目录

软件测试 从零开始.....2

如何做好单元测试.....13

为盈利而测试.....20

软件测试的常识.....32

如何提高软件的质量.....36

浅谈功能测试用例模板设计.....51

成功测试自动化的 7 个步骤.....55

缺陷漏测分析：测试过程改进.....77

Rational Robot 的启动.....89

JAVA 测试模式.....95

软件测试 从零开始

王 威

Wangwei@withub.org

【摘要】本文面向软件测试新手，从测试前的准备工作、测试需求收集、测试用例设计、测试用例执行、测试结果分析几个方面给出建议和方法。鉴于国内的软件开发、测试不规范的现状，本文为软件测试新手提供了若干个软件测试的关注点。

【关键词】软件测试、测试用例、测试需求、测试结果分析

引言

几年前，从学校毕业后，第一份工作就是软件测试。那时候，国内的软件企业大多对软件测试还没有什么概念，书店里除了郑人杰编写的《计算机软件测试技术》之外，几乎没有其它的软件测试相关书籍，软件测试仅仅在软件工程的教材中作为一个章节列出来，因此，我对软件测试一无所知。不过，在正式走上工作岗位之前，公司提供了为期两周的系统的软件测试技术专题培训，对接下来的软件测试工作有很大的指导意义。现在，我继续从事软件测试的培训与咨询服务，在这个过程中，亲眼目睹了很多软件测试新手面对的困惑，他们初涉软件测试行业，没有接受系统的培训，对软件测试一无所知，既不知道该测试什么，也不知道如何开始测试。下面针对上述情况，给出若干解决办法。

一、测试准备工作

在测试工作伊始，软件测试工程师应该搞清楚软件测试工作的目的是什么。如果你把这个问题提给项目经理，他往往会这样回答："发现我们产品里面的所有 BUG，这就是你的工作目的"。作为一名软件测试新手，如何才能发现所有的 BUG？如何开始测试工作？即便面对的是一个很小的软件项目，测试需要考虑的问题也是方方面面的，包括硬件环境、操作系统、产品的软件配置环境、产品相关的业务流程、用户的并发容量等等。该从何处下手呢？

1. 向有经验的测试人员学习

如果你进入的是一家运作规范的软件公司，有独立的软件测试部门、规范的软件测试流程、软件测试技术有一定的积累，那么，恭喜你！你可以请求测试经理委派有经验的测试人员作为你工作上的业务导师，由他列出软件测试技术相关书籍目录、软件测试流程相关文档目录、产品业务相关的文档目录，在业务导师的指导下逐步熟悉软件测试的相关工作。其实，在很多运作规范的软件公司，已经把上述的师父带徒弟的方式固化到流程中。

如果你进入的是一个软件测试一片空白的软件企业，那么，也恭喜你！你可以在这里开创一片自己的软件测试事业，当然，前提是老板确实认识到软件测试的重要性，实实在在需要提高产品的质量。这时候，可以到国内的软件测试论坛和相关网站上寻找软件测试资源，这种情况下，自学能力和对技术的悟性就至关重要了。

2. 阅读软件测试的相关书籍

现在，中文版的软件测试书籍越来越多，有的是国人自己写的，有的是翻译国外经典之作。可以到 www.chinapub.com 或者 www.cnforyou.com 等网络购书的站点查找软件测试相关的书籍。目前，从国外引入的软件测试书籍有很多经典之作，但是，翻译成中文后，翻译质量对阅读效果有很大的影响。

3. 走读缺陷跟踪库中的问题报告单

如果您所在的公司已经有软件缺陷跟踪库了，无论采用的是商用工具，如 ClearQuest、TestDirecter 等工具，还是采用的 Bugzilla、Mantis 等开源工具，这都无关紧要，缺陷跟踪库中的缺陷报告单才是有价值的。缺陷跟踪库中的问题报告单是软件测试工程师工作绩效的集中体现，同时也是软件产品问题的集中体现。一般来说，缺陷报告单中最关键的几个部分包括：第一部分是发现缺陷的环境，包括软件环境、硬件环境等；第二部分是缺陷的基本描述；第三部分是开发人员对缺陷的解决方法。通过对上述缺陷报告单的三个部分作仔细分析，不知不觉你已经吸收了其他软件测试人员的工作经验，并掌握了软件产品常见的基本问题。这是迅速提高软件测试经验的好方法。

4. 走读相关产品的历史测试用例

如果你所在的公司有测试用例管理系统，那么，走读相关产品的

软件测试用例是迅速提高测试用例设计水平的一条捷径。走读测试用例也是有技巧的。测试用例写作一般会包括测试用例项和根据测试用例项细化的测试用例，下面举例说明。“测试用户登录的功能”是一个测试项，该测试项的目的是测试用户登录功能是否正确，是否能够完成正常的登录功能，是否能够对非法用户名和密码做异常处理等等。因此，根据该用例项，可以设计出若干个测试用例，大多数情况下，测试用例项和测试用例是一对多的关系。

通过走读测试用例项目，你可以掌握应该从哪些功能点着手未来的测试工作；通过走读软件测试用例，你可以了解如何根据被测试的功能点开展软件测试用例的设计工作，包括如何确定测试用例的输入、测试用例的操作步骤和测试用例的输出结果等。

总之，走读其他软件测试人员设计的优秀软件测试用例，是提高自身用例设计水平的好方法。

5. 学习产品相关的业务知识

软件测试人员不仅要掌握软件测试技术相关知识，对产品相关的业务知识也要学习。这很好理解，如果从事财务软件的测试工作，一定要学习财务知识；如果从事通讯产品测试工作，那么相关的通讯理论知识也是必须的；如果从事银行软件的测试，银行的业务流程也是不可或缺的知识点。

因此，在学习软件测试技术的同时，千万不要忽略产品相关业务知识的学习。如果你是一个软件测试技术专家，但是对产品业务知识一无所知，那么也只能测试出来纯粹的软件缺陷，而面对眼前出现的产品业务相关的缺陷，很可能是视而不见，如此这般，软件测试的效果会大打折扣。

二、识别测试需求

识别测试需求是软件测试的第一步。如果开发人员能够提供完整的需求文档和接口文档，那固然好。可以根据需求文档中描述的每个功能项目的输入、处理过程和输出，来设计测试用例。如果开发人员没有提供软件需求文档，那该如何是好？下面给出几个有效的方法：

1. 主动获取需求

开发人员通常不会更好地考虑软件测试，如果没有开发流程的强

制规定，他们通常是不愿意提供任何开发文档，即便有强制规定，需求文档也未必能够真正指导软件系统测试工作。因此，需要测试人员发挥主观能动性，与相关的软件开发项目经理和软件开发人员保持沟通，了解软件实现的主要功能是什么，并记录得收集到的信息。一般来说，开发人员即便没有提供相关需求文档，也会保存一些简单的过程文档，主动向开发人员索要这些文档，可以作为测试的参考。此外，可以与公司的技术支持人员交流，技术支持人员是最贴近用户的人，因此，通过交流可以获取第一手的用户使用感受，在测试的过程中会更加贴近用户。

当拿到相关的资料后，从哪些方面分析需求？如何与开发人员交流需求？其实，只要把握需求分析的几个关键的点就可以解决问题：输入、处理过程、输出、性能要求、运行环境，下面针对每一个项目逐一分析：

软件输入：与该需求相关的一切可能输入，可以从这几方面考虑，输入来源、输入参数的数量、输入参数的度量单位、输入参数的时间要求、输入参数的精度和输入参数的有效输入范围。在测试用例设计中，这部分内容作为测试用例输入的依据。

处理过程：描述对输入数据所执行的所有操作和如何获得输出的过程。测试人员了解处理过程即可，在测试过程中发现 BUG 时候，如果对处理过程了解的深入，对定位问题根源有很大的帮助。

软件输出：描述每个需求的输出结果，包括输出的位置（如计算机显示器、打印机，文件），输出参数的数量、输出参数的度量单位、输出参数的时序、输出参数精确度、输出参数的有效输出范围、错误消息。在测试用例设计中，这部分内容作为测试用例的预期输出。

性能要求：与该需求相关的性能要求，比如“插入 ATM 取款卡后，3 秒钟内弹出提示用户取款的图形界面”。3 秒钟这一限制，就是对需求的基本性能要求。

运行环境：软件的运行所需的环境，包括硬件平台的要求、操作系统的要求、数据库的要求，以及其它相关支撑软件的要求。

2. 确认需求的优先级

确认需求的优先级是很必要的，如果在产品进度比较紧的情况下，测试人员可以考虑优先测试优先级高的需求项，如果进度允许，

那么在测试优先级低的需求项，如果进度不允许，那么就放弃测试优先级低的需求项。如果软件公司有规范的流程支撑，开发人员在提供软件需求文档的时候，应该在文档中确定需求的优先级。但是，如果开发人员连基本的软件需求文档都没有提供，又怎能指望他们确定软件需求的优先级？如果是这样，需求的优先级只能由测试人员完成了。

3. 加入开发小组的邮件群组

测试人员需要通晓被测试产品，但是，产品在开发的过程中往往是不断变化的。如果软件开发团队有一套变更控制流程，测试人员会对产品的变更了如指掌。如果没有变更控制，那就要采用其他的土方法了。如果公司里面有自动化办公系统，也许采用的是 Lotus Notes 系统，也许使用的是 E-mail 系统，测试人员应该加入到开发人员的邮件群组中。当开发人员通过邮件讨论问题、通知召开技术会议的时候，测试人员可以及时知晓，如果必要，可以参加开发人员的技术会议。即便公司里面有了软件变更控制流程，加入到开发邮件群组也是一个很好的习惯。

4. 与开发人员为邻

建议测试人员与开发人员为邻。我所在的测试组曾经与开发组是在相邻的写字间里，开发人员与测试人员的关系非常融洽，抛去同事关系，大家还是不错的朋友。不管开发人员有什么样的活动，测试人员都能第一时间获得信息。无论从事软件测试工作，还是从事其它的工作，与工作中上下游环节的同事保持良好的个人关系对工作有很大便利。一般的公司内部都存在部门墙，良好的人际关系是打通部门墙的手段之一。向领导建议测试人员与开发人员为邻，这很必要。

三、测试用例设计

测试需求收集完毕后，开始测试设计。测试用例是什么？测试用例就是一个文档，描述输入、动作、或者时间和一个期望的结果，其目的是确定应用程序的某个特性是否正常的工作。设计测试用例需要考虑以下问题：

1. 测试用例的基本格式

软件测试用例的基本要素包括测试用例编号、测试标题、重要级别、测试输入、操作步骤、预期结果，下面逐一介绍。

用例编号：测试用例的编号有一定的规则，比如系统测试用例的编号这样定义规则：PROJECT1-ST-001，命名规则是项目名称＋测试阶段类型（系统测试阶段）＋编号。定义测试用例编号，便于查找测试用例，便于测试用例的跟踪。

测试标题：对测试用例的描述，测试用例标题应该清楚表达测试用例的用途。比如“测试用户登录时输入错误密码时，软件的响应情况”。

重要级别：定义测试用例的优先级别，可以笼统的分为“高”和“低”两个级别。一般来说，如果软件需求的优先级为“高”，那么针对该需求的测试用例优先级也为“高”；反之亦然，

测试输入：提供测试执行中的各种输入条件。根据需求中的输入条件，确定测试用例的输入。测试用例的输入对软件需求当中的输入有很大的依赖性，如果软件需求中没有很好的定义需求的输入，那么测试用例设计中会遇到很大的障碍。

操作步骤：提供测试执行过程的步骤。对于复杂的测试用例，测试用例的输入需要分为几个步骤完成，这部分内容在操作步骤中详细列出。

预期结果：提供测试执行的预期结果，预期结果应该根据软件需求中的输出得出。如果在实际测试过程中，得到的实际测试结果与预期结果不符，那么测试不通过；反之则测试通过。

软件测试用例的设计主要从上述 6 个域考虑，结合相应的软件需求文档，在掌握一定测试用例设计方法的基础上，可以设计出比较全面、合理的测试用例。具体的测试用例设计方法可以参见相关的测试书籍，白盒测试方法和黑盒测试方法在绝大多数的软件测试书籍中都有详细的介绍，这里不作赘述。

2. 重用同类型项目的测试用例

如果我看得远，那是因为我站在巨人的肩上 －－牛顿。

一般来说，每个软件公司的项目可以分为固定的几大类。可以按业务类型划分，比如 ERP 软件、产品数据管理软件、通信软件、地

理信息系统软件等等；可以按软件结构来划分，比如 B/S 架构的软件、C/S 架构的软件、嵌入式软件等等。参考同类别软件的测试用例，会有很大的借鉴意义。如果，公司中有同类别的软件系统，千万别忘记把相关的测试用例拿来参考。如果，系统非常接近，甚至经过对测试用例简单修改就可以应用到当前被测试的软件。“拿来主义”可以极大的开阔测试用例设计思路，也可以节省大量的测试用例设计时间。

3. 利用已有的软件 Checklist

在上面一个小节中，按照不同的规则划分了不同的软件类型。每种类型的软件都有一定的测试规范，比如，WEB 软件系统在系统测试过程中，会有一系列的范式，比如针对 Cookie 就会有很多测试点。在设计测试用例的时候，不妨到网上去搜索相关的 Checklist，不过国内外的网站很少有这方面的资料，即便有，也不是特别系统。可以先找一份粗糙的 Checklist，然后，在设计测试用例的时候不断的去完善它，以作为下次测试用例设计的基础。

4. 加强测试用例的评审

测试用例设计完毕后，最好能够增加评审过程。同行评审是 CMM3 级的一个 KPA，如果因为公司没有通过 CMM3 级，就不开展同行评审是不恰当的。测试用例应该由产品相关的软件测试人员和软件开发人员评审，提交评审意见，然后根据评审意见更新测试用例。如果认真操作这个环节，测试用例中的很多问题都会暴露出来，比如用例设计错误、用例设计遗漏、用例设计冗余、用例设计不充分等等；如果同行评审不充分，那么，在测试执行的过程中，上述本应在评审阶段发现的测试用例相关问题，会给测试执行带来大麻烦，甚至导致测试执行挂起。

5. 定义测试用例的执行顺序

在测试用例执行过程中，你会发现每个测试用例都对测试环境有特殊的要求，或者对测试环境有特殊的影响。因此，定义测试用例的执行顺序，对测试的执行效率影响非常大。比如某些异常测试用例会导致服务器频繁重新启动，服务器的每次重新启动都会消耗大量的时间，导致这部分测试用例执行也消耗很多的时间。那么在编排测试用例执行顺序的时候，应该考虑把这部分测试用例放在最后执行，如果在测试进度很紧张的情况下，如果优先执行这部分消耗时间的异常测

试用例，那么在测试执行时间过了大半的时候，测试用例执行的进度依然是缓慢的，这会影响到测试人员的心情，进而导致匆忙地测试后面的测试用例，这样测试用例的漏测、误测就不可避免，严重影响了软件测试效果和进度。因而，合理地定义测试用例的执行顺序是很有必要的。

四、测试用例执行

测试用例设计完毕后，接下来的工作是测试执行，测试执行中应该注意以下几个问题：

1. 搭建软件测试环境，执行测试用例

测试用例执行过程中，搭建测试环境是第一步。一般来说，软件产品提交测试后，开发人员应该提交一份产品安装指导书，在指导书中详细指明软件产品运行的软硬件环境，比如要求操作系统系统是 Windows 2000 pack4 版本，数据库是 Sql Server 2000 等等，此外，应该给出被测试软件产品的详细安装指导书，包括安装的操作步骤、相关配置文件的配置方法等等。对于复杂的软件产品，尤其是软件项目，如果没有安装指导书作为参考，在搭建测试环境过程中会遇到种种问题。

如果开发人员拒绝提供相关的安装指导书，搭建测试中遇到问题的时候，测试人员可以要求开发人员协助，这时候，一定要把开发人员解决问题的方法记录下来，避免同样的问题再次请教开发人员，这样会招致开发人员的反感，也降低了开发人员对测试人员的认可程度。

2. 测试执行过程应注意的问题

测试环境搭建之后，根据定义的测试用例执行顺序，逐个执行测试用例。在测试执行中需要注意以下几个问题：

全方位的观察测试用例执行结果：测试执行过程中，当测试的实际输出结果与测试用例中的预期输出结果一致的时候，是否可以认为测试用例执行成功了？答案是否定的，即便实际测试结果与测试的预期结果一致，也要查看软件产品的操作日志、系统运行日志和系统资源使用情况，来判断测试用例是否执行成功了。全方位观察软件产品的输出可以发现很多隐蔽的问题。以前，我在测试嵌入式系统软件的

时候，执行某测试用例后，测试用例的实际输出与预期输出完全一致，不过在查询 CPU 占用率地时候，发现 CPU 占用率高达 90%，后来经过分析，软件运行的时候启动了若干个 1ms 的定时器，大量的消耗的 CPU 资源，后来通过把定时器调整到 10ms，CPU 的占用率降为 7%。如果观察点单一，这个严重消耗资源的问题就无从发现了。

加强测试过程记录：测试执行过程中，一定要加强测试过程记录。如果测试执行步骤与测试用例中描述的有差异，一定要记录下来，作为日后更新测试用例的依据；如果软件产品提供了日志功能，比如有软件运行日志、用户操作日志，一定在每个测试用例执行后记录相关的日志文件，作为测试过程记录，一旦日后发现问题，开发人员可以通过这些测试记录方便的定位问题。而不用测试人员重新搭建测试环境，为开发人员重现问题。

及时确认发现的问题：测试执行过程中，如果确认发现了软件的缺陷，那么可以毫不犹豫的提交问题报告单。如果发现了可疑问题，又无法定位是否为软件缺陷，那么一定要保留现场，然后知会相关开发人员到现场定位问题。如果开发人员在短时间内可以确认是否为软件缺陷，测试人员给予配合；如果开发人员定位问题需要花费很长的时间，测试人员千万不要因此耽误自己宝贵的测试执行时间，可以让开发人员记录重新问题的测试环境配置，然后，回到自己的开发环境上重现问题，继续定位问题。

与开发人员良好的沟通：测试执行过程中，当你提交了问题报告单，可能被开发人员无情驳回，拒绝修改。这时候，只能对开发人员晓之以理，做到有理、有据，有说服力。首先，要定义软件缺陷的标准原则，这个原则应该是开发人员和测试人员都认可的，如果没有共同认可的原则，那么开发人员与测试人员对问题的争执就不可避免了。此外，测试人员打算说服开发人员之前，考虑是否能够先说服自己，在保证可以说服自己的前提下，再开始与开发人员交流。

3. 及时更新测试用例

测试执行过程中，应该注意及时更新测试用例。往往在测试执行过程中，才发现遗漏了一些测试用例，这时候应该及时的补充；往往也会发现有些测试用例在具体的执行过程中根本无法操作，这时候应该删除这部分用例；也会发现若干个冗余的测试用例完全可以由某一

个测试用例替代，那么删除冗余的测试用例。

总之，测试执行的过程中及时地更新测试用例是很好的习惯。不要打算在测试执行结束后，统一更新测试用例，如果这样，往往会遗漏很多本应该更新的测试用例。

4. 提交一份优秀的问题报告单

软件测试提交的问题报告单和测试日报一样，都是软件测试人员的工作输出，是测试人员绩效的集中体现。因此，提交一份优秀的问题报告单是很重要的。软件测试报告单最关键的域就是“问题描述”，这是开发人员重现问题，定位问题的依据。问题描述应该包括以下几部分内容：软件配置、硬件配置、测试用例输入、操作步骤、输出、当时输出设备的相关输出信息和相关的日志等。

软件配置：包括操作系统类型版本和补丁版本、当前被测试软件的版本和补丁版本、相关支撑软件，比如数据库软件的版本和补丁版本等。

硬件配置：计算机的配置情况，主要包括 CPU、内存和硬盘的相关参数，其它硬件参数根据测试用例的实际情况添加。如果测试中使用网络，那么网络的组网情况，网络的容量、流量等情况。硬件配置情况与被测试产品类型密切相关，需要根据当时的情况，准确翔实的记录硬件配置情况。

测试用例输入\操作步骤\输出：这部分内容可以根据测试用例的描述和测试用例的实际执行情况如实填写。

输出设备的相关输出信息：输出设备包括计算机显示器、打印机、磁带等等输出设备，如果是显示器可以采用抓屏的方式获取当时的截图，其他的输出设备可以采用其它方法获取相关的输出，在问题报告单中提供描述。

日志信息：规范的软件产品都会提供软件的运行日志和用户、管理员的操作日志，测试人员应该把测试用例执行后的软件产品运行日志和操作日志作为附件，提交到问题报告单中。

根据被测试软件产品的不同，需要在“问题描述”中增加相应的描述内容，这需要具体问题具体分析。

五、测试结果分析

软件测试执行结束后，测试活动还没有结束。测试结果分析是必不可少的重要环节，“编筐编篓，全在收口”，测试结果的分析对下一轮测试工作的开展有很大的借鉴意义。前面的“测试准备工作”中，建议测试人员走读缺陷跟踪库，查阅其他测试人员发现的软件缺陷。测试结束后，也应该分析自己发现的软件缺陷，对发现的缺陷分类，你会发现自己提交的问题只有固定的几个类别；然后，再把一起完成测试执行工作的其他测试人员发现的问题也汇总起来，你会发现，你所提交问题的类别与他们有差异。这很正常，人的思维是有局限性，在测试的过程中，每个测试人员都有自己思考问题的盲区和测试执行的盲区，有效的自我分析和分析其他测试人员，你会发现自己的盲区，有针对性的分析盲区，必定会在下一轮测试用避免盲区。

总结：

限于文章的篇幅，本文不可能给出一个类似于 checklist 的指导性的软件测试新手入门。无论从事软件测试还是从事其它的工作，技术上的和技巧上的问题都可以通过查询相关的软件测试技术书籍获取，掌握一套基本的方法论是最重要的。以上文字，都是作者从事软件测试工作积累的经验之谈，如发现谬误之处请不吝指出。

如何做好单元测试

周　峰

Zhoufeng@withub.org

【摘要】单元测试是软件开发过程中重要的质量保证活动，单元测试的质量将很大程度上影响软件产品的最终质量。本文从组织、流程和技术三个方面来阐述了做好单元测试的一些关键因素，可以作为软件企业开展单元测试活动的参考。

【关键字】单元测试，组织，流程，技术

前言

单元测试是对软件基本组成单元进行的测试，是属于白盒测试的范畴，它主要通过对代码的逻辑结构进行分析来设计测试用例。在动态测试手段中，单元测试是一种非常高效的测试方法，并且是软件测试周期中第一个进行的测试。从成本角度考虑，缺陷发现越早越好，加强单元测试力度有利于降低缺陷定位和修复难度，从而降低缺陷解决成本，同时加强单元测试也减轻了后续集成测试和系统测试的负担。根据业界的统计，一个BUG在单元测试阶段发现花费是1的话，到集成测试就变为10，到系统测试就高达100，到实际推向市场量产后就高达1000。但单元测试在目前国内软件企业中开展得并不好，一方面是由于对单元测试重视程度不够，测试投入不足，另一方面是由于在单元测试实践方面积累得也不够，单元测试处于一种摸索状态。

软件的质量由组织、流程和技术三个维度来决定，任何一个维度都不能单独决定软件的质量。好的组织结构可以保证流程的顺利实施，好的流程能提高软件开发的规范性和可控性，从而提高软件开发的效率和质量，而采用了好的技术和有好的技术的载体——人，则从根本上保证了软件的质量。

总而言之，组织、流程和技术是软件质量三角，本文将从这三个方面对如何做好单元测试进行论述。

一、 组织结构应该保证测试组参与单元测试

目前无论是工业界还是学术界都认为单元测试应该由开发人员开展，这是因为从单元测试的过程看，单元测试普遍采用白盒测试的

方法，离不开深入被测对象的代码，同时还需要构造驱动模块、桩函数，因此开展单元测试需要较好的开发知识。从人员的知识结构、对代码的熟悉程度考虑，开发人员具有一定的优势。

单元测试由开发人员进行能带来一些特别的收益。我们知道，在实践中开发人员进行单元测试一般推荐采用交叉测试的方法，例如由被测单元的调用方进行该单元的测试，即尽量避免对自己的代码进行单元测试。这种交叉的测试安排可以避免测试受开发思路影响太大，局限于原来的思路不容易发现开发过程中制造的问题；二来也达到一个技术备份或充分交流的目的，这对组织非常有利。即使不采用交叉测试的方法，而安排单元的生产者自行开展单元测试，也是有很大的优越性的，其最大的优点是快速，且能更好的实现“预防错误”。在人员紧张的情况下这种自行测试的安排也是不错的选择。

从经验值来看，单元测试投入和编码投入相比基本上是一比一，如果由专职测试队伍来进行单元测试，维持这样庞大的单一任务队伍显然是不合适的。

以上谈的是由开发人员进行单元测试的优点，其中主要是从单元测试的效率角度来考虑。但是从单元测试效果的角度考虑，必须从组织结构上保证测试组参与单元测试，这是因为：

首先，从目前国内企业普遍现状来看，测试人员质量意识要高于开发人员，测试人员参与单元测试能够提高测试质量。

其次，对被测系统越了解，测试才有可能越深入，测试人员参与单元测试，将使得测试人员能够从代码级熟悉被测系统，这对测试组后期集成测试和系统测试活动非常有帮助，会很大的提升集成测试和系统测试质量。

测试组以何种方式参与单元测试，应该结合软件组织的实际情况来定。如果软件组织测试资源充分，测试人员对开发人员的比例较高，那么可以由测试人员独立承担部分重要模块的单元测试工作；如果测试资源不足，测试人员对开发人员的比例较低，那么可以采取由测试人员进行单元测试计划、单元测试设计的工作，而单元测试的实现和执行由开发人员来完成；而如果测试资源非常缺乏，连单元测试计划、单元测试设计都无法承担，那么测试组至少应该参与开发组的各相关单元测试文档、单元测试报告的评审，保证单元测试的质量。

