check(system(A).page(G).label(H), expresult(I))
```

最后，相关人员打开本次测试结果的归档系统，将相关错误的处理结果更新上去，所有测试结果和处理结果都将作为项目资产保存起来。

三、如何实现上述理想模型

1、架构框架的能力要求

1) 丰富的手工测试和自动化测试经验，真正理解测试人员需要什么，架构为谁而做。

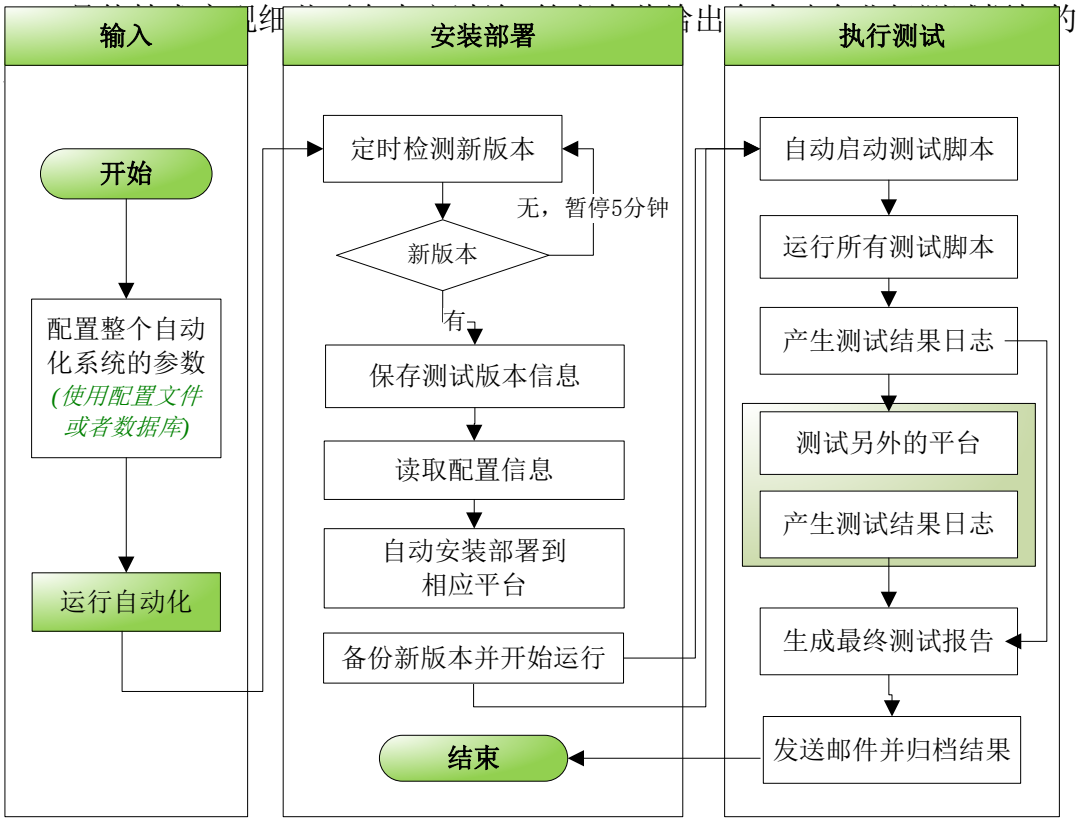
2) 软件设计和架构的能力，理解软件架构的精髓。

3) 能将复杂的程序设计简单化，提供给测试人员类似自然语言编程的简单方便。

4) 究竟由开发人员还是测试人员来做架构呢？我们需要的是一个理解软件

测试精髓的开发人员或者理解程序设计精髓的测试人员。

2、基本功能如何实现？



3、这样就完美了吗？

答案是肯定的：那就是还不完美，框架的改善没有尽头。比如我们不单想全自动，也想按需测试，比如指定本次测试哪个平台，在什么时候开始测试，测试哪些模块甚至指定某一部分测试用例来执行，这一切，均可通过一个应用系统来完成。

这一切，听起来很美，但是我们一定要注意到，我们为何要设计框架？因为我们想让自动化更加自动，我们想要测试脚本的开发和维护变得简单，“简单”很重要，如果我们把一个框架设计得异常复杂的话，它就背离了我们的初衷。

四、注意事项

1、并非测试人员写的代码越少越好

大家可能会觉得一个好的框架应该是让测试人员写很少的代码，最好不写代码，只写一些关键字，如测试人员只需要输入：“按钮，确定，单击，已成功提

交”或者“文本框，用户名，输入，Jack，已成功登录”这样的文本即可实现自动化测试，这个初衷没错，但是这样的话我们框架的设计就变得非常复杂，维护也很麻烦，而遇到复杂的操作流程的话，远没有办法用几个简单的关键字来实现。框架应该使工作简化，但是得有个度，如果太过简化了，则框架本身会变得很复杂甚至臃肿。

2、框架的目的是简化任务而非提升自动化程度

框架是为了提供统一的规范，并且将常用的功能进行封装，最好能利用面向对象编程的特性进行封装，使我们的测试脚本的开发和维护变得更简单，并且将关注点放在如何操作业务流程是而不是对象识别及功能实现上，这样起到快速开发自动化测试代码的目的。如果一味地使用框架来提升自动化测试程度使框架本身变得很重，那么有点舍本逐末，得不偿失。

3、企业应该构建适合自己的框架

我们能够找到一些现成的框架，每一个框架都宣称自己的实现思路最好的，这点不否认，但是人家觉得好的东西未必适合你的企业，企业应该分析自己的产品特点，借鉴别的框架的精髓并尽量简单地开始设计自己的框架，刚开始不可太复杂，在使用中慢慢完善。

4、花钱买解决方案而非买工具

企业不应该将钱花在购买大型自动化测试工具上，工具是没有办法帮助解决企业自动化测试的难题的，即使这个工具号称自己是业界最好的，它仍然只是个工具，没有感情，没有智能。企业真正需要的，是解决方案。我们不一定需要 QTP，不一定需要 RFT，说不定一套基于开源框架的自动化架构比如 Java+WatiJ，比如 AutoIt+WaitR，比如 C#+WatiN+UIAutomation 等已经满足我们的需要。

所以，如果需要实现自动化，企业最需要的，不是一个工具，而是解决方案。

由于篇幅所限，本文无法将上述模型的具体实现展现给大家，如有兴趣，可关注 51Testing 测试论坛 RFT 版块，笔者正在连载 RFT 教程，并计划展示如何使用 RFT 构建自动化测试框架的细节。

软件自动化测试理论及实施经验

作者：柳胜

软件自动化测试根源于目前软件行业一个越来越突出的问题，即软件质量与成本的矛盾。很多软件公司已经看到了这个潮流，很早就开始做软件自动化测试的预研和实施，比如微软，oracle 等已经在企业内部测试团队整合了自动化测试流程，实施了自动化测试框架，并且已经 按时更新换代，进入稳定应用的周期。

但在国内，由于软件自动化测试时间不长，测试人员技能水平等因素影响，软件自动化测试的研究和实施大多还处于一个起步 摸索的阶段，我们看到普遍的情形是“做的人不少，成功的不多”，这种现状一方面有技术的问题，另外也有方向上的问题。因为在业界可供借鉴的成功实施经验或 案例比较少，所以在起步阶段可能就会走弯路，这是摸索必然要付出的“学费” 。

那么怎样能够把握软件自动化测试方向和思路呢？本文将理论结合实际，来谈谈如何实施自动化测试。

一、实施测试自动化前的几个问题

1、问题1：是否实施测试自动化的决策依据是什么？

一直做手工测试的团队，为什么要选择做自动化测试？我们常看到这样的理由：

A. 节省手工测试的人力和时间成本

B. 有助于提升测试团队的技术力量

C. 能够生成直观的图形化报表

D. 我不知道，领导要求做的

如果是你来负责决策是否自动化测试，而且 ABCD 都是考虑因素，贪心将注定你最后什么也不会得到。正确的答案是只有一个 A，只有 A 才是我们自动化测试的目标。

这是一个非常简单而朴素的道理：自动化测试的横空出世，是为了帮助手工测试解决繁重的工作压力，我们在自动化测试实施中所做的任何工作的目的都是最终能够节省手工测试的成本，如果离开了这个指导思想，自动化测试就成了浓妆艳抹，面目全非的女人。

而且，这个指导思想应该一直贯穿在自动化测试整个生命周期，在实施前期作为决策的依据，在中期作为实施的原则，在后期则是评估是否成功的度量标准。

要怎么判断是否“节省了软件测试成本”，甚至度量“节省了多少测试成本”呢？下面，我们来看一下自动化测试的效益分析。

2、问题2：如何估算分析自动化测试效益？

在市场经济的今天，做任何事情都讲究经济效益，效益就是投入产出比，投入越少，产出越多，效益就越好。怎样度量测试自动化的效益呢？当然，要做的第一步就是要找出测试自动化的投入成本和收益。

1) 测试自动化实施成本

很多没有实施经验的人经常进入一个认识上的误区，即认为自动化测试主要花费在前期开发上，而一旦测试程序运行起来，就可以高枕无忧，一劳永逸了。这是一个错误的观点。

究其根本，自动化测试的开发在本质上是一种软件开发行为，但它同时又具有测试的特质。我们每一轮测试的软件版本和其代码都是变化的，本质上来说，每次回归测试都是不同的软件，因此自动化测试的脚本也要考虑并适应这种变化，这就是自动化测试独特的维护成本。

自动化测试实施成本的计算公式：

自动化实施成本=前期的开发成本+后期的维护成本

前期的开发成本又可进一步细分为人力成本，时间成本，工具购买成本，培训成本等等。

后期的维护成本也是多种多样，如软件变化引起测试程序的变更成本，测试程序的扩展成本，测试报告的诊断成本等等。

需要注意的是，前期成本显而易见，而后期维护成本其实往往是自动化测试实施成败的命门。笔者见过最多的案例就是后期的维护成本过高导致自动化测试项目最终宣告失败。

2) 自动化测试的收益分析

自动化测试的收益是比较复杂的，比如在情感因素上，实施自动化测试可以提升测试团队的技术能力，地位和影响力等等。但其最切实的收益是节省了手工测试工作量。

很显然，自动化测试程序开发完成后，它运行的次数越多，它替代手工测试的次数也越多，也就节约了手工测试的成本越多。所以在这里，我们用自动化测试的运行次数来度量自动化测试的收益。

自动化测试的收益=自动化测试运行的次数

在这个收益度量理论基础上，我们如果想提高自动化测试的收益，无非有如下几个途径：

