软件测试基础知识与软件测试基本流程

（完整版）

Q：什么是软件测试？软件测试的目的是什么？

A：IEEE 软件测试定义为：使用人工和自动手段来运行或测试某个系统的过程，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异。

该定义明确提出了软件测试以检验是否满足需求为目标。

软件测试的目的在于发现错误；一个好的测试用例在于发现从前未发现的错误；一个成功的测试是发现了从前未发现的错误的测试。

所以更为合适的定义是：测试是为发现错误而执行程序的过程。

Q：什么是软件缺陷？

A：满足下列五个规则之一才称为软件缺陷：

1）软件未达到产品说明书标明的功能。

2）软件出现了产品说明书指明不会出现的错误。

3）软件功能超出产品说明书指明的范围。

4）软件未达到产品说明书虽未指出但应该达到的目标。

5）软件测试人员认为软件难以理解、不易使用、运行速度缓慢，或者最终用户认为不好。

Q：什么黑盒测试？黑盒测试方法都包括哪些？

A：黑盒测试意味着测试要在软件的接口处进行。是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。因此黑盒测试又叫功能测试或数据驱动测试。

黑盒测试方法包括：等价类划分、边界值分析、因果图分析、错误推测法、功能图分析等

Q：什么白盒测试？白盒测试方法包括哪些？

A：白盒测试是对软件的过程性细节做细致的检查。是把测试对象看做一个打开的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序状态，确定实际状态是否与预期的状态一致。因此白盒测试又称为结构测试或逻辑驱动测试。

白盒测试方法包括：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖等

Q：软件测试策略都包含哪些？

A：根据软件测试工作的测试策略，一般将软件测试过程分为：单元测试、集成测试、系统测试、验收测试四个大的阶段。

Q：什么是单元测试？

A：单元测试是对软件中的基本组成单位进行的测试，如一个模块、一个过程等等。它是软件动态测试的最基本的部分，也是最重要的部分之一，其目的是检验软件基本组成单位的正确性。一个软件单元的正确性是相对于该单元的规约(详细设计)而言的。因此，单元测试以被

测试单位的规约为基准。单元测试的主要方法有控制流测试、数据流测试、排错测试、分域测试等等。

Q：什么是集成测试？

A：集成测试是在软件系统集成过程中所进行的测试，其主要目的是检查软件单位之间的接口是否正确。它根据集成测试计划，一边将模块或其他软件单位组合成越来越大的系统，一边运行该系统，以分析所组成的系统是否正确，各组成部分是否合拍。集成测试的策略主要有自顶向下和自底向上两种。

测试的热情。

测试新人最应该具备的就是测试的热情。因为这是一项事业，一项庞大的工程。系统的灵魂师！

Q：什么是系统测试？

A：统测试是对已经集成好的软件系统进行彻底的测试，以验证软件系统的正确性和性能等满足其规约所指定的要求，检查软件的行为和输出是否正确并非一项简单的任务，它被称为测试的“先知者问题”。因此，系统测试应该按照测试计划进行，其输入、输出和其他动态运行行为应该与软件规约进行对比。软件系统测试方法很多，主要有功能测试、性能测试、随机测试等等。

Q：什么是验收测试？

A：验收测试旨在向软件的购买者展示该软件系统满足其用户的需求。它的测试数据通常是系统测试的测试数据的子集。所不同的是，验收测试常常有软件系统的购买者代表在现场，甚至是在软件安装使用的现场。这是软件在投入使用之前的最后测试。

Q：什么是自动化测试？

A：一般我们谈到的自动化测试，其实是有两种说法的，一种是 Test Automation，翻译过来叫测试自动化，侧重说明将测试用自动化设计和实现的过程；另外一种是 Automated Testing/Test，翻译过来叫自动化测试，侧重说明自动的测试软件，可以是自动测试软件的功能或者性能等。

表面上看两种是有区别的，但现在我们用的多了，在提到自动化测试时，也就不区分了，基本上代表了一个意思，即：自动化测试是通过工具（程序）来对软件进行测试，一般不需要人为干预或干预很少。

Automated Testing/Test Automation:

1、Testing employing software tools which execute tests without manual intervention. Can be applied in GUI, performance, API, etc. testing.

使用自动化测试工具来进行测试，这类测试一般不需要人干预，通常在 GUI、性能等测试中用得较多。

2、The use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions. Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.

使用软件来控制测试的执行，实际输出和预期输出的对比，测试前提条件的构建，以及其他测试控制条件和测试报告功能。通常，测试自动化涉及自动化对一个已经使用了正式的测验流程的手工过程。

显而易见，第二种定义具体，且涵盖了多数情况，特别是只提及软件，而不是一定是“自动化测试工具”，而且不一定自动化测试步骤才叫自动化测试，很多情况下测试前提条件的自动化也是很重要而且很值得自动化的。

什么是 Alpha 和 Beta 测试？

Alpha 测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试。开发者负责记录发现在错误和使用中遇到的问题。总之，Alpha 测试是在受控的环境中进行的。

Beta 测试由软件的最终用户们在一个或多个客房场所进行。与 Alpha 测试不同，开发者通常在 Beta 测试的现场，因 Beta 测试是软件在开发者不能控制的环境中的“真实”应用。用户 Beta 测试过程中遇到的一切问题（真实在或想像的），并且定期把这些问题报告给开发者。接收到在 Beta 测试期间报告的问题之后，开发者对软件产品进行必要的修改，并准备向全体客户发布最终的软件产品。

什么是功能测试？

Functional testing (功能测试)，也称为 behavioral testing（行为测试），根据产品特征、操作描述和用户方案，测试一个产品的特性和可操作行为以确定它们满足设计需求。本地化软件的功能测试，用于验证应用程序或网站对目标用户能正确工作。使用适当的平台、浏览器和测试脚本，以保证目标用户的体验将足够好，就像应用程序是专门为该市场开发的一样。

功能测试也叫黑盒子测试或数据驱动测试,只需考虑各个功能，不需要考虑整个软件的内部结构及代码.一般从软件产品的界面、架构出发，按照需求编写出来的测试用例，输入数据在预期结果和实际结果之间进行评测,进而提出更加使产品达到用户使用的要求。

什么是性能测试？

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

什么是冒烟测试？

冒烟测试 (smoke testing)，据说是微软起的名字。在《微软项目求生法则》一书第 14 章“构建过程”关于冒烟测试，就是开发人员在个人版本的软件上执行目前的冒烟测试项目，确定新的程序代码不出故障。

冒烟测试的名称可以理解为该种测试耗时短，仅用一袋烟功夫足够了。也有人认为是形象地类比新电路板功基本功能检查。任何新电路板焊好后，先通电检查，如果存在设计缺陷，电路板可能会短路，板子冒烟了。

冒烟测试的对象是每一个新编译的需要正式测试的软件版本，目的是确认软件基本功能正常，可以进行后续的正式测试工作。冒烟测试的执行者是版本编译人员。

在一般软件公司的软件编写过程中，内部需要编译多个版本 (Build)，但是只有有限的几个版本需要执行正式测试（根据项目开发计划），这些需要执行的中间测试版本，在刚刚编译出来后，软件编译人员需要进行基本性能确认测试，例如是否可以正确安装/卸载，主要功能

是否实现，是否存在严重死机或数据严重丢失等 Bug。如果通过了该测试，则可以根据正式测试文档进行正式测试。否则，就需要重新编译版本，再次执行版本可接收确认测试，直到成功。

新版本的基本功能确认检查的测试，有的公司称为版本健康检查 (Build Sanity Check)。对于编译的本地化软件新版本，除了进行上面提到的各种测试检查，还要检查是否在新的本地化版本中正确包含了全部应该本地化的文件。可以通过采用文件和目录结构比较工具，首先比较源语言版本和本地化版本的文件和目录中的文件数目、文件名称和文件日期等，这个过程称为版本镜像检查 (Build Image Check)。其次，分别安装源语言版本和本地化版本，比较安装后的文件和目录结构中的文件数目、文件名称和文件日期等，这个过程称为版本安装检查 (Build Installing Check)。

什么是随机测试？TOP [浏览:6 次]

在软件测试中除了根据测试样例和测试说明书进行测试外，还需要进行随机测试 (Ad-hoc testing)，主要是根据测试者的经验对软件进行功能和性能抽查。随机测试是根据测试说明书执行样例测试的重要补充手段，是保证测试覆盖完整性的有效方式和过程。

随机测试主要是对被测软件的一些重要功能进行复测，也包括测试那些当前的测试样例 (TestCase) 没有覆盖到的部分。另外，对于软件更新和新增加的功能要重点测试。须注意针对一些特殊情况点、特殊的使用环境和可能并发性问题进行检查。尤其对以前测试发现的重大 Bug，进行再次测试，可以结合回归测试 (Regressive testing) 一起进行。

理论上，每一个被测软件版本都需要执行随机测试，尤其对于最后的将要发布的版本更要重视随机测试。随机测试最好由具有丰富测试经验的熟悉被测软件的测试人员进行测试。对于被测试的软件越熟悉，执行随机测试越容易。只有不断的积累测试经验，包括具体的测试执行和对缺陷跟踪记录的分析，不断总结，才能提高。

什么是动态测试和静态测试？

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。所谓软件的动态测试，就是通过运行软件来检验软件的动态行为和运行结果的正确性。目前，动态测试也是公司的测试工作的主要方式。

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语法、结构、过程、接口等来检查程序的正确性。对需求规格说明书、软件设计说明书、源程序做结构分析、流程图分析、符号执行来找错。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。

什么是测试用例？

测试用例（Test Case）是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。

测试用例（Test Case）目前没有经典的定义。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

测试用例（Test Case）是将软件测试的行为活动做一个科学化的组织归纳. 目的是能够将软件测试的行为转化成可管理的模式；同时测试用例也是将测试具体量化的方法之一.

软件测试类型有哪些？TOP［浏览:36 次 ］

软件测试是指使用人工或者自动的手段来运行或测定某个软件产品系统的过程，其目的是在于检验是否满足规定的需求或者弄清预期的结果与实际结果的区别。本文主要描述软件测试的类型。

1.数据和数据库完整性测试

数据与数据库完整测试是指测试关系型数据库完整性原则以及数据合理性测试。

数据库完整性原即：

主码完整性：主码不能为空；

外码完整性：外码必须等于对应的主码或者为空。

数据合理性指数据在数据库中的类型，长度，索引等是否建的比较合理。

在项目名称中，数据库和数据库进程应作为一个子系统来进行测试。在测试这些子系统时，不应将测试对象的用户界面用作数据的接口。对于数据库管理系统 (DBMS)，还需要进行深入的研究，以确定可以支 1 持测试的工具和技术。

比如，有两张表：部门和员工。部门中有部门编号，部门名称，部门经理等字段，主码为部门编号；员工表中有员工编号，员工所属部门编号，员工名称，员工类型等字段，主码为员工编号，外码为员工所属部门编号，对应部门表。如果在某条部门记录中部门编号或员工记录员工编号为空，他就违反主码完整性原则。如果某个员工所属部门的编号为##，但是##在部门编号中确找不到，这就违反外码完整性原则。

员工类型如下定义：0：职工，1：职员，2：实习生。但数据类型为 Int，我们都知道 Int 占有 4 个字节，如果定义成 char(1).就比原来节约空间。

2.白盒测试

白盒测试是基于代码的测试，测试人员通过阅读程序代码或者通过使用开发工具中的单步调试来判断软件的质量，一般黑盒测试由项目经理在程序员开发中来实现。白盒测试分为动态白盒测试和静态白盒测试