二、 加强单元测试流程规范性

1. 制订单元测试的过程定义

软件质量的提高需要规范的流程，对软件开发过程进行管理也需要依据规范的过程定义。过程定义包含阶段的划分、阶段的入口/出口准则、阶段的输入/输出、角色和职责、模板和查检表等等。将单元测试划分为几个阶段便于对单元测试过程进行控制，体现软件测试可控性。要提高单元测试的质量，首先要制定规范的单元测试过程，开发组、测试组、SCM组、SQA组等可以依据单元测试过程定义开展各自的工作，共同保证单元测试的质量。

单元测试过程的定义需要参照企业的实际情况，例如阶段划分可以分为四个阶段：计划、设计、实现、执行。其中计划阶段应当考虑整个单元测试过程的时间表，工作量，任务的划分情况，人员和资源的安排情况，需要的测试工具和测试方法，单元测试结束的标准及验收的标准等，同时还应当考虑可能存在的风险，以及针对这些风险的具体处理办法，并输出《单元测试计划》文档，作为整个单元测试过程的指导。设计阶段需要具体考虑对哪些单元进行测试，被测单元之间的关系以及同其他模块之间单元的关系，具体测试的策略采用哪一种、如何进行单元测试用例的设计、如何进行单元测试代码设计、采用何种工具等，并输出《单元测试方案》文档，用来指导具体的单元测试操作。实现阶段需要完成单元测试用例设计、脚本的编写，测试驱动模块的编写，测试桩模块的编写工作，输出《单元测试用例》文档、相关测试代码。执行阶段的主要工作是搭建单元测试环境，执行测试脚本，记录测试结果，如果发现错误，开发人员需要负责错误的修改，同时进行回归测试，该阶段结束需要提交《单元测试报告》。

具体进行单元测试过程定义的时候，可以进行一定的裁减，例如可以裁减为设计和执行两个阶段，将《单元测试方案》和《单元测试用例》合二为一。

2. 单元测试工作产品必须纳入配置管理

单元测试工作产品指单元测试完成后应交付的测试文档、测试代码及测试工具等，一般包括但不限于如下工作产品，可以根据实际情况进行适当裁剪：

- 单元测试计划

- 单元测试方案
- 单元测试用例
- 单元测试规程
- 单元测试日报
- 单元测试问题单
- 单元测试报告
- 单元测试输入及输出数据
- 单元测试工具
- 单元测试代码及设计文档

为了保证单元测试工作产品的准确性，需要对测试代码和脚本进行走读或检视，对测试文档进行评审。这些工作产品应该纳入到配置管理，对于其修改要走配置变更流程，并及时发布其配置状态，这样可以保持单元测试工作产品的一致性和可回溯性。

3. 必须制订覆盖率指标和质量目标来指导和验收单元测试

单元测试必须制订一定的覆盖率指标和质量目标，来指导单元测试设计和执行，同时作为单元测试验收的标准。设计用例时，可针对要达到的覆盖率指标来设计用例，而在测试执行时，可以依据覆盖率分析工具分析测试是否达到了覆盖率指标，如果没达到，需要分析哪些部分没有覆盖到，从而补充用例来达到覆盖率指标。而单元测试质量目标的制订，需要符合软件企业的实际过程能力，这依赖于软件企业以前单元测试过程度量数据的积累，不能凭空制造出来。有了以前度量数据的积累，完全可以了解当前组织的单元测试能力，例如单元测试每千行代码发现的缺陷数是多少。如果单元测试统计结果没有落到这个质量目标范围内，说明单元测试过程中某些方面存在一些问题，需要对过程进行审计后找出问题原因进行改进。

这些指标确定下来后，一定要严格推行。会有一些测试人员找出各种理由证明覆盖率指标达不到等等，这需要QA根据实际情况分析指标是否合理。实际证明有一个相对简单的标准也比没有标准要好得多，我们的实践发现，通过推行硬性指标，单元测试发现的问题数目比没有标准前至少增加了2倍。

下面是印度SASKEN公司的质量目标：

印度SASKEN公司质量目标

阶段	组织目标	目标上限	目标下限
HLD （概要设计）	50 Major Defects / 100 pages	55 Major Defects/100 pages	45 Major Defects /100 pages
LLD （详细设计）	40 Major Defects / 100 pages	44 Major Defects/100 pages	36 Major Defects / 100 pages
Unit Test Plan （单元测试计划）	25 Major Defects / 100 pages	27.5 Major Defects /100 pages	22.5 Major Defects / 100 pages
Code Review （代码走读）	**20 Major Defects / KLOC**	**22 Major Defects / KLOC**	**18 Major Defects / KLOC**
Defects during Unit test （单元测试）	**15 Major Defects / KLOC**	**16.5 Major Defects / KLOC**	**13.5 Major Defects / KLOC**
Defects during Integration test （集成测试）	6 Major Defects / KLOC	6.6 Major Defects / KLOC	5.4 Major Defects / KLOC

4. 加强详细设计文档评审

详细设计是单元测试的主要输入，详细设计文档的质量将直接影响到单元测试的质量，所以一定要加强详细设计文档的评审，特别是要写相关测试方案和进行测试用例设计的人员，一定要从写测试用例的角度看这个详设是否符合要求，否则后期进行单元测试设计时会发现无法依据详细设计进行单元测试设计。软件组织可以将详细设计评审的要点以查检表的形式固化下来，这样在详细设计评审的时候依据查检表一项项检查，既提高了评审效率，也能保证评审效果。评审流程需要确定如果不满足查检表n%以上的条件，被评审详细设计文档就不能通过，需要重新设计。

通常详细设计文档有两种形式，一种是流程图的形式，另一种是伪码的形式。用流程图表达的优点是直观，利于单元测试用例设计，缺点是描述性比较差，文档写作麻烦，不利于文档的变更和修改；伪码的方式可能正好相反，文档变更修改简单，可以方便地在任何地方增加文字说明，而且翻译成代码更加便捷，但不直观，不利于进行单元测试用例设计。

详细设计和单元测试设计一定要分离。如果单元测试由测试人员承担，这一点不会有什么问题；如果单元测试由开发人员承担，那么实际操作时可以让项目组内做相同或者相近任务的成员相互交换，根据对方的详设设计对方的单元测试。这样在单元测试开始之前的详细设计评审阶段就要考虑到后面的分工，安排相关的单元测试设计人员参与相关详细设计的评审。

如果代码没有对应的经过评审后的详细设计文档，建议不进行单元测试，而是用代码审查替代单元测试。

开发人员在编码的过程中，可能会发现详设中的问题，并对代码进行修改，这种修改应该回溯到详设，并对详设进行相应的修改，否则到单元测试执行的时候，会发现代码和详设根本对不上，无法执行下去。详设的修改要受控制，要走变更控制流程，它的变更也要经过评审。因为单元测试是详细设计的下游活动，如果详细设计随意更改，单元测试文档很难和其保持一致，这样单元测试也就失去了依据和意义。只有详设也纳入配置管理，才能保证单元测试和详设的一致性。

三、 单元测试者技能的提高

1．加强对单元测试人员的技能培训

单元测试的质量很大程度上决定于进行单元测试的人的技术水平。如果测试者不具备单元测试的知识，那么应该对测试者进行相关的培训。一个没有做过单元测试人，不经过培训初次是很难做好单元测试的。单元测试在详设阶段结束时开始，但是单元测试相关培训应该尽早准备和计划，培训可以分两个阶段，每个阶段的内容类似。第一阶段是写单元测试方案前，培训对象为测试方案的写作者和详设的写作者，这样可以在设计时多考虑可测试性，培训的内容为单元测试基本概念、单元测试分析方法、单元测试用例的写作、单元测试标准的明确；第二阶段为单元测试执行前，对象为测试执行者，培训内容为具体单元测试的执行，包括驱动函数、桩函数的构造、覆盖率测试工具的使用（TrueCoverage、Logiscope等）、利用自动化单元测试框架构造单元测试自动化（TCL、CppUnit、JUnit等）。培训过程中最好结合实例穿插其中，会比较生动，而且增强理解。

通过以上的系统培训，可以统一单元测试方法、明确单元测试的标准、掌握单元测试基本技能，为后期单元测试的顺利开展扫平道路。

2．必须引入工具进行辅助

单元测试非常需要工具的帮助，特别是覆盖率工具不能缺少，否则用例执行后无法得到测试质量如语句覆盖、路径覆盖等情况，也就无法对被测对象进行进一步的分析。应用较广的分析覆盖率的工具有Logiscope、TrueCoverage、PureCoverage等，它们的功能有强有弱，可以根据实际情况采用。

为了提高单元测试的效率，特别是提高进行回归测试时的效率，需要在单元测试中引入自动化。目前常用的方法是采用TCL语言编写扩展指令，构造自己的单元测试自动化。也可以直接采用开源的自动化测试框架如CppUnit、JUnit等。

此外，在单元测试之前，还需要利用PC_Lint对被测代码进行检查，排除代码语法错误，确保进行单元测试的代码已经具备了基本质量，保证单元测试能够顺利进行，提高单元测试执行效率。

3．单元测试者加强对被测软件的全面了解

单元测试的目的除了要发现编码中引入的错误和发现代码与详细设计不一致的地方之外，还有一个目的是为了保证详细设计的质量。因为测试分析和测试用例设计需要依据详细设计来进行，这个过程实际上是对详细设计的重新检视，在这个过程中会发现以前评审中没有发现的问题。

无论是在单元测试的设计活动中还是在单元测试的执行过程中，都需要测试者了解软件的需求和概要，加强对被测软件的全面了解。否则对被测对象了解不深，只能就被测单元的流程而测流程，而对于该流程是否正确就无法保证了。

测试者要注重与开发的交流，这样能对被测单元有更深的了解；同时因为进度的原因，包括详设在内的文档往往来不及更新，所以最新最正确的思想往往存在于开发人员的脑袋里，及时与他们交流才会获得最及时的信息，减少将来更新用例的工作量。

结尾

单元测试是软件开发过程中非常重要的质量保证手段，加强单元测试对提高软件质量具有非常重要的意义。而做好单元测试不是只要掌握单元测试方法就可以了的，这需要从组织、流程和技术三个方面来保证。

为盈利而测试

—谈软件企业进行软件测试的经济目的

郑　岷[1]
Minz32@hotmail.com

【摘要】测试经济学问题和心理学问题是至今仍需业界所关注的一个中心问题之一。自[美] Grenford J. Myers 在 1978 年所著的《The Art of Software Testing》一书中提出之后已有年了，而且软件已成为当今世界的支柱产业，但诸多的软件企业，特别是高层经理、质量主管仍受传统的测试观念所束缚，或把测试看成是一种“软”任务，少点晚点、多点少点不重要，或不愿意对测试进行大的投入和委托第三方进行测试，或不重测试队伍的建设，或单纯地以捕获的缺陷数作为对测试人员进行业绩考核指标，或还是靠高层主管一支笔、一句话来定软件产品发布或交付的时间，…。这些大大影响了这些软件企业自身的竞争力和企业赢利的成长。

所以，作者在本文中着重从测试经济学的视角，以过程建模的方法，给出了业界普遍存在的六个误区，一个软件“变换”过程缺陷导入模型，一个测试和纠错的成本模型和软件测试成本最小化停止条件。

【关键字】软件测试；测试经济学；测试管理；测试停止条件

引言：

1作者简介：

作者师从于 James A. Whittaker (《实用软件测试指南 How to Break Software》的作者)。于 1997 年 12 月取得佛罗里达理工学院（Florida Institute of Technology）计算机科学硕士学位，2000 年完成博士学位的课程学习，进入微软工作，研究方向为软件测试。

工作单位：
COM+ Legacy Test Group, Microsoft Corporate.
Redmond, WA 98052 USA

邮政地址：
9732 227th Way NE
Redmond WA 98053

[美] Grenford J. Myers 在 1978 年所著的《The Art of Software Testing》一书中，首先用一章的篇幅论述了程序测试的心理学和经济学问题。他认为要淡软件测试问题，“最为重要的却是经济学问题和心理学问题”。并在列举了当时大多数人使用的完全错误的“测试”定义所导致的危害之后，给出了被业界普遍公认和引用的“较为恰当的定义”，即“程序测试是为了发现错误而执行程序的过程”。

Grenford J. Myers 所以把测试经济学问题和心理学问题看得如此之重，因为软件测试至今仍是由人为主体的一系列活动，而人类的活动却具有高度的目的性，即可以说绝大多数的人类行为是由“目标驱动”的。一个正确的测试目的，可引发一系列为实现该目的的有效测试行为和结果；相反，一个不正确的测试目的，可引发一系列无效的或有害的测试行为和结果。

一、什么是软件测试

在谈到软件测试时，业界还是都引用 Grenford J. Myers 在《The Art of Software Testing》一书中表述的观点：

① 程序测试是为了发现错误而执行程序的过程；

② 测试是为了证明程序有错，而不是证明程序无错误；

③ 一个好的测试用例是在于它能发现至今未发现的错误；

④ 一个成功的测试是发现了至今未发现的错误的测试。

但是软件测试 决不等同于找 BUG。不幸的是，仅凭字面意思理解这些观点，认为发现错误似乎是软件测试的唯一目的，这样就可能会产生一些误导。认为查找不出错误的测试就是没有价值了；一些软件企业在制定对测试组（人员）的业绩考核标准时，简单地把捕捉到缺陷个数作为主要考核指标。然而事实并非如此简单。那些把寻找缺陷当作是软件测试的唯一目的的测试管理人员往往会陷入一个又一个测试管理误区。

以下列出是作者总结的六个值得业界重视的误区。

误区一：忽视对正常输入的测试。

软件缺陷最常见的地方是哪里？一般来说软件缺陷最为密集的地方就是当用户输入非正常输入组合时。一方面软件设计人员较容易忽略一些极难发生的情况，另一方面设计与开发人员往往会把更多的精力用在功能实现上，容错与错误处理往往是开发上的薄弱环节。所

以当测试人员将测试目标订在缺陷数量上时，重点测试非正常输入显然就是最好的手段。

然而对于用户来说，真正影响使用的却是正常输入组合激发的缺陷。而这种缺陷，一个就已经太多了。

例如对于一个记事本软件，也许没有人会去注意为什么将字体大小设置到大于 49151 的正整数时，每个字会突然变成一个小点。也许很少人会去在乎若将字体大小设置到超过窗口高度时右方的滚动卷标不能滚动到一个能让你看到某一个字的下半截的位置。但若使用一般的剪贴功能会将一段话搞成乱码，那每一个用户就都会跳起来了。

误区二：忽视设计阶段的参与与评估

在多数企业里，软件的设计阶段往往没有软件测试人员的参与，甚至有些企业在编码几乎完成了才开始组织测试团队。原因无他，设计还没定型呢，怎么找缺陷哪？

事实上设计上的缺陷往往是耗用成本最高，也是最难在开发后期修复的缺陷。而一个软件的质量与它有多大的设计缺陷有着密不可分的联系。而有经验的测试人员的质量意识，安全意识，对用户需求的了解及分析能力，对于打造高品质的软件设计都有着不可忽视的作用。

同时对于技术可行性与用户需求之间的设计妥协，测试人员也必须要有充分的评估和理解。这样在最紧张的开发后期，才不会把大量的（测试及开发人员的）宝贵时间用于记录和处理那些已在设计阶段确定放弃的功能。

误区三：忽视测试计划与测试文档的建立及维护。

设计测试计划与建立测试文档常常会被看作是浪费时间。即使做了也只是为了交差，要么敷衍了事，要么找个范本一抄，略作修改便万事大吉。真正测试起来还是全靠灵感与直觉。

但软件测试的另一个重要目的是确认及评估一个软件是否能够符合用户的需求。而对软件质量的保证不能只是靠测试人员一开金口，而要靠详细的测试计划以及完整的测试结果来取得用户的信任。

误区四：忽视缺陷的分析，报告及跟踪。

找到缺陷本身并不能提高软件的质量。只有缺陷被正确修复了才

能真正提高软件的质量。作为一个好的测试人员不仅仅要善于发现缺陷，同时还要善于描述缺陷的表现特征，并积极跟踪缺陷的处理过程，确保缺陷被正确诊断及修复。不负责任的缺陷报告常常会给软件开发人员带来很多不必要的麻烦甚至误导。这会严重影响开发人员对测试人员的信任与尊重。

误区五：错误的测试目标及测试终止条件。

没有缺陷的软件是不存在的。同时也没有方法可以知道一个软件中到底有多少个缺陷。一个简单的文本处理软件经过三个月的测试找到了100个缺陷，那说明这个软件总共有多少缺陷呢？也许是101个，你找到了99%的缺陷，也许是10000个你只找到1%的缺陷。又或连着两个星期只找到一两个缺陷，那么说是软件质量已达到较高水平了呢，还是你的测试方法走进了死胡同呢？谁也说不清。

误区六：不懂得合理调配使用测试人员的知识技能结构。

除去发现缺陷的洞察力外，好的测试人员还应该拥有很好的组织能力，准确的表达能力，严密的逻辑推理能力，以及对软件开发过程，软件使用对象，软件开发技术等的较深入地了解。同时具备所有素质的测试人员也许比凤毛麟角还要难找。所以在构建一个测试团队的时候就一定要合理搭配拥有各种能力的测试人员，并在使用的时候注意发挥各自的特长。如果让所有测试人员一心一意地找 BUG，那不仅仅是压制了其他方面的人才，整个测试团队的实际绩效必然会大打折扣，甚至会影响到整个软件产品的顺利开发。

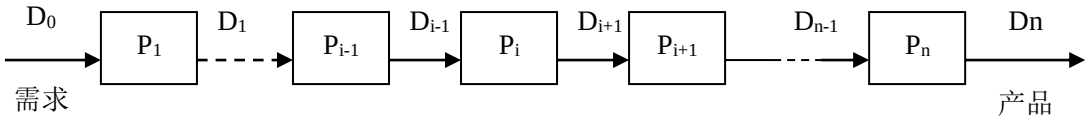
二、软件质量与软件测试

为了定量地论述软件测试的经济目的，本节首先剖析一下软件质量与软件测试的关系。随着人们对软件特性、软件测试与软件质量的认识的深化，尤其是对质量认识的深化，任何一个软件企业要使自己提供给用户的产品达到并保持稳定的质量水平，不仅要关注对软件产品在发布或交付之前进行严格的软件测试，而要从软件需求获取开始，对整个软件开发过程都要引入严格的质量管理，在每个开发阶段均要引入软件测试，并且这种质量管理不能停留在早期的质量检验型管理，而要转为全面的质量管理和第三方的质量认证或评估。

显见，在 1983 年 IEEE 的软件测试定义“使用人工或自动手段来

运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。”中只是明确指出：1)软件测试是运行或测定某个系统的过程；2)软件测试的目的是为了检验软件系统是否满足需求。把测试视作检验的一种手段和活动。

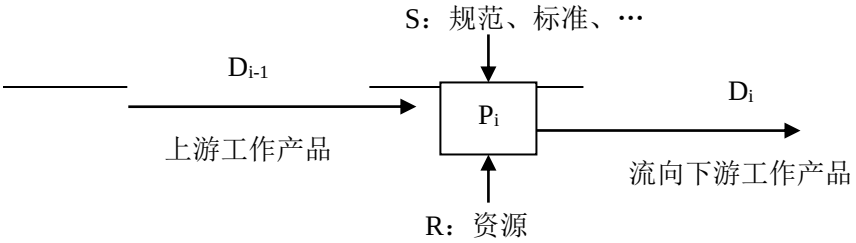
实际上，业界很早就从早期的狭义的软件测试概念和实施范围作了拓展。软件测试的对象不再限于可运行的程序代码，而拓展到质量管理中所涉及的在整个开发各阶段的输入、输出、过程和中间工作产品，测试活动也不限于对可运行代码的测试活动，而包括复查、走查、评审、验证等。事实上，软件整个开发过程，是一系列 “变换”或“处理和活动”过程 P1、P2、…、Pn 所组成的半序序列(参见图—1)。



图－1

图－1 中 D0 是源头（用户需求），D1、…、Dn-1 是诸中间“变换”过程的工作产品，Dn 为最终工作产品。显见，每个“变换”过程都有它的输入和输出，都可能导入这样或哪样的缺陷（Bugs），而且这种缺陷按缺陷扩大模型还会逐级扩大，尤其是在上游工作产品频繁出现更改时。不过这些中间工作产品大多只是以静态文档形态给出的。对它们的测试通常也只能采用 “静态测试”方法，其中包括工程测试（非正式的内部评审）、正式测试（正式评审）、审核测试（验证、审计）和检查性测试（检测、确认）等。

为此，为了保证最终工作产品的质量，必须关注对每一个中间“变换”过程以及它们输入和输出的质量的检测和评估，尽可能防止缺陷向下游传布和扩散。以下再通过一个“变换”过程来分析一下每个“变换”过程可能导入的质量隐患：



图－2

显见，在每一个“变换”过程中导入缺陷途径是多种多样的，例如：

上游工作产品中隐含的缺陷；

规范、标准、…中隐含的缺陷；

资源、工具引入的缺陷，其中主要是由“变换”过程的操作者（组、个人）引入的缺陷，因为据大量统计资料统计，即使一个有经验的程序员，在编码过程中，平均每写 7-10 行程序就会有引入 1 个缺陷。事实上，对每一过程的操作者，导入缺陷有诸多的客观的和主观的原因，其中包括对上游工作产品、规范、标准、…以及引用外部构件的理解上的差异，操作者对知识、经验和技能方面的欠缺和习惯性、过失性错误，以及团队沟通和版本管理、工具使用中差错和过失。

由上可知，对“变换”过程 Pi 流向下游工作产品 Di 中所导入缺陷个数可用以下公式表示：

$$N_i=k_iN_{i-1}+NS_i+NR_i \quad \ldots\ldots \quad (1)$$

其中

N_i - 是“变换”过程 P_i 的工作产品 D_i 中所隐含缺陷数；

k_i - 是“变换”过程 P_i 对上游工作产品 D_{i-1} 中所隐含缺陷数的扩大系数 k_i，通常 k_i 在 2～10 之间；

NS_i– 是由于“变换”过程 P_i 的规范、标准、…缺陷所引入的缺陷数

NR_i – 是由于“变换”过程 P_i 所用资源、工具、技术所引入的缺陷数，显见它与参与本“变换”过程 P_i 的人员特性及状态有极大的相关性。

这样就可从公式(1)导得，

$$N_1=k_1N_0+NS_1+NR_1$$

$$N_2=k_2N_1+NS_2+NR_2$$

……

$$N_n=k_nN_{n-1}+NS_n+NR_n$$

最终产品 Dn 中所包含的缺陷个数为

$$N_n= k_1k_2\ldots k_nN_0+k_2k_3\ldots k_nNS_1+k_2k_3\ldots k_nNR_1+\ldots+NS_n+NR_n \quad \ldots \quad (2)$$

显见，当 n=5、k1=k2=k3=k4=k5=3 时，在需求阶段引入了一个

缺陷，而且未能尽早于以排除，那未最终经逐级扩大在最终产品中可产生 35 = 243 个缺陷。

可是，一旦在某个环节导入缺陷后，有些是难以被显露的，即使通过软件测试被外显，被发现，有的也难以捕获到它。只有捕获到它，才能设法纠正它。不幸的是，有时即使找到了，而且也采取了补救措施，结果由于不良的补救措施又会引发新的缺陷。就象人从向上的自动扶梯一步一步往下走一样，可能你往下走了一步（排除了一个缺陷），结果自动扶梯又往上滚动好几步（又引发了多个缺陷），这样一来，推倒重来的事是常有的。

现设在 Di 中每发现和修复一个缺陷的平均成本分别是 CTi 和 CRi，如果诸缺陷都留到最终产品 Dn 才进行测试和排除，那末总的发现和修复成本为：

$$C_n=(k_1k_2\ldots k_nN_0+k_2k_3\ldots k_nNS_1+k_2k_3\ldots k_nNR_1+\ldots+NS_n+NR_n)\times(CT_n+CR_n) \quad (3)$$

所以，随着软件产品规模、复杂性增长，加上在启动项目时，一些需求往往是模糊的。一般的软件组织要化一半以上的精力来查找和修复错误。加上测试时间往往难以预计，没完没了的产品缺陷常常是造成超支和超期的主要原因。

可是对最终用户来说最关心的当然还是最终软件产品的功能、可靠性、效率、易于使用、易于移植性。而对一个实际软件项目企业高层管理者说，他们自然更多注重总的质量和项目赢利，而不是某一特性，而要权衡质量（Q）、成本（C）、进度（T）。所以，对一个项目经理、软件工程组以及测试组来说，同样要重视软件测试的质量效益。

三、软件测试的经济目的

事实上，作为软件开发企业来说，投入人力，资金搞软件测试的最终目的还是离不开经济效益。而对与测试项目的管理也不能离开这个大前提。软件测试的经济效益主要来自以下两个方面。一是满足用户需求，提高产品的竞争力，最终提高产品的销售量。二是尽早发现缺陷，降低售后服务成本。而软件测试的最终目的就是使它带来的经济效益最大化。

1. 满足用户需求，提高产品的竞争力，最终提高产品的销售量。

一个学生编写程序可能是为了学习语言，算法，技巧等知识。一个科研人员编写程序可能是为了验证某个理论，计算某个结果，或实

现某个算法。而一个软件企业生产一个软件则只有一个目的 – 盈利。同其他所有企业一样，要盈利，首先要把用户当作上帝。只有满足了用户的需求，并能使用户用的方便、用的放心，用户才会心甘情愿地掏钱。所以作为软件品质的最后把关者，软件测试要把用户的想法放在第一位。从设计测试计划到构造测试用例，从实际进行测试到跟踪缺陷处理，都要从用户的角度去考虑每一个问题。

