一、目的与适用范围

1、目的

软件测试是软件工程的重要组成部分，测试工作的质量直接影响软件产品的生命力。测试工作的标准化是软件质量保证(Quality Assurance)重要而且必须的环节。制定本标准的目的在于使测试流程更标准，测试过程更规范。从而使整个软件生产纳入更系统化、更专业化的轨道。

2、适用范围

本标准适用于软件测试流程的管理和测试的具体操作过程。本标准的使用者可以是企业内部的测试人员和开发人员。

二、测试方法

软件测试的方法和技术是多种多样的。以下将介绍比较常用的一些测试方法：

1、静态测试

静态方法是指不运行被测程序本身，仅通过分析或检查源程序的文法、结构、过程、接口等来检查程序的正确性。静态方法通过程序静态特性的分析，找出欠缺和可疑之处，例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。静态测试结果可用于进一步的查错，并为测试用例选取提供指导。

2、动态测试

动态方法是指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能，这种方法由三部分组成：构造测试实例、执行程序、分析程序的输出结果。

3、黑盒测试

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。黑盒测试方法主要有等价类划分、边值分析、因—果图、错误推测等，主要用于软件确认测试。

“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

4、白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能，白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

“白盒”法全面了解程序内部逻辑结构、对所有逻辑路径进行测试。“白盒”法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数是天文数字。但即使每条路径都测试了仍然可能有错误。第一，穷举路径测试决不能查出程序违反了设计规范，即程序本身是个错误的程序。第二，穷举路径测试不可能查出程序中因遗漏路径而出错。第三，穷举路径测试可能发现不了一些与数据相关的错误。

5、ALAC(Act-like-a-customer)测试

ALAC 测试是一种基于客户使用产品的知识开发出来的测试方法。ALAC 测试是基于复杂的软件产品有许多错误的原则。最大的受益者是用户，缺陷查找和改正将针对哪些客户最容易遇到的错误。

6、单元测试方法

6.1 单元测试任务

单元测试任务包括：

* 模块接口测试；
* 模块局部数据结构测试；
* 模块边界条件测试；
* 模块中所有独立执行通路测试；
* 模块的各条错误处理通路测试。

模块接口测试是单元测试的基础。只有在数据能正确流入、流出模块的前提下，其他测试才有意义。

6.2 接口测试

测试接口正确与否应该考虑下列因素：

* 输入的实际参数与形式参数的个数是否相同；
* 输入的实际参数与形式参数的属性是否匹配；
* 输入的实际参数与形式参数的量纲是否一致；
* 调用其他模块时所给实际参数的个数是否与被调模块的形参个数相同；
* 调用其他模块时所给实际参数的属性是否与被调模块的形参属性匹配；
* 调用其他模块时所给实际参数的量纲是否与被调模块的形参量纲一致；
* 调用预定义函数时所用参数的个数、属性和次序是否正确；
* 是否存在与当前入口点无关的参数引用；
* 是否修改了只读型参数；
* 对全程变量的定义各模块是否一致；
* 是否把某些约束作为参数传递。

如果模块内包括外部输入输出，还应该考虑下列因素：

* 文件属性是否正确；

* OPEN/CLOSE 语句是否正确；

* 格式说明与输入输出语句是否匹配；

* 缓冲区大小与记录长度是否匹配；

* 文件使用前是否已经打开；

* 是否处理了文件尾；

* 是否处理了输入/输出错误；

* 输出信息中是否有文字性错误；

6.3 数据测试

检查局部数据结构是为了保证临时存储在模块内的数据在程序执行过程中完整、正确。局部数据结构往往是错误的根源，应仔细设计测试用例，力求发现下面几类错误：

* 不合适或不相容的类型说明；

* 变量无初值；

* 变量初始化或省缺值有错；

* 不正确的变量名（拼错或不正确地截断）；

* 出现上溢、下溢和地址异常。

除了局部数据结构外，如果可能，单元测试时还应该查清全局数据（例如 FORTRAN 的公用区）对模块的影响。

6.4 控制流测试

在模块中应对每一条独立执行路径进行测试，单元测试的基本任务是保证模块中每条语句至少执行一次。此时设计测试用例是为了发现因错误计算、不正确的比较和不适当的控制流造成的错误。此时基本路径测试和循环测试是最常用且最有效的测试技术。计算中常见的错误包括：