2.1 静态白盒测试

利用眼睛，浏览代码，凭借经验，找出代码中的错误或者代码中不符合书写规范的地方。比如，代码规范中规定，函数必须为动宾结构。而黑盒测试发现一个函数定义如下：

```
Function NameGet(){
….
}
```

这是属于不符合开发规范的错误。

有这样一段代码:

```
if (i<0) & (i>=0)
…
```

这段代码交集为整个数轴，IF 语句没有必要

```
I=0;
while(I>100){
```

```
J=J+100;
T=J*PI;
}
```

在循环体内没有 I 的增加,bug 产生。

2.2 动态白盒测试

利用开发工具中的调式工具进行测试。比如一段代码有 4 个分支，输入 4 组不同的测试数据使 4 组分支都可以走通而且结果必须正确。

看一段代码

```
if(I<0){
P1
}else{
P2
}
```

在调试中输入 I=-1,P1 程序段通过， P2 程序段未通过，属于动态黑盒测试的缺陷

3.功能测试

功能测试指测试软件各个功能模块是否正确，逻辑是否正确。

对测试对象的功能测试应侧重于所有可直接追踪到用例或业务功能和业务规则的测试需求。这种测试的目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰当。此类测试基于黑盒技术，该技术通过图形用户界面 (GUI) 与应用程序进行交互，并对交互的输出或结果进行分析，以此来核实应用程序及其内部进程。功能测试的主要参考为类似于功能说明书之类的文档。

比如一个对电子商务系统，前台用户浏览商品-放入购物车-进入结账台，后台处理订单，配货，付款，发货，这一系列流程必须正确无误的走通，不能存在任何的错误。

4.UI 测试

UI 测试指测试用户界面的风格是否满足客户要求，文字是否正确，页面美工是否好看，文字，图片组合是否完美，背景是否美观，操作是否友好等等

用户界面 (UI) 测试用于核实用户与软件之间的交互。UI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。另外，UI 测试还可确保 UI 中的对象按照预期的方式运行，并符合公司或行业的标准。包括用户友好性，人性化，易操作性测试。UI 测试比较主观，与测试人员的喜好有关

比如：页面基调颜色刺眼；用户登入页面比较难于找到，文字中出现错别字，页面图片范围太广等都属于 UI 测试中的缺陷，但是这些缺陷都不太严重。

5.性能测试

性能测试主要测试软件测试的性能，包括负载测试，强度测试，数据库容量测试，基准测试以及基准测试

5.1 负载测试

负载测试是一种性能测试指数据在超负荷环境中运行，程序是否能够承担。

在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关的方面。

比如，在 B/S 结构中用户并发量测试就是属于负载测试的用户，可以使用 webload 工具，模拟上百人客户同时访问网站，看系统响应时间，处理速度如何？

5.2 强度测试

强度测试是一种性能测试，他在系统资源特别低的情况下软件系统运行情况。这类测试往往可以书写系统要求的软硬件水平要求。

实施和执行此类测试的目的是找出因资源不足或资源争用而导致的错误。如果内存或磁盘空间不足，测试对象就可能会表现出一些在正常条件下并不明显的缺陷。而其他缺陷则可能由于争用共享资源（如数据库锁或网络带宽）而造成的。强度测试还可用于确定测试对象能够处理的最大工作量。

比如：一个系统在内存 366M 下可以正常运行，但是降低到 258M 下不可以运行，告诉内存不足，这个系统对内存的要求就是 366M。

5.3 数据库容量测试

数据库容量测试指通过存储过程往数据库表中插入一定数量的数据，看看相关页面是否能够及时显示数据。

数据库容量测试使测试对象处理大量的数据，以确定是否达到了将使软件发生故障的极限。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测试对象正在为生成一份报表而处理一组数据库记录，那么容量测试就会使用一个大型的测试数据库，检验该软件是否正常运行并生成了正确的报表。做这种测试通常通过书写存储过程向数据库某个表中插入一定数量的记录，计算相关页面的调用时间。

比如，在电子商务系统中，通过 insert customer 往 user 表中插入 10 000 数据，看其是否可以正常显示顾客信息列表页面，如果要求达到最多可以处理 100 000 个客户，但是顾客信息列表页面不能够在规定的时间内显示出来，就需要调整程序中的 SQL 查询语句；如果在规定的时间内显示出来，可以将用户数分别提高到 20 000 ， 50 000, 100 000 进行测试。

5.4 基准测试

基准测试与已知现有的系统进行比较，主要检验是否与类似的产品具有竞争性的一种测试。

如果你要开发一套财务系统软件并且你已经获得用友财务系统的性能等数据，你可以测试你这套系统，看看哪些地方比用友财务系统好，哪些地方差？以便改进自己的系统，也可为产品广告提供数据。

5.5 竞争测试

软件竞争使用各种资源（数据纪录，内存等），看他与其他相关系统对资源的争夺能力。比如：一台机器上即安装您的财务系统，又安装用友财务系统。当 CPU 占有率下降后，看看是否能够强过用友财务系统，而是自己的系统能够正常运行？

6. 安全性和访问控制测试

安全性和访问控制测试侧重于安全性的两个关键方面：

应用程序级别的安全性，包括对数据或业务功能的访问

系统级别的安全性，包括对系统的登录或远程访问。

6.1 应用程序级别的安全性

可确保：在预期的安全性情况下，主角只能访问特定的功能或用例，或者只能访问有限的数据。例如，可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性，测试就可确保“用户类型一”能够看到所有客户消息（包括财务数据），而“用户二”只能看见同一客户的统计数据。

比如 B/S 系统，不通过登入页面，直接输入 URL,看其是否能够进入系统？

6.2 系统级别的安全性

可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问。

比如输入管理员账户，检查其密码是否容易猜取，或者可以从数据库中获得？

7.故障转移和恢复测试

故障转移和恢复测试指当主机软硬件发生灾难时候，备份机器是否能够正常启动，使系统是否可以正常运行，这对于电信，银行等领域的软件是十分重要的。

故障转移和恢复测试可确保测试对象能成功完成故障转移，并能从导致意外数据损失或数据完整性破坏的各种硬件、软件或网络故障中恢复。

故障转移测试可确保：对于必须持续运行的系统，一旦发生故障，备用系统就将不失时机地“顶替”发生故障的系统，以避免丢失任何数据或事务。

恢复测试是一种对抗性的测试过程。在这种测试中，将把应用程序或系统置于极端的条件下（或者是模拟的极端条件下），以产生故障（例如设备输入/输出 (I/O) 故障或无效的数据库指针和关健字）。然后调用恢复进程并监测和检查应用程序和系统，核实应用程序或系统和数据已得到了正确的恢复。一定要注意主备定时备份

比如电信系统，突然主机程序发生死机，备份机器是否能够启动，使系统能够正常运行，从而不影响用户打电话？

8.配置测试

又叫兼容性测试。配置测试核实测试对象在不同的软件和硬件配置中的运行情况。在大多数生产环境中，客户机工作站、网络连接和数据库服务器的具体硬件规格会有所不同。客户机工作站可能会安装不同的软件例如，应用程序、驱动程序等而且在任何时候，都可能运行许多不同的软件组合，从而占用不同的资源。（如浏览器版本，操作系统版本等）

下面列出主要配置测试

8.1 浏览器兼容性

测试软件在不同产商的浏览器下是否能够正确显示与运行；

比如测试 IE，Natscape 浏览器下是否可以运行这套软件？

8.2 操作系统兼容性

测试软件在不同操作系统下是否能够正确显示与运行；

比如测试 WINDOWS98,WINDOWS 2000,WINDOWS XP,LINU, UNIX 下是否可以运行这套软件？

8.3 硬件兼容性

测试与硬件密切相关的软件产品与其他硬件产品的兼容性，比如该软件是少在并口设备中的，测试同时使用其他并口设备，系统是否可以正确使用.

比如在 INTER,舒龙 CPU 芯片下系统是否能够正常运行？

这样的测试必须建立测试实验室，在各种环境下进行测试。

9.安装测试

安装测试有两个目的。第一个目的是确保该软件在正常情况和异常情况的不同条件下: 例如，进行首次安装、升级、完整的或自定义的安装_都能进行安装。异常情况包括磁盘空间不足、缺少目录创建权限等。第二个目的是核实软件在安装后可立即正常运行。这通常是指运行大量为功能测试制定的测试。

安装测试包括测试安装代码以及安装手册。安装手册提供如何进行安装，安装代码提供安装一些程序能够运行的基础数据。

10.多语种测试

又称本地化测试，是指为各个地方开发产品的测试，如英文版，中文版等等，包括程序是否能够正常运行，界面是否符合当地习俗，快捷键是否正常起作用等等，特别测试在 A 语言环境下运行 B 语言软件（比如在英文 win98 下试图运行中文版的程序），出现现象是否正常。

本地化测试还要考虑：

l 当语言从 A 翻译到 B，字符长度变化是否影响页面效果。比如中文软件中有个按键叫“看广告”，翻译到英文版本中为 “View advertisement”可能影响页面的美观程度

l 要考虑同一单词在各个国家的不同意思，比如 football 在英文中为足球，而美国人使用中可能理解为美式橄榄球。

l 要考虑各个国家的民族习惯，比如龙个美国中被理解邪恶的象征，但翻译到中国，中国人认为为吉祥的象征。

11.文字测试

文字测试测试软件中是否拼写正确，是否易懂，不存在二义性，没有语法错误；文字与内容是否有出入等等，包括图片文字。

比如：“比如，请输入正确的证件号码！”何谓正确的证件号码，证件可以为身份证，驾驶证，也可为军官证，如果改为“请输入正确的身份证号码！”用户就比较容易理解了。

12.分辨率测试

测试在不同分辨率下，界面的美观程度,分为 800*600，1024*768，1152*864，1280*768，1280*1024，1200*1600 大小字体下测试。一个好的软件要有一个极佳的分辨率，而在其他分辨率下也都能可以运行。

13 发布测试

主要在产品发布前对一些附带产品，比如说明书，广告稿等进行测试

13.1 说明书测试

主要为语言检查，功能检查，图片检查

语言检查：检查说明书语言是否正确，用词是否易于理解；

功能检查：功能是否描述完全，或者描述了并没有的功能等；

图片检查:：检查图片是否正确

13.2 宣传材料测试

主要测试产品中的附带的宣传材料中的语言，描述功能，图片

13.3 帮助文件测试

帮助文件是否正确，易懂，是否人性化。最好能够提供检索功能。

13.4 广告用语

产品出公司前的广告材料文字，功能，图片，人性化的检查

14 文档审核测试

文档审核测试目前越来越引起人们的重视，软件质量不是检查出来的，而是融进软件开发中来。前置软件测试发越来越受到重视。请看一个资料：

文档审核测试主要包括需求文档测试，设计文档测试，为前置软件测试测试中的一部分。

14.1 需求文档测试

主要测试需求中是否存在逻辑矛盾以及需求在技术上是否可以实现；

14.2 设计文档测试

测试设计是否符合全部需求以及设计是否合理。

总结

据美国软件质量安全中心 2000 年对美国一百家知名的软件厂商统计，得出这样一个结论：软件缺陷在开发前期发现比在开发后期发现资金，人力上节约 90%；软件缺陷在推向市场前发现比在推出后发现资金，人力上节约 90%。所以说软件的缺陷应该尽早发现。不是所有的软件都要进行任何类型的软件测试的，可以根据产品的具体情况进行组装测试不同的类型。

什么是测试用例？

测试用例（Test Case）是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。

测试用例（Test Case）目前没有经典的定义。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

测试用例（Test Case）是将软件测试的行为活动做一个科学化的组织归纳. 目的是能够将软件测试的行为转化成可管理的模式；同时测试用例也是将测试具体量化的方法之一.