首先，用户花钱购买软件实际上是购买软件的功能。所以软件测试首先要保证用户所需的功能都能正确而可靠地实现。在设计测试计划时应从用户实际可能使用本软件的基本流程或典型使用脚本出发，分析各个功能的重要性和相关性，再根据现有资源及时间构造和筛选测试用例。

其次，要想保住老用户、吸引新用户，产品的质量是关键。对于一个传统工业企业来说，一个产品的质量也许可以用一连串的行业指标来界定；而对软件企业来说，一个软件的质量是完全由用户来评价的。用户对软件质量的评价通常包含下列五个方面：

- 可靠性
- 安全性
- 性能
- 易用性
- 外观

因而软件测试也要针对这几个方面构造相应的测试用例。

2. 尽早发现缺陷，降低后继质量成本

那么是不是说找缺陷就不重要了呢？当然不是。软件测试的另一个经济目标是尽早发现缺陷，降低修复及售后服务成本。显然，每一个已发布产品中的缺陷除了会影响产品及企业的声誉外，还会直接增加产品的售后服务成本。无论是派人到现场调试，或研发、发布补丁程序都要远比在发布前的修复成本昂贵数十倍，甚至数百倍。

事实上，许多统计资料表明，开发过程每前进一步，发现和修复一个缺陷的平均成本要提高 10 倍。在代码复查阶段，平均 1-2 分种能发现和修复一个缺陷，在初始测试阶段要 10-20 分钟。在集成测试时要花费 1 个小时或更多，在系统测试时要花 10-40 个小时。所以软件测试除了可以揭示和评估软件产品的可靠性之外的首要任务，就是要在开发过程中尽可能早地找到可能存在的各种缺陷，并找出最佳的

解决方案。一旦在某一“变换”过程的工作产品成了次品，那末后继的所有“变换”过程的工作产品也永远是一个次品。

现在再来分析上节所导出的成本公式(3)

$$C_n = (k_1k_2\ldots k_nN_0+k_2k_3\ldots k_nNS_1+k_2k_3\ldots k_nNR_1+\ldots+NS_n+NR_n)\times(CT_n+CR_n)$$

再设定不再是把软件测试和修复都留到最后，而是前移到每个“变换”过程 Pi 后，且设定对每个中间工作产品的缺陷捕获率和缺陷修复率分别为 RTi 和 RRi，那末整个软件产品的测试和缺陷修复总成本为：

$$\begin{aligned}CT &= CT_1+CT_2+CT_3+\ldots+CT_n\\ &= N_0\times RT_1(CT_1+ RR_1\times CR_1)\\ &+ (N_0(1-RT_1\times RR_1)+NS_1+NR_1))\times RT_2(CT_2+ RR_2\times CR_2)\\ &+ (N_0(1-RT_1\times RR_1)+NS_1+NR_1))\times((1-RT_2\times RR_2)+NS_3+NR_3))RT_3(CT_3+ RR_3\times CR_3)\\ &+\ldots\ldots\\ &+ (N_0(1-RT_1\times RR_1)+NS_1+NR_1))\times((1-RT_2RR_2)+NS_3+NR_3))\times\ldots\\ &\times(\ (1-RT_{n-1}\times RR_{n-1})+NS_n+NR_n)\ldots))\times RT_n(CT_n+ RR_n\times CR_n) \qquad \ldots\ldots \qquad (4)\end{aligned}$$

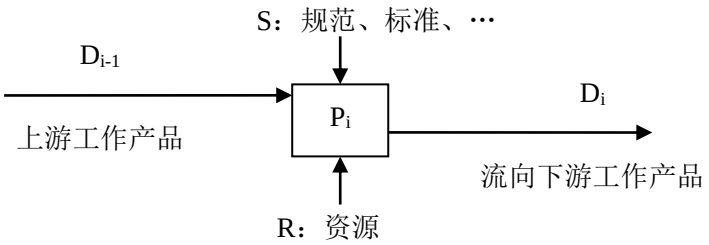
由(4)可知，当

$$N_0=NS_1=NR_1= NS_2=NR_2=\ldots=NS_n=NR_n=0$$

时，有

$$CT=0$$

显见，这就是人们期望的在每个“变换”过程 P_i 中均实现最理想的“零”缺陷目标的结果。但是，这是不可能的，因为至今软件项目涉及的诸多的“变换”过程 P_i 还是人工介入的，尤其是由多人或众多团队介入后，引入缺陷是难免的，所以，控制成本的中心任务之一就是如何预防和减少对每一“变换”过程 P_i 所引入的缺陷数。现在再回到上节中的“变换”过程 P_i 模型--图－2，即：



图—3

首先，在上游工作产品 D_{i-1} 流入“变换”过程 P_i 前，由该过程的操作者共同参与对 D_{i-1} 评估或测试，尽量避免或减少经输入端导入和引发的缺陷数，其中导入的缺陷数是指上游工作产品 D_{i-1} 自身中所含的

缺陷数，而引发的缺陷数是指由于上游工作产品 D_{i-1} 中所含的缺陷或因为该过程的操作者对上游工作产品 D_{i-1} 理解上的差异而引发的。

其次，由于 S：规范、标准、...的缺陷，如无完整、清晰的、可操作的文档化的东西，有些团队甚至只有非文挡化的约定，以至造成该“变换”是一种非稳定的过程，是因人而异的，这就使这种“变换”过程 Pi 所流向下游的工作产品 Di 必定是极不稳定的。

最后，来分析 R：资源对过程质量的影响，其中虽然涉及到该过程所用的外部构件、工具、工作平台等，但这里重点要分析的是该过程的操作者-人。根据[美]Watts S.Humphrey 在他的《Introduction to the Personal Softwere Process》，即 PSP 一书中指出：学生和工程师一般在设计阶段每小时引入 1~3 个缺陷，在编码阶段每小时引入 5~8 个缺陷。在代码复查阶段一般每小时发现 6~12 个缺陷，在测试阶段每小时排除 2~4 个缺陷。而经过 PSP 培训后，所引入的缺陷可减少 50%，缺陷排除效率提高近 4 倍。又指出 Motorila 的工程师在学习了 PSP 后，第一次在测试时只发现 1 个缺陷。

实际上，排除一个缺陷的时间要比测试一个缺陷多 5 到 6 倍。当然，系统越复杂，排除缺陷的费用越高。以 Micrisoft NT 系统为例，共修复了 30000 个缺陷，测试一共花费了 250 个人-年，平均 16 个小时修复 1 个缺陷。

显然测试和排除缺陷不仅要耗费人工、花钱，而且还要拖延时间，所以对缺陷的测试和排除要综合考虑质量（Q）、时间进度（T）与成本（C）。保证工作质量的原则之一是每一次要开发出合格的产品。切忌一开始就匆忙进行设计、编码，首先要有一个好的、合格的需求，然后按需求特性确定一个适用的软件需求开发过程及每个“变换”过程 Pi 应遵循的规范、标准，在选择合适的开发团队、操作人员和其它资源后，还要进行上岗培训，它包括对上游工作产品进行“变换”应遵循的规范、标准及所选择的工具、构件等。随后再启动后继的变换过程及质量监测和控制点，以保证每一个变换过程都能产生更好的、完整的文档。以防止由于不清楚或不完整引入的缺陷。这些都是软件质量保证和测试策划过程必须与以关注的。

例如，以设计过程为例。一个有经验的工程师在进行设计时，经常在各个设计层次之间动态地进行切换：①首先要理解一定抽象层次上是如何工作的，然后才能在高层设计中放心用。②如一个功能过去用过就可不考虑它的细节，否则要进行详细设计，甚至做一个原型加

以验证后，才可放心地回到高层设计。③由于难以把设计阶段和实现阶段严格区分开来，所以要关注的是设计活动，即过程，而且除了要按照支持该过程的规范、标准，把该过程的最终结果完整地、正确地在设计文档中给予详细地表述，即评估、测试及后继的下游过程是可视化的，还要在给出最终设计文档前的活动中适当引入同行评审来过滤掉不良设计思路或碰撞出好的思路，这也是十分有效的。

现在我们再来考虑评估、测试活动的实际效益问题。事实上，软件工作产品中越复杂或所含的缺陷愈少，测试发现缺陷的效率越低，因此测试、修复缺陷的花费越大。加上软件产品中确切的缺陷数是无法知道的，无休止的测试也是不可取的。这就涉及到两个方面的问题–：第一是这些缺陷可能带来的损失到底有多大； 第二是这种缺陷被激发的可能性有多大。如果两个问题的答案都是“极小”，那么就根本不应当列入考虑之中。

四、何时应当停止测试

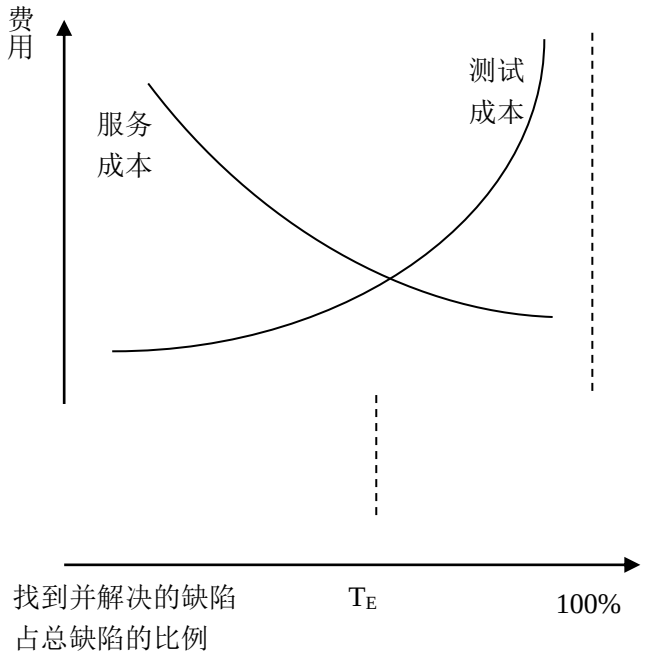
软件测试还有一个很重要的问题 – 何时应当停止测试？对于许多企业来说，终止测试的时间往往是随产品的预定发布时间或交付时间来定的。甚至有的地方产品发布或交付前一周还在写新功能。每一个测试人员都在叫测试时间不够，但又没有谁能说出到底多少时间才算够。

其实问题也很简单，一个合理的测试终止条件只能来源于一个清晰的测试目标。如果测试的目标是找到所有的缺陷，那么无论多少时间都是不够的。而要从测试的两个经济目标来看，合理的终止条件应当是由以下两点组成：

测试是为了保证软件的质量，而软件的质量标准是由用户来决定的。这个标准应当在软件开发初期由用户需求调查所得，如果每一需求项都列出了可测试的、被共同利益者认可的标准和写入测试计划之中的测试用例，这样软件测试结束的第一个必要条件就是所有在测试计划中所列出的测试项和标准（常见的有：必要及重要的功能通过测试，某种公认的认证测试用例包测试通过，连续无故障运行超过一定时间等）都被通过。

通常来说，早期找到并排除的缺陷越多，总的售后服务的成本就越少，但密集的测试就会导致测试成本的增高。而对测试来说，随着旧的缺陷不断地被找到并修复，软件的质量也就越来越好，后继的服

务成本就越来越低，相应地，新缺陷也就越来越难找，即测试成本越来越难高。所以售后服务的成本和测试的成本大致可以用下图来表示：



图－4

由图－4可知，当找到并解决的缺陷占总缺陷的比例达到T_E时，可终止测试。因为再要通过测试去发现一个缺陷成本比发布后再去维护的成本要高了。其实从企业利润的角度来看，就要使这两部分的成本之和最小。当然在实际情况下，这两条曲线是无法准确估计的。人们往往按拇指规则，即假设残留的缺陷数与最后一阶段排除的缺陷数相等，启用这样一个较为合理的中止条件：当一段时间内（通常是一个星期）测试不出的新缺陷时，就可中止寻找缺陷了；或按本文提出的测试效益规则，即当找到的新缺陷的实际价值低于相同时间的测试运行费用、或测试成本与维护成本之和达最小值时、或经3至5倍企业同类软件开发项目的平均单位缺陷测试时间内测试不出的新缺陷时就可中止找缺陷了。

结尾

随着软件行业的进一步发展，对软件测试及软件测试管理的要求也越来越高，尤其是在当前发达国家渐渐兴起软件外包浪潮的关键时刻，谁能提供低价可靠的软件开发服务，谁就能抢先占领市场，而高效经济的软件测试则必然会成为软件企业竞争力的最大助益。

软件测试的常识

张　华

Thinker@csai.com.cn

软件开发和使用的历史已经留给了我们很多由于软件缺陷而导致的巨大财力、物力损失的经验教训。这些经验教训迫使我们这些测试工程师们必须采取强有力的检测措施来检测未发现的隐藏的软件缺陷。

生产软件的最终目的是为了满足客户需求，我们以客户需求作为评判软件质量的标准，认为软件缺陷（Software Bug）的具体含义包括下面几个因素：

1. 软件未达到客户需求的功能和性能；
2. 软件超出客户需求的范围；
3. 软件出现客户需求不能容忍的错误；
4. 软件的使用未能符合客户的习惯和工作环境。

考虑到设计等方面的因素，我们还可以认为软件缺陷还可以包括软件设计不符合规范，未能在特定的条件（资金、范围等）达到最佳等。可惜的是，我们中的很多人更倾向于把软件缺陷看成运行时出现问题上来，认为软件测试仅限于程序提交之后。

在目前的国内环境下，我们几乎看不到完整准确的客户需求说明书，加以客户的需求时时在变，追求完美的测试变得不太可能。因此作为一个优异的测试人员，追求软件质量的完美固然是我们的宗旨，但是明确软件测试现实与理想的差距，在软件测试中学会取舍和让步，对软件测试是有百益而无一弊的。

下面是一些软件测试的常识，对这些常识的理解和运用将有助于我们在进行软件测试时能够更好的把握软件测试的尺度。

一、 测试是不完全的（测试不完全）

很显然，由于软件需求的不完整性、软件逻辑路径的组合性、输入数据的大量性及结果多样性等因素，哪怕是一个极其简单的程序，要想穷尽所有逻辑路径，所有输入数据和验证所有结果是非常困难的一件事情。我们举一个简单的例子，比如说求两个整数的最大公约数。其输入信息为两个正整数。但是如果我们将整个正整数域的数字进行

一番测试的话，从其数目的无限性我们便可证明是这样的测试在实际生活中是行不通的，即便某一天我们能够穷尽该程序，只怕我们乃至我们的子孙都早已作古了。为此作为软件测试，我们一般采用等价类和边界值分析等措施来进行实际的软件测试，寻找最小用例集合成为我们精简测试复杂性的一条必经之道。

二、 测试具有免疫性（软件缺陷免疫性）

软件缺陷与病毒一样具有可怕的“免疫性”，测试人员对其采用的测试越多，其免疫能力就越强，寻找更多软件缺陷就更加困难。由数学上的概率论我们可以推出这一结论。假设一个 50000 行的程序中有 500 个软件缺陷并且这些软件错误分布时均匀的，则每 100 行可以找到一个软件缺陷。我们假设测试人员用某种方法花在查找软件缺陷的精力为 X 小时 /100 行。照此推算，软件存在 500 个缺陷时，我们查找一个软件缺陷需要 X 小时，当软件只存在 5 个错误时，我们每查找一个软件缺陷需要 100X 小时。实践证明，实际的测试过程比上面的假设更为苛刻，为此我们必须更换不同的测试方式和测试数据。该例子还说明了在软件测试中采用单一的方法不能高效和完全的针对所有软件缺陷，因此软件测试应该尽可能的多采用多种途径进行测试。

三、 测试是“泛型概念”（全程测试）

我一直反对软件测试仅存在于程序完成之后。如果单纯的只将程序设计阶段后的阶段称之为软件测试的话，需求阶段和设计阶段的缺陷产生的放大效应会加大。这非常不利于保证软件质量。需求缺陷、设计缺陷也是软件缺陷，记住“软件缺陷具有生育能力”。软件测试应该跨越整个软件开发流程。需求验证（自检）和设计验证（自检）也可以算作软件测试（建议称为：需求测试和设计测试）的一种。软件测试应该是一个泛型概念，涵盖整个软件生命周期，这样才能确保周期的每个阶段禁得起考验。同时测试本身也需要有第三者进行评估（信息系统审计和软件工程监理），即测试本身也应当被测试，从而确保测试自身的可靠性和高效性。否则自身不正，难以服人。

另外还需指出的是软件测试是提高软件产品质量的必要条件而非充分条件，软件测试是提高产品质量最直接、最快捷的手段，但决不是一个根本手段。

四、 80-20 原则

80%的软件缺陷常常生存在软件 20%的空间里。这个原则告诉我们，如果你想使软件测试有效地话，记住常常光临其高危多发“地段”。在那里发现软件缺陷的可能性会大的多。这一原则对于软件测试人员提高测试效率及缺陷发现率有着重大的意义。聪明的测试人员会根据这个原则很快找出较多的缺陷而愚蠢的测试人员却仍在漫无目的地到处搜寻。

80-20 原则的另外一种情况是，我们在系统分析、系统设计、系统实现阶段的复审，测试工作中能够发现和避免 80%的软件缺陷，此后的系统测试能够帮助我们找出剩余缺陷中的 80%，最后的 5%的软件缺陷可能只有在系统交付使用后用户经过大范围、长时间使用后才会曝露出来。因为软件测试只能够保证尽可能多地发现软件缺陷，却无法保证能够发现所有的软件缺陷。

80-20 原则还能反映到软件测试的自动化方面上来，实践证明 80%的软件缺陷可以借助人工测试而发现，20%的软件缺陷可以借助自动化测试能够得以发现。由于这二者间具有交叉的部分，因此尚有 5%左右的软件缺陷需要通过其他方式进行发现和修正。

五、 为效益而测试

为什么我们要实施软件测试，是为了提高项目的质量效益最终以提高项目的总体效益。为此我们不难得出我们在实施软件测试应该掌握的度。软件测试应该在软件测试成本和软件质量效益两者间找到一个平衡点。这个平衡点就是我们在实施软件测试时应该遵守的度。单方面的追求都必然损害软件测试存在的价值和意义。一般说来，在软件测试中我们应该尽量地保持软件测试简单性，切勿将软件测试过度复杂化，拿物理学家爱因斯坦的话说就是：Keep it simple but not too simple。

六、 缺陷的必然性

软件测试中，由于错误的关联性，并不是所有的软件缺陷都能够得以修复。某些软件缺陷虽然能够得以修复但在修复的过程中我们会难免引入新的软件缺陷。很多软件缺陷之间是相互矛盾的，一个矛盾的消失必然会引发另外一个矛盾的产生。比如我们在解决通用性的缺陷后往往会带来执行效率上的缺陷。更何况在缺陷的修复过程中，我

们常常还会受时间、成本等方面的限制因此无法有效、完整地修复所有的软件缺陷。因此评估软件缺陷的重要度、影响范围，选择一个折中的方案或是从非软件的因素（比如提升硬件性能）考虑软件缺陷成为我们在面对软件缺陷时一个必须直面的事实。

七、 软件测试必须有预期结果

没有预期结果的测试是不可理喻的。软件缺陷是经过对比而得出来的。这正如没有标准无法进行度量一样。如果我们事先不知道或是无法肯定预期的结果，我们必然无法了解测试正确性。这很容易然人感觉如盲人摸象一般，不少测试人员常常凭借自身的感觉去评判软件缺陷的发生，其结果往往是把似是而非的东西作为正确的结果来判断，因此常常出现误测的现象。

八、 软件测试的意义-事后分析

软件测试的目的单单是发现缺陷这么简单吗？如果是“是”的话，我敢保证，类似的软件缺陷在下一次新项目的软件测试中还会发生。古语说得好，“不知道历史的人必然会重蹈覆辙”。没有对软件测试结果进行认真的分析，我们就无法了解缺陷发生的原因和应对措施，结果是我们不得不耗费的大量的人力和物力来再次查找软件缺陷。很可惜，目前大多测试团队都没有意识到这一点，测试报告中缺乏测试结果分析这一环节。

结论：

软件测试是一个需要“自觉”的过程，作为一个测试人员，遇事沉着，把持尺度，从根本上应对软件测试有着正确的认识，希望本文对读者对软件测试的认识有所帮助。

如何提高软件的质量

古　乐2　史九林
Kulerj@hotmail.com

【摘要】软件质量是软件产品的灵魂。本文全面介绍了质量的概念，提出了从流程、技术、组织管理、人员技能发展等多个角度提高软件质量的重要性；并对目前国际上流行的 CMM 标准进行了介绍，提出了使用 PSP 和 TSP 来实现 CMM 的方法。本文最后还给出了中小型软件公司在提高软件质量方面的一个初步思路。

【关键字】质量管理，软件开发过程模型，软件分析和设计方法，软件测试，CMM

如何提高软件的质量已经不是一个纯粹的技术问题，而是一个工程的问题。自从计算机诞生以来，相应的软件开发就存在了。由于早期的计算机运行性能较低，软件的可编程范围也较狭窄，因此质量问题就没有那么突出。50年代后期到60年代，高级语言的相继诞生并得到了广泛的应用，随之而来的是软件规模也越来越庞大，越来越复杂。伴随着软件应用的越来越广泛，软件的质量问题就变得越来越突出。根据美国国家宇航局NASA的统计，在80年代初，软件引起的故障与硬件引起的故障，其比率约为1.1:1.0，到了80年代末，这一比率已达到2.5:1.0。因此如何提高软件的质量成为软件工程研究的一个重点。自从软件危机产生以来，出现了很多提高产品质量的理论和方法，有的从技术角度出发，例如：面向对象技术的产生和推广，第四代语言的诞生等等；有的从自动化工具入手，例如：CASE工具、过程控制软件、自动化管理平台等；有的从过程模型角度出发，例如：迭代模型、螺旋模型、RUP、IPD、净室软件工程等；也有从管理角度出发的，例如：团队管理、绩效管理、PSP、TSP等；也有从测试角度出发的，例如：加强全流程的测试等；一些相应的规范和标准也孕育而生，例如：ISO9000系列、CMM、QMS等。然而每一种技术都不是绝对的，软件质量的提高应该是一个综合的因素，需要从每个方面进行改进，同时还需要兼顾成本和进度。

2 本文作者长期从事软件工程，软件质量管理和软件测试方面的技术研究，有着丰富的理论知识和实践经验，欢迎各位在这方面有兴趣的公司管理人员和技术人员来信讨论，并愿意提供这方面的咨询。

一、什么是质量？

作为软件产品的销售人员，市场人员或维护人员经常会受到客户这样那样的指责或抱怨，客户说：你们产品的质量太差，不稳定等等。那么什么是质量呢？我们该如何来衡量质量呢？

质量具有三个维度：

- 符合目标。目标是客户所定义的，符合目标即判断我们是不是在做需要做的事情。
- 符合需求。即产品是不是在做让它做的事情。
- 符合实际需求。实际的需求包括用户明确说明的和隐含的需求。

ISO关于质量的定义表示如下：

“一个实体（产品或服务）的所有特性，基于这些特性可以满足明显的或隐含的需要。”

注意，在这个定义中包含明显的需求和隐含的需求。而往往我们会忽略隐含的需求。因此在控制一个产品的质量的过程中必须关注这些隐含的需求，并给予应有的验证。

另一方面因为我们的产品是为客户提供服务的，因此凡是不满足客户需求的，我们都认为是一个失效（failure）。所以我们的产品必须始终围绕着客户的需求进行开发和验证。

这里我们谈到客户，其实在一个软件的需求收集过程中需要关注客户和用户。而我们经常会忽略客户与用户之间的区别。那么谁是客户？谁是用户呢？简单的来说，客户是真正能够决定是否购买你软件的人，而用户是实际使用软件的人。了解了这个区别，对于你在分析需求的重要性的时候就可以进行参考。同时在产品质量验证的时候也可以做出不同的权衡。另一方面我们在考虑我们用户需求的时候，往往只考虑了实际使用软件的人员，而忽略了其它一些人员对软件的要求或对软件造成的潜在竞争，这包括维护人员的要求、系统管理人员的要求、软件上下游人员的要求、先前版本的情况、市场上竞争对手的软件情况等。

每个人提到质量的时候，经常会遇到下列矛盾，在这些矛盾中隐含着对质量的承诺【5】：

- 质量需要一个承诺，尤其是高层管理者的承诺。但为了得到质

量，高层管理者必须和其雇用的员工进行紧密合作；

- 许多人相信没有缺陷的产品和服务是不可能的。但是控制在一定级别的缺陷数是正常并可接受的；
- 质量经常是和成本紧密联系在一起，一个高质量的产品同时也意味着高投入。这是设计的质量和一致性质量的一个矛盾；
- 一个高的质量要求需求规格说明书足够详细，以便产品可以根据这些规格说明书进行定量的分析。然而许多组织没有能力或者不愿意产生如此详细程度的规格说明书；
- 技术人员经常相信规范和标准会束缚他们的创造力，因此就不遵照标准做事。然而如果要得到高质量的产品，就必须遵循良好定义的标准和过程。

二、流程对质量的贡献

好了，既然已经了解了什么是质量，那么怎么才能改进软件产品的质量呢？从一个企业的长远发展来看，首先应当从流程抓起，规范软件产品的开发过程。这是一个软件企业从小作坊的生产方式向集成化、规范化的大公司迈进的必经之路，也是从根本上解决质量问题，提高工作效率的一个关键手段。