① 自动化测试脚本必须保持稳定的长时间运行，运行的时间越长，运行次

数才可能增加。

② 自动化测试脚本应该是被频繁执行的那部分测试案例。这样的案例转化成脚本，自然也会被频繁地执行，会增加运行的次数。

自动化测试收益成本比计算公式如下：

$$p=\frac{k*n}{c1+c2}$$

图 收益成本比计算公式

k=手工执行自动化测试案例的所花费的时间成本

n=自动化测试案例执行的次数

c1=花费在 automation 前期的（时间成本+人力成本+金钱成本）

c2=花费在 automation 后期的（时间成本+人力成本+金钱成本）

3) 自动化测试的收益分析计算案例

假设，我们自动化一个测试案例，所需的代码行数为 150 行（包括测试脚本代码和框架代码），而一个成熟的自动化测试团队的脚本开发效率是每人每天完成 30 行（包括相关代码和文档），显然，一个案例的前期开发成本工为 5 人天。

而一个策略得当，代码健壮的脚本，其理想的维护成本不应该超过开发成本的 20%：

后期维护成本=前期维护成本×20%=5 人天×20%=1 人天

自动化测试实施理想总成本=前期开发成本+后期维护成本=5 人天+1 人天=6 人天

而一份自动化测试脚本开发完成后，假设在其生命周期内，其回归运行次数为 500 次，如果此测试案例手工执行所花费的工作量是 10 分钟人。那么，自动化测试的预期收益为：

理想收益=脚本运行次数×手工执行成本=500×10 分钟人=5000 分钟人=14 人天

显然，对本系统实施测试自动化后，我们预期的收益成本比为

收益成本比=总收益/总成本=14/6=2.3

2.3 大于 1，说明对本测试案例实施自动化是划算的，可以减轻手工工作量。

二、分阶段规划实施自动化测试框架

基于笔者近几年对业界自动化测试实施状况的观察和了解，个人倾向于把自动化测试的成熟程度可以分为如下三个层次阶段，每个阶段是上一个阶段的进化，又是下一个阶段的基础。

1、测试的自动化——以工具为中心

在自动化测试诞生之初，测试人员最迫切的愿望就是通过自动化测试的实施来摆脱重复的工作量，即回归测试中的案例执行工作。怎样完成这个从测试案例到测试程序的转化呢？显然，使用已有的自动化测试工具解决方案是最有效的。

因此，这个阶段最明显的特征是所有的自动化测试实施工作是以工具为中心，在这个阶段的过程中，要完成的工作如下：

1) 选择工具

软件测试团队学习和掌握自动化工具的基本知识，并能根据当前被测产品的软件特征和项目特点，选择合适的自动化测试工具族。

2) 使用工具

测试工程师基于工具来开发生成自动化测试脚本，通过运行脚本，来完成测试案例的执行工作。

在这样一个从手工测试到自动化的显著转变之后，测试工程师们击掌相庆，原本以为可以从此高枕无忧，一劳永逸了，但等到自动化测试运行起来之后，他们痛苦地发现，事实远非想像那么简单，减少手工执行案例的工作量是以增加自动化开发和维护工作量为代价的，这是以前未有预料到的。自动化运行过程中出现的各种各样的问题考验他们的耐性和智慧，甚至甚于重复手工测试的苦恼。比如，常见的有如下问题：

a. 如何提高的测试程序或脚本的复用性以获得自动化测试的高收益？

b. 如何解决测试自动化的健壮性问题？

c. 自动化测试脚本的类型和数量越来越多，怎样有效组织管理和调度这些脚本？

d. 自动化测试如何与手工测试整合？

等等。

这些问题是助产士，它们促成软件自动化测试第二个阶段的到来。

2、自动化的测试——以框架为中心

如果说第一阶段是在一个点上下功夫，那么第二阶段就是在一条线上作战了。“自动化的测试”的内涵更加丰富，它意在将软件测试里的所涉及的各个环节作为一个 统一的整体考虑，从测试案例的管理，测试案例的执行到测试报告的展现都有相应的策略及自动化实现，故称“自动化的测试”。在这里，自动化测试框架横空出世，自动化测试框架是一系列策略思想，规范和代码的集合。

目前业界流行的框架理念有数据驱动框架，关键字驱动框架，经常被提到的还有模块化框架，测试库框架等等。但这些框架在笔者看来更像是框架的设计原则思想，关注点在实现层面，称不上真正的“测试框架”。真正的“测试框架”应该是有血有肉，是测试团队的战略层次上的规划和完整的解决方案。有关框架

详细介绍，会在本文的续二中集中介绍。

3、无需人工干预的自动化回归测试——以流程为中心

这个阶段可以说是软件自动化测试炉火纯青的时候了，达到了”天人和一”，经过多年修炼，在这里软件自动化测试和软件开发再次做一个整合，从自动化流程上， 能够达到真正的测试驱动开发，比如 coding 与 unit testing 做整合。目前已经达到这个阶段的有微软，IBM 等。

三、自动化测试实施经验谈

目前国内软件公司软件自动化测试的实施情况大多处于第一阶段或从第一向第二过渡的阶段。在对于软件测试框架的理解和实现上，出现了不同的思路，甚至偏差。所以我在下面着重谈谈软件自动化测试框架的设计原则和一些风险的规避。

1、框架要集中精力解决自动化测试中的突出问题

看起来这是个很简单明了的原则，不是么？自动化测试框架本质是一个软件产品，它旨在满足自动化测试中的必然需求，而不是大包大揽软件工程中所有的问题。我在某个测试网站曾看过一个测试框架的设计原型，真叫人叹为观止，从脚本代码的版本同步管理到 bug 的提交和存储，都通通塞入框架解决方案之中，包罗万象， 无所不能。我虽猜不中这框架的开始，但我能猜到这框架的结局，无非两种情形，要么技术资源不够而流产，要么实施起来太过复杂，自动化成本大大高于产出，总之结局一个字：死。

一个经验丰富的自动化测试架构师应该既能够敏锐捕捉到当前自动化测试中的关键问题，又能巧妙利用现有企业资源为我所用，在上面的例子中，可以把脚本版本管 理工作交给企业中已有的源码管理系统，比如 clearcase，Vss，把 bug 的存储交给 bug 管理系统，而框架集中精力解决自动化测试中的无人值守运行，报告生成等问题，总之在框架设计的初期阶段，一个明确切实的实施目标是十分重要的。

2、框架设计中的扩展考虑

毫无疑问，自动化测试框架是一个软件，在设计时同样遵循软件设计原则：模块的高内聚和低耦合，这些都是基本原则，不再赘述。需要注意的一点是由于框架作为 企业环境的一部分，必然涉及与其他系统的多种接口，比如和 case 数据库连接实现自动化测试案例自动获取，和 mail 服务器连接实现邮件实时通知，和 bug 数据库连接实现 bug 的自动提交，因此，在接口设计上要尽量保证扩展性和兼容性。

3、框架实现中的数据驱动思想

数据驱动在这里有两层意思，第一，框架实现上完全数据驱动，不能存在hard-code 的问题；第二，框架要提供规范和策略，使得脚本同样必须遵守数据驱动 的原则。

4、与手工测试的整合

我们的企业不是试验的学校，也不是花拳绣腿的作秀场，在企业里做事往往从务实和实际效益出发，以结果为导向，因此，“自动化测试”在我们看来，重点在“测试”，而“自动化”只是一种手段而已。我们更关心测试整体的效果，不是么？因此，自动化测试与手工测试如何整合，相得益彰，是一个重要的问题。

有关自动化测试框架的更多内容，请参看本文续《软件自动化测试框架实例研究》。

参考文献

《软件自动化测试框架设计与实践》 柳胜著 人民邮电出版社

真心爱测试，努力去工作

作者：Tengmy