灰盒测试

灰盒测试，确实是介于白盒测试与黑盒测试之间的，可以这样理解，灰盒测试关注输出对于输入的正确性，同时也关注内部表现，但这种关注不象白盒那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，有时候输出是正确的，但内部其实已经错误了，这种情况非常多，如果每次都通过白盒测试来操作，效率会很低，因此需要采取这样的一种灰盒的方法。

灰盒测试结合了白盒测试和黑盒测试的要素.它考虑了用户端、特定的系统知识和操作环境。它在系统组件的协同性环境中评价应用软件的设计。

灰盒测试由方法和工具组成，这些方法和工具取材于应用程序的内部知识和与之交互的环境，能够用于黑盒测试以增强测试效率、错误发现和错误分析的效率。

灰盒测试涉及输入和输出，但使用关于代码和程序操作等通常在测试人员视野之外的信息设计测试。

什么是测试计划？

测试计划包含项目范围内的测试目的和测试目标的有关信息。此外，测试计划确定了实施和执行测试时使用的策略，同时还确定了所需资源。定义一个测试项目的过程，以便能够正确的度量和控制测试

软件测试类型有哪些？

软件测试是指使用人工或者自动的手段来运行或测定某个软件产品系统的过程，其目的是在于检验是否满足规定的需求或者弄清预期的结果与实际结果的区别。本文主要描述软件测试的类型。

软件测试类型有哪些？

软件测试是指使用人工或者自动的手段来运行或测定某个软件产品系统的过程，其目的是在于检验是否满足规定的需求或者弄清预期的结果与实际结果的区别。本文主要描述软件测试的类型。

1.数据和数据库完整性测试

数据与数据库完整测试是指测试关系型数据库完整性原则以及数据合理性测试。

数据库完整性原即：

主码完整性：主码不能为空；

外码完整性：外码必须等于对应的主码或者为空。

数据合理性指数据在数据库中的类型，长度，索引等是否建的比较合理。

在项目名称中，数据库和数据库进程应作为一个子系统来进行测试。在测试这些子系统时，不应将测试对象的用户界面用作数据的接口。对于数据库管理系统 (DBMS)，还需要进行深入的研究，以确定可以支 1 持测试的工具和技术。

比如，有两张表：部门和员工。部门中有部门编号，部门名称，部门经理等字段，主码为部门编号；员工表中有员工编号，员工所属部门编号，员工名称，员工类型等字段，主码为员工编号，外码为员工所属部门编号，对应部门表。如果在某条部门记录中部门编号或员工记录员工编号为空，他就违反主码完整性原则。如果某个员工所属部门的编号为##，但是##在部门编号中确找不到，这就违反外码完整性原则。

员工类型如下定义：0：职工，1：职员，2：实习生。但数据类型为 Int，我们都知道 Int 占有 4 个字节，如果定义成 char(1).就比原来节约空间。

2.白盒测试

白盒测试是基于代码的测试，测试人员通过阅读程序代码或者通过使用开发工具中的单步调试来判断软件的质量，一般黑盒测试由项目经理在程序员开发中来实现。白盒测试分为动态白盒测试和静态白盒测试

2.1 静态白盒测试

利用眼睛，浏览代码，凭借经验，找出代码中的错误或者代码中不符合书写规范的地方。比如，代码规范中规定，函数必须为动宾结构。而黑盒测试发现一个函数定义如下：

```
Function NameGet(){
….
}
```

这是属于不符合开发规范的错误。

有这样一段代码:

```
if (i<0) & (i>=0)
…
```

这段代码交集为整个数轴，IF 语句没有必要

```
I=0;
while(I>100){
J=J+100;
T=J*PI;
}
```

在循环体内没有 I 的增加,bug 产生。

2.2 动态白盒测试

利用开发工具中的调式工具进行测试。比如一段代码有 4 个分支，输入 4 组不同的测试数据使 4 组分支都可以走通而且结果必须正确。

看一段代码

```
if(I<0){
P1
}else{
P2
}
```

在调试中输入 I=-1,P1 程序段通过， P2 程序段未通过，属于动态黑盒测试的缺陷

3.功能测试

功能测试指测试软件各个功能模块是否正确，逻辑是否正确。

对测试对象的功能测试应侧重于所有可直接追踪到用例或业务功能和业务规则的测试需求。这种测试的目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰当。此类测试基于黑盒技术，该技术通过图形用户界面 (GUI) 与应用程序进行交互，并对交互的输出或结果进行分析，以此来核实应用程序及其内部进程。功能测试的主要参考为类似于功能说明书之类的文档。

比如一个对电子商务系统，前台用户浏览商品-放入购物车-进入结账台，后台处理订单，配货，付款，发货，这一系列流程必须正确无误的走通，不能存在任何的错误。

4.UI 测试

UI 测试指测试用户界面的风格是否满足客户要求，文字是否正确，页面美工是否好看，文字，图片组合是否完美，背景是否美观，操作是否友好等等

用户界面 (UI) 测试用于核实用户与软件之间的交互。UI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。另外，UI 测试还可确保 UI 中的对象按照预期的方式运行，并符合公司或行业的标准。包括用户友好性，人性化，易操作性测试。UI 测试比较主观，与测试人员的喜好有关

比如：页面基调颜色刺眼；用户登入页面比较难于找到，文字中出现错别字，页面图片范围太广等都属于 UI 测试中的缺陷，但是这些缺陷都不太严重。

5.性能测试

性能测试主要测试软件测试的性能，包括负载测试，强度测试，数据库容量测试，基准测试以及基准测试

5.1 负载测试

负载测试是一种性能测试指数据在超负荷环境中运行，程序是否能够承担。

在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关的方面。

比如，在 B/S 结构中用户并发量测试就是属于负载测试的用户，可以使用 webload 工具，模拟上百人客户同时访问网站，看系统响应时间，处理速度如何？

5.2 强度测试

强度测试是一种性能测试，他在系统资源特别低的情况下软件系统运行情况。这类测试往往可以书写系统要求的软硬件水平要求。

实施和执行此类测试的目的是找出因资源不足或资源争用而导致的错误。如果内存或磁盘空间不足，测试对象就可能会表现出一些在正常条件下并不明显的缺陷。而其他缺陷则可能由于争用共享资源（如数据库锁或网络带宽）而造成的。强度测试还可用于确定测试对象能够处理的最大工作量。

比如：一个系统在内存 366M 下可以正常运行，但是降低到 258M 下不可以运行，告诉内存不足，这个系统对内存的要求就是 366M。

5.3 数据库容量测试

数据库容量测试指通过存储过程往数据库表中插入一定数量的数据，看看相关页面是否能够及时显示数据。

数据库容量测试使测试对象处理大量的数据，以确定是否达到了将使软件发生故障的极限。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测试对象正在为生成一份报表而处理一组数据库记录，那么容量测试就会使用一个大型的测试数据库，检验该软件是否正常运行并生成了正确的报表。做这种测试通常通过书写存储过程向数据库某个表中插入一定数量的记录，计算相关页面的调用时间。

比如，在电子商务系统中，通过 insert customer 往 user 表中插入 10 000 数据，看其是否可以正常显示顾客信息列表页面，如果要求达到最多可以处理 100 000 个客户，但是顾客

信息列表页面不能够在规定的时间内显示出来，就需要调整程序中的 SQL 查询语句；如果在规定的时间内显示出来，可以将用户数分别提高到 20 000 ， 50 000, 100 000 进行测试。

5.4 基准测试

基准测试与已知现有的系统进行比较，主要检验是否与类似的产品具有竞争性的一种测试。

如果你要开发一套财务系统软件并且你已经获得用友财务系统的性能等数据，你可以测试你这套系统，看看哪些地方比用友财务系统好，哪些地方差？以便改进自己的系统，也可为产品广告提供数据。

5.5 竞争测试

软件竞争使用各种资源（数据纪录，内存等），看他与其他相关系统对资源的争夺能力。比如：一台机器上即安装您的财务系统，又安装用友财务系统。当 CPU 占有率下降后，看看是否能够强过用友财务系统，而是自己的系统能够正常运行？

6. 安全性和访问控制测试

安全性和访问控制测试侧重于安全性的两个关键方面：

应用程序级别的安全性，包括对数据或业务功能的访问

系统级别的安全性，包括对系统的登录或远程访问。

6.1 应用程序级别的安全性

可确保：在预期的安全性情况下，主角只能访问特定的功能或用例，或者只能访问有限的数据。例如，可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性，测试就可确保“用户类型一”能够看到所有客户消息（包括财务数据），而“用户二”只能看见同一客户的统计数据。

比如 B/S 系统，不通过登入页面，直接输入 URL,看其是否能够进入系统？

6.2 系统级别的安全性

可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问。

比如输入管理员账户，检查其密码是否容易猜取，或者可以从数据库中获得？

7.故障转移和恢复测试

故障转移和恢复测试指当主机软硬件发生灾难时候，备份机器是否能够正常启动，使系统是否可以正常运行，这对于电信，银行等领域的软件是十分重要的。

故障转移和恢复测试可确保测试对象能成功完成故障转移，并能从导致意外数据损失或数据完整性破坏的各种硬件、软件或网络故障中恢复。

故障转移测试可确保：对于必须持续运行的系统，一旦发生故障，备用系统就将不失时机地“顶替”发生故障的系统，以避免丢失任何数据或事务。

恢复测试是一种对抗性的测试过程。在这种测试中，将把应用程序或系统置于极端的条件下（或者是模拟的极端条件下），以产生故障（例如设备输入/输出 (I/O) 故障或无效的数据库指针和关健字）。然后调用恢复进程并监测和检查应用程序和系统，核实应用程序或系统和数据已得到了正确的恢复。一定要注意主备定时备份

比如电信系统，突然主机程序发生死机，备份机器是否能够启动，使系统能够正常运行，从而不影响用户打电话？

8.配置测试

又叫兼容性测试。配置测试核实测试对象在不同的软件和硬件配置中的运行情况。在大多数生产环境中，客户机工作站、网络连接和数据库服务器的具体硬件规格会有所不同。客户机工作站可能会安装不同的软件例如，应用程序、驱动程序等而且在任何时候，都可能运行许多不同的软件组合，从而占用不同的资源。（如浏览器版本，操作系统版本等）

软件测试 V 模型

————————————————————————————————————

V 模型是最具有代表意义的测试模型

V 模型是软件开发瀑布模型的变种，它反映了测试活动与分析和设计的关系 。

从左到右，描述了基本的开发过程和测试行为，非常明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系 。

左边依次下降的是开发过程各阶段，与此相对应的是右边依次上升的部分，即各测试过程的各个阶段。

用户需求 验收测试

需求分析和系统设计 确认测试和系统测试

概要设计 集成测试

详细设计 单元测试

编码

V 模型问题：

1.测试是开发之后的一个阶段。

2.测试的对象就是程序本身。

3.实际应用中容易导致需求阶段的错误一直到最后系统测试阶段才被发现。

4.整个软件产品的过程质量保证完全依赖于开发人员的能力和对工作的责任心，而且上一步的结果必须是充分和正确的，如果任何一个环节出了问题，则必将严重的影响整个工程的质量和预期进度

W 模型由 Evolutif 公司公司提出，相对于 V 模型，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动。W 模型由两个 V 字型模型组成，分别代表测试与开发过程，图中明确表示出了测试与开发的并行关系。

W 模型强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发是同步进行的。

W 模型有利于尽早地全面的发现问题。例如，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。 但 W 模型也存在局限性。在 W 模型中，需求、设计、编码等活动被视为串行的，同时，测试和开发活动也保持着一种线性的前后关系，上一阶段完全结束，才可正式开始下一个阶段工作。这

样就无法支持迭代的开发模型。对于当前软件开发复杂多变的情况，W 模型并不能解除测试管理面临着困惑。

什么是开发的缺陷管理?