软件产品的开发同其它产品（如汽车）的生产有着共同特性，即需要按一定的过程来进行生产。在工业界，流水线生产方式被证明是一种高效且能够比较稳定地保证产品质量的一种方式。通过这种方式，不同的人员被安排在流程的不同位置，最终为着一个目标共同努力，这样可以防止人员工作间的内耗，极大的提高工作效率。并且由于其过程来源于成功的实例，因此其最终的产品质量能够满足过程所设定的范围要求。软件工程在软件的发展过程中吸取了这个经验并把它应用到了软件开发中，这就形成了软件工程过程，简单的说就是开发流程。

无论做什么事情，都有一个循序渐进的过程，从计划到策略再到实现。软件流程就是按照这种思维来定义开发过程，它根据不同的产品特点和以往的成功经验，定义了从需求到最终产品交付的一整套流程。流程告诉我们该怎么一步一步去实现产品，可能会有那些风险，如何去避免风险等等。由于流程来源于成功的经验，因此，按照流程进行开发可以使得我们少走弯路，并有效的提高产品质量，提高用户的满意度。

目前流行的流程方法有很多种，不同的过程模型适合于不同类型的项目。瀑布模型是应用的最为广泛的一种模型，也是最容易理解和掌握的模型，然而它的缺陷也是显而易见的。遗漏的需求或者不断变更的需求会使得该模型无所适从。然而，对于那些容易理解但很复杂的项目，采用瀑布模型会是比较适合的，因为你可以按部就班的去处理复杂的问题。在质量要求高于成本和进度要求的时候，该模型表现的尤其突出。

螺旋模型是也是一个经典模型，它关注于发现和降低项目的风险【8】。螺旋型项目从小的规模开始，然后探测风险，制定风险控制计划，接着确定下一步项目是否还要继续，然后进行下一个螺旋的反复。该模型的最大优点就是随着成本的增加，风险程度随之降低。然而螺旋模型的缺点是比较复杂，且需要管理人员有责任心，专注以及有管理方面经验。

RUP（Rational Unified Process）是Rational公司提出的一套开发过程模型，它是一个面向对象软件工程的通用业务流程【9】。它描述了一系列相关的软件工程流程，它们具有相同的结构，即相同的流程构架。RUP 为在开发组织中分配任务和职责提供了一种规范方法，其目标是确保在可预计的时间安排和预算内开发出满足最终用户需求的高品质的软件。RUP具有两个轴，一个是时间轴，这是动态的。另一个是工作流轴，这是静态的。在时间轴上，RUP划分了四个阶段：初始阶段、细化阶段、构造阶段和发布阶段。每个阶段都使用了迭代的概念。在工作流轴上，RUP设计了六个核心工作流程和三个核心支撑工作流程，核心工作流轴包括：业务建模工作流、需求工作流、分析设计工作流、实现工作流、测试工作流和发布工作流。核心支撑工作流包括：环境工作流、项目管理工作流和配置与变更管理工作流。具体可以参考图1。RUP 汇集现代软件开发中多方面的最佳经验，并为适应各种项目及组织的需要提供了灵活的形式。作为一个商业模型，它具有非常详细的过程指导和模板。但是同样由于该模型比较复杂，因此在模型的掌握上需要花费比较大的成本。尤其对项目管理者提出了比较高的要求。

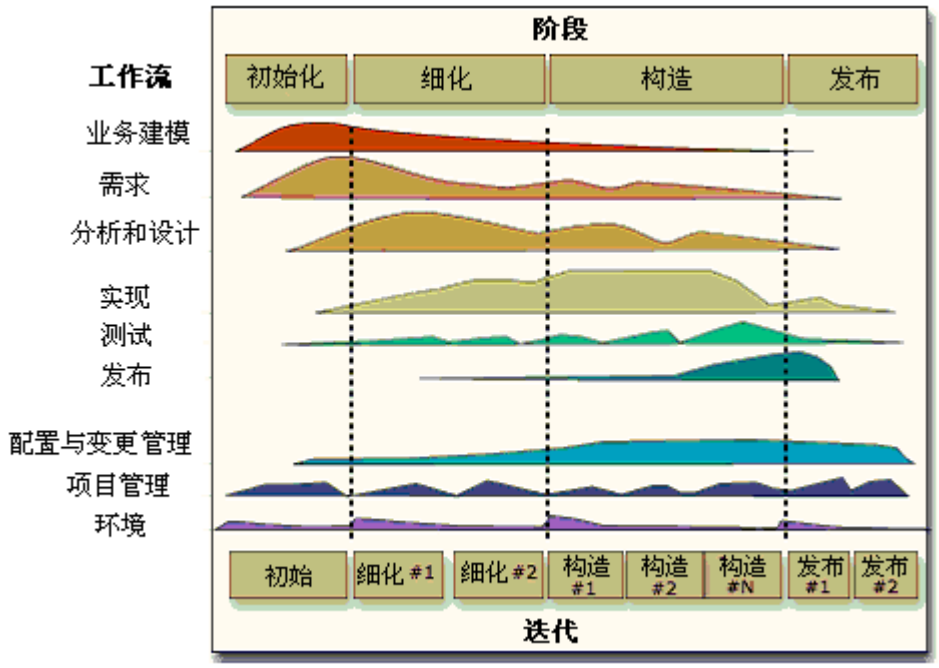


图1、RUP工作流程示意图

IPD（Integrated Product Development）流程是由IBM提出来的一套集成产品开发流程，非常适合于复杂的大型开发项目，尤其涉及到软硬件结合的项目。IPD从整个产品角度出发，流程综合考虑了从系统工程、研发（硬件、软件、结构工业设计、测试、资料开发等）、制造、财务到市场、采购、技术支援等所有流程。是一个端到端的流程。在IPD流程中总共划分了六个阶段（概念阶段、计划阶段、开发阶段、验证阶段、发布阶段和生命周期阶段），四个个决策评审点（概念阶段决策评审点、计划阶段决策评审点、可获得性决策评审点和生命周期终止决策评审点）以及六个技术评审点，具体可以参考图2。IPD流程是一个阶段性模型，具有瀑布模型的影子。该模型通过使用全面而又复杂的流程来把一个庞大而又复杂的系统进行分解并降低风险。一定程度上，该模型是通过流程成本来提高整个产品的质量并获得市场的占有。由于该流程没有定义如何进行流程回退的机制，因此对于需求经常变动的项目该流程就显得不大适合了。并且对于一些小的项目，也不是非常适合使用该流程。

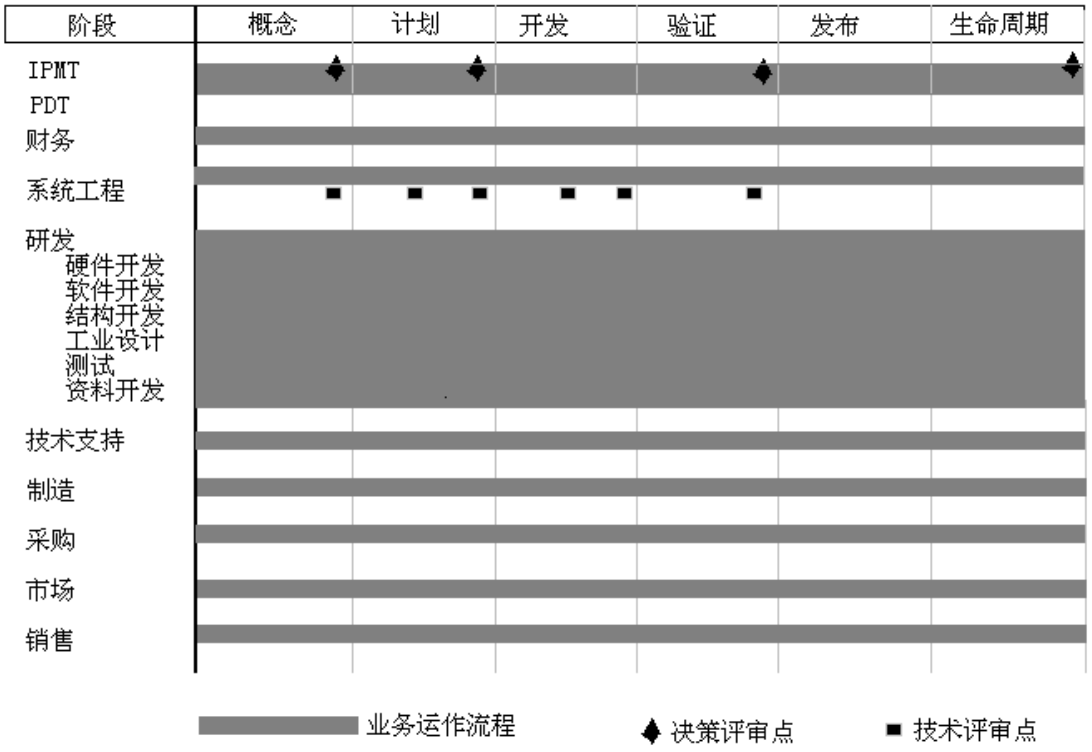


图 2、IPD 流程示意图

三、流程与技术

流程和成功不是等价的。没有流程就成功是不可能得到保证，但有了流程并不意味着肯定能够成功。这恐怕是很多迷信于流程的人所不能接受的。但这的确是个事实。记得有个做了将近30多年的需求分析专家说过：即使是一个已经达到CMM4级的公司，也完全有可能做不好需求分析。为什么？技术，技术是成功的另外一个必要条件。就好比现在你要从上海到北京去，流程给你指出了最短的路径，技术提供给你最快的交通工具。两者结合就是完美。

对于软件开发来说，要保证软件的质量，需要掌握多方面的技术，包括分析技术、设计技术、编码技术和测试技术等等。在国内有一个普遍的非正常现象，就是大家觉得只有编程能力才是玩电脑的真正技能。就好像造一套房子，其它都不重要，只要砖瓦匠有高超的技能就行了。尽管这个比喻会打击很多程序员的自尊心，但这的确是一个事实。我们缺少系统级的工程师，在分析和设计方面的工作做得很不扎实。

需求是一个项目的灵魂。模棱两可的需求带来不可避免的后果便是返工——重做一些你认为已做好的事情。返工会耗费开发总费用的

40%，而70%～85%的重做是由于需求方面的错误所导致的（leffingwell1997）【10】。想像一下如果你能减少一半的返工会是怎样的情况？你能更快地开发出产品，在同样的时间内开发更多、更好的产品，甚至能偶尔回家休息休息。在《软件需求》一书中关于如何进行需求分析给出了比较详细的介绍【7】，RUP中关于需求的指导也是很实用的。

设计是最能体现一个工程师能力和水平的环节。一个好的设计基本上决定了产品的最终质量。设计是把需求转换成系统的一个关键步骤，它需要从自然语言描述的需求中寻找出设计的基础单元，构建出整个系统的构架。在RUP中关于系统构架师和设计师的定位是相当高的。关于设计方面的技能涉及面是很广的，包括传统的结构化设计到面向对象设计。设计人员需要掌握一定的建模技术。UML是国际上比较流行的一种建模语言【11】。在嵌入式方面，SDL也是一种非常好的选择。《设计模式》是在设计思想方面总结的非常出色的一本书【6】，作为一名设计人员（尤其是面向对象设计人员）必须要好好研究一下。但是对这些模式的应用应当讲究一种自然的应用，千万不要因为模式而去设计模式，否则会适得其反。

现在的程序员热中于掌握多种编程语言，或者讲究语言的过分技巧化，而往往忽略了编程语言的规范化。不规范的语言应用给程序的可理解性、可维护性以及可测试性带来了大的伤害，进而损害了产品的质量。某公司曾对中国程序员和印度程序员做过一个测验，这个测验要求参加者对一组数进行排序。测试结果发现，印度程序员设计的程序使用的算法并不是最优，但却是最不容易出错的，并且几个程序员写出来的代码如出一辙。而几个中国程序员写出的代码，有的非常漂亮，很精练，效率很高；有的却很冗杂，还有错误。如果大家是在做研究性的项目或纯粹兴趣性的项目，那么充分发挥自己的编程天才也无可厚非。然而，对于一个软件公司，产品最终是要交给用户的，需要遵循的是一个软件产品的开发工程。因此这类软件的开发需要遵循一定的编程规范，毕竟开发的软件不是自己用，还需要和别人的集成，还需要给以后版本重用和维护。

测试的技术将在第五节进行阐述。总之流程很关键，技术也很重要，我的观点是：鱼和熊掌，两者都不能放。

四、全面质量管理

自从Deming的全面质量管理（TQM）原则在日本工业界获得了巨大成功之后，这个原则迅速被传播到了世界各个地方，同样，全面质量管理原则也被应用到了软件开发当中。如前面提到的，软件开发也是一个工程性的工作，因此必须提高整个工程的质量。

产业界的大量研究（TRW、Nippon Electric和Mitre Corp.以及其它一些公司）表明设计活动引入的错误占软件过程中出现所有错误（和最终的缺陷）数量的50％到65％。根据IBM的研究表明，假定在分析阶段发现的错误其改正成本为1个单位的话，那么在测试之前（设计编码阶段）发现一个错误的修改成本约为6.5个货币单位，在测试时（集成测试，系统测试和验收测试）发现一个错误的修改成本约为15个货币单位，而在发布之后（已经交到用户手上）发现一个错误的修改成本约为60到100个货币单位。同样该比例也适用用于发现一个错误需要的时间。我们可以看下面两条曲线图：

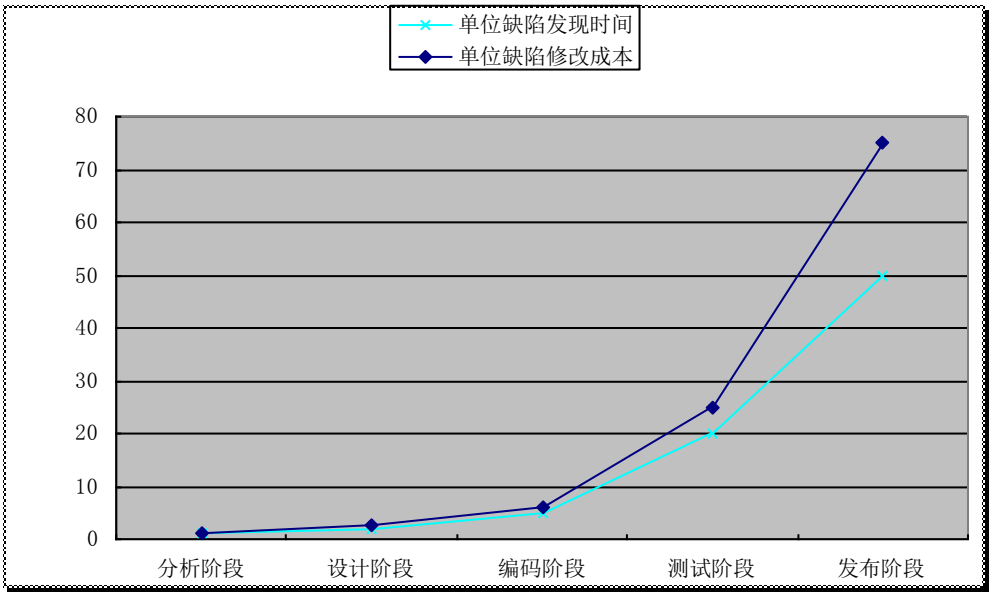


图 3、缺陷代价曲线

为了提高产品质量，缩短产品开发进度，节约产品开发成本，必须尽早的进行产品质量控制。全面质量控制要求在过程的每个阶段每个步骤上都要进行严格的验证和确认活动。

什么是验证？**验证**就是要用数据证明我们是不是在正确的制造产品。注意这里强调的是过程的正确行【12】。

什么是确认？**确认**就是要用数据证明我们是不是制造了正确的产品。注意这里强调的是结果的正确性。

IEEE给出的验证和确认过程可以用下图来表示。验证和确认是一个广泛的概念，感兴趣的读者可以参考IEEE Std 1012-1998。

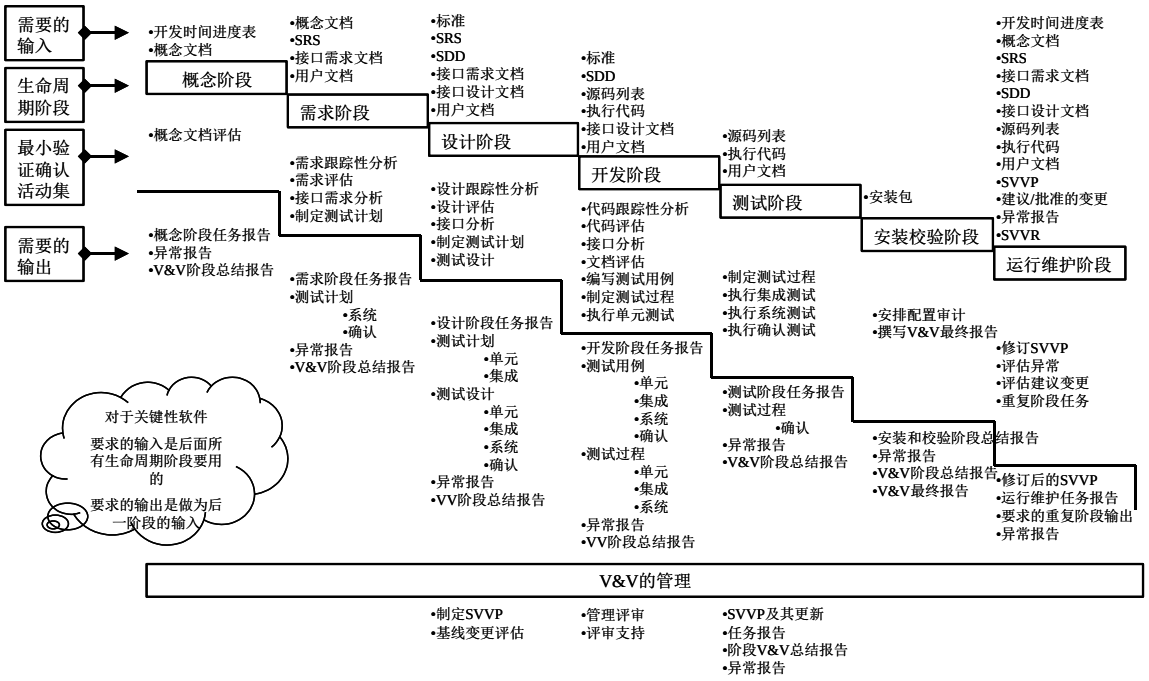


图 4、验证和确认模型

五、关注测试

软件测试是软件质量控制中的关键活动。业界的统计数据表明，测试的成本大约占软件开发总成本的50%左右。

软件测试的目的是要发现软件中的错误。一个好的测试是发现至今没有被发现的错误。传统的软件测试专注于动态测试范畴，如：单元测试，集成测试和系统测试。而测试工程的发展已经进入到了全流程的测试，包括开发过程前期的静态测试。

一般我们可以把测试分为白盒测试和黑盒测试。**白盒测试**：顾名思义，白盒测试应当是透明的。的确，该类测试是根据程序代码的内部逻辑结构来设计测试用例进行测试。那么什么是测试用例？ 一个**测试用例**就是一个文档，描述输入、动作、或者时间和一个期望的结果，其目的是确定应用程序的某个特性是否正常的工作。

黑盒测试：看了白盒测试的解释，我想你很快就能猜出黑盒测试是不考虑程序内部结构情况的。事实上也是这样。黑盒测试是根据规格说明书进行的测试。**规格说明书**记录了用户的需求。比如用户希望在编辑器中增加查找功能，那么我们把该需求写入规格说明书，根据该项要求，直接调用应用程序的该项功能进行测试，而不管其内部是

用什么算法实现的。

白盒和黑盒这两类测试是从完全不同的出发点，并且是两个完全对立点，反映了事物的两个极端，两种方法各有侧重，不能替代。但是在现代测试理念中，这两种测试往往不是决然分开的，一般在白盒测试中交叉使用黑盒测试的方法，在黑盒测试中交叉使用白盒测试的方法。

常见的白盒测试是单元测试。**单元测试**是测试中最小单位的测试。简而言之，就是拿一个函数出来，加上驱动模块，桩模块，让它能够运行起来，然后设计一些用例测试其内部的控制点（如：条件判断点，循环点，选择分支点等）。**驱动模块**是模拟调用被测函数的函数。**桩函数**是模拟当前测试函数所调用的函数。

常见的黑盒测试包括：集成测试，系统测试。**集成测试**是在单元测试的基础上，将所有模块按照设计要求（如根据结构图）组装成为子系统或系统，进行集成测试。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。

系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统定义不符合或与之矛盾的地方。系统测试的测试用例应根据需求分析说明书来设计，并在实际使用环境下来运行。系统测试的内容极其广泛，包括功能测试、协议测试、性能测试、压力测试、容量测试等等。有关测试方面的概念可以参考本人已出版的《软件测试技术概论》。

软件测试是产品最终交付到用户之前的最后一道防线，有着举足轻重的地位。然而，做好软件测试却是不容易的，一方面你需要同时掌握软件开发的技能和软件测试方面的技能；另一方面产品必须给予测试充分的独立性和资源保证。

六、成功的铁三角

在一个软件企业中，如果能够良性的发展，必须关注组织，流程和人三者之间的关系。组织是流程成功实施的保障，好的组织结构能够有效的促进流程的实施；流程对于产品的成功有着关键的作用，一个适合于组织特点和产品特点的流程能够极大的提高产品开发的效率和产品质量，反之则会拖延产品开发进度，并且质量也无法得到保证；对企业来说，人是最宝贵的财富，它们是技术的载体。对于一个

软件公司来说，无论是开发人员还是测试人员，都非常关心其今后的发展通道，如果有一条清晰的技术发展线为其指明今后的职业发展方向的话，这可以大大激励员工的士气和工作积极性。另外技术发展的方向应该与现在的开发流程和规范相结合，这样有利于专业技能的提高。

总之，组织，流程和人这三者是一个企业成功的铁三角，理想的情况下它们彼此促进，糟糕的情况下它们彼此制约。

七、国际上流行的质量标准

最早进入国内的质量标准是ISO系列。在软件方面主要使用ISO9000系列标准。ISO9000是一个非常完整的标准，并且定义了供应商设计和交付一个有质量产品的能力所需要的所有元素。ISO9002涵盖了对供应商控制设计和开发活动所认为重要的质量标准。ISO9003用于证明供应商在检视和测试期间检测和控制产品不一致性的能力。ISO9004描述和ISO9001、ISO9002和ISO9003相关的质量标准，并提供了一个完整的质量查检表。

软件能力成熟度模型是目前国内软件企业中非常受欢迎的一个质量标准。并且该标准已经成为业界一个事实上的标准。CMM为软件组织提供了一个指导性的管理框架。在这个框架的指导下：

- 软件组织可以对其软件开发、维护过程获得控制。
- 软件组织可以推进其软件工程更为科学、推进软件过程管理更为卓越。
- CMM通过确定当前软件过程管理的成熟度，通过标识软件的质量和过程改进中关键的、要害的问题，可以指导软件组织选择正确的软件过程改进策略。
- CMM将其焦点，聚焦在一系列具体的软件过程活动上，并以侵略方式（Aggressively）达到这些活动。一个软件组织就可以稳定地、持续地改进其整个软件组织过程，使得其软件过程管理能力取得持续地、持久地不断争长提高。

在CMM中，把软件工厂分为五个等级：初始级、可重复级、已定义级、管理级和优化级。其中：

初始级：软件过程是未加定义的随意过程，项目的执行是随意甚至是混乱的。也许，有些企业制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，且执行没有政策、资源等方面的保

证时，那么它仍然被视为初始级。

可重复级：人们根据多年的经验和教训，总结出软件开发的首要问题不是技术问题而是管理问题。因此，第二级的焦点集中在软件管理过程上。一个可管理的过程则是一个可重复的过程，可重复的过程才能逐渐改进和成熟。可重复级的管理过程包括了需求管理、项目管理、质量管理、配置管理和子合同管理五个方面；其中项目管理过程又分为计划过程和跟踪与监控过程。通过实施这些过程，从管理角度可以看到一个按计划执行的且阶段可控的软件开发过程。

已定义级：要求制定企业范围的工程化标准，并将这些标准集成到企业软件开发标准过程中去。所有开发的项目需根据这个标准过程裁剪出与项目适宜的过程，并且按照过程执行。过程的裁剪不是随意的，在使用前必须经过企业有关人员的批准。

管理级：所有过程需建立相应的度量方式，所有产品的质量（包括工作产品和提交给用户的最终产品）需要有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品。量化控制将使软件开发真正成为一种工业生产活动。

优化级：的目标是达到一个持续改善的境界。所谓持续改善是指可以根据过程执行的反馈信息来改善下一步的执行过程，即优化执行步骤。如果企业达到了第五级，就表明该企业能够根据实际的项目性质、技术等因素，不断调整软件生产过程以求达到最佳。

美国国防部规定，重要性级别高的软件应该由质量级别高的企业承担。不同等级的软件公司提交的软件，其软件质量也相差很大，国外的一份统计资料如下：

表1、CMM级别与软件质量关系表格

每千行软件的缺陷数目	软件过程成熟度等级	软件准时提交的百分比	每人每月生产的程序行数	软件需要返工的百分比	平均软件失效时间（近似）
大于10	初始级	<=50	Z	>=45	2到60分钟
小于10	可重复级	90	1.5Z	20	1-160小时
小于1	已定义级	99	2.5Z	10	不确定
小于0.1	管理级	降低开发时间到1/2	5Z	5	不确定
小于0.01	优化级	降低开发时间到1/4	10Z	<=2	近似完全可靠

对于很多已经推行或者准备推行CMM的公司来说，CMM的起步是很难的，因此Humphrey又提出了PSP（Person Software Process）和TSP（Team Software Process）【2】【3】。