摘要：软件测试是一个具有创造性的行业，需要真正了解和喜欢它的人来用思想和灵魂进行不断的创作。真心的喜欢测试，不断的提升自己，你才能在自己的软件测试道路上，越走越宽！

关键字：软件测试，软件测试工程师

序言：

不知不觉中，踏入软件测试这个行业已经将近六载了。悉数以前走过的路，有很多的感慨。我不知道在软件测试这个行业里面奋斗着人群中，有多少人是清楚了解并且乐意进入的，不清楚有多少人是因为受到了所谓的‘软件测试工程师’是一个门槛很低但是薪资很高的宣传才选择进入的，也不了解有多少人是为了“仓廪实”而踏入的。

时而感叹因为目的不同而进入软件测试大军的人们，经过实践的历练，岁月的打磨，有了那么多不一样的结果。

或许，世界本来就是不公平的，在某些人的眼中，看别人潇洒的谈笑，自己却要如此艰涩的生活着；或许，世界本来就是公平的，你善待了你的生命，虔诚的为了自己的理想去努力，即使不能硕果累累，取得的成绩也可圈可点。

在 2010 年的春天即将来到这座靠海的城市之际，也想用一支拙笔，写出些什么，以期让自己更明白未来的路，也让身边的在软件测试行业里面奋斗着的朋友，有所裨益。

——Tengmy　2010 年 3 月于大连

软件测试这个行业，发展到 2010 年的这个春天，已经颇具规模，记得 2004 年我在懵懂之中加入这个行业时候，在大连这片土地上，还真没有多少具有专职软件测试团队的软件企业的存在。而今，大连的软件业虽然还是倾向于外包，但是基本上有软件企业项目的地方就能找到专业的软件人才。也算是历史的一大进步了。

其实我一直觉得，为人做事的基本原则就是为自己去工作，所以我一向比较推崇那句据说是智联的广告语：既然一辈子，有半辈子在工作，那就找一个喜欢的吧。

我也一直认为，找工作和谈恋爱是一样的，需要靠缘分。天下间三百六十行，

你能投身到此，总有三分缘分。

我的这篇小文也旨在于给刚刚加入软件测试行业的朋友一个启发，给跟我一样在软件测试行业里面奋斗的朋友一个让自己能够继续努力下去的理由。

软件测试——创造艺术的天堂，我们相遇

软件测试，在神州大地上，如今从业者何止千万，但是不同的人对这个行业有不同的感叹。有人在抱怨，软件测试的枯燥，重复和无味，有人在唏嘘，软件测试的“没有技术含量”，有人在叹息，软件测试高手的路如此漫长，也有人在自己努力的道路上一路辛苦一路歌。

我一直认为，软件测试这个在中国因为种种原因发展滞后于软件开发的行业，是一个创造艺术的天堂。我们属于用自己的学识和灵魂创造艺术的人。

软件测试，其实并没有外人看得那么简单。软件测试的产生来源于开发出来的软件项目/产品质量的不过关。软件测试所进行的主要工作就是在软件开发过程中的排错。但是这种排错的工作却并不是初涉入这个行业或者说不了解这个行业所说得那样，仅仅是一个寻找 bug 的工作，任何人，都能找到出来，区别在于，熟手找到的速度和数量大于生手而已。

软件测试，根本的原则就是站在客户/终端用户的角度上来衡量和评价软件产品的质量。所以了解客户的需求是最基本的要求。如果不了解需求，不知道自己要测试的软件产品的基本功能，组成，之间的相互联系，你的工作不过是机械的操作而已。

软件测试，需要我们用自己的思想去理解客户的需求，并且把客户的需求幻化成一个一个可以测试的功能点。并且需要你用自己的思维去想，在有限的测试时间之内，如何设计测试用例，能够保证最佳有效的测试覆盖，检测出软件产品中的种种功能，性能的隐患来。

而这种质量/数量/时间考虑之下的测试的用例的设计，测试计划地制定，测试的执行过程，都是一个不能机械的规则化的东西，是需要你根据具体的项目，具体的需求，因地制宜进行创造，进行设计。然后像一个猎人一样，用你的嗅觉和判断力，找到你要的猎物。我时常说，软件测试对我来说其实是一个挖掘宝藏的过程。宝藏隐藏的越深，条件越复杂，级别越高，找到了，我会越开心。

软件测试，在测试地进行中，我们需要对自己的测试 case 按照需求的变更进行有效的调整，你需要对阶段性的测试成果进行评估，然后对测试工作的下一个阶段进行调整性的部署。

测试结束之后，我们更需要进行测试项目的评估，分析测试的成果，分析软件产品的风险，分析自己测试方针上的有效性和失误，然后再一个项目/阶段中

进行改进。应该说，无论项目是否有严格的测试评估要求，作为项目参与者的我们，都需要去做。因为没有思考和总结，岂是很难让自己有一个全面的考量，也就难以进一步开展工作。每一个项目，每一个阶段都是按照一样的方针去实施的话，发现了问题也没有让你对部署进行调整的话，在一定程度上肯定会影响测试的效果。

此外，根据不同的测试项目，我们还需要在测试之初和测试进行中不断的学习业务的知识。不懂得测试的相关业务，不了解客户的实际业务操作流程，不知道测试数据在系统中的流转，你的测试也是失败的。也会让你在测试的时候不知道测试的优先顺序，也许会制定错误的测试方案，把大量的时间耗费在没有多少价值的测试中去，不会去准确定位终端客户关注的那些东西。

还有，做好测试，我们还需要自己去分析各种发现的主客观缺陷，去分析和定位缺陷的产生以及原因，然后跟开发，客户做好沟通。所以，你需要知道网络安全的知识，需要了解数据库的操作，需要能够至少看懂一些项目里面的代码以便于你自己对缺陷进行初步的定位。这些需要你具备的不但是很好的沟通能力，需要具备良好的逻辑思维能力进行分析，还需要你具备跟开发人员对话的基础，了解项目的进度，了解业务/产品中哪些功能是级别比较高的，从而能有的放矢的进行你的工作安排。

对于发现的缺陷，你还需要进行后续的追踪，对于修改之后的版本，你还需要再重新测试的时候去跟开发团队沟通，了解他们的修改过程以便于自己在重新测试的时候判断是否要扩大测试范围以防止新的缺陷被引进。

当然，身为测试人员，你还需要根据不同的需求，去学会使用各种缺陷管理工具，测试用例，测试计划的管理工具。如果是自动化/性能测试的人员，你还需要具备某些脚本的编写能力。

以上种种，岂是一个简单的不需要任何动脑的人就能做到的？所以我说，之所以你觉得自己的工作没有挑战性，没有技术能力，谁都可以做，是因为，你没有真正的去喜欢并且努力过。

虽然就目前来说，软件测试的具体工作是比较庞杂的，但是为之付出了种种努力的人却很难量化的看到自己的劳动成果，很难像开发人员一样看着自己的辛苦变成成型的产品。这样的特点也让不少的测试从业人员觉得委屈，好像在为他人做嫁衣裳。如果从这样的心理出发，在心理不认同的情况下，无论怎样都是难以做好这份工作。

所以，如何让测试从业人员能够从心理认同自己的工作，如何让测试从业人员工作的更有成就感也是摆在我们面前的一个需要思考和解决的课题。

对于存在这样想法的同行，我只能说，想办法提升你自己，路是自己走出来

的。学会去参与更多，收获更多，然后你可能就没有这样的抱怨了。走出自我提升的第一步，或许你就会发现，世界已经开始变得不同。

但是，从另一个角度上讲，测试像网，包罗万千，测试的复杂度和深度是难以预测的。技术，业务，沟通缺一不可。想做测试，学习了实践了就不难，想做好，绝非易事。

软件测试——喜欢并且提高才能有自己的天空

有的朋友说，我们看不到客户，无法了解需求；有的说，跟开发人员不熟悉，没有办法问到自己需要的信息，有的说，我们的测试不规范，我找不到任何有价值的信息，有的说我们的测试团队有严格的作业规定，我没有机会去做你说的那些事情……

的确，不同的公司有不同的文化，不同的项目有不同的规则。但是无论面什么样的事情，我始终认为，只要你想，你一定可以做到。