软件中的缺陷(Defect 或 BUG)是软件开发过程中的“副产品”。通常，缺陷会导致软件产品在某种程度上不能满足用户的需要。每一个软件开发团队都必须知道如何妥善处理软件中的缺陷，这关系到软件生存、发展的质量根本。可遗憾的是，并非所有的软件开发团队都知道如何有效地管理软件中的缺陷。

软件缺陷管理是在软件生命周期中为确保缺陷被跟踪和管理所进行的活动。狭义地讲，BUG 是写程序过程中造成的错误。广义地讲，BUG 是影响客户正常使用的任何问题。就是说，BUG 不仅仅是编程中出现的问题，还包括客户需求和功能规范等方面。

(1)缺陷管理的目标

一般而言，缺陷的跟踪和管理需要达到以下两个目标：一是确保每个被发现的缺陷都能够被解决，二是收集缺陷数据并根据缺陷趋势曲线识别和预防缺陷的频繁发生。

在谈到缺陷管理时，一般人都会只想到如何修正缺陷，而对根据缺陷分析进行有效预防缺陷却很容易忽视。其实，在一个运行良好的项目开发中，缺陷数据的收集和分析是很重要的，从缺陷数据中可以得到很多与软件质量相关的数据。例如通过缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式。常见的的缺陷数据统计图表包括缺陷趋势图、缺陷分布图、缺陷及时处理情况统计表等。

(2)缺陷管理重在预防缺陷

正如我们所知，BUG 应该尽早地在开发过程中被发现。修正处于开发阶段的 BUG 的成本远远低于修正处于验收阶段的 BUG，而相对与修正已经发布给客户的产品 BUG 的成本更是可以忽略不计。因此，越晚修正 BUG，需要重做的事情就越多。

对很多人来说，零缺陷的软件产品似乎是不切实际的。因此，我们总是听到许多软件开发人员说：“软件永远有 BUG”。软件产品含有 BUG 并不奇怪，不幸的是发布一个包含很多 BUG 的产品给客户仍然不让人感到惊讶，这就是一件值提深思的事情了。

事实上，每个软件开发团队都可以通过一些简单的方法，在不增加额外资源的情况下预防 BUG。为了能够预防 BUG，我们首先需要了解 BUG 的来源。软件 BUG 可以分为几个类别：第一类 BUG 可能是随机的，它们通常是因为一时的疏忽造成的。尽管这些 BUG 可能由于其随机性很难预防。但是，适当的分析将有助于避免这些 BUG。另一类的 BUG 来自于需求误解、开发环境的错误或者纯粹由于缺乏解决问题的相关技术，这类 BUG 共同的特点是都来自于开发人员。

但有一个好消息是，软件中的 BUG 往往倾向于重复出现，即使是一个随机出现的 BUG。软件 BUG 的不断出现不仅表现在同一个开发人员的工作上，而且表现在同一个项目上。这当然不是说项目中的每一个开发人员都会犯同样的错误。但是，至少其中一些的错误足以成为经常性出现的问题。因此，BUG 的预防尤为重要。

缺陷管理的核心：缺陷分析

缺陷预防的着眼点在于缺陷的共性原因(Common Cause)。通过寻找、分析和处理缺陷的共性原因，实现缺陷预防。BUG 预防并不是一个不切实际的目标，但是不能期望它在一夜之间发生。我们在开发过程中应该积极为开发小组提供缺陷分析，使 BUG 逐渐改善。因此，缺陷管理的最终目标是预防 BUG，不断提高整个开发团队的技能和实践经验，而不只是修正它们。

BUG 预防策略非常简单和容易实现，策略是发现 BUG，找出 BUG 的根源，然后寻找一个方法来预防类似的 BUG 在将来出现。这策略并不需要昂贵的花费，但是却可带来极大的额外价值。

(1)BUG 记录

BUG 分析的第一步是记录 BUG，值得注意的是记录 BUG 不应该满足于记录 BUG 的表面症状。测试的一个重要职责就是试图发现 BUG 的根本原因，在测试时不应将产品看作一个黑盒，而应该像开发人员那样了解产品的内在，包括深入源代码，理解产品的设计和实现。

(2)利用 BUG 分析了解开发质量趋势

对于测试出来的 BUG 进行缺陷分类，找出那些关键的缺陷类型，进一步分析其产生的根源，从而针对性的制定改进措施。缺陷分析非常关键的一步就是寻找一个预防类似缺陷再次发生的方法。这一方法不仅涉及到开发、测试人员，还涉及到不直接负责代码编写的资深开发人员。利用这一阶段的实践成果，开发人员可以预防 BUG 的发生，而不仅仅是修正这些 BUG。

BUG 预防分析是整个 BUG 分析过程的核心。这一阶段总结出的实践可以在更广泛的范围内预防潜在的缺陷。由于分析结果的广泛应用性，分析某个具体 BUG 的投入将很容易被收回。在这个时候，BUG 分析提供了两个非常重要的参数，一个是缺陷数量的趋势，另一个是缺陷修复的趋势。缺陷趋势就是将每月新生成的缺陷数、每月被解决的缺陷数和每月遗留的缺陷数标成一个趋势图表。

一般在项目的开始阶段发现缺陷数曲线会呈上升趋势，到项目中后期被修复缺陷数曲线会趋于上升，而发现缺陷数曲线应总体趋于下降。同时处于 OPEN 状态的缺陷也应该总体呈下降趋势，到项目最后，三条曲线都趋向于零。项目经理可通过持续观察这张图表，确保项目开发健康发展。同时，通过分析预测项目测试缺陷趋于零的时间，以制定产品质量验收和发布的时间。

实际上，BUG 分析图表会告诉我们很多有价值的信息。比如说，可分析开发和测试在人力资源的配比上是否恰当，可以分析出某个严重的缺陷所造成的项目质量的波动。对于异常的波动，如本来应该越测试越收敛的，却到了某个点发现的故障数反而呈上升趋势，那么意味着往往有一些特殊事件的发生。通过对测试缺陷分析，能够给予我们很多改进研发和测试工作的信息。

什么是软件测试中的缺陷度量？

缺陷度量

缺陷度量是软件度量的一部分，其本身并不能发现缺陷、剔除缺陷，但是有助于这些问题的解决。另外，当正确、持续地进行了缺陷度量时，产品以及过程的质量属性的数据为实施和管理过程改进活动提供了有效的基础。

数据的质量等因素，我们在本章 7.4 节中已经考虑了，这里仍将遵循。

什么是缺陷度量

软件产品质量度量，主要集中在软件的缺陷度量上。

缺陷度量就是对项目过程中产生的缺陷数据进行采集和量化，将分散的缺陷数据统一管理，使其有序而清晰，然后通过采用一系列数学函数，对数据进行处理，分析缺陷密度和趋势等信息，从而提高产品质量和改进开发过程。一般来说，在软件质量保证过程中，需要度量的缺陷数据包括 6 大类缺陷发现手段发现的所有缺陷。如测试相关的缺陷，需要度量包括测试投入的工作量和成本数据、测试任务完成情况、测试规模数据、测试结果数据（包括缺陷数据、覆盖率数据）等。

（1）组织级缺陷度量，目的是了解组织的整体缺陷情况，了解客户对组织的质量满意度，建立组织基线，确定改进活动。

（2）项目级缺陷度量，目的是了解项目实时质量情况（很多项目只在最后度量，包括那些迭代式开发的项目，实际上为时已晚），预测缺陷造成的发布后维护工作量，了解客户对项目的质量满意度。

（3）个体缺陷度量，目的是了解个体缺陷产生的详细原因，并实施行动进行改进。

前两种度量大家接触较多，但第三种度量常常被忽略。这常常导致：

● 项目反复得到关于自己的质量评价，但很难了解如何去提高；

● 测试组常常能做一些改进（如增加测试覆盖、延长测试周期）来提高缺陷排除效率，但开发组没有降低缺陷产生数量的有效措施；

● 软件开发遵循了编码规范，但似乎对提高质量没有太多帮助。

什么是灰盒测试？

灰盒测试，确实是介于黑盒测试和白盒测试二者之间的，可以这样理解，灰盒测试关注输出对于输入的正确性，同时也关注内部表现，但这种关注不象白盒那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，有时候输出是正确的，但内部其实已经错误了，这种情况非常多，如果每次都通过白盒测试来操作，效率会很低，因此需要采取这样的一种灰盒的方法。

灰盒测试结合了白盒测试盒黑盒测试的要素.它考虑了用户端、特定的系统知识和操作环境。它在系统组件的协同性环境中评价应用软件的设计。

灰盒测试由方法和工具组成，这些方法和工具取材于应用程序的内部知识盒与之交互的环境，能够用于黑盒测试以增强测试效率、错误发现和错误分析的效率。

灰盒测试涉及输入和输出，但使用关于代码和程序操作等通常在测试人员视野之外的信息设计测试。

软件测试用例设计综合策略

---

1. Myers 提出了使用各种测试方法的综合策略：

1)在任何情况下都必须使用边界值分析方法，经验表明用这种方法设计出测试用例发现程序错误的能力最强。

2)必要时用等价类划分方法补充一些测试用例。

3)用错误推测法再追加一些测试用例。

4)对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度，如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

5)如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

2.测试用例的设计步骤

1)构造根据设计规格得出的基本功能测试用例；

2)边界值测试用例；

3)状态转换测试用例；

4)错误猜测测试用例；

5)异常测试用例；

6)性能测试用例；

7)压力测试用例。

3.优化测试用例的方法

1)利用设计测试用例的 8 种方法不断的对测试用例进行分解与合并；

2)采用遗传算法理论进化测试用例；

3)在测试时利用发散思维构造测试用例。

什么是软件测试框架

测试框架是一组自动化测试的规范、测试脚本的基础代码，以及测试思想、惯例的集合。可用于减少冗余代码、提高代码生产率、提高代码重用性和可维护性。

测试框架的好处在于：提高开发速度，提升测试代码的执行效率；提高软件代码质量，同时引入重构概念，让代码更干净和富有弹性；提升系统的可信赖度，作为回归测试的一种实现方法支持修复后“再测试”，确保代码的正确性。

常用的测试框架分类包括自动化测试框架和单元测试框架。根据所用开发平台不同，也可使用不同的测试框架展开测试。

什么是静态测试

静态测试，英文是 Static Testing。

静态测试指测试不运行的部分，例如测试产品说明书，对此进行检查和审阅.。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。

静态测试常用工具有：Logiscope、PRQA；

静态测试技术 -- 代码走查、代码审查和技术评审

静态测试又可分为代码走查（Walkthrough），代码审查(Inspection)，技术评审(Review)。

代码走查（Walkthrough）

开发组内部进行的，采用讲解、讨论和模拟运行的方式进行的查找错误的活动。

代码审查 (Inspection)

开发组内部进行的，采用讲解、提问并使用编码模板进行的查找错误的活动。一般有正式的计划、流程和结果报告。

技术评审(Review)