* 误解或用错了算符优先级；

* 混合类型运算；

* 变量初值错；

* 精度不够；

* 表达式符号错。

比较判断与控制流常常紧密相关，测试用例还应致力于发现下列错误：

* 不同数据类型的对象之间进行比较；

* 错误地使用逻辑运算符或优先级；

* 因计算机表示的局限性，期望理论上相等而实际上不相等的两个量相等；

* 比较运算或变量出错；

* 循环终止条件或不可能出现；

* 迭代发散时不能退出；

* 错误地修改了循环变量。

6.5 出错处理测试

一个好的设计应能预见各种出错条件，并预设各种出错处理通路，出错处理通路同样需要认真测试，测试应着重检查下列问题：

* 输出的出错信息难以理解；
* 记录的错误与实际遇到的错误不相符；
* 在程序自定义的出错处理段运行之前，系统已介入；
* 异常处理不当；
* 错误陈述中未能提供足够的定位出错信息。

6.6 边界条件测试

边界条件测试是单元测试中最后，也是最重要的一项任务。众的周知，软件经常在边界上失效，采用边界值分析技术，针对边界值及其左、右设计测试用例，很有可能发现新的错误。

7、集成测试的基本方法

某设计人员习惯于把所有模块按设计要求一次全部组装起来，然后进行整体测试，这称为非增量式集成。这种方法容易出现混乱。因为测试时可能发现一大堆错误，为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，新旧错误混杂，更难断定出错的原因和位置。与之相反的是增量式集成方法，程序一段一段地扩展，测试的范围一步一步地增大，错误易于定位和纠正，界面的测试亦可做到完全彻底。下面讨论两种增量式集成方法。

7.1 自顶向下集成

自顶向下集成是构造程序结构的一种增量式方式，它从主控模块开始，按照软件的控制层次结构，以深度优先或广度优先的策略，逐步把各个模块集成在一起。深度优先策略首先是把主控制路径上的模块集成在一起，至于选择哪一条路径作为主控制路径，这多少带有随意性，一般根据问题的特性确定。

自顶向下集成测试的具体步骤为：

* 以主控模块作为测试驱动模块，把对主控模块进行单元测试时引入的所有桩模块用实际模块替代；
* 依据所选的集成策略（深度优先或广度优先），每次只替代一个桩模块；
* 每集成一个模块立即测试一遍；
* 只有每组测试完成后，才着手替换下一个桩模块；
* 为避免引入新错误，须不断地进行回归测试（即全部或部分地重复已做过的测试）；
* 从第二步开始，循环执行上述步骤，直至整个程序结构构造完毕。

自顶向下集成的优点在于能尽早地对程序的主要控制和决策机制进行检验，因此较早地发现错误。缺点是在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，因此测试并不充分。解决这个问题有几种办法，第一种是把某些测试推迟到用真实模块替代桩模块之后进行，第二种是开发能模拟真实模块的桩模块；第三种是自底向

上集成模块。第一种方法又回退为非增量式的集成方法，使错误难于定位和纠正，并且失去了在组装模块时进行一些特定测试的可能性；第二种方法无疑要大大增加开销；第三种方法比较切实可行。

7.2 自底向上集成

自底向上测试是从“原子”模块（即软件结构最低层的模块）开始组装测试，因测试到较高层模块时，所需的下层模块功能均已具备，所以不再需要桩模块。

自底向上综合测试的步骤分为：

* 把低层模块组织成实现某个子功能的模块群（cluster）；
* 开发一个测试驱动模块，控制测试数据的输入和测试结果的输出；
* 对每个模块群进行测试；
* 删除测试使用的驱动模块，用较高层模块把模块群组织成为完成更大功能的新模块群；
* 从第一步开始循环执行上述各步骤，直至整个程序构造完毕。

自底向上集成方法不用桩模块，测试用例的设计亦相对简单，但缺点是程序最后一个模块加入时才具有整体形象。它与自顶向综合测试方法优缺点正好相反。因此，在测试软件系统时，应根据软件的特点和工程的进度，选用适当的测试策略，有时混和使用两种策略更为有效，上层模块用自顶向下的方法，下层模块用自底向上的方法。

此外，在集成测试中尤其要注意关键模块，所谓关键模块一般都具有下述一或多个特征：①对应几条需求；②具有高层控制功能；③复杂、易出错；④有特殊的性能要求。关键模块应尽早测试，并反复进行回归测试。

8、确认测试的基本方法

8.1 确认测试标准

实现软件确认要通过一系列黑盒测试。确认测试同样需要制订测试计划和过程，测试计划应规定测试的种类和测试进度，测试过程则定义一些特殊的测试用例，旨在说明软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、准确，人机界面和其他方面（例如，可移植性、兼容性、错误恢复能力和可维护性等）是否令用户满意。