我的第一份工作是在一个日企做黑盒手机软件测试。刚刚进入软件测试的我那个时候对软件测试也一无所知，甚至一开始我的职位都是：手机操作员。跟我一个奋斗在测试部门的同伴们，每天都在不断的自我抱怨中拿着那些别人写好的测试用例，一遍一遍测试那个小小的手机的若干功能。大家都似乎认同一个道理：这就算测试，看式样书，运行自己写或者别人写的 case,机械的去碰运气一样的去寻找或许存在的 bug。

我也曾迷茫过，我也曾跟他们一样机械的工作着并且在心理质疑过自己的选择。但是我最终还是觉得，即便是在这样的环境里面，我能学到应该不止这些。我能做的应该也不止这样，否则我的路在哪里？

于是我在工作之余，会习惯性的去阅读客户的各种式样书，去看开发文档。去了解手机相关的业务知识和通信的知识。。渐渐的我发现，我能做到的能想到的越来越多了。我发现软件测试这份看起来简单无比的工作，其实暗藏着很多的玄机。你有多少想法，就能创造出多少可能。虽然大家做着外表上一样的动作，但是带着想法去做，带着思维去做的，收获到的绝对是不一样的果实。也就是这样的感觉让我喜欢上了软件测试这份工作，并且不断地试图去学习更多，了解更多。

还有，比较感谢这个时代，我可以在网络的天地中寻找到自己需要的很多知识。经由网络去了解外面的世界。让我在当时那个基本上对软件测试一无所知的人，迅速的补充了很多的养分。虽然有一些知识是当时的那个环境所不能实践的，虽然有一些看起来基本上一辈子都能难碰触到的。

人其实是无法预料未来的，但能把握现在。如果你想未来能走得更远，更宽，

那么今天就一定要努力。这也是我当时想的，并且努力去做的。甚至在大家的嘲讽中去一步步走向属于我的明天。事实证明，我的想法并没有错。只有你喜欢它，并且为之努力，你才能真正有属于你的那片天空。

软件测试——自我修炼，给自己一个努力的理由

走到 2010 年，回顾将近六载的软件测试从业岁月。我没有后悔过。我很感谢那个偶然进入这个行业的机会，很感谢自己的努力，因为努力，所以我才能拥有更多，从而多了更多自我选择的机会。

如果没有当初自己的主动探求，不断地思索实践和总结，我想我不会在软件测试这个行业里面停留太久。其实喜欢与否，有的时候也在于你是否有兴趣去探求，去思考和了解这个行业里面的浮在表面之外那些东西。不了解，就很难喜欢，不喜欢，又有什么机会去因为兴趣而未知努力呢？

同样的，如果在嵌入式手机软件测试那个领域，如果我没有努力的提升自己的管理能力和外语能力，没有努力的去学习应用软件以及相关测试工具的知识，没有一点点深入的去学习更多更广泛的测试的理论并且不断的引入到我自己的测试实践当中，我想，也没有现在的发展。

在任何一个行业，都需要不断的努力和突破自己，给自己设立一个个小小的里程碑，然后让自己的视野不断的得到拓展。让自己的能力不断地得到提升，你才能在未来走得更好。

就我个人的经验而言，我努力提高自己的理由就是因为对于简单的测试执行的不甘心。

每个人都会在某个时候因为某些情况而觉得有些情绪，有些不甘。而处理这种情绪的方法大家都会不同。我个人比较喜欢从自己身上找各种原因，然后去提高我的自己。

首先不要人云亦云的附和某些声音。某一份工作是否值得做很多时候不是别人的看法而是自己的选择。既然选择了，就要做出一些努力。你可以好高骛远，天马行空的去设计千百个美好的未来，归根到底，还是要努力做好每一天的事情。只有自己真正的投入了，确切地了解了，你才知道这份工作里面到底有哪些‘机关’，藏着哪些你不知道的东西。哪些是能成就你的，哪些是你不感兴趣的，哪些是你在努力也无法弄明白的。

然后就是不要被动的等待着别人的指令，自己想办法去学习更多。古人有云：不在其位，不谋其政。我觉得到了当今社会，这句话需要改为：不在其位，不谋其政，要具其能。之所以这样说，理由很简单，举一个团队里面很普通的例子。你现在不是团队的领导者，但是你处处给自己创造机会学习，而且积极协助管理

者去做一些事情，具备了一定的团队管理能力，当有一天，有了这样的机会，具备这样素质和潜质的你肯定会有优先胜出权。

再次，从实践上讲，你可以在你的项目里面找一个你看的起的人作为你超越的目标，这也是让你不去懈怠和沦落的一个很有效的方法。然后向他学习直到超越他。如果有一天你发现，很多以前不清楚的东西你明白了，你的成长依然没有改变的薪资或者其他你觉得你应该获得的，那么离开的时候，你也不会有一无所获的遗憾。

最后，知其然而究其所以然的工作习惯一定要有，只有这样，你才能不断地找到乐趣。也只有这样，你才能让自己的大脑迅速的旋转起来，学会去思考，然后一步步走向自我的提升。对于别人的介意和质疑，不用太多的抱怨，用你的实际行动，做给他们看吧！

真心爱测试，努力去工作，寻找自己的价值所在，不断的提升自己，让自己成为可替代率越来越低的人，这也是我们这些奋斗在软件测试行业里的小人物，在当今需要做到的事情了。

提高软件测试效率的方法探讨

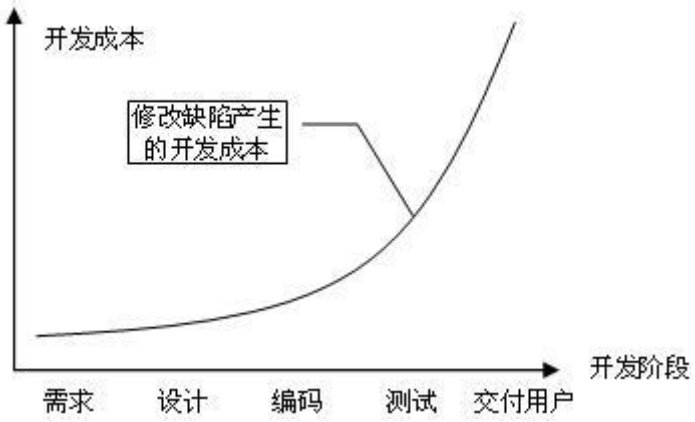
作者：丁志义

摘要：有位大师曾经问我，如何快速发现软件中的 BUG？在当时有限时间情况下，我只说了测试者经验、熟悉需求等几个方面，显示这样的回答没能令他满意。软件测试有无银弹？有无高效的测试方法能尽快尽多发现软件中的缺陷？本文汲取部分软件测试工作者经验并结合笔者工作经历，就提高测试效率的一些方法进行归类探讨，以期和同行共享。

关键词：软件测试，效率

前言：

软件缺陷暴露得越早，越能降低开发和维护成本。研究表明软件寿命周期中，暴露缺陷的阶段与修改缺陷产生的开发成本之间的对应关系如下图所示：



软件测试作为一项工作，提高它的效率意义重大。

怎样才能最快发现缺陷呢？我个人认为没有这样的利器，没有这样的银弹，这是个伪命题。要相信没有最好只有更好。就像刘翔你怎样让他跑的最快呢？显然只有和别人比较时候才能分出谁最快。当他打破了世界记录又打破了自己的记录时，是在不断超越自己。软件测试也一样，随着管理和技术的进步，我们的工作效率也会不断提高。就像成功没有秘诀，但成功一定有方法样法，软件测试工作也有他的规律和方法可寻。同时在测试的不同阶段，测试工作内容重点各不相同，需要行使不同的测试策略。所以，要想提高测试效率或者更加高效的尽可能早的多发现缺陷，只能是各种方法或手段的联合应用，而不是只有一味良方。

工作效率又分为个人和团队效率。很多方法是在团队组员中同时使用的，这样提升了每个人的效率，团队效率自然上升，有些策略是让管理者在测试过程中

去实施，最终还是由测试组员执行，因此本文不再具体细分哪些原则或方法是针对个人还是针对测试团队。

一、测试效率的度量

怎样才是高效率的测试？软件测试目的是发现 BUG，并保证 BUG 得以修复，降低质量风险。测试员工作的主要内容是进行软件测试，优秀的测试员发现有效缺陷数量比普通测试员一定高，这个是不争的事实。所以我们可以简化衡量测试员在软件测试阶段的工作效率指标，那就是在一定时间内的有效 BUG 数量，或平均每日或每周 BUG 数量。当然缺陷的质量、重要紧急程度也很重要，但我们可以通过培训，把缺陷的标准进行统一让测试员填写的缺陷更加规范。如果你的团队中任务分配均匀合理，通过 BUG 数量的多少就更能准确反应各测试员工作效率。如下图，显然测试员 C 本月的工作效率最高。

三位测试员发现有效缺陷数量分布

测试员	第一周	第二周	第三周	第四周	每周平均
A	70	73	53	60	64