开发组、测试组和相关人员(QA、产品经理等)联合进行的，采用讲解、提问并使用编码模板进行的查找错误的活动。一般有正式的计划、流程和结果报告。

实际工作，我们完全不必要被概念所束缚住，根据项目的实际情况来决定采取什么的静态测试形式，不用严格去区分到底是代码走查，代码审查和还是技术评审。

静态分析往往需要借助白盒测试工具（如 Logiscope,C++ Test）来自动检测。

什么是代码走查

代码走查(code walkthrough)和代码审查(code inspection)是两种不同的代码评审方法,

代码审查是一种正式的评审活动，而代码走查的讨论过程是非正式的。

最近对项目组进行代码评审，发觉需要对代码评审中找到的问题进行一下分类，大概可以分成以下几类问题：

1. Comment

注释没写，或者格式不对，或者毫无意义

2. Coding Standard

没遵守代码规范

3. Existing Wheel

重复现成的代码，或者是开源项目，或者公司已有代码

4. Better practice

Java 或者开源项目，有更好的写法

5. Performance bottle and Improvement

性能瓶颈和提高

6. Code Logic Error

代码逻辑错误

7. Business Logic Error

业务逻辑错误

代码审查列出问题的类型，并有解决情况报告

首先我们要明确为什么要进行代码走查活动，我以为其目的主要有：

1）、通过代码走查活动，及时了解程序员编写的代码是否符合设计要求以及编码规范；

2）、通过代码走查活动，及时了解程序员在编码过程中遇到的问题，并给以协助，从而达到有效、透明地掌控项目进度的目的；

3）、通过代码走查活动，及时了解代码中可以重用的代码，并将其提取为公共方法或模块，提高代码的可重用性以弥补当前人员设计能力不足的现状。

要满足上面的三个目的，显然仅仅依靠工具是不能够满足要求的。那么如何执行代码走查活动才会有效呢？

首先，在系统设计阶段，我们需要明确系统架构、编码规范等技术要求，来制定出代码走查活动需要的 Checklist（对于编码规范，当可以利用工具来进行检查时，准备的 Checklist 中就不需要将工具可以检查的要点再逐一列出来。）下图就是一个 Checklist 的示例。

第二步是确定代码走查时发现问题的记录方式。可以使用文档的方式来记录（这在很多项目中使用），也可以使用缺陷跟踪系统来记录。

当准备工作完成，且项目进入 Coding 阶段后，我们就可以正式开始执行代码走查活动了。为了改变以前那种事后检查的弊端，我们将代码走查活动前移到与程序员的 Coding 同步进行。这样做就是为了及时发现问题及时解决问题。实施的步骤如下：

1）、负责代码走查的人员从建构库中获取需要走查的代码；

2）、阅读代码，并根据前面准备好的 Checklist 对代码进行检查，看代码是否符合相关的技术要求，以及是否满足业务需求，发现的问题及时记录下来；

TIPS: 通常可以在阅读代码之前或者阅读完代码之后，利用工具来进行必要的 Check。可以利用的工具有：Checkstyle, CodePro, FindBugs, Metrics, JDepend 等等。

3）、阅读代码的过程中，如果发现有可供提取的可重用方法或模块，及时记录并通报给项目的架构负责人，由其负责可重用方法的编写；

4）、及时向程序员通报代码走查的结果，并由程序员对发现的问题进行修改。必要时对代码走查中发现的问题需及时向程序员进行讲解和指导；

5）、跟踪代码走查中发现的问题的解决进展，直到问题均关闭；

6）、每日重复以上的步骤，直到所有功能的编码全部结束为止。

通过以上代码走查活动的说明，可看出代码走查人员在项目中承担着比较重要的角色。因此安排合适的人来进行代码走查也显得格外重要，可以说直接关系着代码走查活动的最终成效。

通常我们可考虑安排功能的设计者来负责该功能的代码走查。这样有几个好处：

一是功能的设计者对于功能的业务需求比较清楚，这样在做代码走查时就容易了解程序员编写的代码是否能够满足设计的要求和业务需求。其实从另外一个角度来看也是对设计的一种检查；

二是通常功能的设计者都是较资深的人员，可以为程序员提供有效地指导和协助，从另外一个角度也是对程序员的 On-Job Training。

在实施代码走查的过程中，我们还需要借助工具来提高我们的效率，但切忌过分依赖工具或者仅仅只靠工具。同时也需要转变为了代码走查而代码走查的倾向，因为那样就不能发挥代码走查的作用，并最终达到代码走查的目的。

从实践来看，代码走查时记录问题的方式也影响到代码走查的效率。这里向大家介绍一个在 Eclipse 中进行代码走查的插件——Jupiter。它提供了一种简单而便捷的方式来记录和跟踪代码走查时发现的问题。

Jupiter 将走查结果以 XML 文件的形式存入 SCM 系统中，并且每条代码走查的意见直接关联对应代码，可以做方便的代码跳转。最新的安装包可以在 Jupiter 的网站上下载到：http://csdl.ics.hawaii.edu/tools/jupiter

下图是 Eclipse 中安装 Jupiter 后的界面。

在项目中使用 Jupiter 时，需要先进行配置。所有配置信息都保存在一个.jupiter 文件中，并需将之提交到 SCM 系统。

配置完成后，可按照以下步骤来使用：

1）、个人走查（Individual Phase） 代码走查者在本地对代码进行走查，然后建立相关的 Review Issue。走查结果会自动保存在一个以.review 为扩展名的 XML 文件中。该文件保存在 Eclipse 项目所在目录的某个子目录下，通常是项目根目录下的 Review 子目录。走查完毕，走查者需将.review 文件提交到 SCM 系统。

2）、团队走查（Team Phase） 个人更新本地工程以获得其他走查者的.review 文件，然后选择“Team Phase”。这样，所有在个人走查阶段建立的 Review Issue 都将会在本地呈现。此阶段主要完成问题的分析与指派。最后，将修改后的全部.review 文件重新提交到 SCM 系统。

3）、修改阶段（Rework Phase） 被查代码的所有者，根据走查者的意见对代码做出修改，然后修改 Review Issue 的状态。

由于使用 Jupiter 建立 Review Issue 是可以与代码关联的，并且是在 Eclipse 的集成环境下，相比较而言，我们记录 Issue 就会更加方便，而修改代码时也更容易将 Issue 与代码相对应。

按照以上介绍的方法来进行代码走查的项目，在代码品质上都有了普遍地提高。那么，你所在的项目呢？行动起来，相信代码走查这个锦囊能够为你的项目走向成功奠定一个很好的基础。

根据有关职位统计资料显示，在国外大多数软件公司，1 个软件开发工程师就需要辅有 2 个软件测试工程师。目前，软件测试自动化技术在我国则刚刚被少数业内专家所认知，而这方面的专业技术人员在国内更是凤毛麟角。根据对近期网络招聘 IT 人才情况的了解，许多正在招聘软件测试工程师的企业很少能够在招聘会上顺利招到合适的人才。

随着中国 IT 行业的发展，产品的质量控制与质量管理正逐渐成为企业生存与发展的核心。从软件、硬件到系统集成，几乎每个中大型 IT 企业的产品在发布前都需要大量的质量控制、测试和文档工作，而这些工作必须依靠拥有娴熟技术的专业软件人才来完成。而软件测试工程师就是其中之一。

据了解，由于软件测试工程师处于重要岗位，所以必须具有电子、电机类相关专业知识背景，并且还应有两年以上的实际操作经验。他们应熟悉中国和国际软件测试标准，熟练掌握和操作国际流行的系列软件测试工具，能够承担比较复杂的软件分析、测试、品质管理等任务，并能独立担任测试、品质管理部门的负责人。一般情况，软件测试工程师可分为测试工程师、高级测试工程师和资深测试工程师三个等级。

在具体工作过程中，测试工程师的工作是利用测试工具按照测试方案和流程对产品进行功能和性能测试，甚至根据需要编写不同的测试工具，设计和维护测试系统，对测试方案可能出现的问题进行分析和评估。对软件测试工程师而言，必须具有高度的工作责任心和自信心。任何严格的测试必须是一种实事求是的测试，因为它关系到一个产品的质量问题，而测试工程师则是产品出货前的把关人，所以，没有专业的技术水准是无法胜任这项工作的。同时，由于测试工作一般由多个测试工程师共同完成，并且测试部门一般要与其他部门的人员进行较多的沟通，所以要求测试工程师不但要有较强的技术能力而且要有较强的沟通能力。

什么是软件功能测试

Functional testing (功能测试)，也称为behavioral testing（行为测试），根据产品特征、操作描述和用户方案，测试一个产品的特性和可操作行为以确定它们满足设计需求。本地化软件的功能测试，用于验证应用程序或网站对目标用户能正确工作。使用适当的平台、浏览器和测试脚本，以保证目标用户的体验将足够好，就像应用程序是专门为该市场开发的一样。

功能测试也叫黑盒子测试或数据驱动测试,只需考虑各个功能，不需要考虑整个软件的内部结构及代码.一般从软件产品的界面、架构出发，按照需求编写出来的测试用例，输入数据在预期结果和实际结果之间进行评测,进而提出更加使产品达到用户使用的要求。

什么是软件性能

软件的性能是软件的一种非功能特性，它关注的不是软件是否能够完成特定的功能，而是在完成该功能时展示出来的及时性。由于感受软件性能的主体是人，不同的人对于同样的软件能有不同的主观感受，而且不同的人对于软件性能关心的视角也不同。由于目前网络应用非常普遍，因此下面将介绍网络应用软件性能的指标和软件性能的视角。

1.软件性能的指标

（1） 响应时间

响应时间是指系统对请求作出响应的时间。直观上看，这个指标与人对软件性能的主观感受是非常一致的，因为它完整地记录了整个计算机系统处理请求的时间。由于一个系统通常会提供许多功能，而不同功能的处理逻辑也千差万别，因而不同功能的响应时间也不尽相同，甚至同一功能在不同输入数据的情况下响应时间也不相同。所以，在讨论一个系统的响应时间时，人们通常是指该系统所有功能的平均时间或者所有功能的最大响应时间。当然，往往也需要对每个或每组功能讨论其平均响应时间和最大响应时间。

对于单机的没有并发操作的应用系统而言，人们普遍认为响应时间是一个合理且准确的性能指标。需要指出的是，响应时间的绝对值并不能直接反映软件的性能的高低，软件性能的高低实际上取决于用户对该响应时间的接受程度。对于一个游戏软件来说，响应时间小于 100 毫秒应该是不错的，响应时间在 1 秒左右可能属于勉强可以接受，如果响应时间达到 3 秒就完全难以接受了。而对于编译系统来说，完整编译一个较大规模软件的源代码可能需要几十分钟甚至更长时间，但这些响应时间对于用户来说都是可以接受的。

（2）系统响应时间和应用延迟时间

虽然软件性能指标本身只涉及软件性能的度量，但考虑到软件性能测试的主要目的是测试和改善所开发软件的性能，对于复杂的网络化的软件而言，简单地用响应时间进行度量就不一定合适了。

考虑一个普通的网站系统。开发该网站系统时，软件开发实际上只集中在服务器端，因为客户端的软件是标准的浏览器。虽然用户看到的响应时间时使用特定客户端计算机上的特定浏览器浏览该网站的响应时间，但是在讨论软件性能时更关心所开发网站软件本身的“响应时间”。也就是说，可以把用户感受到的响应时间划分为“呈现时间”和“系统响应时间”，前者是指客户端的浏览器在接收到网站数据时呈现页面所需的时间，而后者是指客户端接收到用户请求到客户端接收到服务器发来的数据所需的时间。显然，软件性能测试更关心“系统响应时间”，因为“呈现时间”与客户端计算机和浏览器有关，而与所开发的网站软件没有太大的关系。