确认测试的结果有两种可能，一种是功能和性能指标满足软件需求说明的要求，用户可以接受；另一种是软件不满足软件需求说明的要求，用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正，因此必须与用户协商，寻求一个妥善解决问题的方法。

8.2 配置复审

确认测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必须的细节。

8.3 α、β测试

事实上，软件开发人员不可能完全预见用户实际使用程序的情况。例如，用户可能错误的理

解命令，或提供一些奇怪的数据组合，亦可能对设计者自认明了的输出信息迷惑不解，等等。因此，软件是否真正满足最终用户的要求，应由用户进行一系列“验收测试”。验收测试既可以是非正式的测试，也可以有计划、有系统的测试。有时，验收测试长达数周甚至数月，不断暴露错误，导致开发延期。一个软件产品，可能拥有众多用户，不可能由每个用户验收，此时多采用称为α、β测试的过程，以期发现那些似乎只有最终用户才能发现的问题。

α测试是指软件开发公司组织内部人员模拟各类用户行对即将面市软件产品（称为α版本）进行测试，试图发现错误并修正。α测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作并尽最大努力涵盖所有可能的用户操作方式。经过α测试调整的软件产品称为β版本。紧随其后的β测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用β版本，并要求用户报告异常情况、提出批评意见。然后软件开发公司再对β版本进行改错和完善。

9、系统测试的基本方法

9.1 恢复测试

恢复测试主要检查系统的容错能力。当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。恢复测试首先要采用各种办法强迫系统失败，然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化(reinitialization)、检查点(checkpointing mechanisms)、数据恢复(data recovery)和重新启动 (restart)等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。

9.2 安全测试

安全测试检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。例如，①想方设法截取或破译口令；②专门定做软件破坏系统的保护机制；③故意导致系统失败，企图趁恢复之机非法进入；④试图通过浏览非保密数据，推导所需信息，等等。理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值。此时非法侵入者已无利可图。

9.3 强度测试

强度测试检查程序对异常情况的抵抗能力。强度测试总是迫使系统在异常的资源配置下运行。例如，①当中断的正常频率为每秒一至两个时，运行每秒产生十个中断的测试用例；②定量地增长数据输入率，检查输入子功能的反映能力；③运行需要最大存储空间（或其他资源）的测试用例；④运行可能导致虚存操作系统崩溃或磁盘数据剧烈抖动的测试用例，等等。

9.4 性能测试

对于那些实时和嵌入式系统，软件部分即使满足功能要求，也未必能够满足性能要求，虽然从单元测试起，每一测试步骤都包含性能测试，但只有当系统真正集成之后，在真实环境中才能全面、可靠地测试运行性能系统性能测试是为了完成这一任务。性能测试有时与强度测试相结合，经常需要其他软硬件的配套支持。

10、回归测试方法

回归测试的价值在于它是一个能够检测到回归错误的受控实验。当测试组选择缩减的回归测试时，有可能删除了将揭示回归错误的测试用例，消除了发现回归错误的机会。然而，如果采用了代码相依性分析等安全的缩减技术，就可以决定哪些测试用例可以被删除而不会让回归测试的意图遭到破坏。

选择回归测试策略应该兼顾效率和有效性两个方面。常用的选择回归测试的方式包括：

10.1 再测试全部用例：

选择基线测试用例库中的全部测试用例组成回归测试包，这是一种比较安全的方法，再测试全部用例具有最低的遗漏回归错误的风险，但测试成本最高。全部再测试几乎可以应用到任何情况下，基本上不需要进行分析和重新开发，但是，随着开发工作的进展，测试用例不断增多，重复原先所有的测试将带来很大的工作量，往往超出了我们的预算和进度。

10.2 基于风险选择测试：

可以基于一定的风险标准来从基线测试用例库中选择回归测试包。首先运行最重要的、关键的和可疑的测试，而跳过那些非关键的、优先级别低的或者高稳定的测试用例，这些用例即便可能测试到缺陷，这些缺陷的严重性也仅有三级或四级。一般而言，测试从主要特征到次要特征

10.3 基于操作剖面选择测试：

如果基线测试用例库的测试用例是基于软件操作剖面开发的，测试用例的分布情况反映了系统的实际使用情况。回归测试所使用的测试用例个数可以由测试预算确定，回归测试可以优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障。这种方法可以在一个给定的预算下最有效的提高系统可靠性，但实施起来有一定的难度。