B	65	64	48	43	55
C	75	76	60	65	69

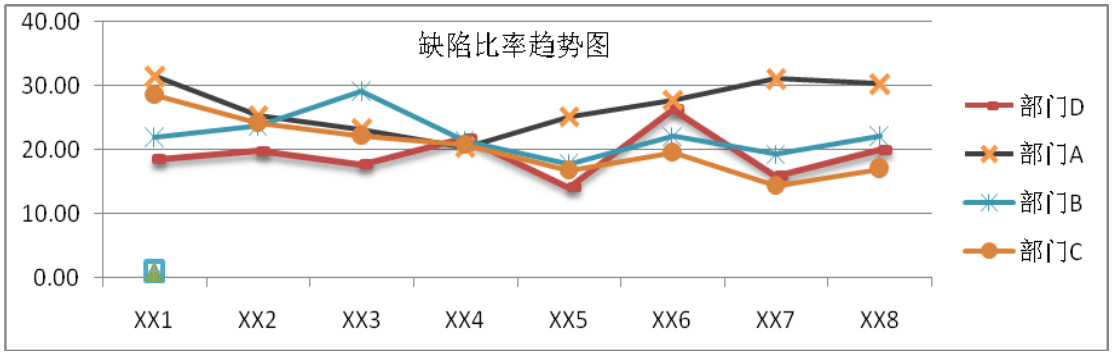
对与团体工作效率，若企业只有一个测试部门，无法比较谁快谁慢，只能是定性的认为在某个时点应该完成哪些测试内容，如没有完成则可认为效率低，提前完成认为效率高，但也不绝对，因为软件测试和软件开发一样有着很多不确定因素。

对大型应用软件企业，开发部门可能有好几个，各开发部门有自己独立的测试团队。不同测试团队的工作效率，可看他是否在里程碑事件前完成对应的测试内容，并提交对应的工作成果来定性判断效率的高低。

我们还可以从软件上市后用户的反馈问题中，分析是 BUG 性质问题占所有修改问题的比率，用这一指标来间接衡量各测试部门工作效率的高低，比率越低效率越高。参考下表和趋势图，从中看出在 XX8 产品发版后部门 C 的缺陷比率最小，测试工作效率应是最高。

各版本产品发版后支持问题中缺陷问题数占部门产品问题%

开发部门	XX1	XX2	XX3	XX4	XX5	XX6	XX7	XX8
部门 A	31.56	25.33	23.27	20.41	25.20	27.78	31.09	30.28
部门 B	22.05	23.79	29.23	21.43	17.76	22.12	19.32	22.22
部门 C	28.60	24.17	22.19	20.80	16.74	19.60	14.41	17.10
部门 D	18.48	19.75	17.65	21.82	14.07	26.32	15.90	20.00



注：XXX 表示产品版本

二、测试工作的七项效率原则

1. 主动思考，积极行动，尽早参与项目，做好前期准备，“有备”才能“无患”；

2. 一开始就牢记目标，不迷失方向，什么时间点完成某个测试项目要牢记；

3. 重要的事情放在首位（但常常将紧急的事情放在首位），学会时间管理；

4. 先理解人，后被人理解，测试是发现缺陷让产品更完美，而不是故意找茬；

5. 寻求双赢，积极配合开发人员工作，如帮助他发现问题规律，努力赢得开发人员支持；如哪些地方可能会有问题，需要加强测试；

6. 互相合作，追求 1+1>2，测试团队人员密切配合，促进测试整体进度；

7. 终生学习，自我更新，不断进步。

三、效率低下人群七大习惯

如果你觉得以上七条效率原则很抽象枯燥，那看看效率极低人群之七大习惯吧，如你能把这七个习惯规避掉，你的工作效率效率也会大大提高。

1. 缺席。当你每天都坚持写作或绘画，你就会快速的得到提高。如果你参加更多的约会，你遇到心上人的机会就会大大提高。仅仅只是更多出席就会使你

的生活大大不同。而缺席却会使你毫无收获。缺席情况如：这项测试我不干；只想学习参考别人的，自己却不去总结。

2. 拖拖拉拉。把一天最重要的事情先干完，把复杂的事情动手去分解，而不是像无头苍蝇一样没有头绪。

3. 做一些无关紧要的事情。除了拖拖拉拉以外，你最容易陷入的不良习惯之一就是忙于一些无关紧要的事情之中。为了提高效率需要学习时间管理方法。

4. 多虑。因为多虑而很少采取行动。陷于无穷的分析之中只会使你虚度光阴。虽然行动之前加以思考在一定程度上对你有所帮助，但你现在需要做的就只是停止思考，然后去做那些你应该做的事情。

5. 凡事过于消极。当你凡事都从消极方面考虑时，你的积极性就会被大大打击。测试是一项烦琐的工作，如果一直想着重复的劳动，会压抑心情，因此你必须得乐观点，选择积极的态度，在工作中寻找快乐。例如发现了 BUG，我就有成就感，这样产品才会按我们理想的那样越来越完美，而不是越来越差。

6. 固执己见，与世隔绝。解决方法之一就是打开心胸。开阔视野，从他人和自己的错误中汲取教训，从书籍等资源中获取知识，和别人多交流，三人行必有我师。

7. 持续信息过剩。信息过剩并不是说你过多的阅读，本处指的是输入信息的过剩。为了可以集中精力，清晰思考并付诸行动，你就需要在吸取信息时更有选择性。当你工作时尽可能的避免那些分散注意力的事物：关掉电话，断开网络和 QQ，关上大门。你就会不可思议的发现，居然可以完成这么多的事情。三心二意是干不好事情的。

四、改善或提高测试效率的几种常用方法

1. 注重计划，在到门口前将钥匙拿出来。

孙子兵法有云：夫未战而庙算胜者，得算多也；未战而庙算不胜者，得算少也。良好的计划是成功的一半，所谓神机妙算，就是在事情开始前进行周密的分析推演，这样事情才能按预想那样顺利进行。要想在测试中忙而不乱，顺利完成，我们必须制定周密的测试计划。需要怎么测试，哪些项目或内容需要测试，什么时间进行什么测试内容，由谁去完成，做到心中有数，责任明确。

测试计划包括项目开发计划、测试工作计划，日常工作计划等内容，在开发计划中主要把握各里程碑时间点，熟悉有哪些主要任务需要完成，合理安排测试资源，如果人力资源短缺，做到提前预防，如新员工招聘、能力提升培训等；测试工作计划是在整个产品从定义开发到上市发版过程中各阶段测试要做的工作安排，包括但不限于项目开发计划中测试工作内容，如参与各项评审活动，进行各种培训，如业务需求培训、开发过程培训、测试规范培训等，更重要的是什么

时间开始各项功能测试，由谁去测试，并将计划下发给团队中测试人员，好做些环境准备，时间安排等。日常工作计划，一般是一周末总结上周测试内容后，对下周工作进行具体性的安排，这样的安排更具体更高效，每个测试员会进行针对性的工作。当每个人按计划完成测试内容，测试工作才能按目标进行，而非随心所欲的无组织无纪律进行。注意一定不要在周一才将你的计划发下去，要打有准备之仗，要在到门口前将钥匙拿出来。

除了测试管理者要做工作计划，每位测试员工也应该有自己的工作计划，哪怕是很简短几个字的工作日志安排，都比无序工作效率高，工作才不盲目被动。

2. 注重测试用例和方案的编写及有效使用，不将其仅作为存档或应付检查的资料。

可将测试用例和测试方案理解为测试计划的一部分。

精心准备的测试用例和方案，一定是在消化过产品功能需求后进行的。在编写测试用例时还没有开始测试，没有紧迫感，因此有足够的时间让你去想，去分析你要测试的功能点，有哪些路径要去覆盖，要准备什么数据，预期结果是什么。各种测试方法都可在用例中体现，如正交分析法，边界值法，非法和常规数值等都会考虑进来。可以用 EXCEL 表格列出各测试项目内容，如果你能抽象出产品测试中的共性，可将测试内容进行一些归类，这样按项目去测试，效果会更好。表格中涉及谁去测试，第一次测试是谁，什么时间进行，下一次测试时间和测试员又是谁。参考如下表：

测试项目	一次测试人	时间	二测试人	时间	三测试人	时间	是否完成
工作流	A		B		C		是
角色功能权限确认	B		C		A		
权限（功能/模版/操作员/部门）	C						
单据格式	A						
数据精度	B						
……	C						

测试用例和计划应该被有效评审。百密一疏，谁都可能会有错误或疏漏，因此请组内同行和需求甚至开发人员提些问题，将遗漏的测试内容补齐，将使我们的用例更加完善，减少测试真空。

不是为了编写而编写，评审后的测试用例应该在测试时被有效使用，可以将部分用例提交给开发人员作为单元测试用例使用，更主要的是测试员自己要按用例步骤要求去执行。因为我们在编写用例时是经过深思熟滤的，用例是严谨科学