如果仔细分析这个例子，还可以把“系统响应时间”进一步分解为“网络传输时间”和“应用延迟时间”，其中前者是指数据（包括请求数据和响应数据）在客户端和服务器端进行传输的时间，而后者是指网站软件实际处理请求所需的时间。类似的，软件性能测试也更关心

“应用延迟时间”。实际上，这种分解还可以继续下去，如果该网站系统使用了数据库，我们可以把“数据库延迟时间”分离出来，如果该网站系统使用了中间件，还可以把“中间件延迟时间”也分离出来。

以上的时间分解实际上有两方面的目的。首先，人们通常希望把与所开发软件直接相关的延迟时间和与所开发软件爱你不直接相关的延迟时间分离开，因为改善前者往往需要开发人员修改程序代码，而改善后者不需要开发人员修改代码，很多时候，开发人员对后者甚至是无能为力的。其次，详细的分解有助于开发人员分析哪些部分是影响软件性能的主要因素，以便于实时性能改善方案。

(3)吞吐量

吞吐量是指系统在单位时间内处理请求的数量。对于无并发的应用系统而言，吞吐量与响应时间成严格的反比关系，实际上此时吞吐量就是响应时间的倒数。前面已经说过，对于单用户的系统，响应时间（或者系统响应时间和应用延迟时间）可以很好地度量系统的性能，但对于并发系统，通常需要用吞吐量作为性能指标。

对于一个多用户的系统，如果只有一个用户使用时系统的平均响应时间是 t，当有你 n 个用户使用时，每个用户看到的响应时间通常并不是 n×t，而往往比 n×t 小很多（当然，在某些特殊情况下也可能比 n×t 大，甚至大很多）。这是因为处理每个请求需要用到很多资源，由于每个请求的处理过程中有许多不走难以并发执行，这导致在具体的一个时间点，所占资源往往并不多。也就是说在处理单个请求时，在每个时间点都可能有许多资源被闲置，当处理多个请求时，如果资源配置合理，每个用户看到的平均响应时间并不随用户数的增加而线性增加。实际上，不同系统的平均响应时间随用户数增加而增长的速度也不大相同，这也是采用吞吐量来度量并发系统的性能的主要原因。一般而言，吞吐量是一个比较通用的指标，两个具有不同用户数和用户使用模式的系统，如果其最大吞吐量基本一致，则可以判断两个系统的处理能力基本一致。

（4）并发用户数

并发用户数是指系统可以同时承载的正常使用系统功能的用户的数量。与吞吐量相比，并发用户数是一个更直观但也更笼统的性能指标。实际上，并发用户数是一个非常不准确的指标，因为用户不同的使用模式会导致不同用户在单位时间发出不同数量的请求。一网站系统为例，假设用户只有注册后才能使用，但注册用户并不是每时每刻都在使用该网站，因此具体一个时刻只有部分注册用户同时在线，在线用户就在浏览网站时会花很多时间阅读网站上的信息，因而具体一个时刻只有部分在线用户同时向系统发出请求。这样，对于网站系统我们会有三个关于用户数的统计数字：注册用户数、在线用户数和同时发请求用户数。由于注册用户可能长时间不登陆网站，使用注册用户数作为性能指标会造成很大的误差。而在线用户数和同事发请求用户数都可以作为性能指标。相比而言，以在线用户作为性能指标更直观些，而以同时发请求用户数作为性能指标更准确些。

（5）资源利用率

资源利用率反映的是在一段时间内资源平均被占用的情况。对于数量为 1 的资源，资源利用率可以表示为被占用的时间与整段时间的比值；对于数量不为 1 的资源，资源利用率可以表示为在该段时间内平均被占用的资源数与总资源数的比值。

2.软件性能的视角

（1）用户视角

对用和而言，性能就是响应时间。用户甚至不关心响应时间中哪些是软件造成的，哪些是硬件造成的。但用和感受到的响应时间既有客观成分，也有主观成分，甚至是心理因素 。

（2）管理员视角

管理员需要使用软件提供的管理功能等手段来方便普通用户使用。这类用户首先关注普通用户感受到的软件性能。其次，管理员需要进一步关注如何利用管理功能进行性能调优。

（3）开发人员视角

开发人员的视角与管理员的视角基本一致，但开发人员需要更深入地关注软件性能。在开发过程中，开发人员希望能够尽可能地开发出高性能的软件。

什么是软件的性能测试

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

性能测试在软件的质量保证中起着重要的作用，它包括的测试内容丰富多样。中国软件评测中心将性能测试概括为三个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，三方面有效、合理的结合，可以达到对系统性能全面的分析和瓶颈的预测。

- 应用在客户端性能的测试

应用在客户端性能测试的目的是考察客户端应用的性能，测试的入口是客户端。它主要包括并发性能测试、疲劳强度测试、大数据量测试和速度测试等，其中并发性能测试是重点。

并发性能测试是重点

并发性能测试的过程是一个负载测试和压力测试的过程，即逐渐增加负载，直到系统的瓶颈或者不能接收的性能点，通过综合分析交易执行指标和资源监控指标来确定系统并发性能的过程。负载测试（Load Testing）是确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统组成部分的相应输出项，例如通过量、响应时间、CPU 负载、内存使用等来决定系统的性能。负载测试是一个分析软件应用程序和支撑架构、模拟真实环境的使用，从而来确定能够接收的性能过程。压力测试（Stress Testing）是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

并发性能测试的目的主要体现在三个方面：以真实的业务为依据，选择有代表性的、关键的业务操作设计测试案例，以评价系统的当前性能；当扩展应用程序的功能或者新的应用程序将要被部署时，负载测试会帮助确定系统是否还能够处理期望的用户负载，以预测系统的未来性能；通过模拟成百上千个用户，重复执行和运行测试，可以确认性能瓶颈并优化和调整应用，目的在于寻找到瓶颈问题。

什么是数据驱动测试

黑盒测试（Black-box Testing，又称为功能测试或数据驱动测试）是把测试对象看作一个黑盒子。利用黑盒测试法进行动态测试时，需要测试软件产品的功能，不需测试软件产品的内部结构和处理过程。

采用黑盒技术设计测试用例的方法有：等价类划分、边界值分析、错误推测、因果图和综合策略。

黑盒测试注重于测试软件的功能性需求，也即黑盒测试使软件工程师派生出执行程序所有功能需求的输入条件。黑盒测试并不是白盒测试的替代品，而是用于辅助白盒测试发现其他类型的错误。

黑盒测试试图发现以下类型的错误：

1）功能错误或遗漏；

2）界面错误；

3）数据结构或外部数据库访问错误；

4）性能错误；

5）初始化和终止错误。

黑盒测试的测试用例设计方法：

- 等价类划分方法
- 边界值分析方法
- 错误推测方法
- 因果图方法
- 判定表驱动分析方法
- 正交实验设计方法
- 功能图分析方法

什么是容量测试

通过性能测试，如果找到了系统的极限或苛刻的环境中系统的性能表现，在一定的程度上，就完成了负载测试和容量测试。容量还可以看作系统性能指标中一个特定环境下的一个特定性能指标，即设定的界限或极限值。

容量测试的目的是通过测试预先分析出反映软件系统应用特征的某项指标的极限值（如最大并发用户数、数据库记录数等），系统在其极限状态下没有出现任何软件故障或还能保持主要功能正常运行。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。软件容量的测试能让软件开发商或用户了解该软件系统的承载能力或提供服务的能力，如某个电子商务网站所能承受的、同时进行交易或结算的在线用户数。知道了系统的实际容量，如是不能满足设计要求，就应该寻求新的技术解决方案，以提高系统的容量。有了对软件负载的准确预测，不仅能对软件系统在实际使用中的性能状况充满信心，同时也可以帮助用户经济地规划应用系统，优化系统的部署。

什么是嵌入式测试

通常嵌入式系统对可靠性的要求比较高。嵌入式系统安全性的失效可能会导致灾难性的后果，即使是非安全性系统，由于大批量生产也会导致严重的经济损失。这就要求对嵌入式系统，包括嵌入式软件进行严格的测试、确认和验证。随着越来越多的领域使用软件和微处理器控制各种嵌入式设备，对门益复杂的嵌入式软件进行快速有效的测试愈加显得重要。

软件测试的目的是保证软件满足需求规格说明。系统失效是系统没有满足一个或多个正式需求规范中所要求的需求项。嵌入式软件有其特殊的失效判定准则，但是，嵌入式软件测试的日的与非嵌入式软件是相同的。在嵌入式系统设计中，软件正越来越多地取代硬件，以降低系统的成本，获得更大的灵活性，这就需要使用更好的测试方法和工具进行嵌入式和实时软件的测试。

软件性能测试的步骤

在每种不同的系统架构的实施中，开发人员可能选择不同的实现方式，造成实际情况纷繁复杂。我们不可能对每种技术都详细解说，这里只是介绍一种方法提供给你如何选择测试策略，从而帮助分析软件不同部分的性能指标，进而分析出整体架构的性能指标和性能瓶颈。

由于工程和项目的不同，所选用的度量,评估方法也有不同之处。不过仍然有一些通用的步骤帮助我们完成一个性能测试项目。步骤如下

1. 制定目标和分析系统
2. 选择测试度量的方法
3. 学习的相关技术和工具
4. 制定评估标准
5. 设计测试用例
6. 运行测试用例
7. 分析测试结果

• 制定目标和分析系统

每一个性能测试计划中第一步都会制定目标和分析系统构成。只有明确目标和了解系统构成才会澄清测试范围，知道在测试中要掌握什么样的技术。

目标：

1. 确定客户需求和期望
2. 实际业务需求
3. 系统需求

系统组成

系统组成这里包含几方面含义：系统类别，系统构成，系统功能等。了解这些内容的本质其实是帮助我们明确测试的范围，选者适当的测试方法来进行测试。

系统类别：分清系统类别是我们掌握什么样的技术的前提，掌握相应技术做性能测试才可能成功。例如：系统类别是 bs 结构,需要掌握 http 协议，java，html 等技术 。或者是 cs 结构，可能要了解操作系统，winsock，com 等。所以甄别系统类别对于我们来说很重要。

系统构成：硬件设置，操作系统设置是性能测试的制约条件，一般性能测试都是利用测试工具模仿大量的实际用户操作，系统在超负荷情形下运作。不同的系统构成性能测试就会得到不同的结果。

系统功能：系统功能指系统提供的不同子系统，办公管理系统中的公文子系统，会议子系统等，系统工能是性能测试中要模拟的环节，了解这些是必要的。

・选择测试度量的方法

经过第一步，将会对系统有清醒的认识。接下来我们将把精力放在软件度量上，收集系统相关的数据。

度量的相关方面：

* 制定规范

* 制定相关流程，角色，职责

* 制定改进策略

* 制定结果对比标准

・学习的相关技术和工具

性能测试是通过工具，模拟大量用户操作，对系统增加负载。所以需要掌握一定的工具知识才能进行性能测试。大家都知道性能测试工具一般通过 winsock,http 等协议纪录用户操作。而协议选择是基于软件的系统架构实现（web 一般选择 http 协议,cs 选择 winsock 协议），不同的性能测试工具，脚本语言也不同，比如 rational robot 中 vu 脚本用类 c 语言实现。

开展性能测试需要对各种性能测试工具进行评估，因为每一种性能测试工具都有自身的特点，只有经过工具评估，才能选择符合现有软件架构的性能测试工具。确定测试工具后，需要组织测试人员进行工具的学习，培训相关技术。

