第一部分 软件测试综述

软件测试-机械工业出版社 （美）Ron Patton 著 周予滨 姚静等译
雪舞奉天读书笔记

说真的，这本书真的很不错，里面的一些定义很权威的，而且话不罗嗦，讲的都是重点，美中不足的在测试用例设计方法那块不完整。许多人在推荐入门看什么书的时候都提到此书，为了方便新手学习（其实我也是新手哈哈），我决定把我以前的读书笔记敲出来贴在网上，写的不是太全，主要是我觉得不错的东西。在此感谢此书作者和翻译人员！

软件测试读书笔记之一软件测试背景

一.软件缺陷的正式定义：

符合下边 5 个规则的才能叫做软件缺陷。
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范围。
4.软件未达到产品说明书虽未指出但应达到的目标。
5.软件测试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。

二.软件缺陷的产生原因：

导致软件缺陷最大的原因是产品说明书；第二大来源是设计方案；三是代码；四是某些软件缺陷产生的条件被错误地认定。

三.软件缺陷的修复费用：

随时间增长，修复软件缺陷的费用是呈几何数级增长的，随时间推移，数十倍增长。

四.软件测试人员的目的：

软件测试远的目标就是发现软件缺陷，尽可能早一些，并确保其得以修复。

五.怎么成为优秀测试员：
1.探索精神
2.故障排除能手
3.不懈努力
4.创造性
5.追求完美
6.判断准确
7.老练稳重
8.说服力
9.除了这些素质，在软件编程方面受过的教育也是重要的。
10.软件的功能为了解决现实问题，因此，教学烹饪航空木工医疗等知识都将对查找该领域软件的缺陷有莫大帮助

软件测试读书笔记之二软件开发过程

一.测试文挡包括：
1.测试计划
2.测试案例
3.软件缺陷报告
4.归纳，统计和总结。

二.软件产品由哪些部分组成（都是要测的哦，当然我国许多软件都无法达到这么多部分~呵呵）
1. 最终产品（光盘/软盘/程序...）
2.帮助文件
3.用户手册
4.样本和示例
5.标签和帖子
6.产品支持信息
7.图标和标志
8.错误信息
9.广告和宣传材料
10.安装
11.说明文件
这些都是要测试的，书中尤其提到了不要忘了测试错误提示信息（错误提示信息是软件产品最容易忽视的部分，通常是有程序员而不是训练有素的稿手来写的。这
些信息很少照顾到修复软件缺陷的需要，还常常造成麻烦。软件测试员也难以找到并显示全部信息。在软件中不要加入吓人和不友好的错误提示信息。）

三.软件开发模式

1.大棒式：所有精力都在开发软件和编写代码上

2.边写边改式：没有时间做好，总有时间返工哈哈！这句话经典，测试者几乎每天都

拿到一个新版本，新版本出来的时候，旧版本还没测完！而新版本还包含新的或者经过修改的功能）

3.流水式：创意-分析-设计-开发-测试-最终产品，只许前进不能后退！

4.螺旋式：开始不必详细定义所有细节。从小开始，定义重要功能，努力实现，接受

客户反馈，然后进入下一阶段。（一个螺旋包括 6 个步骤：1.确定目标，可选方案和限

制条件；2.指出并解决风险；3.评估方案；4.本阶段开发和测试；5.计划下一阶段；

6.确定进入下一阶段的方法。 ）测试一直在进行，知道最后宣布成功！

软件测试读书笔记之三软件测试的实质

一.测试人员要知道的几个'交通规则'和'生活法则'~

1.完全测试是不可能的。A.输入量太大；B.输出结果太多；C.软件实现途径太多；D.软件说明书没有客观标准。从不同角度看，软件缺陷标准不同。

2.软件测试是有风险行为。

3.测试无法显示潜伏的软件缺陷。

4.找到的软件缺陷越多，就说明软件缺陷越多。

5.老用一种药，害虫都有抵抗力，程序也如此，如在螺旋开发模式中，每一个轮回都

会对软件进行测试，几回合后，该发现的都发现了，找不到什么错误了。这要求我们必

须不断编写不同的新测试程序，对程序的不同部分进行测试，以找到更多的缺陷。

6.并非所有的软件缺陷都能修复：A.没有足够的时间；B.不算真正的缺陷；C.修复风险太大；D.不值得修复

7.难以说清的软件缺陷

8.产品说明书不断变化：软件测试员必须想到产品说明书可能改变。

9.测试员做的工作不受欢迎，因为工作就是挑错！所以我们要懂得怎么和开发的相处：

A.早点找出缺陷；B.控制情绪；C.多交流，不要总是报告坏消息。

10.软件测试是一项讲究条理的技术专业。

二.软件测试的术语和定义

这里引用下网上的术语总结，对原作者表示歉意和谢意和敬意！（不知道是谁）

1.精确和准确：A.精确参照物是目标。与目标越接近，就越准确；B：准确参照物是

每次实施的结果。几次结果相互之间越接近，表示越精确。但与目标可能相去甚远.