的，所以不能在进行测试时将用例摆在一边，而另外再想一套测试路径。你可以在按照用例的步骤执行完后，再发散一下思维，补充一下测试思路，而不是随意的不按套路的测试。经过这样有效的用例执行，发现再多的问题也不可怕，你要测试的功能点或项目都已经覆盖了，还会有什么大的问题呢？

要及时更新你的用例执行情况，这样能真实反映产品开发进度和稳定水平，如用例通过就要确定为通过等，而对项目型测试可以在表格中用颜色区分，通过、有问题、未进行等。参考上表，完成并通过测试标为绿色，有部分小问题标为黄色，有严重问题标红色，未进行测试不标注。

3. 注重沟通交流 ，减少阻隔和信息失真。

积极的交流可以减少歧义，达成共识。

例如测试人员和需求的沟通交流，对需求中不是很确切的内容进行明确，便可以进行一些更加具体的测试，防止漏测，或偏测；测试人员和开发交流，知道开发人员认为哪些地方可能会有问题，都会做些针对性的测试，更加快速发现问题；测试同开发的沟通还可减少对立、尴尬等局面，例如你填写的缺陷开发不认为是缺陷而僵持，导致该缺陷长期未解决，这样测试效率显然低下。其实有些歧义是正常的，这个时候要主动寻求需求人员或你的上级进行问题仲裁，而不是将问题长期挂起。

测试人员也可把自己的一些工作计划或工作步骤和开发交流，让开发和测试工作同步。如下周要进行哪些项目测试，开发人员可在本周内完成对应的开发任务，或及时将处理跟踪问题重点切换过来，都将对测试工作起到促进作用。

对异地开发有效的交流沟通意义更大，例如传输给对方数据接口的正确性，控制的理解等，除了电子邮件，最好勤打电话进行沟通交流，防止信息失真，理解错误。

4. 注重培训、学习和经验共享，在学习中提升素质。

测试离不开人，自动化测试只能在一定范围内使用，自动化工具的掌握也离不开人，软件产品质量由测试人员来保证而非工具，所以提高软件测试效率，要优先提高测试人员的素质。同时短板效应告诉我们需要对团队中能力欠佳者要进行培训提升，否则会影响整个测试团队的能力和效率。

培训包括新员工培训、产品需求培训、新方法新理论学习培训等。

新员工以自学和培训相结合，对新测试员工的培训，包括企业文化、产品知识、工作纪律、测试知识技巧、测试管理工具使用等培训都是必要的，认真有效的培训，能快速提升新员工工作能力，迅速转变为生产力，培训应当实用，贴近将要测试的工作内容为最好，其他相关知识可在工作中逐渐学习。

新产品功能需求培训，测试用例编写前，测试人员要对需求进行消化，这需

要需求人员做些针对性的培训，讲解需求目的和场景，让测试人员充分和快速理解需求，达到认知一致。编写测试用例才准确，测试工作才能在正确的道路上进行。

新的测试理论和方法的学习，汲取同行更先进更有效的测试方法，学习新的测试工具。同时学会及时总结经验，提取测试工作中的共性，把最新的工作经验共享给组员，让他人少走弯路。学习可以采取自学和集中培训方式，也可采取互相培训等方式，让组员人人参与，达到共同提高。采取开放的头脑风暴的方式进行学习培训，效果会更好。

通过培训学习和总结，团队能力将更大提升。通过互相学习团队协作精神也会更大发扬。

5. 注重时间管理，正确的时间做正确的事。

大凡成功者他也一定是高效人士，他对自己的时间安排利用也是恰到好处。每个测试人员如能将一天的工作进行时间象限划分，并按优先级先后顺序依次做最重要最紧急的事情、重要但不紧急事情、紧急但不重要事情、不重要也不紧急事情，他的工作效率不高都很难。

对整个测试产品而眼，当时间紧迫，尤其是测试后期回归验证测试时，来不及进行全面测试，要抓住重点，按优先级次序进行，而不能眉毛胡子一把抓，否则会丢了西瓜。先测试核心功能再测试辅助功能，先测试常用功能再测试不常用功能、先测试影响大的再测试影响小的、先测试变更过的，再测试未变更过的等等策略。

另外测试人员长时间高负荷工作，总有疲劳倦怠的时候，例如下午或者周一，效率低下等，那我们可以把培训、学习、无需动太多脑筋的测试工作安排在这个时期，让大家得到缓冲。有研究表明，上班族每周工作效率由高而低依序是：星期五（35.78%）、星期二（30.89%）、星期三（13.98%）、星期一（13.49%）、星期四（5.86%）。如果你的企业也符合这样的规律，那就在星期四多安排一些学习培训的事情吧，停止测试也是可以的。合适的时间做合适的事情，往往会事半功倍。

6. 注重工作文化氛围，标杆激励，人文关怀，提升士气 。

在组织中也可以采取很多激励工作热情的措施。在你的组织中开展类似劳动竞赛的活动，例如按每日或每周填写 BUG 数进行排名 ，选择出测试员 TOP10、TOP3 来，并张榜公布，形成看板，给优秀者加个大红花。这样的标杆效应不容小视，当下一周你从 TOP10 消失时你一定会有些失落惭愧吧，你一定会想办法高效地去挖掘深藏其间的 BUG，让你再次闪亮登场。

当团队中出现士气低落或者有疲惫感时，你需要给测试放个假。抽个晚上安

排你的组员出去小聚一下，唱个卡拉 OK，或者一起喝茶喝咖啡看个足球赛。过程中可以稍微总结一下前期工作和对下一步工作的安排，但不能太多。这种方式将让每一位测试人员得到很好的身心放松和调节，目的是为了明天更高效的投入工作，而不是走神，或者胡衍。测试工作是紧张严肃的，但更需要团结和活泼。

良好的企业文化，宽松的工作氛围，温馨的人文关怀都是不可缺少的，只有这样，才会有团队精神，或则都是单干，无法实现 1+1>2；要关心你的组员个人心情状况等，如果他有个人私事需要办理，一定要准假。最浅显的道理：人是社会的人，谁能没点事情能够呢？再说工作是为了更好的生活，生活出现问题需要调节，不让他去处理好个人私事怎么会有好的心情工作？怎么会能高效的测试？但这一点往往会被忽略，会被严格的考勤制度掩盖。

7. 利用交叉测试提升软件测试效率。

软件测试员长期重复测试同一功能或项目会出现疲劳和厌烦，会丧失激情，甚至不思进取。由于疲劳导致的精力不集中，很容易略过一些重要的测试环节，甚至面对显然的错误也“视而不见”，这是“审丑疲劳”。同时，在软件开发测试过程中，尤其是对同一产品，同一组开发人员和同一组测试人员长时间配合，测试人员会在开发人员多次说服和解释下而被同化。还有研究表明，人有定位效应。简单地说测试人员的定位效应就是测试过的功能不再进行认真测试。

针对“审丑疲劳”、同化现象以及定位效应等工效下降问题，必须采取措施，提高测试积极性和工作效率。除了将测试用例多次完整执行、必要的士气鼓舞等激励措施外，交叉测试便是一种有效的办法。

交叉测试可以很大程度的避免定位效应。测试人员在互相交换任务时，反复测试某一功能的机会大大减少，从而也就不会“主观的”认为某些功能没有缺陷。因为测试工作内容的变化，对新测试人员来说更有新鲜感、挑战性和刺激性，测试人员对未知领域和未驾御的程序将更乐于测试。测试内容的变化对应配合的开发人员一般也会对应改变，这样“审丑疲劳”和“同化现象”都将大大降低。不同测试人员的不同视角和方法能将隐藏的缺陷快速找出，容易发现更多问题。实践表明一个“新手”发现问题会更多更快。

通过这样的交叉测试，“漏网之鱼”会越来越少。

关于交叉测试意义，怎样进行交叉测试设计、交叉测试时点、执行和分析等，详见笔者发表于 51testing 的另一篇文章《利用交叉测试提升软件测试效率》，链接地址：http://www.51testing.com/html/23/n-205923.html

8. 工具的有效使用，让辅助工具发挥它该有的效果和该尽的责任。