CMM是过程改善的第一步，它提供了评价组织的能力、识别优先改善需求和追踪改善进展的管理方式【1】。企业只有开始CMM改善后，才能接受需要规划的事实，认识到质量的重要性，才能注重对员工经常进行培训，合理分配项目人员，并且建立起有效的项目小组。然而，它实现的成功与否与组织内部有关人员的积极参加和创造性活动密不可分。

PSP能够指导软件工程师如何保证自己的工作质量，估计和规划自身的工作，度量和追踪个人的表现，管理自身的软件过程和产品质量。经过PSP学习和实践的正规训练，软件工程师们能够在他们参与的项目工作之中充分运用PSP，从而有助于CMM目标的实现。

TSP结合了CMM的管理方法和PSP的工程技能，通过告诉软件工程师如何将个体过程结合进小组软件过程，并将后者与组织进而整个管理系统相联系；通过告诉管理层如何支持和授权项目小组，坚持高质量的工作，并且依据数据进行项目的管理，向组织展示如何应用CMM的原则和PSP的技能去生产高质量的产品。

软件的生产过程及其它的许多子过程、软件的开发者和用户、以及系统的使用中存在着巨大的变化和不同，要使一个软件过程对软件生产的改善真正有所帮助，其框架应是由CMM、TSP和PSP组成的一个完整体系，即从组织、群组和个人三个层次进行良好的软件工程和管理实践的指导和支持。总而言之，单纯实施CMM，永远不能真正做到能力成熟度的升级，只有将实施CMM与实施PSP和TSP有机地结合起来，才能发挥最大的效力。

八、如何起步？

质量改进需要花费成本，因此改进的途径需要视不同公司的规模、业务、财务状况、人员技术水平等多方面综合进行考虑。一般建议中型以上的较大的软件公司实施CMM体系。而对于一些小型的软件公司可以采取比较实际的，相对成本较少，且容易操作的方面进行，这些方面大致如下：

- 实施简洁的开发过程体系，根据不同业务特点可以选择瀑布模型，迭代模型等，并在这些模型上进行适当的变化以适应于短

平快的产品开发特点。

- 提高需求分析和设计方面的技术，例如：原型法技术，分析模式，设计模式，面向对象设计，UML等；
- 加强文档化工作。文档是经验的保留，对于一个企业要想获得长期的发展，必须加强文档化工作；
- 加强编程规范工作；
- 进行适当的测试工作，建议进行单元测试和系统测试；
- 实施配置管理工作，加强版本控制；
- 开展走读、评审和检视活动，尤其要加强代码走读，建议进行每日交叉走读活动；
- 进行简单的度量分析获得；建议实施PSP活动；

参考文档

【1】 Watts S. Humphrey ， Manager the Software Process, Addison Wesley, 1990

【2】 Watts S. Humphrey ， The Team Software Process^{SM} (TSP^{SM})，TECHNICAL REPORT CMU/SEI-2000-TR-023 ESC-TR-2000-023，*November 2000*

【3】 Watts S. Humphrey ， The Personal Software Process^{SM} (PSP^{SM})，TECHNICAL REPORT CMU/SEI-2000-TR-023 ESC-TR-2000-022，*November 2000*

【4】 Roger S.Pressman，软件工程实践者的研究方法，机械工业出版社，ISBN： 7111072820

【5】 William E.Lewis，Software Testing and Continuous Quality Improvement，AUERBACH

【6】 Erich Gamma，Richard Helm，Ralph Johnson，John Vlissides，设计模式，机械工业出版社，ISBN-7-111-07575-7

【7】 Karl E.Wiegers，软件需求，机械工业出版社，ISBN：7-111-08127-7

【8】 斯蒂夫·迈克康奈尔，快速软件开发——有效控制与完成进度计划，电子工业出版社，ISBN：7505372858

【9】 Ivar Jacobson，Grady Booch，James Rumbaugh，统一软件开发过程，机械工业出版社，ISBN：7111075722

【10】 Leffingwell, Dean. 1997. “Calculating the Return on Investment

from More Effective Requirements Management.” American Programmer 10(4):13-16

【11】 James Rumbaugh，UML 参考手册，机械工业出版社，ISBN：7111082206

【12】 BOHEM, B.W. Software Engineering Economics, Prentice-Hall, 1981. FENTON, N.E. Software Metrics, Chapman & Hall, 1991.

浅谈功能测试用例模板设计

蔡　琰

Cindycai@withub.org

【摘要】本文介绍测试用例一般要素以及如何根据项目特点设计测试用例模板，用以提高测试用例设计效率和实现测试用例执行结果报告的自动化计算，分析测试用例覆盖率。

【关键字】测试用例　模板　测试覆盖率

测试用例设计和执行是测试工作的核心，也是工作量最大的任务之一，设计良好的测试用例模板能提高测试用例的设计质量，便于跟踪测试用例的执行结果，自动生成测试用例覆盖率报告。这几年测试技术和理论有了长足的发展，就功能测试用例设计要素而言，样式上均大同小异，一般都包含主题、前置条件、执行步骤、期望结果等。

测试用例可以用数据库、Word、Excel、xml 等格式进行管理，市面亦有成熟的商业软件工具和开源工具等，对于一般中小软件企业，使用文档来管理测试用例是较为方便、经济的途径。Word 格式的文档可以满足设计需要，但不利于跟踪和自动统计执行结果报告。下面我将介绍自己在多个项目中设计和改进的 Excel 模版，它可以方便地设计测试用例，记录执行结果并自动统计测试用例覆盖率。图-1 为 Excel 模板。具体细目说明如下：

Test by:				Date:	
Pre Requisite:					
Test Procedure:					
TC ID	Level	Subject	Test Case step	Expected Result	Result
1	Service Provider				
SCENARIO A: Create New Service Provider					
Pre Requisite:					
A1	1				F
A2	1				P
SCENARIO B: scenario-name					
Pre Requisite:					
B1	3				B
END					

图-1 Excel 模板

测试用例 ID —— 用于唯一标识测试用例号，可根据自身需要定义规则，最好易于跟踪和维护；

测试前置条件 —— 如果有则描述之；
测试用例等级 —— 根据需求重要性区分测试用例等级，测试执行阶段可以根据测试用例等级安排测试任务，分为四级：

1. 冒烟测试，即版本确认测试，每个测试版本需通过所有该级测试用例，否则拒绝继续测试；
2. 关键路径测试，每个测试版本需执行该级测试用例，若该级测试用例均通过，意味着软件功能趋于稳定；
3. 可接受级测试，该级测试用例只要执行一次通过即可，该级测试用例通过意味着可以准备发布了；
4. 建议执行的用例，如果有时间，最好执行该级测试用例，但不作为发布的必要条件。

测试用例执行步骤、期望结果；
测试用例执行结果 —— 执行时填写，分为通过、失败、警告、阻塞、忽略。

通过开发 VBA 脚本，可以自动统计每轮测试用例执行结果，如图-2 所示，得到测试用例覆盖率结果报告，用于分析测试结果。

[Module Name] Test Suite Summary

Test by
Test duration

Test Case Level	Total Cases	Run	Pass	Fail	Block	Not Delivery
Level 1 - Smoke Test	2	2	1	1	0	0
Level 2 - Critical Path	0	0	0	0	0	0
Level 3 - Acceptance Criteria	1	0	0	0	1	0
Level 4 - Suggested	0	0	0	0	0	0
Total	3	2	1	1	1	0
By Pct		66.67%	33.33%	33.33%	33.33%	0.00%

Summary Report

图-2 测试用例覆盖率分析报告

测试用例状态转换分析

图-3 显示了一个典型测试用例的生命周期，依据不同类型和规模的项目可以自行定制。

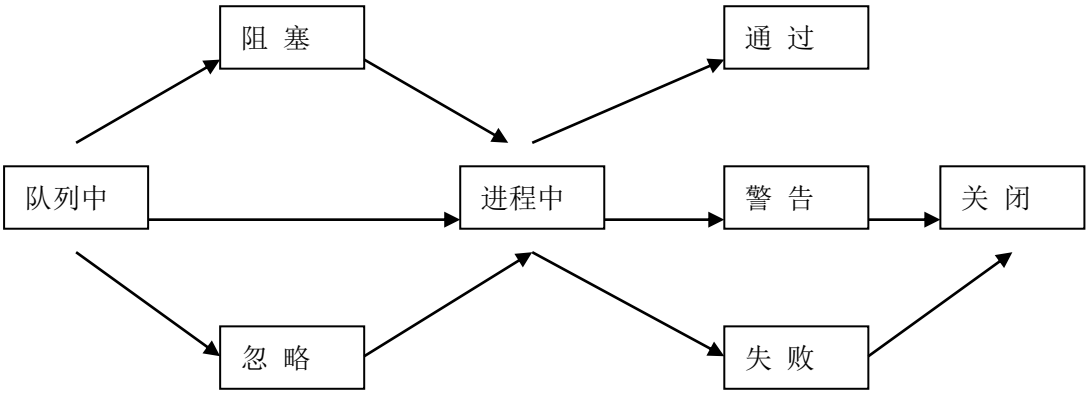


图-3 测试用例生命周期

队列中（In Queue）-- 测试用例在排队等待中；
进程中（In Progress）-- 表示测试正在进行，并且可能会持续一段时间，如果一个测试花费的时间少于一天或两天，只需将它显示在处于排队状态；
阻塞（Block）-- 一些外部条件—如缺少部分功能—将无法执行测试；
忽略（Skip）-- 已经决定（或被告知）跳过这个测试用例；
通过（Pass）-- 终点状态，没问题；
失败（Fail）-- 测试用例执行出错；
警告（Warn）-- 结果处于 Pass 和 Fail 之间，错误严重性等级较轻，不影响功能和性能；
关闭（Closed）-- 以前识别出的错误都已经被修正。

实际项目中，一个测试用例有多个执行步骤，每个步骤可能有不同结果，如步骤 1 通过，步骤 2 失败，步骤 3 被步骤 2 中的失败所阻塞，那么该测试状态如何？单纯指出这个测试用例阻塞或失败都将遗漏重要的信息。因此必须指定双重状态，如 Block/Fail，Block/Warn，Skip/Pass，Skip/Closed 等。然而，如果显示十几个状态，则测试结果可能更难以解释。如何使结果明了又能精确反映实际结果，需要精明选择包括哪些状态。

使用该模板优点：使用维护简便，方便测试任务分配，易于与项目组其他角色交流，结果报告自动生成。

不足之处：测试变更跟踪不方便，每个测试用例的规模不等，所

以测试覆盖率结果只是作为参考，结果百分比不能精确反映工作量，需要具体分析项目情况。这个模版没有跟踪统计缺陷，同时考虑是否使用加权评估缺陷严重性，一个测试用例往往对应几个缺陷的统计分析。

结论：在实际项目中，应该根据项目特点和开发流程定义测试用例各项。在精确和简单两个特性相对立时，需要好好权衡。如果您有好的解决方案，我将很乐意知道。

参考文献

- 《测试流程管理》 北京大学出版社 作者 Rex Black 2001 年 3 月第一版；
- Test Case Manager，Free software tool;
- 参考 **www.stickymind.com** 测试用例模板。

成功测试自动化的 7 个步骤

作者：Bret Pettichord
译者：王　威
Wangwei@withub.org

【摘要】我们对自动化测试充满了希望，然而，自动化测试却经常带给我们沮丧和失望。虽然，自动化测试可以把我们从困难的环境中解放出来，在实施自动化测试解决问题的同时，又带来同样多的问题。在开展自动化测试的工作中，关键问题是遵循软件开发的基本规则。本文介绍自动化测试的 7 个步骤：改进自动化测试过程，定义需求，验证概念，支持产品的可测试性，具有可延续性的设计（design for sustainability），有计划的部署和面对成功的挑战。按照以上 7 个步骤，安排你的人员、工具和制定你的自动化测试项目计划，你将会通往一条成功之路。

一个故事:

我在很多软件公司工作过，公司规模有大有小，也和来自其他公司的人员交流，因此经历过或者听说过影响自动化测试效果的各种各样的的问题。本文将提供若干方法规避可能在自动化测试中出现的问题。我先给大家讲一个故事，以便各位了解自动化测试会出现哪些问题。

以前，我们有一个软件项目，开发小组内所有的人都认为应该在项目中采用自动化测试。软件项目的经理是 Anita Delegate。她评估了所有可能采用的自动化测试工具，最后选择了一种，并且购买了几份拷贝。她委派一位员工——Jerry Overworked 负责自动化测试工作。Jerry 除了负责自动化测试工作，还有其他的很多任务。他尝试使用刚刚购买的自动化测试工具。当把测试工具应用到软件产品测试中的时候，遇到了问题。这个测试工具太复杂，难于配置。他不得不给测试工具的客户支持热线打了几个电话。最后，Jerry 认识到，他需要测试工具的技术支持人员到现场帮助安装测试工具，并找出其中的问题。在打过几个电话后，测试工具厂商派过来一位技术专家。技术专家到达后，找出问题所在，测试工具可以正常工作了。这还算是顺利了。但是，几个月后，他们还是没有真正实现测试自动化，Jerry 拒绝继续从事这个项目的工作，他害怕自动化测试会一事无成，只是浪

费时间而已。

项目经理 Anita 把项目重新指派给 Kevin Shorttimer，一位刚刚被雇佣来做软件测试的人员。Kevin 刚刚获得计算机科学的学位，希望通过这份工作迈向更有挑战性的、值得去做的工作。Anita 送 Kevin 参加工具培训，避免 Kevin 步 Jerry 的后尘——由于使用测试工具遇到困难而变得沮丧，导致放弃负责的项目。Kevin 非常兴奋。这个项目的测试需要重复测试，有点令人讨厌，因此，他非常愿意采用自动化测试。一个主要的版本发布后，Kevin 准备开始全天的自动化测试，他非常渴望得到一个机会证明自己可以写非常复杂的，有难度的代码。他建立了一个测试库，使用了一些技巧的方法，可以支持大部分的测试，这比原计划多花费了很多时间，不过，Kevin 使整个测试工作开展的很顺利。他用已有的测试套测试新的产品版本，并且确实发现了 bug。接下来，Kevin 得到一个从事软件开发职位的机会，离开了自动化的岗位。

Ahmed Hardluck 接手 Kevin 的工作，从事自动化测试执行工作。他发现 Kevin 留下的文档不仅少，并且没有太多的价值。Ahmed 花费不少时间去弄清楚已有的测试设计和研究如何开展测试执行工作。这个过程中，Ahmed 经历了很多失败，并且不能确信测试执行的方法是否正确。测试执行中，执行失败后的错误的提示信息也没有太多的参考价值，他不得不更深的钻研。一些测试执行看起来仿佛永远没有结束。另外一些测试执行需要一些特定的测试环境搭建要求，他更新测试环境搭建文档，坚持不懈地工作。后来，在自动化测试执行中，它发现几个执行失败的结果，经过分析，是回归测试的软件版本中有 BUG，导致测试执行失败，发现产品的 BUG 后，每个人都很高兴。接下来，他仔细分析测试套中的内容，希望通过优化测试套使测试变得更可靠，但是，这个工作一直没有完成，预期的优化结果也没有达到。按照计划，产品的下一个发布版本有几个主要的改动，Ahmed 立刻意识到产品的改动会破坏已有的自动化测试设计。接下来，在测试产品的新版本中，绝大多数测试用例执行失败了，Ahmed 对执行失败的测试研究了很长时间，然后，从其他人那里寻求帮助。经过商讨，自动化测试应该根据产品的新接口做修改，自动化测试才能运转起来。最后，大家根据新接口修改自动化测试，测试都通过了。产品发布到了市场上。接下来，用户立刻打来投诉电话，投诉软件无法工作。大家才发现自己改写了一些自动化测试脚本，导致一些错误提示信息

被忽略了。虽然，实际上测试执行是失败的，但是，由于改写脚本时的一个编程错误导致失败的测试执行结果被忽略了。这个产品终于失败了。

这是我的故事。或许您曾经亲身经历了故事当中某些情节。不过，我希望你没有这样的相似结局。本文将给出一些建议，避免出现这样的结局。

问题

这个故事阐明了几个使自动化测试项目陷入困境的原因：

自动化测试时间不充足：根据项目计划的安排，测试人员往往被安排利用自己的个人时间或者项目后期介入自动化测试。这使得自动化测试无法得到充分的时间，无法得到真正的关注。

缺乏清晰的目标：有很多好的理由去开展自动化测试工作，诸如自动化测试可以节省时间，使测试更加简单，提高测试的覆盖率，可以让测试人员保持更好的测试主动性。但是，自动化测试不可能同时满足上述的目标。不同的人员对自动化测试有不同的希望，这些希望应该提出来，否则很可能面对的是失望。

缺乏经验：尝试测试自己的程序的初级的程序员经常采用自动化自动化测试。由于缺乏经验，很难保证自动化测试的顺利开展。

更新换代频繁（High turnover）：测试自动化往往需要花费很多时间学习的，当自动化测试更新换代频繁的时候，你就丧失了刚刚学习到的自动化测试经验。

对于绝望的反应：在测试还远没有开始的时候，问题就已经潜伏在软件中了。软件测试不过是发现了这些潜伏的问题而已。就测试本身而言，测试是一件很困难的工作。当在修改过的软件上一遍接一遍的测试时，测试人员变得疲劳起来。测试什么时候后结束？当按照计划的安排，软件应该交付的时候，测试人员的绝望变得尤其强烈。如果不需要测试，那该有多好呀！在这种环境中，自动化测试可能是个可以选择的解决方法。但是，自动化测试却未必是最好的选择，他不是一个现实的解决方法，更像是一个希望。

不愿思考软件测试：很多人发现实现产品的自动化测试比测试本身更有趣。在很多软件项目发生这样的情况，自动化工程师不参与到软件测试的具体活动中。由于测试的自动化与测试的人为割裂，导致很多自动化对软件测试并没有太大的帮助。

关注于技术：如何实现软件的自动化测试是一个很吸引人的技术问题。不过，过多的关注如何实现自动化测试，导致忽略了自动化测试方案是否符合测试需要。

遵守软件开发的规则

你可能了解 SEI（软件工程研究所）提出的 CMM（能力成熟度模型）。CMM 分为 5 个界别，该模型用来对软件开发组织划分等级。Jerry Weinberg（美国著名软件工程专家）也创建了自己的一套软件组织模型，在他的组织模型中增加了一个额外的级别，他称之为零级别。很显然，一个零模式的组织实际上也是开发软件；零模式组织中，在开发人员和用户之间没有差别[Weinberg 1992]。恰恰在这类组织环境中，经常采用自动化测试方法。因此，把资源用于自动化测试，并且把自动化测试当作一个软件开发活动对待，把软件测试自动化提升到一级。这是解决测试自动化的核心的方案。我们应该像运作其他的开发项目一样来运作软件自动化测试项目。

像其他软件开发项目一样，我们需要软件开发人员专注于测试自动化的开发任务；像其他软件开发项目一样，自动化测试可以自动完成具体的测试任务，对于具体的测试任务来说，自动化开发人员可能不是这方面的专家，因此，软件测试专家应该提供具体测试任务相关的咨询，并且提供测试自动化的需求；像其他软件开发项目一样，如果在开发编码之前，对测试自动化作了整体设计有助于测试自动化开发的顺利开展。像其他软件开发项目一样，自动化测试代码需要跟踪和维护，因此，需要采用源代码管理。像其他软件开发项目一样，测试自动化也会出现 BUG，因此，需要有计划的跟踪 BUG，并且对修改后的 BUG 进行测试。像其他软件开发项目一样，用户需要知道如何使用软件，因此，需要提供用户使用手册。

本文中不对软件开发做过多讲述。我假定您属于某个软件组织，该组织已经知道采用何种合理的、有效的方法开发软件。我仅仅是推动您在自动化测试开发过程中遵守已经建立的软件开发规则而已。本文按照在软件开发项目中采用的标准步骤组织的，重点关注测试自动化相关的事宜和挑战。

1. 改进软件测试过程
2. 定义需求
3. 验证概念

4. 支持产品的可测试性
5. 可延续性的设计（design for sustainability）
6. 有计划的部署
7. 面对成功的挑战

步骤一：改进软件测试过程

如果你负责提高一个商业交易操作的效率，首先，你应该确认已经很好的定义了这个操作的具体过程。然后，在你投入时间和金钱采用计算机提供一套自动化的商业交易操作系统之前，你想知道是否可以采用更简单、成本更低的的方法。同样的，上述过程也是用于自动化测试。我更愿意把“测试自动化”这个词解释成能够使测试过程简单并有效率，使测试过程更为快捷，没有延误。运行在计算机上的自动化测试脚本只是自动化测试的一个方面而已。

例如，很多测试小组都是在回归测试环节开始采用测试自动化的方法。回归测试需要频繁的执行，再执行，去检查曾经执行过的有效的测试用例没有因为软件的变动而执行失败。回归测试需要反复执行，并且单调乏味。怎样才能做好回归测试文档化的工作呢？通常的做法是采用列有产品特性的列表，然后对照列表检查。这是个很好的开始，回归测试检查列表可以告诉你应该测试哪些方面。不过，回归测试检查列表只是合于那些了解产品，并且知道需要采用哪种测试方法的人。

在你开始测试自动化之前，你需要完善上面提到的回归测试检查表，并且，确保你已经采用了确定的的测试方法，指明测试中需要什么样的数据，并给出设计数据的完整方法。如果测试掌握了基本的产品知识，这会更好。确认可以提供上面提到的文档后，需要明确测试设计的细节描述，还应该描述测试的预期结果，这些通常被忽略，建议测试人员知道。太多的测试人员没有意识到他们缺少什么，并且由于害怕尴尬而不敢去求助。这样一份详细的文档给测试小组带来立竿见影的效果，因为，现在任何一个具有基本产品知识的人根据文档可以开展测试执行工作了。在开始更为完全意义上的测试自动化之前，必须已经完成了测试设计文档。测试设计是测试自动化最主要的测试需求说明。不过，这时候千万不要走极端去过分细致地说明测试执行的每一个步骤，只要确保那些有软件基本操作常识的人员可以根据文档完成测试执行工作既可。但是，不要假定他们理解那些存留在你头

脑中的软件测试执行的想法，把这些测试设计的思路描述清楚就可以了。

我以前负责过一个软件模块的自动化测试工作。这个模块的一些特性导致实现自动化非常困难。当我了解到这项工作无需在很短的时间内完成后，决定制定一个详细回归测试设计方案。我仔细地检查了缺陷跟踪库中与该模块相关的每个已经关闭的缺陷，针对每个缺陷，我写了一个能够发现该问题的测试执行操作。我计划采用这种方法提供一个详细的自动化需求列表，这可以告诉我模块的那一部分最适合自动化测试。在完成上述工作后，我没有机会完成测试自动化的实现工作。不过，当我们需要对这个模块做完整回归测试的时候，我将上面提到的文档提供给若干只了解被测试产品但是没有测试经验的测试人员。依照文档的指导，几乎不需要任何指导的情况下，各自完成了回归测试，并且发现了 BUG。从某种角度看，这实际上是一次很成功的自动化测试。在这个项目中，我们与其开发自动化测试脚本，还不如把测试执行步骤文档化。后来，在其它项目中，我们开发了自动化测试脚本，发现相关人员只有接受相关培训才能理解并执行自动化测试脚本，如果测试自动化设计的很好，可能会好一些。不过，经过实践我们总结出完成一份设计的比较好的测试文档，比完成一份设计良好的测试脚本简单的多。

另外一个提高测试效率的简单方法是采用更多的计算机。很多测试人员动辄动用几台计算机，这一点显而易见。我之所以强调采用更多的计算机是因为，我曾经看到一些测试人员被误导在单机上努力的完成某些大容量的自动化测试执行工作，这种情况下由于错误的使用了测试设备、测试环境，导致测试没有效果。因此，自动化测试需要集中考虑所需要的支撑设备。

针对改进软件测试过程，我的最后一个建议是改进被测试的产品，使它更容易被测试，有很多改进措施既可以帮助用户更好的使用产品，也可以帮助测试人员更好的测试产品。稍后，我将讨论自动化测试的可测试需求。这里，我只是建议给出产品的改进点，这样对手工测试大有帮助。

一些产品非常难安装，测试人员在安装和卸载软件上花费大量的时间。这种情况下，与其实现产品安装的自动化测试，还不如改进产品的安装功能。采用这种解决办法，最终的用户会受益的。另外的一个处理方法是考虑开发一套自动安装程序，该程序可以和产品一同发

布。事实上，现在有很多专门制作安装程序的商用工具。

另一些产品改进需要利用工具在测试执行的日志中查找错误。采用人工方法，在日志中一页一页的查询报错信息很容易会让人感到乏味。那么，赶快采用自动化方法吧。如果你了解日志中记录的错误信息格式，写出一个错误扫描程序是很容易的事情。如果，你不能确定日志中的错误信息格式，就开始动手写错误扫描程序，很可能面临的是一场灾难。不要忘记本文开篇讲的那个故事中提到的测试套无法判断测试用例是否执行失败的例子。最终用户也不愿意采用通过搜索日志的方法查找错误信息。修改错误信息的格式，使其适合日志扫描程序，便于扫描工具能够准确的扫描到所有的错误信息。这样，在测试中就可以使用扫描工具了。

通过改进产品的性能对测试也是大有帮助的。很显然的，如果产品的性能影响了测试速度，鉴别出性能比较差的产品功能，并度量该产品功能的性能，把它作为影响测试进度的缺陷，提交缺陷报告。

上面所述的几个方面可以在无需构建自动化测试系统的情况下，大幅度的提高测试效率。改进软件测试过程会花费你构建自动化测试系统的时间，不过改进测试过程无疑可以使你的自动化测试项目更为顺利开展起来。