2.验证和合法性检查：A.验证保证软件符合产品说明书的过程 B.合法性检查保证软
件满足用户要求的过程.
3.质量和可靠性：可靠性只是质量的一个方面。A.质量可能包含功能是否齐全，产
品能否在各种机器上运行，软件公司有没有技术支持，甚至包装盒的色彩，可靠性
或者软件产品是否经常毁坏数据可能也很重要，但不绝对。B.可靠性：你自己想吧，
我没找到定义哈哈~
4.测试和质量评判(QA)：A.软件测试员的目标是找出软件缺陷，尽可能造一些，确保得以修复；B.软件质量评判人员的主要指责是创建和加强促进软件开发并防止
软件缺陷的标准和方法

第二部分 测试基础

软件测试读书笔记之四检查产品说明书

一.开始测试

1.A：黑盒测试：软件测试员只需知道软件要做什么，无法看到如何运作。只进行输入操作来得到输入结果。
B：白盒测试：软件测试员可以访问程序员的代码，并通过检查代码来协助测试。
2.A：静态测试：测试不运行的部分—只是检查和审阅。
B：动态测试：指通常意义上的测试—运行和使用软件。
3.测试产品说明书属于静态黑盒测试。

二.对产品说明书进行高级审查

测试产品说明书第一步不是去找软件缺陷，而是在一个高度上审视。审查产品说明书是为了找出根本性大问题，疏忽或遗漏之处。
1.占在客户角度思考:设身处地的为客户着想,测试的时候把自己当成客户。
2.研究现有的标准和规范:软件测试员的任务不是定义润件要符合何种标准和规范，而是观察，检验是否套用正确的标准，没有遗漏。
3.审查和测试同类软件:同类软件有助于制订测试条件和测试方法，还可能暴露

没想到的潜在问题。

三.产品说明书的低级测试技术

1.优秀产品说明书应当具有的 8 个属性

A.完整。是否有遗漏和丢失？完全吗？单独使用是否包含全部内容？

B.准确。解决方案正确吗？目标明确吗？有没有错误？

C.精确、不含糊、清晰。描述是否一清二楚？还是自说自话？ 容易看懂和理解吗？

D.一致。产品功能描述是否自相矛盾？与其他功能有无冲突？

E.贴切。描述功能的陈述是否必要？有没有多余信息？功能是否原来的客户要求？

F.合理。在特定预算和进度下，以现有人力、物力和资源能否实现？

G.代码无关。是否坚持定义产品，而不是定义其所依赖的设计、架 构和代码？

H.可测试。特性能否测试？测试员建立验证操作的测试错误程序是否提供足够的信息？

2.产品说明书 7 个用语检查清单

A.总是、每一种、所有、没有、从不。

看到此类绝对或肯定的切实认定的叙述，可以着手设计针锋相对的案例。

B.当然、因此、明显、显然、必然。

这些话意图诱使接受假定情况。不要中了圈套。

C.某些、有时、常常、通常、经常、大多、几乎。

这些话太过模糊。“有时”发生作用的功能无法测试

D.等等、诸如此类、依此类推。

以这样的词结束的功能清单无法测试。功能清单要绝对或者解释明确。

E.良好、迅速、廉价、高效、稳定。

这些是不确定的说法，不可测试。如果在产品说明书出现，必须要求进一步指明含义。

F.已处理、已拒绝、已忽略、已消除。

这些说法可能会隐藏大量需要说明的功能。

G.如果...那么...（没有否则）。

缺少配套的否则，想一想，“如果”没有发生会怎样呢？

软件测试读书笔记之五闭着眼睛测试软件

一.动态黑盒测试

1.不深入代码细节的软件测试方法称为动态黑盒子测试。它是动态的，因为程序正在运行；它是黑盒子，因为测试时不知道程序如何工作。测试工作就是进行输入，接受输出，检验结果。