没有什么比使用工具效率更高了，虽然软件测试中离不开测试员的手工测试，自动化测试也只是在一定范围或一定时间段内使用，但追求自动化测试一直

是我们不懈的努力目标。工具能重复使用，夜间使用和不间断的连续使用，而人却不行，所以让测试工具成为我们测试员的好帮手，来辅助我们，那是必须的。测试工具使用越多，你的测试效率就越高。相信谁都知道 Ghost 这个工具带来的好处吧。

例如在回归测试时，没有太多时间让我们全部测试，那就用 Robot、QTP 等工具来进行功能按钮测试吧；例如用 Loadruner 模拟我们测试场景，代替我们很难完成的大用户场景下的性能测试，发挥它的神气效果吧；你还可以录制脚本来进行冒烟测试，因为冒烟测试关注的是主要功能流程，是快速验证新的安装盘是否可以被大多数测试者使用，测试的功能点非常固定，用自动化工具进行辅助测试再合适不过了；如果你的企业软件数据库有面向老客户的升级要求，那升级完毕后用数据库对比工具 dbexplor 等先分析一下数据升级的正确性，再进行功能验证等等。总之要充分发挥这些测试工具的作用。

当然也可自行开发一些针对性的测试小工具，如果没有开发能力，那把你的美好想法形成需求文档，告诉和你配合的开发人员，让他帮你实现吧。这样的小工具很有效果，例如笔者联合开发人员开发了一款文件版本比较工具，可以进行软件新旧版本下文件的版本、只读属性、大小等信息比较，防止老客户卸载升级后因同名文件版本错乱带来应用上的错误等。对于安装后文件数量上百上千甚至上万个，如果进行手工比较是不可想象的。这类小工具的开发，需要测试者从工作经验中提取测试对象共性，分析容易实现工具测试，且小工具本身容易开发才行。

9. 标准化，让测试环境具有互换性。

规范的测试流程和测试工作制度是我们顺利开展工作的基础，是测试员和开发人员甚至和需求人员共同遵守的法则,这个是工作制度的标准化。

而测试环境的标准搭配（如谁的机器 IP 地址是多少），和测试数据的标准化，更会降低测试跟开发人员信息交流成本。例如一个规范的基础档案命名，让所有的测试人员和开发人员都知道该档案的主要属性，这样在使用该数据和分析该档案在后续计算错误中就不需要再去查看该档案属性，开发登陆到谁的机器看到的数据都一样，不再需要信息的二次转换，对开发处理问题和日常沟通都是很好的促进作用。

为此我们可在测试数据标准化上下工夫。通过大家讨论确定该有什么样的基础数据，数据的命名编号要规范，可通过特别方法加以释义便于记忆，做到一看到某个代号便知道它有哪些属性，就如同标准件螺栓螺母一样。可在同一个测试环境下做出来，然后导入到不同机器中，这样一套标准的全面的基础数据大家共享，做到不同测试人员不同机器上测试用基础数据一样,让数据具有互换性。在

后续测试中只需要做日常业务单据即可。

软件测试的标准化将是一项任重而道远的工作，以上仅是抛砖引玉。

五、结束语

提升工作和测试效率的方法远远不止这些。但如果能将上述效率原则、改善测试效率的方法在工作实际中具体应用，并在不同阶段实施不同的测试策略，在工作中发挥每位员工的智慧，让组员参与效率改进，相信一定能提升每个测试员工的工作效率，从而促进团队工作效率的改进，达到 1+1>2 的效果。

浅谈软件开发中的人，过程与技术

作者：茹炳晟

摘要： 人是软件开发的执行者。过程是软件开发的体制。技术是软件开发的精髓。三者缺一不可，却是以人这个根本原动力为核心的。

关键字：执行者，体制，精髓，根本原动力

前言：

在业界，一种普遍的看法是：人、过程和技术构成了软件开发的“铁三角”。然而，三者孰重孰轻，历来多有分歧。本文拟以此为论题，从科学技术哲学的角度谈一下笔者的看法。

三者中，过程与技术蕴意颇深，容后再述。我们先看看人：

一、人：软件开发的执行者

对于人在软件开发中的作用，笔者的第一印象是：人是软件开发的执行者。这一印象虽不为众多“人本位者”所乐见，在笔者看来却是中肯的。

“执行者”这个称谓，看似贬低了人的作用，实则不然。须知，执行者不仅包括被认为是业内蓝领的程序员和测试人员，还应该包括荣为白领的项目经理、系统分析师、系统架构师、系统设计师等等。君不见大大小小的各类公司站在前台的一号人物，名何？CEO 是也。CEO 即首席执行官，或称执行总裁。虽则首席，虽则总裁，也只是一个执行者而已。在公司中，大到 CEO，小到普通职员，都是所谓的“劳方”，都是公司意志的执行者。（公司意志即是资本的意志，其间种种，不在本文论题之内，略过不言。）同样的，当讨论领域是软件开发时，上述所有的开发人员都只是软件开发的执行者，执行的是软件项目的意志。何谓软件项目的意志？很简单的两个字——需求。这是有大背景的。目下的经济为市场经济，市场经济的特点就是以需求为中心。需求是生产之母，这是市场的铁律，也是市场化的软件开发的铁律。所以，软件项目的意志只能是需求，在此意志下的所有技术活动（即软件开发）只能是执行这一意志，那么软件开发人员就都是执行者了。当然也有人会提出：“你太片面了，除了软件开发者，还有一类人——提出需求的利益关系人，不也是人吗？所以人不仅仅是执行者，更是主宰者！”遇到这个问题，我们首先要明确现在所论述的领域，是软件开发，而不是软件项目，两者有联系，但是也有区别。就软件开发而言，一切都是从需求开始的，至于提出需求的人，只是被看作需求的来源，是需求分析师们交流的对象，而非开

发的一部分。换句话说，如果给软件开发划一条边界，需求提出人在边界之外，只有软件开发人员才在边界之内，也只有他们才是软件开发中的人！

作为执行者，人的作用是巨大的。所有的开发，没有了人都是空谈。人的素质如何、人的状态如何，人与人的关系如何，都直接决定了软件开发的成败。这样的例子比比皆是，就不一一道来了。这里就拿军队作为比喻。如果将软件开发比做打仗，那么开发者们就是军队中的军官和士兵，哪怕军纪再严明、武器再先进，只要军官差劲，士兵孱弱，那么这支军队就注定失败了。反之，游击队也能打败正规军，小米加步枪也能打败飞机大炮，人的作用可见一斑。

当然，人再重要，也只是“铁三角”的一个顶点。在软件开发中，过程同样不能忽视，这是因为：

二、过程：软件开发的体制

过程是软件开发的体制。此言一出，必遭不满。唯过程论者一定会跳出来痛批笔者偏见。然笔者仍不知悔改，自以为这种说法是最贴切的。

不满大多是冲着“体制”这两个字来的。在当今的中国大地上，“体制”这两个字带者浓厚的贬义色彩。政府部门腐败丛生，答曰：“这是体制问题”；造导弹的不及卖茶叶蛋的，答曰：“这是体制问题”；中超造反，世界杯淘汰，依旧答曰：“这是体制问题”。不管大事小事，难事易事，出了问题只要推给体制就万事大吉了。反正千错万错都是体制的错，因此你没错我也没错。体制不能改也改不了（真要改了谁做替罪羊啊？），所以错误的体制永远存在，属不可抗力，非人力所能及。

这里不讨论中国的国情，只是借此理解“体制”两字的分量。其实，说得通俗些，体制就是固化下来的行事规则与经验。因为曾经取得过成功，并且也吸取了一定的教训以规避失败，因此在相同的情况下，体制会带来更大的成功概率，并形成权威，使后来者减少扯皮，提高效率。所以，体制不仅不是贬义词，相反蕴涵了大大的褒义（只是被用滥了，变成了大反派。）。现在看看软件开发中的过程，与体制的含义正是完全吻合。因此，合适的过程，指导人们正确的完成各种开发活动，一步步迈向成功的彼岸；而不合适的过程，则把软件开发推入无底深渊。