步骤二：定义需求

在前面的故事中，自动化工程师和自动化测试的发起者的目标存在偏差。为了避免这种情况，需要在自动化测试需求上保持一致。应该有一份自动化测试需求，用来描述需要测试什么。测试需求应该在测试设计阶段详细描述出来，自动化测试需求描述了自动化测试的目标。很多人认为自动化测试显然是一件好事情，但是，他们不愿意对自动化测试的目标给出清晰的描述。下面是人们选用自动化测试的几个原因：

- 加快测试进度从而加快产品发布进度
- 更多的测试
- 通过减少手工测试降低测试成本
- 提高测试覆盖率
- 保证一致性
- 提高测试的可靠性
- 测试工作可以由技术能力不强测试人员完成

- 定义测试过程，避免过分依赖个人
- 测试变得更加有趣
- 提高了编程技能

开发管理、测试管理和测试人员实现自动化测试的目标常常是有差别的。除非三者之间达成一致，否则很难定义什么是成功的自动化测试。

当然，不同的情况下，有的自动化测试目标比较容易达到，有的则比较难以达到。测试自动化往往对测试人员的技术水平要求很高，测试人员必须能理解充分的理解自动化测试，从而通过自动化测试不断发现软件的缺陷。不过，自动化测试不利于测试人员不断的积累测试经验。不管怎么样，在开始自动化测试之前应该确定自动化测试成功的标准。

手工测试人员在测试执行过程中的一些操作能够发现不引人注意的问题。他们计划并获取必要的测试资源，建立测试环境，执行测试用例。测试过程中，如果有什么异常的情况发生，手工测试人员立刻可以关注到。他们对比实际测试结果和预期测试结果，记录测试结果，复位被测试的软件系统，准备下一个软件测试用例的环境。他们分析各种测试用例执行失败的情况，研究测试过程可疑的现象，寻找测试用例执行失败的过程，设计并执行其他的测试用例帮助定位软件缺陷。接下来，他们写作缺陷报告单，保证缺陷被修改，并且总结所有的缺陷报告单，以便其他人能够了解测试的执行情况。

千万不要强行在测试的每个部分都采用自动化方式。寻找能够带来最大回报的部分，部分的采用自动化测试是最好的方法。或许你可能发现采用自动化执行和手动确认测试执行结果的方式是个很好的选择，或许你可以采用自动化确认测试结果和手工测试执行相结合和方式。我听到有人讲，除非测试的各个环节都采用自动化方式，否则不是真正意义上的自动化测试，这真是胡言乱语。如果仅仅是为了寻找挑战，可以尝试在测试的每个环节都采用自动化方法。但是，如果寻找成功测试的方法，请关注那些可以快速建立的，可以反复利用的自动化测试。

定义自动化测试项目的需求要求我们全面地、清楚地考虑各种情况，然后给出权衡后的需求，并且可以使测试相关人员更加合理的提出自己对自动化测试的期望。通过定义自动化测试需求，距离成功的自动化测试近了一步。

步骤三：验证概念

在前面的故事当中，那个自动化测试人员在对测试方向一片茫然的情况下一头扎进了自动化测试项目中。不过，在项目的进行中，他得到了来自各个方面的支持。

你可能还没有认识到这一点，不过，你必须验证自动化测试项目的可行性。验证过程花费的时间往往比人们预期的要长，并且需要来自你身边的各种人的帮助。

很多年前，我从事一个测试自动化项目的工作，参加项目的人员有各种各样的好点子。我们设计了一个复杂的自动化测试系统，并且非常努力工作去实现系统的每个模块。我们定期的介绍测试自动化的设计思路和工作进度，甚至演示已经完成的部分功能。但是，我们没有演示如何利用该套测试自动化系统如何开展实际的测试工作。最后，整个项目被取消了，此后，我再也没有犯这个错误。

你需要尽可能快地验证你采用的测试工具和测试方法的可行性，站在产品的角度验证你所测试的产品采用自动化测试的可行性。这通常是很困难的，需要尽快地找出可行性问题的答案，需要确定你的测试工具和测试方法对于被测试的产品和测试人员是否合适。你需要做是验证概念——一个快速、有说服力的测试套可以证明你选在测试工具和测试方法的正确性，从而验证了你的测试概念。你选择的用来验证概念的测试套是评估测试工具的最好的方式。

对于很多人来说，自动化测试意味着 GUI 自动化测试，我不同意这种观点。我曾经做过 GUI 和非 GUI 自动化测试，并惊奇的发现这两类测试的测试计划有很大的互补性。不过，GUI 测试工具很昂贵、并且过分讲究。选择合适的 GUI 测试工具是很重要的，因为，如果没有选择合适的测试工具，你会遇到很多不可预测的困难。Elisabeth Hendrickson 曾经写过一篇关于选择测试的工具的指导性文章[Hendrickson 1999]。我建议在评估测试工具中，找出能够验证你的想法的证据是很重要的环节。这需要测试工具至少有一个月试用期，你可能打算现在购买一份测试工具，然后直到评估完成后再购买更多份。你需要在付出大笔金钱购买测试工具的之前，找出工具存在的问题。这样，你可以从测试工具供应商得到更好的帮助，当你打算更换工具的时候，你不会感觉很为难。

下面是一些候选的验证概念的试验：

回归测试：你准备在每个版本运行同样的测试用例吗？回归测试

是最宜采用自动化测试的环节。

配置测试：你的软件支持多少种不同的平台？你打算在所有支持的平台上测试执行所有的测试用例吗？如果是的，那么采用自动化测试是有帮助的。

测试环境建立：对于大量不同的测试用例，可能需要相同的测试环境搭建过程。在开展自动化测试执行之前，先把测试环境搭建实现自动化。

非 GUI 测试：实现命令行和 API 的测试自动化比 GUI 自动化测试容易的多。

无论采用什么测试方法，定义一个看得见的目标，然后集中在这个目标上。验证你自动化测试概念可以使自动化更进一步迈向成功之路。

步骤四：支持产品的可测试性

软件产品一般会用到下面三种不同类别的接口：命令行接口（command line interfaces，缩写 CLIs)、应用程序接口（API）、图形用户接口（GUI）。有些产品会用到所有三类接口，有些产品只用到一类或者两类接口，这些是测试中所需要的接口。从本质上看，API 接口和命令行接口比 GUI 接口容易实现自动化，去找一找你的被测产品是否包括 API 接口或者命令行接口。有些时候，这两类接口隐藏在产品的内部，如果确实没有，需要鼓励开发人员在产品中提供命令行接口或者 API 接口，从而支持产品的可测试性。

下面，更多多的讲解 GUI 自动化测试相关内容。这里有几个原因导致 GUI 自动化测试比预期的要困难。第一个原因是需要手工完成部分脚本。绝大多数自动化测试工具都有“录制回放”或者“捕捉回放”功能，这确实是个很好的方法。可以手工执行测试用例，测试工具在后台记住你的所有操作，然后产生可以用来重复执行的测试用例脚本。这是一个很好的方法，但是很多时候却不能奏效。很多软件测试文章的作者得出结论“录制回放”虽然可以生成部分测试脚本，但是有很多问题导致“录制回放”不能应用到整个测试执行过程中。[Bach 1996, Pettichord 1996, Kaner 1997, Linz 1998, Hendrickson 1999, Kit 1999, Thomson 1999, Groder 1999].结果，GUI 测试还是主要由手工完成。

第二个原因，把 GUI 自动化测试工和被测试的产品有机的结合

在一起需要面临技术上的挑战。经常要在采用众多专家意见和最新的GUI 接口技术才能使 GUI 测试工具正常工作。这个主要的困难也是GUI 自动化测试工具价格昂贵的主要原因之一。非标准的、定制的控件会增加测试的困难，解决方法总是有的，可以采用修改产品源代码的方式，也可以从测试工具供应商处升级测试工具。另外，还需要分析测试工具中的 BUG，并且给工具打补丁。也可能测试工具需要做相当的定制，以便能有效地测试产品界面上的定制控件。GUI 测试中，困难总是意外出现，让人惊奇。你也可能需要重新设计你的测试以规避那些存在问题的界面控件。

第三个原因，GUI 设计方案的变动会直接带来 GUI 自动化测试复杂度的提高。在开发的整个过程中，图形界面经常被修改或者完全重设计，这是出了名的事情。一般来讲，第一个版本的图形界面都是很糟糕。如果处在图形界面方案不停变动的时候，就开展 GUI 自动化测试是不会有任何进展的，你只能花费大量的时间修改测试脚本，以适应图形界面的变更。不管怎样，即便界面的修改会导致测试修改脚本，你也不应该反对开发人员改进图形界面。一旦原始的设计完成后，图形界面接口下面的编程接口就固定下来了。

上面提到的这些原因都是基于采用 GUI 自动化测试的方法完成产品的功能测试。图形界面接口当然需要测试，可以考虑实现 GUI 测试自动化。不过，你也应该考虑采用其他方法测试产品的核心功能，并且这些测试不会因为图形界面发生变化而被中断，这类测试应该采用命令行接口或者 API 接口。我曾经看到有人选择 GUI 自动化测试，因为，他们不打算修改被测试产品，但是，最终他们认识到必须对产品做修改，以保证 GUI 测试自动化可以正常工作。无论你选择哪种方法，自动化都需要对被测试的产品做修改。因此，采用可编程的接口是最可靠的。

为了让 API 接口测试更为容易，应该把接口与某种解释程序，例如 Tcl、Perl 或者 Python 绑定在一起。这使交互式测试成为可能，并且可以缩短自动化测试的开发周期。采用 API 接口的方式，还可以实现独立的产品模块的单元测试自动化。

一个关于隐藏可编程接口的例子是关于 InstallShield——非常流行的制作安装盘的工具。InstallShield 有命令行选项，采用这种选项可以实现非 GUI 方式的安装盘，采用这种方式，从提前创建好的文件中读取安装选项。这种方式可能比采用 GUI 的安装方式更简单更

可靠。

另一个例子是关于如何避免基于 WEB 软件的 GUI 自动化测试。采用 GUI 测试工具可以通过浏览器操作 WEB 界面。WEB 浏览器是通过 HTTP 协议与 WEB 服务器交互的，所以直接测试 HTTP 协议更为简单。Perl 可以直接连接 TCP/IP 端口，完成这类的自动化测试。采用高级接口技术，譬如客户端 JAVA 或者 ActiveX 不可能利用这种方法。但是，如果在合适的环境中采用这种方式，你将发现这种方式的自动化测试比 GUI 自动化测试更加便宜更加简单。

我曾经受雇在一家公司负责某个产品 GUI 相关的自动化测试，该产品也提供命令行接口，不过，他们已经实现了 GUI 的自动化测试。经过一段时间的研究，我发现找到图形界面中的 BUG 并不困难，不过，用户并不关注图形界面，他们更喜欢使用命令行。我还发现我们还没有针对最新的产品功能（这些功能即可通过 GUI 的方式，也可以通过命令行的方式使用）实现自动化测试。我决定推迟 GUI 自动化测试，扩展命令行测试套，测试新增的产品功能。现在回过头看这个决定，我没有选择 GUI 自动化测试是最大的成功之处，如果采用了 GUI 自动化测试所有的时间和努力都会浪费在其中。他们已经准备好做 GUI 自动化测试了，并且已经购买了一套测试工具和其他需要的东西，但我知道在开展具体的 GUI 自动化测试活动中，会遇到各种各样的困难和障碍。

无论你需要支持图形界面接口、命令行接口还是 API 接口，如果你尽可能早的在产品设计阶段提出产品的可测试性设计需求，未来的测试工作中，你很可能成功。尽可能早的启动自动化测试项目，提出可测试性需求，会使您走向自动化测试成功之路。

步骤五：具有可延续性的设计

在开篇的故事中，我们看到由于自动化工程师把注意力仅仅集中在如何使自动化运转起来，导致测试自动化达不到预期的效果。自动化测试是一个长期的过程，为了与产品新版本的功能和其他相关修改保持一致，自动化测试需要不停的维护和扩充。自动化测试设计中考虑自动化在未来的可扩充性是很关键的，不过，自动化测试的完整性也是很重要的。如果自动化测试程序报告测试用例执行通过，测试人员应该相信得到的结果，测试执行的实际结果也应该是通过了。其实，有很多存在问题的测试用例表面上执行通过了，实际上却执行失败

了，并且没有记录任何错误日志，这就是失败的自动化。这种失败的自动化会给整个项目带来灾难性的后果，而当测试人员构建的测试自动化采用了很糟糕的设计方案或者由于后来的修改引入了错误，都会导致这种失败的测试自动化。失败的自动化通常是由于没有关注自动化测试的性能或者没有充分的自动化设计导致的。

性能：提高代码的性能往往增加了代码的复杂性，因此，会威胁到代码的可靠性。很少有人关心如何对自动化本身加以测试。通过我对测试套性能的分析，很多测试套都是花费大量的时间等候产品的运行。因此，在不提高产品运行性能的前提下，无法更有效的提高自动化测试执行效率。我怀疑测试自动化工程师只是从计算机课程了解到应该关注软件的性能，而并没有实际的操作经验。如果测试套的性能问题无法改变，那么应该考虑提高硬件的性能；测试套中经常会出现冗余，也可以考虑取出测试套中的冗余或者减少一个测试套中完成的测试任务，都是很好的办法。

便于分析：测试自动化执行失败后应该分析失败的结果，这是一个棘手的问题。分析执行失败的自动化测试结果是件困难的事情，需要从多方面着手，测试上报的告警信息是真的还是假的？是不是因为测试套中存在缺陷导致测试执行失败？是不是在搭建测试环境中出现了错误导致测试执行失败？是不是产品中确实存在缺陷导致测试执行失败？有几个方法可以帮助测试执行失败的结果分析，某些方法可以找到问题所在。通过在测试执行之前检查常见的测试环境搭建问题，从而提高测试套的可靠性；通过改进错误输出报告，从而提高测试自动化的错误输出的可分析性；此外，还可以改进自动化测试框架中存在的问题。训练测试人员如何分析测试执行失败结果。甚至可以找到那些不可靠的、冗余的或者功能比较独立的测试，然后安全地将之删除。上面这些都是减少自动化测试误报告警、提高测试可分析性的积极有效的方法。另外，有一种错误的测试结果分析方法，那就是采用测试结果后处理程序对测试结果自动分析和过滤，尽管也可以采用这种测试结果分析方法，不过这种方法会使自动化测试系统复杂化，更重要的是，后处理程序中的 BUG 会严重损害自动化测试的完整性。如果由于自动化测试本身存在的缺陷误把产品中的正常功能视为异常，那该怎么办？我曾经看到测试小组花费开发测试自动化两倍的时间来修改测试脚本，并且不愿意开展培训过程。那些仅仅关注很浅层次测试技术的测试管理者对这种方法很感兴趣，他们排斥采用隐

藏测试复杂度的方法。

综上所述，应该集中精力关注可以延续使用的测试套，从以下几方面考虑，测试的可检视性、测试的可维护性、测试的完整性、测试的独立性、测试的可重复性。

可读性：很多情况下，在测试人员开始测试项目之前，公司已经有了一套测试脚本，并且已经存在很多年了。我们可以称之为“聪明的橡树”(wise oak tree)[Bach 1996]。大家很依赖它，但是并不知道它是什么。由于“聪明的橡树”类型的测试脚本缺乏可读性，在具体应用中，那些脚本常常没有多大的实用价值，越来越让人迷惑。因此，测试人员很难确定他们实际在测试什么，反而会导致测试人员对自身的测试能力有过高的估计。测试人员能够检视测试脚本，并且理解测试脚本究竟测试了什么，这是很关键的。好的文档是解决问题的手段之一，对测试脚本全面分析是另外一个手段。由上面两种方法可以引申出其它的相关方法，我曾经在一个项目中使用过引申之后的方法。在测试脚本中插桩，把所有执行的产品相关的命令记录到日志里。这样，当分析日志确定执行了哪些产品命令，命令采用了何种参数配置时，可以提供一个非常好的测试记录，里面记录了自动化测试执行了什么，没有执行什么。如果测试脚本可读性不好，很容易变得过分依赖并没有完全理解的测试脚本，很容易认为测试脚本实际上做的工作比你想象中的还要多。测试套的可读性提高后，可以更容易的开展同行评审活动。

可维护性：在工作中，我曾经使用过一个测试套，它把所有的程序输出保存到文件中。然后，通过对比输出文件内容和一个已有的输出文件内容的差别，可以称已有的输出文件为“标准文件”（“gold file”）。在回归测试中，用这个方法查找 BUG 是不是明智之举。这种方法太过于敏感了，它会产生很多错误的警告。随着时间的推移，软件开发人员会根据需要修改产品的很多输出信息，这会导致很多自动化测试失败。很明显，为了保证自动化测试的顺利进行，应该在对“标准文件”仔细分析的基础上，根据开发人员修改的产品输出信息对之做相应的修改。比较好的可维护性方法是，每次只检查选定的产品的某些特定输出，而不是对比所有的结果输出。产品的接口变动也会导致原来的测试无法执行或者执行失败。对于 GUI 测试，这是一个更大的挑战。研究由于产品接口变化引起的相关测试修改，并研究使测试修改量最小的方法，测试中采用库是解决问题的方法。当产品发

生变化的时候，只需要修改相关的库，保证测试与产品的变动同步修改即可。

完整性：当自动化测试执行后，报告测试执行通过，可以断定这是真的吗？这就是我称之为测试套的完整性。在前面的故事中，当没有对自动化测试完整性给予应有的关注的时候，发生了富有喜剧性的情况。我们应该在多大程度上相信自动化化测试执行结果？自动化测试执行中的误报告警是很大的问题。测试人员特别讨厌由于测试脚本自身的问题或者是测试环境的问题导致测试执行失败，但是，对于自动化测试误报告警的情况，大家往往无能为力。你期望自己设计的测试对 BUG 很敏感、有效，当测试发现异常的时候，能够报告测试执行失败。有的测试框架支持对特殊测试结果的分类方法，分类包括 PASS，FAIL 和第三种测试结果 NOTRUN 或者 UNRESOLVED。无论你怎么称呼第三种测试结果，它很好的说明了由于某些测试失败导致其他测试用例无法执行而并非执行失败的情况。得到正确的测试结果是自动化测试完整性定义的一部分，另一部分是能够确认执行成功的测试确确实实已经执行过了。我曾经在一个测试队列中发现一个 BUG，这个 BUG 引起测试队列中部分测试用例被跳过，没有执行。当测试队列运行完毕后，没有上报任何错误，我不得不通过走读代码的方式发现了这个 BUG。如果，我没有关注到这个 BUG，那么可能在认识到自动化测试已经出现问题之前，还在长时间运行部分测试用例。

独立性：采用自动化方法不可能达到和手工测试同样的效果。当写手工测试执行的规程时候，通常假定测试执行将会由一个有头脑、善于思考、具有观察力的测试人员完成的。如果采用自动化，测试执行是由一台不会说话的计算机完成的，你必须告诉计算机什么样的情况下测试执行是失败的，并且需要告诉计算机如何恢复测试场景才能保证测试套可以顺利执行。自动化测试可以作为测试套的一部分或者作为独立的测试执行。测试都需要建立自己所需要的测试执行环境，因此，保证测试执行的独立性是唯一的好方法。手工回归测试通常都相关的指导文档，以便一个接着一个有序地完成测试执行，每个测试执行可以利用前一个测试执行创建的对象或数据记录。手工测试人员可以清楚地把握整个测试过程。在自动化测试中采用上述方法是经常犯的错误，这个错误源于“多米诺骨牌”测试套，当一个测试执行失败，会导致后续一系列测试失败。更糟糕的是，所有的测试关联紧密，无

法独立的运行。并且，这使得在自动化测试中分析合法的执行失败结果也困难重重。当出现这种情况后，人们首先开始怀疑自动化测试的价值。自动化测试的独立性要求在自动化测试中增加重复和冗余设计。具有独立性的测试对发现的缺陷的分析有很好的支持。以这种方式设计自动化测试好像是一种低效率的方式，不过，重要的是在不牺牲测试的可靠性的前提下保证测试的独立性，如果测试可以在无需人看守情况下运行，那么测试的执行效率就不是大问题了。

可重复性：自动化测试有时能够发现问题，有时候又无法发现问题，对这种情况实在没有什么好的的处理办法。因此，需要保证每次测试执行的时候，测试是以同样的方式工作。这意味着不要采用随机数据，在通用语言库中构造的随机数据经常隐藏初始化过程，使用这些数据会导致测试每次都以不同的方式执行，这样，对测试执行的失败结果分析是很让人沮丧的。这里有两个使用随机数据发生器的的方法可以避免上述情况。一种方法是使用常量初始化随机数据发生器。如果你打算生成不同种类的测试，需要在可预测，并且可控制的情况下建立不同类型的随机数据发生器。另外一个办法是提前在文件中或数据库中建立生成随机测试数据，然后在测试过程中使用这些数据。这样做看起来似乎很好，但是我却曾经看到过太多违反规则的做法。下面我来解释到底看到了什么。当手动执行测试的时候，往往匆匆忙忙整理文件名和测试数据记录。当对这些测试采用自动化测试方法，该做哪些工作呢？办法之一是，可以为测试中使用的数据记录给固定的命名。如果这些数据记录在测试完成后还要继续使用，那么就需要制定命名规则，避免在不同的测试中命名冲突，采用命名规则是一种很好的方法。然而，我曾经看到有些测试只是随机的命名数据记录，很不幸，事实证明采用这种随机命名的方式不可避免的导致命名冲突，并且影响测试的可重复性。很显然，自动化工程师低估了命名冲突的可能性。下面的情况我遇到过两次，测试数据记录的名字中包含四个随机产生的数字，经过简单的推算如果我们采用这种命名方式的时候，如果测试使用了 46 条记录，会存在 10%冲突的可能性，如果使用 118 条记录，冲突的几率会达到 50%。我认为测试当中使用这种随机命名是出于偷懒的想法——避免再次测试之前写代码清除老的数据记录，这样引入了问题，损害了测试的可靠性和测试的完整性。

测试库：自动化测试的一个通用策略是开发可以在不同测试中应用的测试函数库。我曾经看到过很多测试函数库，自己也写了一些。

当要求测试不受被测试产品接口变动影响的时候，采用测试库方法是非常有效的。不过，根据我的观察测试库已经使用的太多了，已经被滥用了，并且测试库往往设计的不好，没有相关的文档支撑，因此，使用测试库的测试往往很难开展。当发现问题的时候，往往不知道是测试库自身的问题，还是测试库的使用问题。由于测试库往往很复杂，即便在发现测试库存在问题，相关的维护人员也很不愿意去修改问题。通过前文中的论述，可以得出结论，在一开始就应该保证测试库设计良好。但是，实际情况是测试自动化往往没有花费更多的精力去保证一个优良设计的测试库。我曾经看到有些测试库中的功能根本不再使用了或仅仅使用一次。这与极限编程原则保持一致，即"你将不需要它"。这会导致在测试用例之间的代码出现一些重复，我发现微小的变化可能仍然存在，很难与测试库功能协调。你可能打算对测试用例作修改，采用源代码的方式比采用库的方式更容易修改。如果有几个测试，他们有某些共同的操作，我使用剪切和粘贴的方式去复制代码，有的人认为我采用的方法不可理喻。这允许我根据需要修改通用代码，我不必一开始尝试和猜测如何重用代码。我认为我的测试是很容易读懂的，因为阅读者不必关心任何测试库的语法。这种办法的优势是很容易理解测试，并且很方便扩展测试套。在开发软件测试项目的时候，大多数程序员找到与他们打算实现功能类似的源代码，并对源代码做修改，而不是从头开始写代码。同样，在写测试套的过程中可以采用上述方法，这也是代码开发方式所鼓励的方法。我比较喜欢写一些规模比较小的测试库，这些库可以被反复的使用。测试库的开发需要在概念阶段充分定义，并且文档化，从始至终都应该保持。我会对测试库作充分的测试后，才在测试中使用这些测试库。采用测试库是对所有面临的情况作权衡的。千万不要打算写一个大而全的测试库，不要希望有朝一日测试人员会利用你的测试库完成大量的测试，这一天恐怕永远不会到来。

数据驱动测试：把测试数据写入到简单表格中，这种测试技术得到了越来越广泛的应用，这种方法被称为表驱动（table-driven），数据驱动(data-driven)或者“第三代”自动化测试（"third generation" automation）。这需要写一个解析器，用来解释表格中的数据，并执行测试。该测试架构的最主要的好处是，它允许把测试内容写在具有一定格式的表格中，这样方便数据设计和数据的检视。如果测试组中有缺少编程经验的业务专家参与测试，采用数据驱动测试方法是很合

适的。数据驱动测试的解析器主要是由测试库和上层的少量开发语言写成的代码组成的，所以，上面关于测试库的说明放在这里是同样合适的。在针对上面提到的少量代码的设计、开发、测试的工作，还存在各种困难。代码所采用的编程语言是不断发展的。也许，测试人员认为他们需要把第一部分测试的输出作为第二部分测试的输入，这样，加入了新的变量。接下来，也许有人需要让测试中的某个环节运行一百次，这样加入一个循环。你可以采用其他语言，不过，如果你预料到会面临上述情况的时候，那么做好采用一些能够通过公开的渠道获取的编程语言，比如 Perl,Python 或者 TCL，这样比设计你自己的语言要快的多。