10.4 再测试修改的部分：

当测试者对修改的局部化有足够的信心时，可以通过相依性分析识别软件的修改情况并分析修改的影响，将回归测试局限于被改变的模块和它的接口上。通常，一个回归错误一定涉及一个新的、修改的或删除的代码段。在允许的条件下，回归测试尽可能覆盖受到影响的部分。

再测试全部用例的策略是最安全的策略，但已经运行过许多次的回归测试不太可能揭示新的错误，而且很多时候，由于时间、人员、设备和经费的原因，不允许选择再测试全部用例的回归测试策略，此时，可以选择适当的策略进行缩减的回归测试。

三、测试阶段的划分

根据开发过程和实际需求将测试阶段划分为：设计阶段、代码检测单元测试阶段、集成测试阶段、系统测试阶段、验收测试阶段、回归测试（复测）阶段。各阶段中使用的测试方法详见本规范的测试方法。

1、设计阶段

核心工作是对软件产品功能说明书进行检查，软件产品功能说明书是对软件产品最终需要实现的功能的描述。编写软件测试计划。

2、单元测试阶段

单元测试完成对软件最小的结构的测试，一般用来验证模块的功能属性，它利用设计文档作为指导，主要使用白盒测试技术；但也可以测试其它项目，如性能、可用性等等，可使用“黑盒”或“白盒”方法进行。在单元测试中，检查出模块内部的错误是单元测试的主要工作。该阶段的测试工作，由编程组内部人员进行交叉测试（避免编程人员测试自己的程序）。

单元测试过程： 一般认为单元测试应紧接在编码之后，当源程序编制完成并通过复审和编译检查，便可开始单元测试。测试用例的设计应与复审工作相结合，根据设计信息选取测试数据，将增大发现上述各类错误的可能性。在确定测试用例的同时，应给出期望结果。

提高模块的内聚度可简化单元测试，如果每个模块只能完成一个，所需测试用例数目将显著减少，模块中的错误也更容易发现。

3、集成测试阶段

时常有这样的情况发生，每个模块都能单独工作，但这些模块集成在一起之后却不能正常工作。主要原因是，模块相互调用时接口会引入许多新问题。例如，数据经过接口可能丢失；一个模块对另一模块可能造成不应有的影响；几个子功能组合起来不能实现主功能；误差不断积累达到不可接受的程度；全局数据结构出现错误，等等。集成测试是组装软件的系统测试技术，按设计要求把通过单元测试的各个模块组装在一起之后，进行集成测试以便发现与接口有关的各种错误。

4、确认测试阶段

确认测试的目的是向未来的用户表明系统能够像预定要求那样工作。经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是确认测试的任务，即软件的功能和性能如同用户所合理期待的那样。

5、系统测试阶段

计算机软件是基于计算机系统的一个重要组成部分，软件开发完毕后应与系统中其它成分集成在一起，此时需要进行一系列系统测试。包括恢复测试、安全测试、强度测试和性能测试等。在系统测试之前，软件工程师应完成下列工作：

（1） 为测试软件系统的输入信息设计出错处理通路；

（2） 设计测试用例，模拟错误数据和软件界面可能发生的错误，记录测试结果，为系统测试提供经验和帮助；

（3） 参与系统测试的规划和设计，保证软件测试的合理性。

系统测试应该由若干个不同测试组成，目的是充分运行系统，验证系统各部件是否都能工作并完成所赋予的任务。

6、回归测试（复测）阶段

回归测试就是漏洞修复完成后再对软件进行测试，以确保软件没有产生“回归”或因修复而变得更糟，这种测试一般要重新运行最初发现问题的原始测试程序。有关回归测试有两个焦点：有