2.首先要弄清楚作为测试对象的软件要输入什么得到什么，或者操作结果。这就要求有文挡或产品说明书；接下来开始定义测试案例（就是我们常说的测试用例）

3.选择测试案例是软件测试员最重要的任务。不正确的选择可能导致测试量

过大或者过小，甚至测试目标不对。准确评估风险，把不可穷近的可能性减少到可以控制的范围是成功的诀窍。

*4.没有产品说明书的情况下使用探索测试。（这个我觉得很重要，因为国内大部分软件都是这样的，因为国内大部分软件都是这样的，什么说明都没有,没有需求说明,没有产品说明书,没有设计书.....呵呵,这就是有中国特色的软件测试吧~~，遇到这种情况不要烦躁，"把软件当成产品说明书来对待。分步骤地逐项探索软件特性。记录软件执行情况，详细描述功能。在这种情况下，无法像有产品说明书那样完整的测试软件--比如无法断定是否遗漏功能，但是可以进行系统测试。找到软件缺陷几乎是肯定的." 小雪经验总结:这种情况还要多和开发的沟通，在他们那了解软件更多的情况。他们自己写的,没有人比他们知道的多.这种测试会遇到很多你认为逻辑不合理的地方,因为没有需求说明,开发的完全照自己的意思来编写代码.有的是多人编写,每人负责一个模块,模块之间衔接和整个软件的业务逻辑多会有许多问题.

二.通过测试和失败测试

通过测试：确认软件至少能做什么，而不考验其能力。只运用最简单，最直观的测试案例。
失败测试：纯粹为了破坏软件而设计和执行的测试案例。

设计和执行测试案例时，总是首先进行通过测试。在破坏性试验之前看看软件基本功能是否实现是很重要的，否则在正常使用软件时就会奇怪为什么有那么多的软件缺陷。常见的测试案例就是设法迫使软件出现错误提示信息。

三.等价分配

等价分配（等价类划分）：是指分步骤地把过多（无限）的测试案例减小到同样有效的小范围的过程。

等价类别或者等价区间是指测试相同目标或者暴露相同软件缺陷的一组测试案例。在寻找等价区间时，想办法把软件的相似输入、输出、操作分成组。这些组就是等价区间。等价分配的目标是把可能的测试案例组合缩减到仍然足以测试软件的控制范围。因为选择了不完全测试，就要冒一定的风险。如果为了减少测试案例的数量过度进行等价分配，测试的风险就会增加。另外，等价区间的划分没有一定的标准，只要足以覆盖测试对象就行了。

（个人认为这里讲的不是很好，在笔记前我就说了，本书测试用例设计方法上做的不是很好，有关知识大家上网看吧，写的很详细，推荐一个风姿清扬整理的测试用例设计方法~。以后遇到相关测试用例设计的问题我都引用一些比较流行的通俗的知识或者直接省去了`。 我们设计用例数据的时候按照等价类划分方法：

等价类分为有效等价类和无效等价类，有效等价类就是由那些对程序的规格说明有意义的、合理的输入数据所构成的集合；无效等价类就是那些对程序的规格说明不合理的或无意义的输入数据所构成的集合。
划分等价类的方法：下面给出六条确定等价类的原则。
1、在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。

2、在输入条件规定了输入值的集合或者规定了"必须如何"的条件的情况下，可确立一个有效等价类和一个无效等价类。
3、在输入条件是一个布尔量的情况下，可确定一个有效等价类。
4、在规定了输入数据的一组值（假定 n 个），并且程序要对每一个输入值分别处理的情况下，可确立 n 个有效等价类和一个无效等价类。
5、在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。
6、在确知已划分的等价类中各元素在程序处理中的方式不同的情况下，则应再将该等价类进一步的划分为更小的等价类。）

四.数据测试

软件由数据和程序组成。数据包括键盘输入、鼠标单击、磁盘文件、打印输出等等；程序指可执行的流程、转换、逻辑和运算。

对数据进行软件测试，就是在检查用户输入的信息、返回结果以及中间计算结果是否正确。主要根据下列原则来进行等价分配，以合理减少测试案例：边界条件，次边界条件，空值和无效数据。

（个人认为书里介绍边界值这块不是很好，新手还是看下面的吧，流行的比较经典的是边界值分析法：
上点，就是边界上的点，不管它是开区间还是闭区间，就是说，如果该点是封闭的，那上点就在域范围内，如果该点是开放的，那上点就在域范围外；
内点，就是在域范围内的任意一个点；
离点，就是离上点最近的一个点，如果边界是封闭的，那离点就是域范围外离上点最近的点，如果边界是开放的，那离点就是域范围内离上点最近的点。
边界值分析方法的原则：
1、如果输入（输出）条件规定了取值范围，则应该以该范围的边界值及边界附近的值作为测试数据；
2、如果输入（输出）条件规定了值的个数，则用最大个数，最小个数，比最小个数少一，比最大个数多一的数作为测试数据；
3、如果程序规格说明书中提到的输入或输出是一个有序的集合，应该注意选取有序集合的第一个和最后一个元素作为测试数据；
4、如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试数据。）