启发式确认：我曾经看到很多测试自动化没有真正意义上的结果校验，其原因有两个，一个原因是做完全意义上的自动化测试结果确认从技术上讲是很困难的，另外一个原因是通过测试设计规格很难找出自动化测试的预期结果。这很不幸。不过，采用手工校验测试结果的方法是真正意义上的测试校验。标准文件（Gold file）是另外一中校验测试结果的方法。首先，捕获被测试程序的输出，并检视程序的输出，然后，把输出信息文档化，并归档，作为标准文件。以后，自动化测试结果与标准文件作比较，从而达到结果校验的目的。采用标准文件的方法，也有弊端。当产品发生变化，自动化测试的环境配置发生变化，产品的输出发生变化的时候，采用标准文方法，会上报大量的误报告警。做好的测试结果校验方法是，对输出结果的特定内容作分析，并作合理的比较。有时候，很难知道正确的输出结果是什么样的，但是你应该知道错误的输出结果是什么样的。开展启发式的结果校验是很有帮助的。我猜想一些人在自动化测试中设计了大而全的测试结果校验方法，是因为担心如果结果校验漏掉了任何信息，可能导致自动化测试自身出现错误。不过，我们在测试过程中往往采用折衷的做法，没有采用大而全的测试结果校验方法，这样就不得不面对少量漏测情况的出现的风险。自动化测试不能改变这种情况的出现。如果自动化工程师不习惯采用这种折衷的方法，那么他必须找相关人员咨询，寻找一种合适的测试结果校验策略，这需要有很大的创造性。目前有很多技术可以保证在不产生误报告警的情况下，找到被测试产品的缺陷。

把注意力放在通过设计保证测试的可延续性上，选择一个合适的测试体系架构，你将进一步迈向成功的自动化测试。

步骤六：有计划的部署

在前面的故事中，当自动化工程师没有提供打包后的自动化测试程序给测试执行人员，会影响到测试执行，测试执行人员不得不反过来求助自动化工程师指出如何使用自动化测试程序。

作为自动化工程师，你知道如何利用自动化方法执行测试和分析执行失败的结果。不过，测试执行人员却未必知道如何使用自动化测试。因此，需要提供自动化测试程序的安装文档和使用文档，保证自动化测试程序容易安装和配置。当安装的环境与安装的要求不匹配，出现安装错误的时候，能够给出有价值的提示信息，便于定位安装问题。

能够把自动化测试程序和测试套作为产品对待，那真是太好了。你应该对自动化测试程序和测试套开展测试，保证它们不依赖于任何专用的库或者是设备上的任何其他程序。

保证其他测试人员能够随时利用已经提供的自动化测试程序和测试套开展测试工作；保证自动化测试是符合一般测试执行人员的思维习惯的；保证测试执行人员能够理解测试结果，并能够正确分析失败的测试执行结果；这需要自动化工程师提供自动动化测试相关的指导性文档和培训。

作为测试管理者，你希望在自动化工程师离开前，能够识别并修改测试套中的所有问题。自动化工程师迟早会离开的，如果你没有及时的把测试套中的问题提出来，就会面临废弃已有的测试套的决定。

良好的测试套有多方面的用处。良好的测试套支持对产品新版本的测试；良好的测试套在新的软件平台上可以很方便的验证产品的功能；良好的测试套支持每天晚上开始的软件每日构造过程；甚至开发人员在代码 check in 之前，用良好的测试套验证代码的正确性。

测试套的共享也很重要。很难预见以后什么人会继续使用你开发的测试套。因此，尽量让产品开发测试团队中的成员都很容易获得你的测试套。可以把测试套放在公司的内部网络上，这是个很好的办法。这样，大家就不必为了获取一份需要的测试套而四处打听。有些人总是感觉自己的测试套还没有最终完工或者不够完美，而没有拿出来与人分享，这种做法一定要改变，共享出来的测试套不一定非常完美，共享才是关键。

有计划的自动化测试部署，保证你的测试套能够被产品相关人员获取到，你就向成功的自动化测试又迈进了一步。并且你的自动化测

试会被一次又一次的重用。

步骤七：面对成功的挑战

当你完成了所有的事情，测试套已经文档化了，并且文档已经交付了。测试执行人员能够理解要开展的测试，并知道如何完成测试执行。随着你所负责产品的进一步开发和维护，测试被反复重用。虽然，在自动化使测试变简单的同时，也总是使测试过程复杂化。测试人员需要学习如何诊断自动化测试执行失败的情况，如果不这样做，测试执行人员会认为执行失败的情况是由自动化引起，然后，自动化工程师被叫过来帮助诊断每一个执行失败的情况，开发人员往往也会认为执行失败是由于自动化测试自身引起的问题，这样，测试执行人员就不得不学习通过手工的方式，或者通过采用少量脚本的方式重现自动化测试发现的问题，以证明他们确实发现了产品当中的 BUG。

测试套的相关工作还没有结束，为了提高测试覆盖率或者测试新的产品特性，需要增加更多的测试。如果已有的测试不能正常工作，那么需要对之修改；如果已有的测试是冗余的，那么需要删除这部分测试。

随着时间的推移，开发人员也会研究你设计的测试，改进产品的设计并且通过模拟你的测试过程对产品做初步测试，研究如何使产品在第一次测试就通过，这样，你设计的测试很可能无法继续发现新的问题，这种现象被称为一种杀虫剂悖论。这时候，会有人对你的测试有效性提出质疑，那么，你必须考虑是否应该挖掘更严格的测试，以便能够发现开发人员优化之后的产品中的缺陷。

以前，我提到过一个基本上无法实现的设想，设想通过按下一个按钮就完成了所有的测试工作。自动化测试是不是全能的，手工测试是永远无法完全替代的。

有些测试受测试环境的影响很大，往往需要采用人工方法获取测试结果，分析测试结果。因此，很难在预先知道设计的测试用例有多大的重用性。自动化测试还需要考虑成本问题，因此，千万不要陷入到一切测试都采用自动化方法的错误观念中。

我曾经主张保证给与测试自动化持续不断的投入。但是，在开展自动化测试的时候，一个问题摆在面前，测试自动化应该及时的提供给测试执行人员，这个不成问题，但是如何保证需求变更后，能够及时提供更新后的自动化测试就是个大问题了。如果自动化测试与需求

变更无法同步，那么自动化测试的效果就无法保证了，测试人员就不愿意花费时间学习如何使用新的测试工具和如何诊断测试工具上报的错误。识别项目计划中的软件发布日期，然后把这个日期作为里程碑，并计划达到这个里程碑。当达到这个里程碑后，自动化工程师应该做什么呢？如果自动化工程师关注当前产品版本的发布，他需要为测试执行人员提供帮助和咨询，但是，一旦测试执行人员知道如何使用自动化测试，自动化测试工程师可以考虑下一个版本的测试自动化工作，包括改进测试工具和相关的库。当开发人员开始设计产品下一个版本中的新特性的时候，如果考虑了自动化测试需求，那么自动化测试师的设计工作就很好开展了，采用这种方法，自动化测试工程师可以保持与开发周期同步，而不是与测试周期同步。如果不采用这种方式，在产品版本升级的过程中，自动化测试无法得到进一步的改进。

持续在在自动化投入，你会面临成功的挑战，当自动化测试成为测试过程可靠的基础后，自动化测试的道路将会越来越平坦。

参考文献：

Bach, James. 1996. “Test Automation Snake Oil.” *Windows Technical Journal* , (October): 40-44. http://www.satisfice.com/articles/test_automation_snake_oil.pdf

Dustin, Elfriede. 1999. “Lessons in Test Automation.” *Software Testing and Quality Engineering* (September): 16-22. http://www.stickyminds.com/sitewide.asp?ObjectId=1802&ObjectType=ART&Function=edetail

Fewster, Mark and Dorothy Graham. 1999. *Software Test Automation*, Addison-Wesley.

Groder, Chip. “Building Maintainable GUI Tests” in [Fewster 1999].

Kit, Edward. 1999. “Integrated, Effective Test Design and Automation.” *Software Development* (February). http://www.sdmagazine.com/articles/1999/9902/9902b/9902b.htm

Hancock, Jim. 1997 “When to Automate Testing.” *Testers Network* (June). http://www.data-dimensions.com/Testers'Network/jimauto1.htm

Hendrickson, Elisabeth. 1999. “Making the Right Choice: The Features you Need in a GUI Test Automation Tool.” *Software Testing and Quality Engineering Magazine* (May): 21-25. http://www.qualitytree.com/feature/mtrc.pdf

Hoffman, Douglas. 1999. “Heuristic Test Oracles: The Balance Between Exhaustive Comparison and No Comparison at All.” *Software Testing and Quality Engineering*

Magazine (March): 29-32.

Kaner, Cem. 1997. “Improving the Maintainability of Automated Test Suites.” Presented at Quality Week. http://www.kaner.com/lawst1.htm

Linz, Tilo and Matthias Daigl. 1998. “How to Automate Testing of Graphical User Interfaces.” European Systems and Software Initiative Project No. 24306 (June).http://www.imbus.de/html/GUI/AQUIS-full_paper-1.3.html

Jeffries, Ronald E., 1997, “XPractices,” http://www.XProgramming.com/Practices/xpractices.htm

Marick, Brian. 1998. “When Should a Test Be Automated?” Presented at Quality Week. http://www.testing.com/writings/automate.pdf

Marick, Brian. 1997. “Classic Testing Mistakes.” Presented at STAR. http://www.testing.com/writings/classic/mistakes.html

Pettichord, Bret. 1996. “Success with Test Automation.” Presented at Quality Week (May). http://www.io.com/~wazmo/succpap.htm

Thomson, Jim. “A Test Automation Journey” in [Fewster 1999]

Weinberg, Gerald M. 1992. *Quality Software Management: Systems Thinking*. Vol 1. Dorset House.

缺陷漏测分析：测试过程改进

作者：Mary Ann Vandermark
翻译：周峰
Zhoufeng@withub.org

一、漏测的定义

所谓漏测，是指软件产品的缺陷没有被测试组发现而遗漏到了用户那里，却最终被用户所发现。如果产品在用户那里出现问题，产生的后果是非常严重的。在软件开发过程中，缺陷越早被发现，发现和解决缺陷所花的成本就越小。如果缺陷是在测试组测试中发现的而不是被用户使用时发现的，那么所花的成本将小得多。如果缺陷是被开发组在开发过程中发现的，那么所花的代价将更小。因此，进行漏测分析、预防漏测、促使缺陷尽可能在开发过程的早期被发现，是非常有意义的，它有利于降低软件产品成本、提高软件产品质量。

二、漏测分析的目的

进行漏测分析的目的是为了促进软件质量和开发测试过程得到持续改进。具体来讲，就是通过分析开发和测试过程中漏测的缺陷，制定相应的预防措施以避免今后再发生类似的漏测。测试过程的持续改进将提高测试环境的效果和测试执行的效率、降低遗留到用户处的缺陷数和缺陷解决成本，从而提升软件的质量、声誉和销售。在软件产品开发过程中重视漏测分析并参与到漏测分析工作中的团队越多，漏测分析的效果就越好。如果开发和测试团队都重视漏测分析、并密切配合进行漏测分析工作的话，漏测分析将取得非常好的效果。

在实际工作中，漏测分析过程应该重点关注那些普遍、严重而解决成本高的问题。具体来讲，漏测分析的目标是：

- 对漏测进行分类以便于更进一步深入的分析
- 对分类数据进行统计
- 在统计分析的基础上进行全过程的标识和变更
- 在对一些特殊的漏测项进行分析的基础上，对过程的一些局部进行标识和变更
- 运用度量数据说明过程变更的效果

三、如何进行漏测分析

漏测分析活动可以参照下面的建议进行。在熟悉了漏测分析流程以后，需要确定进行漏测分析活动的频度。为了取得较好的效果，最好是遵照一个时间表来定期进行漏测分析活动，一个月进行一次是一个比较合适的频度。

1. 计划

这个过程是针对多项目组联合进行漏测分析而设置的，在联合项目组中实行该过程最有效。如果不可能组建联合项目组进行漏测分析，也可以修改该过程只在测试组内部实行。

制订计划如果不确定关注点的话，这个计划将难以有效实施。漏测分析要想取得理想的效果，就需要计划好进行漏测分析活动的确切的人员数目、活动时间。 过程执行的效果完全取决于执行它的方式，如果不切切实实的做好计划，你的过程将不会得到太多的改进。

实际进行漏测分析活动时，只选择漏测分类的一部分子集进行分析，将有利于更有效的进行漏测分析工作。进行漏测分类前，需要在计划中确定选择哪部分子集进行分析。例如，如果漏测的严重度等级分为一到四级，一级严重度最高，四级严重度最低，那么也许只分析一、二级的漏测最合适，这样可以避免在那些对用户无关紧要的漏测缺陷上花太多的无用功；也可以只分析那些被关闭和修复了的漏测缺陷，因为如果分析那些没有被关闭和修复的缺陷，可能会漏掉一些至关重要的信息；另外，还可以在进行漏测分析之前排除掉重复缺陷和那些由于用户错误操作引起的缺陷，这样就只需要分析那些有效的漏测缺陷，它们才能真正提供开发和测试过程需要改进的信息。

2. 漏测分类

接下来需要将所有的漏测缺陷按照有意义的属性进行分类，当然，前提条件是已经有了一个包含了所有漏测缺陷详细信息的漏测列表。然后需要确定哪些属性分类是对分析有用的。下面给出一些漏测缺陷的属性的例子：

- 漏测产生活动（最有可能发现该漏测缺陷的活动）
  - 开发阶段（原始需求评审、概念评审、设计评审、代码检视、单元测试、模块联调、信息资料开发）
  - 测试阶段（功能测试、系统测试、本地语言测试、设备驱

动测试、安装测试、性能测试、异常测试）

- 产品模块（产生了漏测缺陷的代码模块）
  - 模块 A、B、C 等
- 缺陷影响（对用户使用造成的影响）
  - 系统崩溃、业务中止、数据完整性、命令失效、安装失败、设备/驱动、性能、文档、可用性等
- 引入版本（引入漏测缺陷的代码版本）
- 平台
- 严重级别
- 发现漏测的版本

漏测产生活动是指在软件开发测试过程中，某类被漏测的缺陷最应该在该活动中被发现。设置该项分类的目的是为了便于对产生了漏测的活动进行更进一步的细化分析。

产品模块是指被漏测的缺陷所在的代码模块。

缺陷影响是指漏测缺陷给用户使用时所带来问题的类型。

引入版本是漏测缺陷被引入时的代码版本，它应该是代码第一次引入该缺陷的版本。

平台是指产生漏测的平台或操作系统。

严重级别是指缺陷的严重程度度量，例如：致命、严重、一般、提示。如果你的缺陷跟踪过程还没有包含缺陷的这个属性，那么漏测分析过程应该明确地给出每个缺陷的严重级别。

发现漏测的版本是指该漏测初次被发现并被报告时的软件版本。

进行漏测分类活动时，最好将开发、测试、技术支持以及其他所有产品生命周期中相关部门的代表组织到一起对近期的漏测进行讨论，特别是技术支持人员能够提供很多非常详细的关于漏测缺陷的信息，这对漏测分类非常有帮助。

在项目组进行实际漏测分析活动时，也许不需要按照上面建议的一些属性进行分类，而需要采用其他一些分类标准，这时最好在项目组内集体讨论来决定哪些分类是最适用的。

在漏测缺陷分类活动结束后，需要对分类结果数据进行统计分析。例如，每个漏测缺陷对应了一个漏测产生的活动，这时可以考虑对该活动进行进一步的改进。

3. 分析活动：跟踪工具

进行漏测分析时如果没有缺陷跟踪工具的支持是很困难的。应该采用工具来维护所有不同分类的漏测缺陷数据。Lotus Notes 数据库就是一个不错的工具，它能很方便地将数据按各种不同的方式进行分割，这样你能够对同样一批数据创建各种视图，从而能够从各个角度进行统计分析。

4. 分析活动：统计

统计分析是为了指导全流程过程改进。进行统计分析首先要确定进行统计分析的频度，一般一个季度进行一次统计分析比较合适。进行统计分析时，需要将某个分类的各分类项的数据一一和该分类的所有其它分类项数据进行比较，并且对所有的分类都要进行这样的操作。对那些相对总数比较大的分类项还要进行更进一步细分，进行更进一步的统计分析工作。

5. 分析活动：全流程过程改进

进行统计分析的时候，漏测分析小组需要集合在一起，对统计分析结果进行讨论。基于统计分析结果可以得到各种趋势图，分析小组可以讨论全开发流程中需要改进的意见和方案，然后对那些需要改进的地方作出正式的改进建议，制定改进实施计划，并在随后的会议上，漏测分析小组对变更实施过程进行讨论。可以通过漏测分析数据库或者其他工具进行任务分配和跟踪。这里可以给出两个根据缺陷分析进行全流程改进的例子：第一个例子，如果在系统在故障处理时发现了很多的漏测缺陷，那么进行开发过程全流程改进时，可以考虑增加异常测试组，加强异常测试；第二个例子，如果用户在某硬件平台上使用软件的过程中发现了大量缺陷而测试组却没有该硬件平台，这时需要考虑改进硬件获取过程，增加测试的硬件平台。全流程改进会给软件企业带来巨大的影响，所以一定要取得管理层的支持和同意。

6. 分析活动：局部过程改进

在联合项目组进行漏测分析时，对每个产生了漏测的活动都要选出代表（如：开发活动代表、测试活动代表、文档写作代表等等）。例如：针对“漏测产生活动”属性进行分类时，如果某漏测缺陷被分类到“单元测试”，那么该漏测缺陷应该由开发活动代表对其进行进一步的局部过程分析。所有这些缺陷都列在漏测分析数据库里，每个分类

活动的代表应该列出归属该活动的所有漏测缺陷列表，然后提出这些活动的局部改进计划。举例来说，测试活动代表应该列出所有“漏测产生活动”为“测试”的漏测缺陷，并进行细分，然后将他们分配给测试工程师进行分析；测试工程师将针对所分配的漏测缺陷进行详细分析，找出漏测的原因，然后提出有针对性的改进计划来防止同类缺陷再次被漏测。这些改进计划应该在审核通过后实施，并且整个改进过程应该在数据库中进行跟踪，每个改进计划都应该能和单个缺陷漏测分析结果相对应，测试代表应该推动各改进计划的完成、审核和实施。这里要特别强调的是，这些改进计划不是用来修复缺陷的，因为这些被分析的漏测缺陷应该已经被修复好了，这些改进计划仅仅是在基于某个缺陷漏测原因分析的基础上重新确定测试过程（或开发过程等），它关心的是如何防止该类问题将来再次发生，而不是关心该特定的缺陷在将来是否会再出现（因为它已经被修复了）。例如，局部过程改进计划可以是补充以前没有考虑到的用例，也可以是在测试环境中增加特定的硬件使得测试环境更接近于用户使用环境。在考虑改进计划的时候应该鼓励创造性。

7. 度量

漏测分析过程的最后一步是对改进过程的阶段性实施效果进行测量。本文后面部分将对此进行更详细的论述。

四、漏测分析举例

图-1

*APAR 是描述缺陷属性的一个术语。

图-1 以一个虚拟产品的 Lotus Notes 漏测缺陷数据库作为示例。在本例中，对一个漏测缺陷采用三个标签项来跟踪其特征数据。第一个标签项用于描述缺陷的详细数据，这些数据来自于缺陷跟踪工具，它们从缺陷跟踪工具到漏测分析工具的输入和转换最好是自动完成。此外，还有一个较大的属性域用来描述缺陷的历史和概要，本图中没有显示出来该域。

图-2

图-2 演示的是“开发分析”标签项。针对开发过程，可以对漏测缺陷进行各种不同的分类。在本例中所示例的漏测缺陷被归类为“设计评审过程”遗漏。这个例子演示了对产品的开发过程变更创建“概念评审”的过程。“捕获概率”项用来评估变更实施后可能产生的效果，它能回答“如果在开发过程中已经实施了该变更计划，这个缺陷被捕获到的可能性有多大？”，对此本文将在后面的“测量”小节进行详细探讨。这个表格还设计了变更计划制订的预期日期项和实施该变更计划的预期日期项，Lotus Notes 系统会在该日期自动发送提示信息给变更计划的受理者。

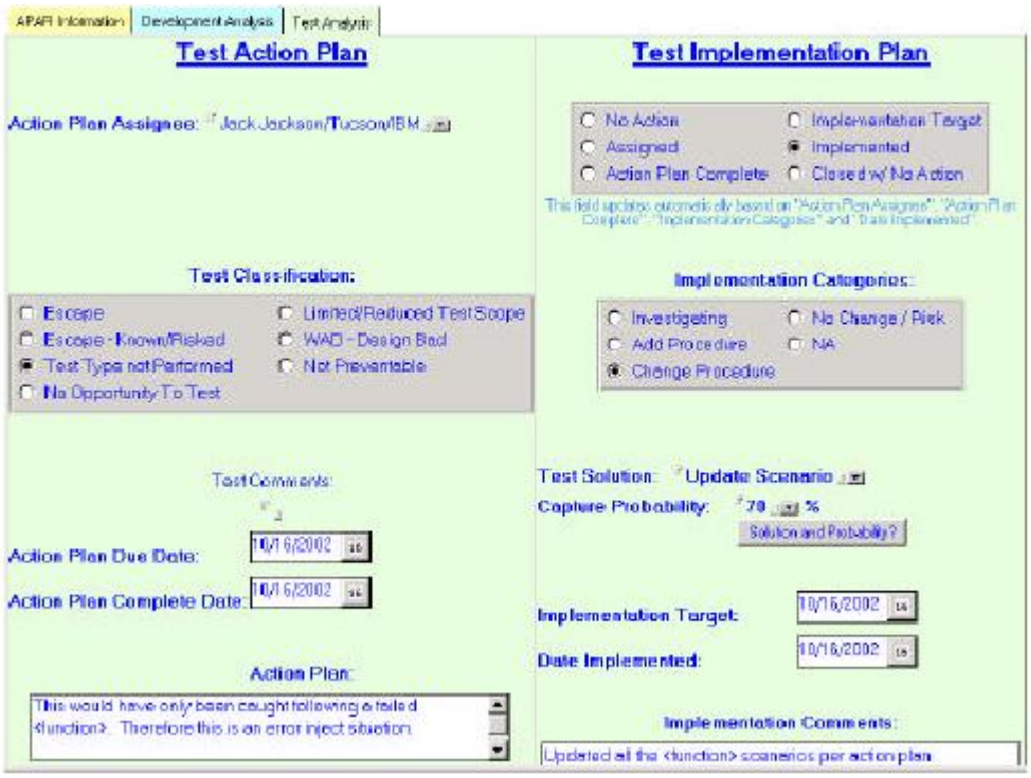


图-3

图-3 演示的是“测试分析”标签项。在这里可以分析用户报告的缺陷是否真的是漏测，换句话说，确定测试过程中是否真的存在漏洞或者该缺陷是否真的值得解决。这是非常重要的。有一些用户发现问题的环境是非常特殊的或生僻的，还有些缺陷解决起来代价很大，并且很难被发现，漏测分析组需要确定是否有必要针对该漏测缺陷进行测试过程改进。如果发现问题的用户是非常重要的客户，并且该用户会经常使用该环境，那么即使发现问题的环境非常特殊，也需要改进测试环境，来尽量符合用户的使用环境。在上面的例子中，缺陷被归类到“未执行的测试类型”，该测试类型发生了漏测。在对数百个漏测缺陷进行统计分析后，如果发现“未执行的测试类型”比例很高，那么可能需要在整个开发过程中增加该类测试类型。这里“捕获概率”项和上面小节描述的含义一样。

五、测量

本节将给出关于测量的一些建议。首先，对于需要改进点，将给出能指导漏测分析组制订合适的改进计划的测量点；接下来，将给出一些评价漏测分析过程效果的方法。您可以采用其中部分或全部建议

来建立自己的测量体系。

1、测量驱动改进

将前面各分类数据和总数比较，得到各分类的比例。下面是一些例子：

图-4 显示的是各代码模块（模块 A－Z）漏测数占总的漏测数的比例。从该饼图上可以清楚地看出超过 50%的漏测来自于 B、C、D、E 四个模块，这个测量结果可以帮助漏测分析组决定是否对这四个模块的开发过程实施改进。

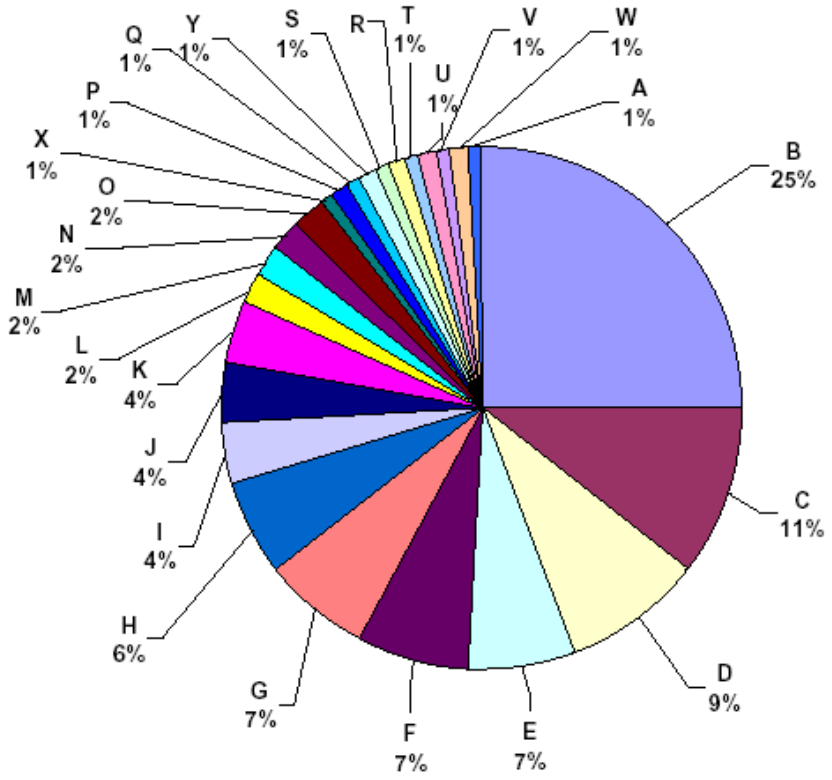


图-4

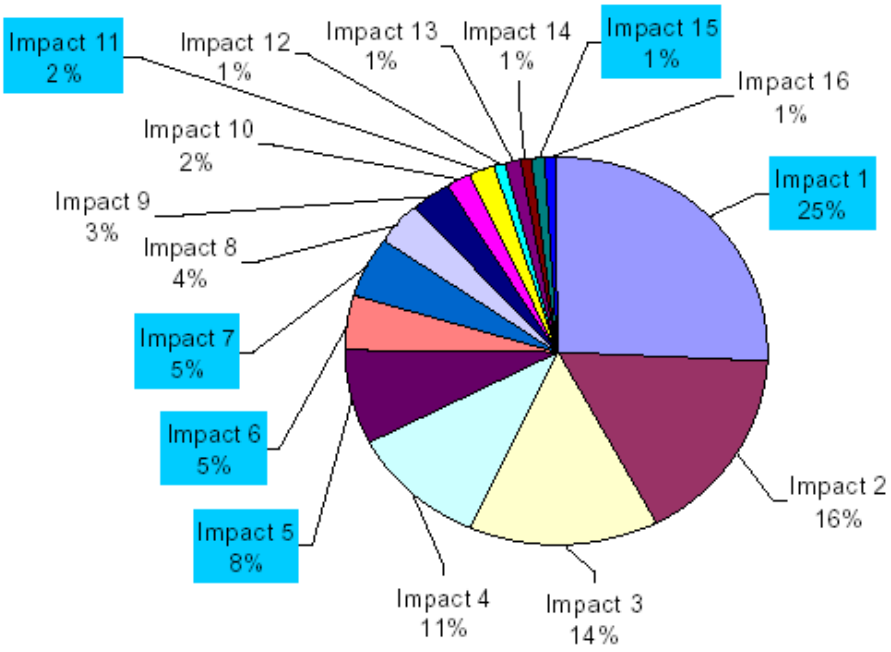


图-5

图-5 分析了漏测缺陷对用户造成的不同影响，如业务中断、系统崩溃、或设备相关问题等。例如，如果“影响 1”是设备相关问题，那么被测软件所在的硬件平台可能需要进行改进；同样，如果蓝色部分是高严重等级的影响类型，那么可以看出漏测是高严重等级影响类型的具体比例是多少。