・制定评估标准

任何测试的目的都是确保软件符合预先规定的目标和要求。性能测试也不例外。所以必须制定一套标准。

通常性能测试有四种模型技术可用于评估：

*线性投射：用大量的过去的，扩展的或者将来可能发生的数据组成散布图，利用这个图表不断和系统的当前状况对比。

*分析模型：用排队论公式和算法预测响应时间，利用描述工作量的数据和系统本质关联起来

*模仿：模仿实际用户的使用方法测试你的系统

*基准：定义测试和你最初的测试作为标准，利用它和所有后来进行的测试结果进行对比

・设计测试用例

设计测试用例是在了解软件业务流程的基础上。设计测试用例的原则是受最小的影响提供最多的测试信息，设计测试用例的目标是一次尽可能的包含多个测试要素。这些测试用例必须是测试工具可以实现的，不同的测试场景将测试不同的功能。因为性能测试不同于平时的测试用例，尽可能把性能测试用例设计的复杂，才有可能发现软件的性能瓶颈。

・运行测试用例

通过性能测试工具运行测试用例。同一环境下作的性能测试得到的测试结果是不准确的，所以在运行这些测试用例的时候，需要用不同的测试环境，不同的机器配置上运行。

・分析测试结果

运行测试用例后，收集相关信息，进行数据统计分析，找到性能瓶颈。通过排除误差和其他因素，让测试结果体现接近真实情况。不同的体系结构分析测试结果的方法也不同，bs

结构我们会分析网络带宽，流量对用户操作响应的影响，而 cs 结构我们可能更关心会系统整体配置对用户操作的影响。

什么是代码审查

代码审查(code review)是软件开发过程的一个阶段，在这个阶段中，代码创造者和审查人员，可能还有质量保证(QA)测试人员，一起进行代码审查。

代码审查(code review)是软件开发过程的一个阶段，在这个阶段中，代码创造者和审查人员，可能还有质量保证(QA)测试人员，一起进行代码审查。能在该阶段中就找出并更正存在的错误，相对来说比较合理，因为如果在开发软件后面的阶段或者软件交付给用户后才来处理、查找和修改程序缺陷的话，会要花费更多成本。

审查人员需要很仔细地检查代码，包括：

1. 缺陷或者潜在缺陷
2. 和整个程序设计的一致性
3. 评论的质量
4. 遵守编码标准

代码审查通常能很好地检测出安全漏洞问题。有一些专门的应用程序可以帮助进行代码审查。自动代码审查系统可以有效地系统化地检测源代码的潜在问题，如缓冲区溢出、竞态条件、内存泄露、代码块大小问题和重复语句等。另外，代码审查也常用于检测补丁质量。

什么是可用性测试

可用性测试是指，让一群有代表性的用户尝试对产品进行典型操作，同时观察员和开发人员在一旁观察，聆听，做记录。该产品可能是一个网站，软件，或者其他任何产品，它可能尚未成型。测试可以是早期的纸上原型测试，也可以是后期成品的测试。

你能从可用性测试获得什么?在每一轮的可用性测试中，你都应该先明确具体的测试问题和目标，针对这些目标进行测试。举例来说，项目刚刚起步，你可以对定量的指标(如时间，错误率和满意度)进行测试，为日后修改网站提供参照。再例如，如果你已经设定了可测量的可用性目标，你可以看看你的产品是否切合这些目标。对于一个典型的可用性测试，你可以：找出该产品的任何的可用性问题从测试参与者的表现收集定量数据确定该产品的用户满意度

可用性测试和以用户为中心的设计的关系?可用性测试是以用户为中心的设计的一个重要组成部分。用户为本的设计过程本身就应该包括对性能和偏好进行评价的一系列测试。

什么时候该做可用性测试?尽早做，经常做。可用性测试可以让设计师和开发团队在产品成形之前尽早发现问题。 问题越早发现和弥补，所造成的损失就越低。这些问题是找到并固定好，越昂贵的补丁程序。随着项目的进展，对设计主体进行改动会变得越来越困难和昂贵。你测试的越多，并就相应测试进行改进，你就可以更加确信你的网站没有偏轨，确信它是符合您的目标和用户的需要的。迭代开发过程——开发原型，测试用户，分析结果，随之修改原型，然后再重复测试、分析、修改周期——是开发一个成功的网站或软件的最好方式。

基准测试

基准测试的关键是要获得一致的、可再现的结果。可再现的结果有两个好处：减少重新运行测试的次数；对测试的产品和产生的数字更为确信。使用的性能测试工具可能会对测试结果产生很大影响。假定测试的两个指标是服务器的响应时间和吞吐量，它们会受到服务器上的负载的影响。服务器上的负载受两个因素影响：同时与服务器通信的连接（或虚拟用户）的数目，以及每个虚拟用户请求之间的考虑时间的长短。很明显，与服务器通信的用户越多，负载就越大。同样，请求之间的考虑时间越短，负载也越大。这两个因素的不同组合会产生不同的服务器负载等级。记住，随着服务器上负载的增加，吞吐量会不断攀升，直到到达一个点。

什么是软件需求

软件需求的定义

IEEE 软件工程标准词汇表(1997 年)中定义需求为：

(1)用户解决问题或达到目标所需的条件或权能(Capability)。

(2)系统或系统部件要满足合同、标准、规范或其它正式规定文档所需具有的条件或权能。

(3)一种反映上面(1)或(2)所描述的条件或权能的文档说明。

需求的层次

下面这些定义是需求工程领域中常见术语的定义说明。

软件需求包括三个不同的层次—业务需求、用户需求和功能需求—也包括非功能需求。业务需求(business requirement)反映了组织机构或客户对系统、产品高层次的目标要求，它们在项目视图与范围文档中予以说明。用户需求(user requirement) 文档描述了用户使用产品必须要完成的任务，这在使用实例(use case)文档或方案脚本(scenario)说明中予以说明。功能需求(functional requirement)定义了开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了业务需求。所谓特性(feature)是指逻辑上相关的功能需求的集合，给用户提供处理能力并满足业务需求。软件需求各组成部分之间的关系如图所示。

单元测试用例设计方法

测试用例的设计在单元测试中占有非常重要的地位，测试用例设计的好坏直接影响到测试的效果。确定测试用例之所以很重要，原因有以下几方面：

（1）测试用例构成了设计和制定测试过程的基础。

（2）测试的“深度”与测试用例的数量成比例。由于每个测试用例反映不同的场景、条件或经由产品的事件流，因而，随着测试用例数量的增加，对产品质量和测试流程也就越有信心。判断测试是否完全的一个主要评测方法是基于需求的覆盖，而这又是以确定、实施和/或执行的测试用例的数量为依据的。

（3）测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。

（4）测试设计和开发的类型以及所需的资源主要都受控于测试用例。测试用例通常根据它们所关联关系的测试类型或测试需求来分类，而且将随类型和需求进行相应地改变。

最佳方案是为每个测试需求至少编制两个测试用例：

（1）一个测试用例用于证明该需求已经满足，通常称作正面测试用例。

（2）另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

单元测试既可以是白盒测试也可以是黑盒测试。白盒测试主要是检查程序的内部结构、逻辑、循环和路径。其常用测试用例设计方法有：逻辑覆盖和基本路径测试。根据覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖，判定覆盖，判定－条件覆盖，条件组合覆盖及路径覆盖等。白盒测试用例设计还可用到：状态转移测试、数据定义－使用测试、等价类划分、边界值分析等。黑盒测试注重对程序功能方面的要求，它只用到程序的规格说明，没有用到程序的内部结构。其常用测试用例方法有：规范（规格）导出、等价类划分、边界值分析法、错误推测法和因果图分析方法。下面将简要介绍各个方法，更详细的说明请读者自行参考相关的测试理论书籍。

最佳方案是为每个测试需求至少编制两个测试用例：

（1）一个测试用例用于证明该需求已经满足，通常称作正面测试用例。

（2）另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

单元测试既可以是白盒测试也可以是黑盒测试。白盒测试主要是检查程序的内部结构、逻辑、循环和路径。其常用测试用例设计方法有：逻辑覆盖和基本路径测试。根据覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖，判定覆盖，判定－条件覆盖，条件组合覆盖及路径覆盖等。白盒测试用例设计还可用到：状态转移测试、数据定义－使用测试、等价类划分、边界值分析等。黑盒测试注重对程序功能方面的要求，它只用到程序的规格说明，没有用到程序的内部结构。其常用测试用例方法有：规范（规格）导出、等价类划分、边界值分析法、错误推测法和因果图分析方法。下面将简要介绍各个方法，更详细的说明请读者自行参考相关的测试理论书籍。

1. 语句覆盖

语句覆盖就是设计若干个测试用例，运行所测程序，使得每一可执行语句至少执行一次。

2. 判定覆盖

判定覆盖就是设计若干个测试用例,运行所测程序,使得程序中每个判断的取 TURE 分支和取 FALSE 分支至少经历一次。

3. 条件覆盖

条件覆盖就是设计若干个测试用例,运行所测程序,使得程序中每个判断的每个条件的可能取值至少执行一次。

4. 判定－条件覆盖

判定－条件覆盖就是设计足够的测试用例,使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行一次。也就是说要求各个判断的所有可能的条件取值组合至少执行一次。

5. 条件组合覆盖

条件组合覆盖就是设计足够的测试用例,运行所测程序,使得每个判断得所有可能得条件取值组合至少执行一次。

6. 路径覆盖

路径测试就是设计足够的测试用例,覆盖程序中所有可能的路径。

7. 规范（规格）导出法

规范导出法是根据相关的规范描述来设计测试用例。每一个测试用例用来测试一个或多个规范陈述语句。一个比较实际的方法是根据陈述规范所用语句的顺序来相应地为被测单元设计测试用例。

8. 状态转移测试法

对于那些以状态机作为模型或设计为状态机的软件，状态转移测试是合适的测试方法。测试用例通过能导致状态迁移的事件来测试状态之间的转换。

9. 数据定义－使用测试法

数据定义是指数据项被赋值的地方，数据使用是指数据项被读或使用的地方。目的是设计测试用例以驱动执行通过数据定义于使用之间的路径。

单元测试用例设计方法

测试用例的设计在单元测试中占有非常重要的地位，测试用例设计的好坏直接影响到测试的效果。确定测试用例之所以很重要，原因有以下几方面：

（1）测试用例构成了设计和制定测试过程的基础。

（2）测试的“深度”与测试用例的数量成比例。由于每个测试用例反映不同的场景、条件或经由产品的事件流，因而，随着测试用例数量的增加，对产品质量和测试流程也就越有信心。判断测试是否完全的一个主要评测方法是基于需求的覆盖，而这又是以确定、实施和/或执行的测试用例的数量为依据的。

（3）测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。

（4）测试设计和开发的类型以及所需的资源主要都受控于测试用例。测试用例通常根据它们所关联关系的测试类型或测试需求来分类，而且将随类型和需求进行相应地改变。

最佳方案是为每个测试需求至少编制两个测试用例：

（1）一个测试用例用于证明该需求已经满足，通常称作正面测试用例。

（2）另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

单元测试既可以是白盒测试也可以是黑盒测试。白盒测试主要是检查程序的内部结构、逻辑、循环和路径。其常用测试用例设计方法有：逻辑覆盖和基本路径测试。根据覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖，判定覆盖，判定－条件覆盖，条件组合覆盖及路径覆盖等。白盒测试用例设计还可用到：状态转移测试、数据定义－使用测试、等价类划分、边界值分析等。黑盒测试注重对程序功能方面的要求，它只用到程序的规格说明，没有用到程序的内部结构。其常用测试用例方法有：规范（规格）导出、等价类划分、边界值分析法、错误推测法和因果图分析方法。下面将简要介绍各个方法，更详细的说明请读者自行参考相关的测试理论书籍。