五.状态测试。

软件状态是指软件当前所处的情况或者模式。软件通过代码进入某一个流程分支，触发一些数据位，设置某些变量，读取某些变量，而转入一个新的状态。软件测试员必须测试软件的状态及其转换。
1.测试软件的逻辑流程。状态测试运用等价分配技术选择状态和分支。因为选择不完全测试，所以要承担一定的风险，但是通过合理选择会减少危险。

2.建立状态转换图。包括的内容有：
A.软件可能进入的每一种独立状态；
B.如果不能断定是否为独立状态，就算它是，以后发现不是，随时把它 T 开；
C.从一种状态转入另一种状态所需的输入和条件。找出什么操作导致的变化；
D.进入或退出某种状态时的设置条件及输出结果。包括显示的菜单和按钮、设置的标志位、产生的打印输出、执行的运算等等。这些是状态转换时发生的部分或全部现象。

3.减少要测试的状态及转换的数量。
A.每种状态至少访问一次；
B.测试看起来最常见最普遍的状态转换；
C.测试状态之间最不常用的分支。
D.测试所有错误状态及其返回值;
E.测试随机状态转换。
4.具体测试的进行。确定要测试的状态及其转换之后,就可以定义测试案例了。测试状态及其转换包括检查所有的状态变量——与进入和退出状态相关的静态条件、信息、值、功能等等。状态变量也许不可见，但是很重要。

(建议看因果图法写测试用例呵呵)

六.失败状态测试

1.竞争条件和时序错乱:在真正的多任务环境中软件设计绝对不能想当然，必须处理随时被中断的情况,能够与其他任何软件在系统中同时运行,并且共享内存、磁盘、通信设备以及其他硬件资源。这一切的的结果就可能导致竞争条件问题.这些问题的几个事件恰好挤在一起,软件未预料到的操作过程被中断,时序就会发生错乱。竞争条件测试难以设计，最好是首先仔细查看状态转换图中的每一个状态，以找出哪些外部影响会中断该状态。考虑要使用数据如果没有准备好，或者在用到时发生了变化，状态会怎样。数条弧线或者直线同时相连的情形如何。
下是要面临竞争条件的典型情形：
A.两个不同的程序同时保存或打开同一个文档。
B.共享同一台打印机、通信端口或者其他外围设备。
C.当软件处于读取或者修改状态时按键或者单击鼠标。
D.同时关闭或者启动软件的多个实例。
E.同时使用不同的程序方位一个共同数据库。

2.重复、压迫和重负
测试的目标是处理那些连程序员都没有想到的恶劣条件下产生的问题的能力。
A.重复测试是不断执行同样的操作。最简单的是不停地启动和关闭程序，或者反复读写数据或者选择同一个操作。这种测试的主要目的是看内存是否不足。如果内存被分配进行某项操作，但操作完成时没有完全释放，就会产生一个常见的软件问题。
B.压迫测试是使软件在不够理想的条件下运行——内存小、磁盘空间少、CPU 速

度慢、调制解调器速率低等等。观察软件对外部资源的要求和依赖程度。压迫测试就是将支持降到最低限度，目的在于尽可能的限制软件的必要条件。
C.重负测试和压迫测试相反。压迫测试是尽量限制软件，而重负测试是尽量提供条件任其发挥。让软件处理尽可能大的数据文件。最大限度的发掘软件的能力，让它不堪重负。比如：软件对打印机或通信端口进行操作，就把能连的都连上；服务器可以处理几千个模拟连接，就按他说的做。
三者应联合使用，同时进行。
注意事项:
A.项目管理员和小组程序员可能不完全接受软件测试员这样打破软件的做法。但是软件测试员的任务就是确保软件在这样恶劣的条件下正常工作，否则就报告软件缺陷。如何以最佳方式报告软件缺陷，使其得到严肃对待和修复，也是一门学问。
B.无数次重复和上千次的连接对于手工操作是不可能的。因而需要借助自动化测试工具来实现。

七.其他黑盒测试技术
1.像新用户那样做,随意操作.
2.在已经找到软件缺陷的地方再找找(80%的缺陷通常集中在 20%的模块)
3.凭借经验、直觉和预感. (软件测试确实是越有经验越吃香啊!,像我们这样的只能好好学习,多多实践,多多积累,不断总结)