没有产生新的漏洞，修复是否确实使缺陷消除。

回归测试的过程：

有了测试用例库的维护方法和回归测试包的选择策略，回归测试可遵循下述过程进行：

* 识别出软件中被修改的部分

* 从原基线测试用例库中排除所有不再适用的测试用例，确定那些对新的软件版本依然有效的测试用例

* 如果必要，生成新的测试用例集，用于测试原来测试用例集无法充分测试的部分

* 依据一定的策略选择测试用例测试被修改的软件。

* 进行测试，并记录测试结果到测试报告

* 分析测试报告

* 修正和测试工作

* 完成测试产品提交配置

四、测试类型的划分

1、功能测试：对软件功能进行的测试，主要检查软件功能是否实现了软件功能说明书（软件需求）上的功能要求。

2、界面测试：对软件的用户界面进行的测试，主要检查用户界面的美观度、统一性、易用性等方面的内容。

3、数据处理测试：对软件数据接口进行的测试，主要检查软件数据处理中输入、处理、输出数据过程。

4、流程测试：按操作流程进行的测试，主要有业务流程、数据流程、逻辑流程、正反流程，检查软件在按流程操作时是否能够正确处理。

5、极限测试：在软件的极限条件下进行的测试，主要有对数据的极限值、边界值操作，对软件进行致命操作等。

6、并发测试：在网络环境、并发环境、多用户条件下对软件进行的测试。

7、安全测试：对软件安全性方面的测试，主要检测软件中加密、解密、数据备份、恢复、病毒检测等问题。

8、性能测试：对软件整体性能的测试，测试内容有适应性、健壮性、可恢复性、灾难恢复能力等

9、安装测试：在不同 PC 条件、操作系统、模拟客户机等条件下进行软件的安装测试，主要检查软件打包或发布之后存在的问题。

五、测试模式

V 型模型，实现测试与软件开发的同步进行。

六、测试—开发工作流程

七、测试工作流程

测试操作流程图

说明：

设计测试用例、执行测试用例详见《测试用例》。

描述软件错误即填写 bug 记录表，详见《BUG 标准》

八、附录

附录一、测试文档

I 测试计划

1．引言

1.1 编写目的

【阐明编写测试计划的目的，指明读者对象。】

1.2 项目背景

【说明项目的来源、委托单位及主管部门。】

1.3 定义

【列出测试计划中所用到的专门术语的定义和缩写词的原意。】

1.4 参考资料

【列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源，可包括：

a. 项目的计划任务书、合同或批文；

b. 项目开发计划；

c. 需求规格说明书；

d. 概要设计说明书；

e. 详细设计说明书；

f. 用户操作手册；

g. 本测试计划中引用的其他资料、采用的软件开发标准或规范。】

2．任务概述

2.1 目标

2.2 运行环境

2.3 需求概述

2.4 条件与限制

3．计划

3.1 测试方案

【说明确定测试方法和选取测试用例的原则。】

3.2 测试项目

【列出组装测试和确认测试中每一项测试的内容、名称、目的和进度。】

3.3 测试准备

3.4 测试机构及人员

【测试机构名称、负责人和职责。】

4．测试项目说明

【按顺序逐个对测试项目做出说明：】

4.1 测试项目名称及测试内容

4.2 测试用例

4.2.1 输入

【输入的数据和输入命令。】

4.2.2 输出

【预期的输出数据。】

4.2.3 步骤及操作

4.2.4 允许偏差

【给出实测结果与预期结果之间允许偏差的范围。】

4.3 进度

4.4 条件

【给出测试对资源的特殊要求，如设备、软件、人员等。】

4.5 测试资料

【说明测试所需的资料。】

5．评价

5.1 范围

【说明所完成的各项测试说明问题的范围及其局限性。】

5.2 准则

【说明评价测试结果的准则。】

II 测试用例

详见《测试用例》

III 测试记录报告

填写测试用例执行报告，详见《测试用例》

IV 测试问题报告

即 BUG 记录表，详见《BUG 标准》

V 测试分析报告

1．引言

1.1 编写目的

【阐明编写测试分析报告的目的，指明读者对象。】

1.2 项目背景

【说明项目的来源、委托单位及主管部门。】

1.3 定义

【列出测试分析报告中所用到的专门术语的定义和缩写词的原文。】

1.4 参考资料

【列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源，可包括：

a. 项目的计划任务书、合同或批文；

b. 项目开发计划；

c. 需求规格说明书；

d. 概要设计说明书；

e. 详细设计说明书；

f. 用户操作手册；

g. 测试计划；

h. 测试分析报告所引用的其他资料、采用的软件工程标准或软件工作规范。】

2．测试计划执行情况

2.1 测试项目

【列出每一测试项目的名称、内容和目的。】

2.2 测试机构和人员

【给出测试机构名称、负责人和参与测试人员名单。】

2.3 测试结果

【按顺序给出每一测试项目的：

a. 实测结果数据；

b. 与预期结果数据的偏差；

c. 该项测试表明的事实；

d. 该项测试发现的问题。】

3．软件需求测试结论

【按顺序给出每一项需求测试的结论。包括：

a. 证实的软件能力；

b. 局限性（即项需求未得到充分测试的情况及原因）。】

4．评价

4.1 软件能力

【经过测试所表明的软件能力。】

4.2 缺陷和限制

【说明测试所揭露的软件缺陷和不足，以及可能给软件运行带来的影响。】

4.3 建议

【提出为弥补上述缺陷的建议。】

4.4 测试结论