把过程看成软件开发的体制，不仅仅是因为两者在现象上是一致的，更重要的是在我们对待它的态度上也要有所借鉴。现在有不少开发人员，在遇到开发中的问题时，往往就会把责任推到过程上，然后感叹一句：“有 CMM（一种软件过程改进的评价标准，其对应的方法论是 RUP）就好了。”接着项目经理不厌其烦，桌子一拍，大叫：“要有 CMM！”于是，便有了 CMM。如此折腾一年后，终于通过了 CMM 认证，依旧遇到开发问题，开发人员仍有的说：“要填这么多

表格，太费时间了；我们不该使用 RUP，改 XP(一种强调迅速应变的软件开发方法论，美其名曰极限编程)就好了。”然后项目经理晕头转向，闭目养神，道：“那就改 XP 吧。”于是，便改了 XP。谁知还是出问题。这时开发人员倒不抱怨了，因为项目已经宣告失败，项目经理也被撤职了。这个不是笑话，而是笔者亲眼所见的。由此可见，过程不能被神话，更不能被乱用。世界上不存在放之四海而皆准的过程，而只有因地置宜、因人置宜地选择合适的过程，并在执行过程中不断地改进该过程，才有可能取得成功；相反，则只能失败，不做他想。

体制二字，真实地体现了过程在软件开发中的地位。然而，光有人与过程还是不够的，最后一个关键的因素是：

三、技术：软件开发的精髓

要说明技术在软件开发中的作用，就得先说说为什么要有软件开发。软件开发，顾名思义，就是要开发软件。那软件是什么？可执行的程序？可带来利润的产品？都对，但都不够全面。从根本上而言，软件，是技术的载体，是被定制的技术，是技术针对某项特定需求的应用。

软件中沉淀着技术，这是显然的，否则别人为什么要买你的软件呢？当然软件中不仅仅沉淀着技术，还包括开发的人力成本、机械成本、时间成本等。但是技术是其中的精髓，也是此软件不同于彼软件的根本所在。软件的竞争力，不在于它的规模，不在于它花费的各种成本，而在于它的功能及非功能特性，而在于这些功能及非功能特性之中蕴涵的技术。比如说，Microsoft 为什么能统治 PC 机的操作系统市场？不是人，不是管理，而是技术，将图形用户界面与操作系统和二为一的 Windows 技术。在 Microsoft 之前，并非没有好的图形用户界面（Apple 的 Macintosh 是这方面的先驱），也并非没有好的操作系统（DEC、IBM 等的 Unix 异常强大）。但是，只有 Microsoft 将图形用户界面技术与操作系统技术结合了起来，应用到 Windows 中，形成了其特有的 Windows 技术，从而异军突起，开创了 Microsoft 帝国。更有甚者，当 Dos 内核的 Windows95/98 无法完全满足市场需求时，Microsoft 不惜撤换整个开发组，改用 Unix 技术重新开发 Windows（Microsoft 称其为 NT，即新技术），于是 WinNT、Win2000、WinXP 等等在新千年中依旧辉煌。可见，Windows 的发展史就是技术在软件过程中作用的最好诠释。

当然，软件开发中的技术不仅仅是指沉淀在软件中的技术，还包括开发技术本身。如开发环境、管理方式、过程工具等等，都是开发技术的应用领域。可以说，软件技术及开发技术，就是软件开发的精髓所在。

一言以蔽之，人、过程和技术在软件开发领域都是不可或缺的。“铁三角”三字正是名副其实，三者缺一不可。那么到底哪个最重要呢？如果仅仅从软件开

发本身来考虑，恐怕难以给出合适的答案，因为厚此薄彼则必然误入歧途。那么似乎本文可以到此为止了。可字数仍然不够诶！（呵呵，开个玩笑）所以继续想——如果撇开软件开发这一具体领域，就一般意义而言，哪个更重要呢？然后再回到软件开发领域，就会发现：

四、再论人：根本原动力

人乃万物之灵。此话同样会引起诸多非议。但是，即使最激烈的批评家也不得不承认其中的合理部分：正是人的情感、认知和创造能力构建了我们所处的人类社会。撇开情感不谈，人的认知与创造能力也是软件开发的根本原动力。

在软件开发中，随处都是人的身影。软件开发的执行者是人，这是显然的了。那么软件开发的过程呢？过程是靠人来实施、来管理的，这无须说，这充其量只是执行的一部分而已。这里想说的是过程本身，它是如何来的呢？过程不是凭空产生的，当过程成为一种体制时，它也是靠人通过不断的创造－使用－改进－再使用－再改进而来的。在过程改进的过程（套用现在的一句行话，这叫做meta-process，即元过程，过程的过程）中，需要的是人，是人在创造着新的过程，是人在推动着过程的改进，是人在使用过程的软件开发中取得了成功！这里要说明一句的是，此人非彼人。此处的“人”不是指“铁三角”一端的人，那是作为一个项目的开发人员、作为软件开发执行者的人。而是所有致力于软件开发这一领域的所有人的总和。这个“人”并不处于一次软件开发的边界之内，而是在将软件开发作为整个领域，包含软件开发的过去、现在和未来，包含无数次软件开发及无数次对于软件开发的研究的前提下，处在这个领域中的一个概念，是历史上曾经出现的、现在正在进行着的、以及日后将会置身其中的所有的开发人员和研究人员的总和。人的认知与创造能力，在这个“人”上体现得淋漓尽致。正是这个人，在最初混乱的软件开发实践中发觉了软件开发过程的重要性，从而逐步创造出各种软件开发过程，并使用之，从而使软件开发秩序化；正是这个人，在使用软件开发过程的实践活动中发现了过程所存在的不足之处，于是吸取教训改进之，从而使软件开发过程更合理、更有效；正是这个人，不局限于过去所创造的过程，怀着扬弃的态度不断地投身于软件开发中去，从而使不断进化的软件开发过程真正成为软件开发“铁三角”的一个顶点，为软件开发、为人类服务。而我们每一个软件开发人员，每一个从事软件开发的研究人员，都是这个人的一部分，都通过自身的活动体现了这个人的作用。所以，我们的整体是具体化的他，而他则是抽象化的我们。

当然，不仅是过程，在技术方面，人所起的作用是类似的。技术作为科学与生产之间的桥梁，其产生、改进、普及直至消亡都离不开人的推动。因此，人是软件开发中当之无愧的根本原动力，软件开发的“铁三角”，包括作为软件开发

执行者的人，作为软件开发体制的过程以及作为软件开发精髓的技术，都是以人这个根本原动力为核心的。概括地说，就是“一个核心，三个顶点，一个都不能少”。

那么，当我们认识到人在软件开发中的核心地位时，应该如何做呢？权以如下文字作为这个问题的答案，以及本文的结尾：

If A thru Z are scored 1 thru 26，

Your knowledge has a score of only 96 (11+14+15+23+12+5+4+7+5)；

And your hardwork just scores 98 (8+1+18+4+23+15+18+11)；

While your attitude is what determines everything of your life (1+20+20+9+20+21+4+5)。

参考文献：

[1] 自然与科学技术哲学，上海交通大学科学史与科学哲学系，上海交通大学出版社，2001。

[2] 简明科学技术史，江晓原，上海交通大学出版社，2001。

[3] Software Engineering –A Practitioner’s Approach（Fifth Edition）, Roger S. Pressman, 2001。

[4] 软件开发的滑铁卢—重大失控项目的经验与教训,Robot Dauglas,电子工业出版社，2000。

[5] Under Pressure and On Time，爱德沙利，机械工业出版社出版。