1. 语句覆盖

语句覆盖就是设计若干个测试用例，运行所测程序，使得每一可执行语句至少执行一次。

2. 判定覆盖

判定覆盖就是设计若干个测试用例,运行所测程序,使得程序中每个判断的取 TURE 分支和取 FALSE 分支至少经历一次。

3. 条件覆盖

条件覆盖就是设计若干个测试用例,运行所测程序,使得程序中每个判断的每个条件的可能取值至少执行一次。

4. 判定－条件覆盖

判定－条件覆盖就是设计足够的测试用例,使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行一次。也就是说要求各个判断的所有可能的条件取值组合至少执行一次。

5. 条件组合覆盖

条件组合覆盖就是设计足够的测试用例,运行所测程序,使得每个判断得所有可能得条件取值组合至少执行一次。

6. 路径覆盖

路径测试就是设计足够的测试用例,覆盖程序中所有可能的路径。

7. 规范（规格）导出法

规范导出法是根据相关的规范描述来设计测试用例。每一个测试用例用来测试一个或多个规范陈述语句。一个比较实际的方法是根据陈述规范所用语句的顺序来相应地为被测单元设计测试用例。

8. 状态转移测试法

对于那些以状态机作为模型或设计为状态机的软件，状态转移测试是合适的测试方法。测试用例通过能导致状态迁移的事件来测试状态之间的转换。

9. 数据定义－使用测试法

数据定义是指数据项被赋值的地方，数据使用是指数据项被读或使用的地方。目的是设计测试用例以驱动执行通过数据定义于使用之间的路径。

数据定义是指数据项被赋值的地方，数据使用是指数据项被读或使用的地方。目的是设计测试用例以驱动执行通过数据定义于使用之间的路径。

软件测试的概念：

作为软件质量控制中的重要一环，软件测试工程师应运而生。软件测试工程师的工作就是利用测试工具按照测试方案和流程对产品进行功能测试和性能测试，甚至根据需要编写不同的测试工具，设计和维护测试系统，对侧四方案可能出现的问题惊醒分析和评估。执行测试用例后，需要跟踪故障，以确保开发的差频频满足需要。手工测试、手工运行软件，发现软件问题的过程；

测试计算器的加法功能，设计测试用例（举一个例子）：输入“2”、点击“+”、输入“3”，点击“=”，预期结果为“5”；

自动测试

上面的一个测试用例也可以使用自动测试工具来测，基本原理是使用自动测试工具把测试过程录制下来（脚本），包括预期结果；下次测试时重放录制下来的脚本，等于使用工具再次运行这个程序，这时候就不需要人工参与；

前面提到公务员查分系统，在测试这个例子的时候，不可能找成千上万的软件测试工程师来模拟用户，只有借助自动测试工具模拟大量用户进行测试；

功能测试

测试软件的功能是否满足用户需求；

例如测试计算器就是典型的功能测试；

性能测试

例 1：写了一个 10000 页的 word 文档，保存起来需要多长时间？如果需要 1 个小时，则说明性能不好；

例 2：例如需要测试给 1000 开 50 次方，计算器计算需要多长时间，如果时间超过 1 分钟，即使最终得出正确结果，用户也不能接受。

黑盒测试

把被测软件看成一个黑盒，不知道软件内部代码怎么写的，根据用户需求进行的测试；

白盒测试

能看到软件的代码，依据软件内部结构（代码）进行的测试（好处：可以通过测试证明内部操作是否符合要求）

什么是 Ad-hoc 测试

“Ad-Hoc” 原意是指 “特定的，一次性的”，这里专指“随机的，自由的”测试。在软件测试中除了根据测试样例和测试说明书进行测试外，还需要进行随机测试(Ad-hoc testing)，主要是根据测试者的经验对软件进行功能和性能抽查。随机测试是根据测试说明书执行样例测试的重要补充手段，是保证测试覆盖完整性的有效方式和过程。

什么是测试驱动开发（Test-driven development）

测试驱动开发（Test-driven development）是现代计算机软件开发方法的一种。利用测试来驱动软件程序的设计和实现。测试驱动开始流行于 20 世纪 90 年代。测试驱动开发是极限编程中倡导的程序开发方法，方法主要是先写测试程序，然后再编码使其通过测试。测试驱动开发的目的是取得快速反馈并使用“illustrate the main line”方法来构建程序。

测试驱动开发的比喻。开发可以从两个方面去看待：实现的功能和质量。测试驱动开发更像两顶帽子思考法的开发方式，先戴上实现功能的帽子，在测试的辅助下，快速实现正确的功能；再戴上重构的帽子，在测试的保护下，通过去除冗余和重复的代码，提高代码重用性，实现对质量的改进。可见测试在测试驱动开发中确实属于核心地位，贯穿了开发的始终。

测试驱动开发中测试的特征

测试驱动开发中需求分析和详细设计的范畴，在代码基本完毕以后，这些测试也成为单元测试的一个部分。

应用领域

新软件的开发，历史系统的维护。

测试驱动开发相关讨论

正面评价

＊ 可以有效的避免过度设计带来的浪费。但是也有人强调在开发前需要有完整的设计再实施可以有效的避免重构带来的浪费。

＊ 可以让开发者在开发中拥有更全面的视角，避免过度实现带来的浪费。

负面评价

＊ 开发者可能只完成满足了测试的代码，而忽略了对实际需求的实现。有实践者认为用结对编程的方式可以有效的避免这个问题。

＊ 会放慢开发实际代码的速度，特别对于要求开发速度的原型开发造成不利。这里需要考虑开发速度需要包含功能和品质两个方面，单纯的代码速度可能不能完全代表开发速度。

＊ 对于 GUI,资料库和 Web 应用而言。构造单元测试比较困难，如果强行构造单元测试，反而给维护带来额外的工作量。有开发者认为这个是由于设计方法，而不是开发方法造成的困难。

＊ 使得开发更为关注用例和测试案例，而不是设计本身。目前，对于这个观点有较多的争议。

＊ 测试驱动开发会导致单元测试的覆盖度不够，比如可能缺乏边界测试。在实际的操作中，和非测试驱动开发一样，当代码完成以后还是需要补充单元测试，提高测试的覆盖度。

什么是代码审查

代码审查(code review)是软件开发过程的一个阶段，在这个阶段中，代码创造者和审查人员，可能还有质量保证(QA)测试人员，一起进行代码审查。

代码审查(code review)是软件开发过程的一个阶段，在这个阶段中，代码创造者和审查人员，可能还有质量保证(QA)测试人员，一起进行代码审查。能在该阶段中就找出并更正存在的错误，相对来说比较合理，因为如果在开发软件后面的阶段或者软件交付给用户后才来处理、查找和修改程序缺陷的话，会要花费更多成本。

审查人员需要很仔细地检查代码，包括：

1. 缺陷或者潜在缺陷
2. 和整个程序设计的一致性
3. 评论的质量
4. 遵守编码标准

代码审查通常能很好地检测出安全漏洞问题。有一些专门的应用程序可以帮助进行代码审查。自动代码审查系统可以有效地系统化地检测源代码的潜在问题，如缓冲区溢出、竞态条件、内存泄露、代码块大小问题和重复语句等。另外，代码审查也常用于检测补丁质量。

什么是软件架构

软件架构是有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。

软件体系结构是构建计算机软件实践的基础。与建筑师设定建筑项目的设计原则和目标，作为绘图员画图的基础一样，一个软件架构师或者系统架构师陈述软件构架以作为满足不同客户需

求的实际系统设计方案的基础。从和目的、主题、材料和结构的联系上来说，软件架构可以和建筑物的架构相比拟。一个软件架构师需要有广泛的软件理论知识和相应的经验来事实和管理软件产品的高级设计。软件架构师定义和设计软件的模块化，模块之间的交互，用户界面风格，对外接口方法，创新的设计特性，以及高层事物的对象操作、逻辑和流程。

软件架构师与客户商谈概念上的事情，与经理商谈广泛的设计问题，与软件工程师商谈创新的结构特性，与程序员商谈实现技巧，外观和风格。

软件架构是一个系统的草图。软件架构描述的对象是直接构成系统的抽象组件。各个组件之间的连接则明确和相对细致地描述组件之间的通讯。在实现阶段，这些抽象组件被细化为实际的组件，比如具体某个类或者对象。在面向对象领域中，组件之间的连接通常用接口来实现。

架构描述语言

架构描述语言(ADL)用于描述软件的体系架构。现在已有多种架构描述语言，如 Wright (由卡内基梅隆大学开发)，Acme (由卡内基梅隆大学开发)，C2 (由 UCI 开发)， Darwin (由伦敦帝国学院开发)。ADL 的基本构成包括组件、连接器和配置。

软件架构一般来说组织成视图，如同在建筑学中的不同种类的蓝图。一些可能的视图有：

* 功能/逻辑视图
* 代码视图
* 开发/结构视图
* 并行/过程/线程视图
* 物理/部署视图
* 用户动作/反馈视图

有许多为描述软件架构的语言被开发出来，但是关于应该采用什么样的符号集和视图系统还没有达成共识。一些人相信 UML 将建立一套软件架构视图的标准。

设计软件模块以及模块之间的通信有很多常用手段，包括

* 客户端服务器
* 分布式计算
* 对等系统
* 黑板
* 隐式调用
* 插件
* 单层系统
* 三层结构
* 结构化(基于模块，但在模块内部是一体的)
* 基于软件构件 (基于模块，在模块内部，通常采用面向对象程序设计方法，slightly less monolithic)
* 面向服务的体系架构

书写专业软件问题报告的技巧

书写软件问题报告的目的是为了正确地重复缺陷或错误，从而在后续工作中可以准确验证并加以处理。因此，基本要求是准确、简洁、完整、规范。为了正确书写专业的软件问题报告，应该注意以下要点：

* 每个软件问题报告只书写一个缺陷或错误

这样可以每次只处理一个确定的错误，定位明确，提高效率，也便于修复错误后方便的进行验证。

* 对错误的描述要做到简洁、准确、完整，揭示错误实质

描述要准确反映缺陷或错误的本质内容，简短明了。为了便于在答数据库中寻找，包含错误发生时的用户界面是个良好的习惯。例如记录对话框的标题、菜单、按钮等控件的名称。

* 明确指明错误类型和严重程度

根据错误的现象，总结判断错误的类型和严重程度，例如，是功能错误？还是界面布局错误？该错误是属于特别严重的错误还是一般错误？是否影响软件的后续开发和发布？

* 每一个步骤尽量只记录一个操作

简洁、条理井然，容易重复操作步骤，以便确认、修复、验证该错误。

* 复现的操作步骤要完整，准确，简短

保证快速准确的重复错误，“完整”即没有缺漏，“准确”即步骤正确，“简短”即没有多余的步骤。

* 附加必要的错误特征图像

为了直观的观察缺陷或错误现象，通常需要附加错误出现的界面，作为附件附着在记录的“附件”部分。为了节省空间，又能真实反映缺陷或错误本质，可以捕捉缺陷或错误产生时的全屏幕，活动窗口和局部区域。

* 附加必要的测试用例

如果打开某个特殊的测试用例而产生的错误，则必须附加该测试用例，从而可以迅速再现缺陷或错误。为了使错误修正者进一步明确缺陷或错误的表现，可以附加修改建议或注解。

* 尽量使用短语和短句，避免复杂句型句式

书写软件问题报告的目的是便于定位错误，因此，要求客观的描述操作步骤，不需要修饰性的词汇和复杂的句型，增强可读性。