呼! 这章怎么这么长啊!排版很乱,有时间再整理吧,对不起大家的眼睛了,再看看这章名字,闭着眼睛..呵呵,看的眼睛痛了就闭眼睛想一会吧,

软件测试读书笔记之六检查代码

软件测试不仅仅是检查产品说明书和闭着眼睛测试软件，还有对软件设计和代码进行测试。因为在测试军队，金融，工业，医药类软件或者在组织严格的开发模式下工作代码和产品检验是例行公事。

一.静态白盒子测试：检查设计和代码
静态测试是指测试非运行部分——检查和审查。白盒测试是指访问代码，能够查看和审查。静态白盒测试是在不执行的条件下有条理地仔细审查软件设计、体系结构和代码，从而找出软件缺陷的过程。有时也称为结构分析。
进行静态白盒子测试的首要原因就是尽早发现软件缺陷，以找出

动态黑盒子测试难以揭示或遇到的软件缺陷；另一个好处是为接受该软件测试的黑盒测试员的测试案例提供思路，他们不必了解代码细节，但是根据审查备注，可以确定似乎有问题或者存在软件缺陷的特性范围。

二.正式审查

正式审查就是进行静态白盒子测试的过程。正式审查含义广泛，从程序员之间的交谈，到代码的严格检查均属于此过程。

有 4 个基本要素：

1.确定问题。审查的目标是找出软件的问题，不仅是出错的项目，还包括遗漏的项目。全部的批评应直指代码，而不是其创建者。合作者不应该互相指责。个人情绪化感觉要保留。

2.遵守规则。 审查要遵守一套固定的规则，规则可能设定要审查的代码量、花费多少时间、哪些内容要做备注等等。其重要性在于合作者了解自己的作用和目标，这有助于使审查进展的更加顺利。

3. 准备。每个合作者需要了解自己的责任和义务，并积极参与审查。在审查过程中找出问题大部分的缺陷是在准备期间发现的，而不是实际审查期间。

4.编写报告。审查小组必须做出总结审查结果的书面报告，并使报告便于开发小组使用。审查结果必须尽快告诉别人，比如发现多少问题，在哪发现的。

正式审查有三种类型：

1.同事审查：召集小组成员进行初次正式审查是最简单的方法就是同时审查。类似于"各抒己见"类型的讨论。常常仅在编写代码的程序员和充当审查者的其他一两个程序员和测试员之间进行；

2.公开陈述：公开陈述是使同事审查正规化的下一步。编写代码的程序员像 5 人小组或其它类似的程序员或测试员正式表述。审查人员应该在审查之前接到软件拷贝，以便检查并编写备注和问题，在审查过程中提

问。
3.检验：检验是最正式的审查类型，具有高度组织化，要求每一个参与者都接受训练。检验与同事审查不同之处在于，表述代码的人不是原来的程序员。这就迫使他学习和了解要表述的材料，从而有可能在检验会议上提出不同的看法和解释。另外的参与者称为检验员，职责是从不同的角度（如用户、测试员或产品支持人员）的角度审查代码。检验会议后，检验员可能再次碰头讨论他们发现的不足之处，并与会议主席共同准备一份书面报告，明确解决问题所必须重做的工作。然后程序员进行修改，由会议验证修改结果，可能要要进行重新检验，以便找到其余的软件缺陷。

三.编码标准和规范

3 个坚持标准和规范的重要原因：
1.可靠性。事实证明按照按规范编写的代码更可靠，软件缺陷将更少。
2.可读性/维护性。符合标准和规范的代码易于阅读，理解和维护。
3.移植性。如果代码符合设备标准，迁移到另一个平台就会容易，甚至没有任何障碍。

标准由 4 个主要部分组成：
1.标题。描述标准包含的主题。
2.标准（或规范）。描述标准（或规范）内容，解释哪些允许，哪些不允许。
3.解释说明。给出标准背后的原因，让人理解这为什么是好的编程习惯
4.示例。给出如何使用此种标准的简单程序示例，这不是必需的。
　　但是，对软件进行正式审查时，测试和注解的对象仅限于错误和缺漏，而不管是否坚持标准或者规范！