通过前面示例的数据库工具，还可以输出大量其他图表，上面所举的两个例子只是最常用的两种分析图。

2、评价漏测分析效果

评价改进的效果需要有精确的数据和一致的分析报告，以下几个数据会被用到：

TFVUD 是用户发现缺陷数（Total Field Valid Unique Defects），即由用户发现的经过了确认的、非重复的、非用户错误操作的、非建议类型的所有缺陷；

PDD 是测试发现缺陷数（Post Development Defects），即在开发完成后的测试周期中发现的缺陷数，但它不包括那些向用户发布后发现的缺陷；

KLOC 表示千行代码。

利用上面数据可以得到以下分析数据：

A. TFVUD

千行变更代码&新代码

应当在产品全生命周期中测量上面的值，用作一个版本和另一个版本在相同时间检查点上进行比较的评价指标。例如，一个季度中，2.0 版本的该测量值应该比 1.0 版本低。进行该项测量的目的是减少单位代码规模中用户发现的的有效问题数。

B. PDD

PDD+TFVUD

在产品全生命周期中同样应当测量上面的值，作为一个版本和另一个版本在相同时间检查点上进行比较的评价指标。例如，一个季度中，2.0 版本的该测量值应该比 1.0 版本高。进行该项测量的目的是推动尽早地在开发过程中发现缺陷，从而降低缺陷的修复成本。

C. 捕获概率

在数据库中有“捕获概率”的属性项（在前面小节进行了详细解释），这是对实施过程变更后防止同类问题再次漏测的效果的一项估计指标。该估计是计划预期效果的基础。通过对各变更的捕获概率取总后求平均，可以得到过程变更后的整体预期效果，这样就能对产品发布后用户问题数的降低程度进行合理的预期。

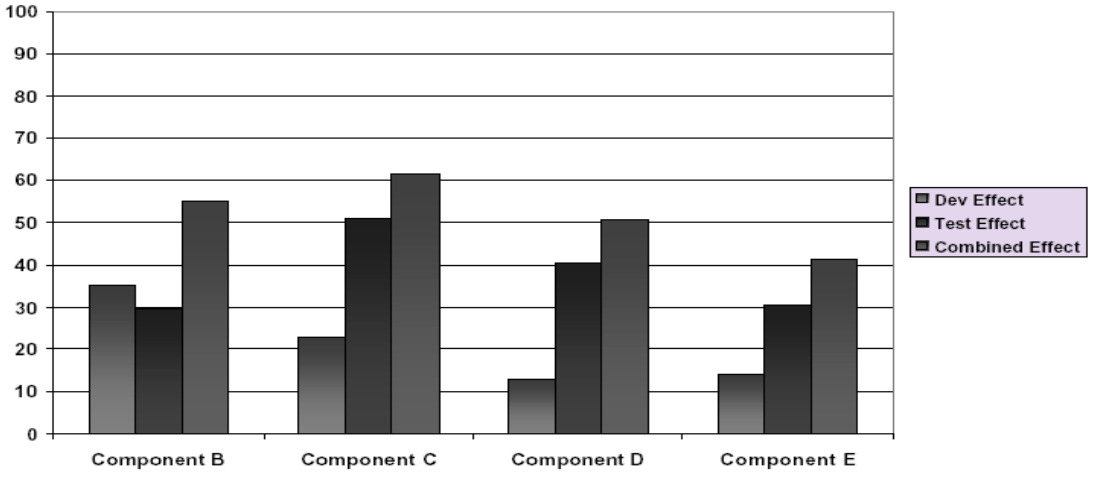


图-6

上图中，模块 B 的开发过程的捕获概率为 35%，测试过程的捕

获概率为 30%。如果开发过程在代码里产生了 100 个缺陷，那么根据捕获概率在开发阶段可能会发现 35 个缺陷，还有 65 个缺陷可能会遗漏到测试阶段，根据测试过程 30%的捕获概率，在测试阶段将可能发现 65*30%＝19.5 个缺陷，那么开发测试阶段总共大概能发现 55 个缺陷。这 55%的概率就是开发测试过程变更后的综合效果估计。用方程式表示上面的过程就是（.35）+(1-.35)(.30)或者 D+(1-D)(T)，这里 D 是开发过程的捕获概率，T 是测试过程捕获概率。本图是基于代码模块的例子，其他分类也可以进行同样的评估工作，如下面图 7。

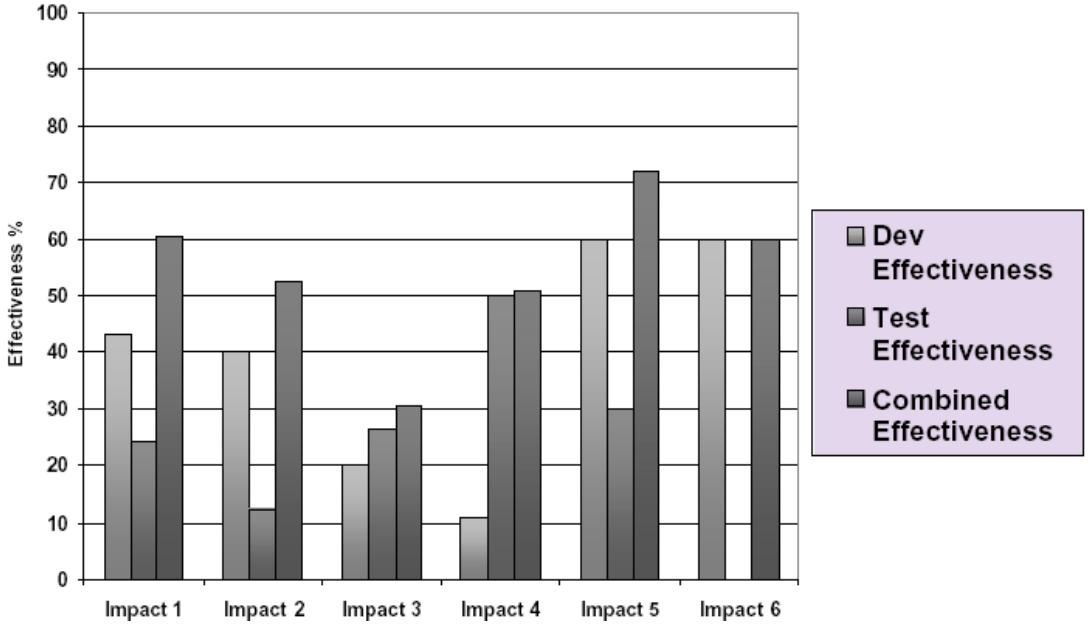


图-7

最后一步是通过对所有综合捕获概率取总后求平均，来预计有效用户缺陷数的减少。首先，选择一组和预期效果相关的重点漏测组。在本例中，假设重点漏测组包含 76 个漏测缺陷，如果针对这 76 个缺陷的综合捕获概率为 52.5%，那么将能预防约 40 个缺陷漏测。假设一年的时间里会有 250 个漏测缺陷，前面 52.5%的捕获概率是一个比较准的数据，那么将能预防 250 个漏测的 16%——约 40 个漏测缺陷，这是对下个版本将会减少的漏测数的最终预测，并且这是最小预测，因为我们只是对重点漏测组进行了预测，这对其他类型的问题可能不适用。如果我们没有作那样的假设，那么预测的漏测数的降低可能是不现实的 52.5%。

六、总结

进行漏测缺陷分析的主要目的就是提高产品质量和用户满意度、降低修复用户发现缺陷的成本。这是通过推动尽可能在软件开发过程的早期发现缺陷来实现的。进行漏测分析活动的软件测试组将会帮助软件开发组改进质量，他们的测试过程将更加完善，测试环境也将更加符合用户实际环境。从漏测分析过程中收集的数据能为测试环境补充硬件等改进活动提供充分的理由。此外，漏测分析过程鼓励项目组间的交流和合作，开发更高质量的软件产品。它还能预测未来的漏测缺陷数，评价自身的效果，来证明所投入的资源是值得的。

Rational Robot 的启动

作者：piaocl3
Pcl2004_19@hotmail.com

【摘要】随着软件产品在各行业应用越来越广泛，软件质量日益得到重视。提高测试水平，测试效率是广大测试工程师追求的目标。Rational 的产品涉及到软件工程各个部分，本文从实际出发介绍控制启动自动化工具 Robot 的方法。文中涉及开发，系统几个方面，希望提供大家一个完美的解决方案。

【关键字】软件测试　自动化测试　Rational Robot　系统启动

这篇文章是基于论坛上一个问题“如何让脚本定时运行”，提供的一个解决方案。我们都听说过自动化测试应用于每日构建，每天在下班之后编译版本，定时启动自动化测试脚本进行功能测试，第二天相关人员收到相应的报告。但是如何启动自动化测试工具呢，这里主要针对于 Rational Robot 的提供一种实现方法。

因为 Rational Robot 支持从命令行启动脚本，所以我们很容易在命令行中启动它。具体参考 Rational Robot User’s Guide。基本的命令行语法结构如下：

Rtrobo.exe [scriptname] [/user userid] [/password PassWrod] [/project Full - path- and – full - projectname] [/play] [/purify] [/quantify] [/coverage] [/build build] [/logfolder foldername] [/log logname] [/nolog] [/close]

元素	描述
Rtrobo.exe	Rational robot 执行文件
scriptname	要运行的脚本名称
/user userid	登陆用户名
/password password	登陆用户名密码，如果用户名没有密码，不用

[3] 岁月匆匆，转眼 27 有余。
现一事无成，深感恐慌。
深知少小不努力，老大徒伤悲。
遂全心投入。
望生活更美好，天涯任我行

	该参数
/Project full path and full projectname	包含脚本的工程名字和工程路径
/play	如果指定了关键字，运行 scriptname 指定的脚本，如果没有指定，只是在编辑器中打开脚本
/purify	和/play 参数一起使用，在 rational Purify 下回放 scriptname 参数指定的脚本
/quantify	和/play 参数一起使用，在 rational quantify 下回放 scriptname 参数指定的脚本
/coverage	和/play 参数一起使用，在 rational purecoverage 下回放 scriptname 参数指定的脚本
/build build	脚本编译名字
/logfolder foldername	存放测试日志文件的文件夹名称，日志文件夹和 build 参数相关联
/log logname	日志文件名称
/nolog	当脚本回放时不输出任何日志文件
/close	回放脚本后关闭 robot

例：用户 admin 启动 testProject 工程脚本 test，命令行语法如下：

rtrobo.exe test /user admin /project c:\Program\robot\testDelphi\testProject.rsp /play

那如何定时启动呢，在 Windows 系统中有“任务计划”提供了定时启动功能，通过它可以预定一些软件在规定的时间运行。利用“任务计划”，可以将任何脚本、程序或文档安排在某个最方便的时间运行。“任务计划”在每次启动 Windows 的时候开始启动并在后台运行。那么我们完全可以利用它来完成“不可能的任务”。

在计划任务中添加任务那么就可以定时启动 Rational robot 运行脚本。操作如下：

1．开始 -〉设置 -〉控制面板

2．打开计划任务向导

3．添加启动任务，设置启动时间

4．修改计划任务运行中参数如下：

"c:\Program Files\Rational\Rational Test\rtrobo.exe" test /user admin /project c:\Program\robot\testDelphi\testProject.rsp /play

Rational Robot 在预定时间定时启动了。问题得到解决。

备注： 运行过程中仍需要设置相关参数，否则阻止运行。比如默认情况下，脚本运行都会启动编译窗口。这里除了命令行设置参数外还可以设置 Robot 中 GUI Playback Option 窗体的 log 页面 use default log information 项。这样在无人职守的情况下，启动测试工具不会由于测试工具自身设置中止运行。

但是实际工作需要更加灵活的功能，比如曾经笔者所在公司开发软件为组态监控软件，利用组态软件开发的一个项目是控制大厦的空调系统在夜里定时重新启动。进行功能测试时候，我们不可能真的等到夜里才进行测试（估计夜里上班公司也不会批准给加班费 ^_^），按照计划任务的实现方法有一定局限性不能随心所欲启动机器进行功能测试。

熟悉 Windows 编程的人都知道 API 函数 ExitWindowsEx 可以实现重新启动系统的功能(具体的用法参看 MSDN)。这样只要调用 ExitWindowEx 函数重新启动系统后启动 Rational Robot 运行需要的脚本就可以达到任何时候启动系统继续进行功能测试的目的。

一般情况下登陆系统时需要输入登陆名和密码，如果不能自动登陆，那么就成为运行脚本进行测试的障碍。在注册表 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon 位置添加 DefaultDomainName,DefaultUserName,DefaultPassword 三项，赋予相应的值。重新启动系统后自动登陆。

那么最后面对的一个问题就是怎样让 robot 随着系统而启动。这个相对来说比较简单，在系统开始启动菜单中建立一个快捷方式就可以实现随着系统启动的功能，其实就是在注册表 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run 中添加键值。附录提供笔者实现的 robot 脚本代码供大家参考。

Brian Bryson 提供组件 – RobotHelperComponent 帮助我们实现利用 Robot 重新启动系统并启动 Rational Robot 运行脚本的功能。该组件下载地址为：

http://www-106.ibm.com/developerworks/rational/library/content/03July/2000/2155/2155_RobotHelperComponent.zip

关于该组件的介绍请看 Brian Bryson 的 Rebooting Your System 一文。该文地址：

http://www-106.ibm.com/developerworks/rational/library/959.html

本文到此告以结束，希望大家对于 Rational Robot 的启动问题有了一个满意的答案。

参考：

1. Microsoft MSDN2003
2. Rational Robot User Guide
3. Rational Robot Tips and Techniques: Rebooting Your System – 作者 Brian Bryson

附录：

```
Const REG_SZ=1
Global Const HKEY_LOCAL_MACHINE =  &H80000002
Declare Function RegOpenKey Lib“advapi32” Alias “RegOpenKeyA”(ByVal hkey As Long, ByVal lpszSubKey As String,phkResult As Long) As Long

Declare Function RegSetValueEx Lib “advapi32” Alias “RegSetValueExA” (ByVal hkey As Long,ByVal lpszValueName As String,ByVal dwReserved As Long,ByVal fdwType As Long, lpbData As Any,ByVal cbData As Long) As Long

Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, lpType As Long, lpData As Any, lpcbData As Long) As Long

Declare Function RegCloseKey Lib “advapi32” Alias “RegCloseKey” (ByVal hkey As Long) As Long
```

编写功能函数

```
Function oRegOpenKey (ByVal hkey As Long, ByVal lpszSubKey As String,phkResult As Long) As Boolean
   Dim lResult As Long
```

```
    On Error GoTo 0 '错误跳转
   lResult=RegOpenKey(hkey,lpszSubKey,phkResult)
   If lResult=0 Then
      oRegOpenKey=True
   Else oRegOpenKey=False
   End If
End Function

Function RegSetStringValue(ByVal hKey As Long, ByVal strValueName As String, ByVal
strData As String, Optional ByVal fLog) As Boolean
      Dim lResult As Long
       On Error GoTo 0
       lResult = RegSetValueEx(hKey, strValueName, 0&, REG_SZ, ByVal strData,
LenB(StrConv(strData, vbFromUnicode)) + 1)
       If lResult = 0 Then
            RegSetStringValue = True
       Else
             RegSetStringValue = False
      End If

End Function
```

调用函数测试代码

```
Sub Main

Dim hkey As Long
Dim MyReturn As Long
MyReturn=oRegOpenkey(HKEY_LOCAL_MACHINE,
"SOFTWARE\Microsoft\Windows\CurrentVersion\Run",hkey)
If hkey=0 Then
      MsgBox "主键未创建或者输入有误，请仔细核对后再次运行本程序！"
      Exit Sub
End If
MyReturn=RegSetStringValue(hkey, "robot","""路径/rtrobo.exe"" test /user admin /project
```

```
c:\Program\robot\testDelphi\testProject.rsp    /play",False)

If MyReturn Then
MsgBox "您的程序已成功添加到系统启动中，再次启动 Windows 时系统将自动运行您的程序！"
Else
MsgBox "代码中存在错误"
End If
RegCloseKey(hkey)

End Sub
```

以上代码在 Rational Robot 2002 上测试通过。

JAVA 测试模式

原著：Marco Torchiano
翻译改编：张　华
Thinker@csai.com.cn

测试模式是一种针对软件测试领域的某种高频率出现问题而采取并经过实践证明行之有效的专门化、高效的解决途径（方法），它在软件理论和实践两者之间起着“桥梁”的作用。在面向对象语言JAVA程序测试的过程中，一个较为棘手的问题就是JAVA类的可视性问题。“信息隐蔽”固然是面向对象语言设计的一个突出的优点，但是同时也给测试带来诸多不便，有关“隐蔽信息”的可测试性成为这类测试的一大突出结症。为此我们针对JAVA类不同的“可视性”要求的场合，采取相应的测试模式来支撑相关JAVA类（包）的测试。本文主要总结五个常用的JAVA测试模式，以飨读者。这五个模式同样适合于其它面向对象类（包）的测试，只不过在具体细节上要考虑与JAVA语言信息隐蔽性的差异。比如：JAVA的可视性以包为界，同一个包内的类对其他类具有相同的存取权限。而C++则以类为界，只有子类和友员函数方可对基类的隐蔽信息进行存取。

我们在下面以图表的方式给出五个常用JAVA测试模式详细描述，有关模式的描述类目分别是模式名称、测试对象、针对问题、约束条件、解决方法、实例、约束解决方式和该测试的设计原理。

模式（一） Main模式

模式名称	Main
测试对象	JAVA类
针对问题	测试人员不知道在何处编写驱动和初始化被测试类的测试代码
约束	测试必须容易运行、测试代码能够访问该类所有的特征（所有的属性和方法）
解决方法	将测试代码放入类的public static void main(String[] args)方法中去
实例	*public static void main(String[] args){* *SomeClass result;*

	// perform the test… *System.out.print("result is..");* *}*
约束解决	该测试代码能够率先被激活和执行、并能够访问被测试类所有的特征
设计原理	JAVA类可以拥有public static void main(String[] args)方法，它是类在JVM中被率先执行的方法，控制着整个类的执行逻辑，main方法能够访问所在类的所有属性和方法

模式（二） toString 模式

模式名称	toString
测试对象	作为运算结果的类
针对问题	测试人员不知道如何检验一个运算对象的中间结果和最终结果
约束	测试结果代表对象内部的一个状态，而该状态必须能够被测试
解决方法	通过使用toString方法来对类状态进行描述，描述结果可以通过打印与预期结果进行比较
实例	*class SomeClass {* *//…* *public String toString(){* *// custom representation* *}* *}*
约束解决	类的内部状态通过字符串来进行表示因而得以解决
设计原理	在基类定义的toString()方法能够提供特定对象的状态的描述，通过将对象状态描述进行打印和显示来判定对象状态是否与预期相符

模式（三）Equal 模式

模式名称	Equal
测试对象	JAVA类
针对问题	作为运算结果的类
约束	测试人员不知道如何检验一个运算对象的中间结果和最终结果
解决方法	定义equals方法比较实际结果与预期结果进行比较
实例	*class SomeClass {* *//...* *public boolean equals(Object o){* *// custom comparison* *}* *}*
约束解决	保持某个状态的对象需要与预期对象进行比较，通过比较后的布尔值来确定是否与预期结果相符合
设计原理	在基类定义的public boolean equals(Object)能够对比对象间的差异并以布尔值形式返回比较结果

模式（四）Internal Tester Class 模式

模式名称	Internal Tester Class
测试对象	JAVA包
针对问题	测试人员不知在何处编写测试代码测试包中的类
约束	测试代码与运行代码必须分开来，测试代码能够访问到类的所有特性，测试代码起到的是一个包的客户端的角色。
解决方法	新建一个类并将该类的包路径与被测试类的包路径相同，然后在该类里写入所有的测试代码
实例	*package theOne;* *public class InternalTestClass {* *//...* *}*
约束解决	所有的测试代码均与运行代码分离，由于它与运行代码在同一

	包路径下，所以它与普通的客户端相比具有更多的可视性
设计原理	将测试代码定义在一个特定的类中。由于测试类与运行类在同一个包路径下，因此它能访问测试类的类的所有特性。因此，它远比客户端测试具有更大的可视性。

模式（五）Extern Tester Class 模式

模式名称	Extern Tester Class 模式
测试对象	JAVA包
针对问题	测试人员不知在何处编写测试代码测试包中的类
约束	测试代码与运行代码必须分开来，测试代码能够访问到类的可视特征，测试代码起到的是一个包的客户端的角色。
解决方法	新建一个类，指定与被测试类不同的包路径，然后在该类里写入所有的测试代码
实例	*package anotherOne;* *public class ExternalTestClass {* *//...* *}*
约束解决	所有的测试代码均与运行代码分离，由于它与运行代码在不同的包路径下，所以它与普通的客户端的可视性相同
设计原理	将测试代码定义在一个特定的类中。由于测试类与运行类不在同一个包路径下，因此它不能访问测试类的所有属性和方法。但是它与客户端具有相同的可视性。因此，它可以替代客户端进行测试。

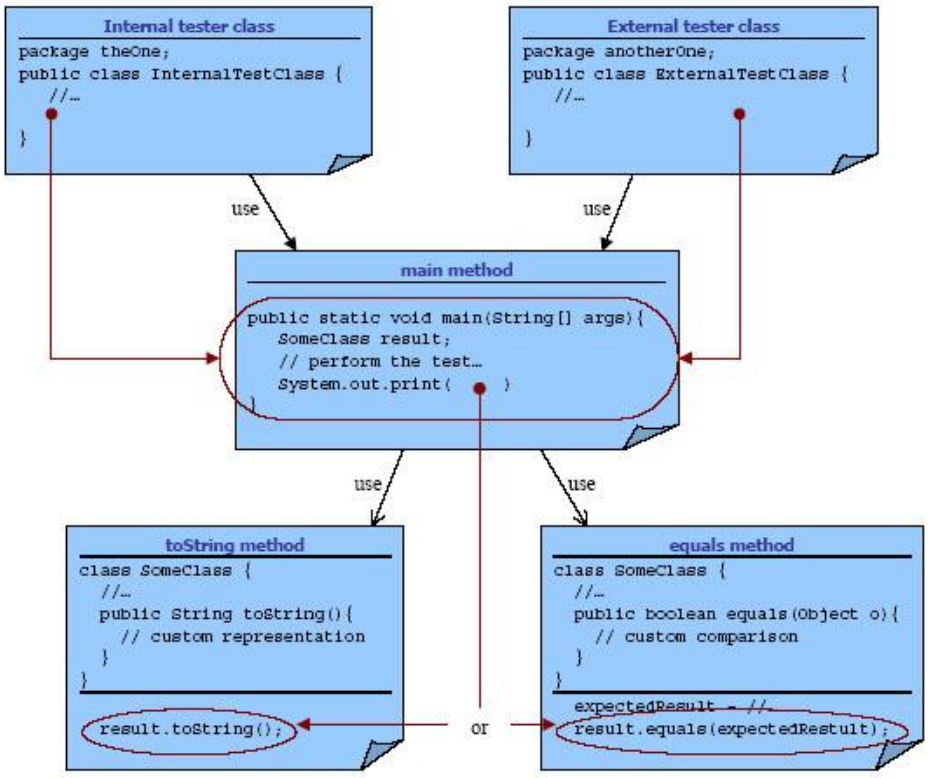


Figure 1: Java testing pattern-language.

图-1 JAVA测试模式图解

了解上述所述的模式有助于测试人员在具体JAVA代码测试中编写测试类，同时上述的这些JAVA测试模式还可以借助JUnit这样的测试框架来实现。

注：本文根据Marco Torchiano的《Patterns for Java Program Testing》删减改编而成