四.通用代码审查清单
1.数据引用错误
数据引用错误是指使用未经正确地初始化的变量、常量、数组、字符串或记录。
A. 是否引用了未初始化的变量？

B. 数组和字符串的下标是整数值吗？下标总是在数组和字符串大小范围之内吗？
C.在检索操作或者应用数组下标时是否包含"丢掉一个"这样的潜在错误？
D.是否在应该使用常量的地方使用了变量？
E.变量是否被赋予不同类型的值？
F.为引用的指针分配内存了吗？
G.一个数据结构是否在多个函数或者子程序中引用，在每一个引用中明确定义结构了吗？
2.数据声明错误
数据声明错误是指不正确地声明或使用变量和常量。
A.所有变量都赋予正确的长度、类型和存储类了吗？
B. 变量是否在声明的同时进行了初始化？是否正确初始化并与其类型一致？
C.变量有相似的名称吗？
D.存在声明过、但从未引用或者只引用过一次的变量吗？
E.在特定模块中所有变量都显式地声明了吗？
3.计算错误
计算错误是指基本的数学逻辑问题。
A.计算中是否使用了不同数据类型的变量，如整数与浮点数相加？
B.计算中是否使用了数据类型相同但字节长度不同的变量？
C.计算时是否了解和考虑到编译器对类型或长度不一致的变量的转换规则？
D.赋值的目的变量是否小于赋值表达式的值？
E.在数值计算过程中是否可能出现溢出？
F.除数或模是否可能为零？
G.对于整型算术运算或某些计算，特别是除法的代码处理是否会丢失精度？
H. 变量的值是否超过有意义的范围？
I. 对于包含多个操作的表达式，求值次序是否混乱，运算优先级对吗？需要加括号使其清晰吗？
4.比较错误
小于、大于、等于、不等于、真、假、比较和判断错误很可能是边界条件问题。
A.比较得正确吗？
B.存在分数或者浮点数之间的比较吗？如果有，精度问题会影响比较吗？1.00000001 和 1.00000002 极其接近，它们相等吗？
C.每一个逻辑表达式都正确地表达了吗？逻辑计算如期进行了吗？求值次序有疑问吗？
D.逻辑表达式的操作数是逻辑值吗？
5.控制流程错误
控制流程错误是指编程语言中循环等控制结构未按预期方式工作，通常由计算或者比较错误直接或间接造成。
A.程序中的语句组是否对应？
B.程序、模块、子程序和循环能否终止？如果不能，可以接受吗？
C.可能存在永远不停的循环吗？
D.循环可能从不执行吗？如果是这样，可能接受吗？
E.对于多分支语句，索引变量能超出可能的分支数目吗？如果超出，该情况能正确处理吗？

F.是否存在"丢掉一个"错误，导致意外进入循环？
6.子程序参数错误
子程序参数错误的来源是软件子程序不正确地传递数据。
A.子程序接收的参数类型和大小与调用代码发送的匹配吗？次序正确吗？
B.如果子程序有多个入口点，引用的参数是否与当前入口点没有关系？
C.常量是否当作形参传递，意外在子程序中改动？
D. 子程序是更改了仅作为输入值的参数？
E.每一个参数的单位是否与相应的形参匹配？
F.如果存在全局变量，在所有引用子程序中是否有相似的定义和属性？
7.输入/输出错误
输入/输出错误包括文件读取、接受键盘或鼠标输入以及向输出设备写入错误等。
A.软件是否严格遵守外设读写数据的专用格式？
C. 软件是否处理外设未连接、不可用、或者读写过程中存储空间占满等情况？
D.软件以预期的方式处理预计的错误吗？
E.检查错误提示信息的准确性、正确性、语法和拼写了吗？
8.其他错误
A.软件是否使用其他外语？是否处理扩展 ASCII 字符？是否需用统一编码取代 ASCII？
B. 软件是否需要移植到其他编译器？
C.是否考虑了兼容性，以使软件能够运行于不同数量的可用内存、不同的内部硬件、不同的外设等？
D.程序编译是否产生"警告"或者"提示"信息？这些信息通常指示语句有疑问。

软件测试读书笔记之七带上 X 光眼镜检查软件

一.动态白盒子测试

用一句话来概括，动态白盒测试是指利用查看代码功能和实现方式得到的信息来确定哪些要测试，哪些不要测试，如何开展测试。动态白盒测试的另一个常用名称是结构化测试，因为软件测试员可以查看并使用代码的内部结构，从而设计和执行测试。

动态白盒测试不仅仅是查看代码，还包括直接参数和控制软件。它包括四部分：
1.直接测试底层功能、过程、子程序和库。即应用程序接口（API）
2.以完整程序的方式从顶层测试软件，但是要根据对软件运行的了解调整测试案例。
3.从软件获得读取变量和状态信息的访问权，以便确定测试与预期结果是否相符，同时，强制软件以正常测试难以实现的方式运行。
4. 估算执行测试时"命中"的代码量和具体代码，然后调整测试，去掉多余的，补充遗漏的。

二、动态白盒子测试和调试

测试和调试是不同的。白盒测试的目标是寻找软件缺陷，调试的目的是修复它们。然而它们在隔离软件缺陷的位置和原因上确实存在交叉现象。测试员应该把问题缩减为能够演示软件缺陷的最简化测试案例。在白盒测试中，甚至要包含那些值得怀疑的代码行信息。进行调试的程序员从这里继续，判断到底是什么导致的软件缺陷，并设法修复。

一定要分清软件测试员和程序员的工作。程序员编写代码，测试员寻找软件缺陷，可能还要编写一些代码来驱动测试，然后程序员修复软件缺陷。要进行这样的底层测试，就要使用与程序员相同的工具。如果程序已经编译过，就要使用同样的编辑器，但是采用不同的设置，以加强错误检测功能。

软件测试员可能会使用代码级的调试器来单步跟踪程序，观察变量，设置断点，等等。对于要求合法性检查的独立代码模块，还有编写测试程序进行测试。

三.分段测试

从测试的角度看，产生高额费用有两个原因：
A.难以甚至不可能找出导致问题的原因
B.某些软件缺陷掩盖了其他软件缺陷。

1.单元和集成测试

独立代码段分别建立和测试，然后集成并重新测试。以最小模块为单位的测试叫单元测试或者模块测试。等到经过单元测试，底层的软件缺陷被找出并修复之后，就集成在一起，对模块组进行集成测试。这个不断增加的测试过程继续进行，加入的软件片段逐渐增多，直至整个产品-至少是产品的主要部分--在称为系统测试的过程中一起测试。

采取这种测试策略很容易隔离软件缺陷。在单元级发现问题时，问题肯定就在那个单元中。如果多个单元集成发现软件缺陷，那么它一定与模块之间的交互有关。当然这个也有例外。

这种递增测试有两种方法：
A.自底向上：要编写测试驱动模块，测试驱动模块以将来真正模块同样的方式挂接，向处于测试的模块发送测试案例数据，接受返回结果，验证结果是否正确。采取这种方式，可以对整个软件进行测试，为它提供全部类型和数量的数据，甚至高层难以发送的数据。
B.自顶向下：有点像小规模的大棒测试，先测试高层的软件，然后测试它们下一层的。

注意：在进行白盒子测试之前，一定要根据说明书建立黑盒子测试案例。用这种方式可以看出真正测试模块的用意。如果先从模块的白盒子角度建立测试案例，

检查代码，就会偏向模块工作方式建立测试案例。程序员或许错误地解释说明，于是测试案例会不对。虽然仔细测试了模块，但是可能不准确，因为没有测试预期的操作。

四.数据覆盖

看了下笔记，发现很乱，取精华，去糟粕，为了不继续误'倒'大家，我把网上流行的经典白盒测试用例设计方法 COPY 过来~

白盒测试的方法：总体上分为静态方法和动态方法两大类。

静态分析是一种不通过执行程序而进行测试的技术。静态分析的关键功能是检查软件的表示和描述是否一致,没有冲突或者没有歧义。

动态分析的主要特点是当软件系统在模拟的或真实的环境中执行之前、之中和之后 , 对软件系统行为的分析。动态分析包含了程序在受控的环境下使用特定的期望结果进行正式的运行。它显示了一个系统在检查状态下是正确还是不正确。在动态分析技术中,最重要的技术是路径和分支测试。下面要介绍的六种覆盖测试方法属于动态分析方法。

1.语句覆盖：语句覆盖是最起码的结构覆盖要求，语句覆盖要求设计足够多的测试用例，使得程序中每条语句至少被执行一次。

2.判定覆盖：判定覆盖又称为分支覆盖，它要求设计足够多的测试用例，使得程序中每个判定至少有一次为真值，有一次为假值，即：程序中的每个分支至少执行一次。每个判断的取真、取假至少执行一次。

3.条件覆盖：条件覆盖要求设计足够多的测试用例，使得判定中的每个条件获得各种可能的结果，即每个条件至少有一次为真值，有一次为假值。

4.判定/条件覆盖：设计足够多的测试用例，使得判定中每个条件的所有可能结果至少出现一次，每个判定本身所有可能结果也至少出现一次。

5.条件组合覆盖：要求设计足够多的测试用例，使得每个判定中条件结果的所有可能组合至少出现一次。

6.路径覆盖：设计足够的测试用例，覆盖程序中所有可能的路径。

（具体例子请看我写的测试试卷一答案整理的大题和白盒子测试方法举例）

总结：白盒测试是一种被广泛使用的逻辑测试方法，是由程序内部逻辑驱动的一种单元测试方法。只有对程序内部十分了解才能进行适度有效的白盒测试。但是贯穿在程序内部的逻辑存在着不确定性和无穷性，尤其对于大规模复杂软件。因此我们不能穷举所有的逻辑路径，即使穷举也未必会带来好运（穷举不能查出程序逻辑规则错误，不能查出数据相关错误，不能查出程序遗漏的路径）。

那么正确使用白盒测试，就要先从代码分析入手，根据不同的代码逻辑规则、语句执行情况，选用适合的覆盖方法。任何一个高效的测试用例，都是针对具体测试场景的。逻辑测试不是片面的测试正确的结果或是测试错误的结果，而是尽可能全面地覆盖每一个逻辑路径。

（原书是数据范围，代码范围，条件范围.....看来作者真不是专业测试人员啊，

书中有些术语翻译的不是很好，由于本人水平有限，没能都改正，建议英语好的人还是看英文版本的，不太好的用此笔记做辅助看，我相信效果会好一点。很抱歉，我还没看英文的呢，打算先把笔记写完再看英文的，先误'倒'一下新手哈哈~ 在次希望大家指正错误，我会及时改的！谢谢~）

软件测试读书笔记之八配置测试

一.配置综述

如果刚准备开始从事软件测试工作，首先的一个任务是配置测试。要保证测试的软件使用尽量多样化的硬件组合。配置测试是指使用各种硬件来测试软件操作的过程。

我们常用有如下配置：个人计算机；部件；外设；接口；可选项和内存；设备驱动程序。

如果准备开始进行软件的配置测试，就要考虑哪些配置与程序的关系最密切。这是必不可少的，因为并不是所有的生产硬件的商家都遵照一套标准来设计硬件。

1.分离配置缺陷

判断缺陷是配置问题还是普通缺陷的方法：在另一台配置完全不同的机器上执行相同

的操作。如果缺陷没产生，那就很可能是配置问题了，如果缺陷在多种配置中产生，应该是普通的缺陷（BUG）

判断缺陷是开发程序的问题还是硬件的问题，要找出问题所在：

（1）软件可能包含在多种配置中都会出现的缺陷。

（2）软件可能包含只在某一个特殊配置中出现的缺陷。

（3）硬件设备或者其设备驱动程序可能包含仅由软件揭示的缺陷。

（4）硬件设备或者其设备驱动程序可能包含一个借助许多其它软件才能看到的缺陷- 尽管它可能对测试的软件特别明显。

前两种情况，由开发小组负责修复缺陷。后两种情况，责任不太清晰。但是即使是硬件的问题，都是开发小组的责任，因为客户不关缺陷是怎么产生的，他们只要求在自己的系统配置中能正常运行。

2. 计算工作量

配置测试工作量可能非常大，我们不可能把会出现的配置都测试。减少麻烦的答案是等价类划分。需要找出一个方法把巨大的配置可能性减少的尽可能控制的范围。由于没有完全测试，因此存在一定的风险，但这正式软件测试的特点！

二.执行任务

确定测试哪些设备和如何测试的决定过程是相当直观的等价类划分工作。什么重要，怎样才会成功，是决定的内容。计划配置测试时采用的一般过程如下：

1.确定所需的硬件类型
2.确定哪些硬件，型号和驱动程序可用
3.确定可能的硬件特性，模式和选项
4.将确定后的硬件配置缩减为可控制的范围
5.明确使用硬件配置的软件唯一特性
6.设计在每一种配置中执行的测试用例
7.在每种配置中执行测试
8.反复测试直到小组对结果满意为止

三．获得硬件

即使把要配置的硬件可能性用等价类划分到最低限度，仍然需要 N 多硬件的，没那么多钱怎么办？
（1）只买可以或者将会经常使用的配置。
（2）与硬件生产商联系，看能否租借甚至白送
（3）问公司内部人有什么硬件，是否允许进行测试。为了完成配置测试，甚至要开车到乡下，但这仍然比买要便宜多了
1.明确硬件标准
大概意思就是了解硬件说明书的一些细节，有助于做出更多清晰的等价划分决定。
2.对其他硬件进行配置测试
根据从设备使用者，项目经理或者销售人员的输入建立硬件的等价区间，写测试用例，收集所选硬件，执行测试。

总结：进行配置测试是软件测试新手经常被分配到的任务，因为它容易定义；是基本组织技巧和等价分配技术的敲门砖；是与其它项目小组成员合作的任务；是管理员快速验证结果的手段。