王轶辰 等编著

软件测试从入门到精通

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

软件测试从入门到精通

王轶辰　等编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

随着软件应用越来越广泛，如何提高软件的质量和可靠性成为软件工作者必须应对的挑战。而软件本身具有“看不见摸不着”的特点，使得对软件的验证和测试与对其他产品的验证和测试大相径庭。本书从软件测试的基本概念讲起，循序渐进地为读者讲解软件生命周期的各个测试阶段应该完成的任务和采用的方法。书中涉及的项目实例多为作者及所在团队参与的课题，具有很强的指导和借鉴意义。希望读者能够从这本书中获取足够的软件测试知识，成为合格的软件测试工作者。

本书适合软件测试的初学者与具有一定测试经验的人员使用。

图书在版编目（CIP）数据

软件测试从入门到精通/王轶辰等编著.一北京：电子工业出版社，2010.7
ISBN 978-7-121-11326-0

Ⅰ.①软… Ⅱ.①王… Ⅲ.①软件—测试 Ⅳ.①TP311.5

中国版本图书馆CIP数据核字（2010）第131757号

责任编辑：李红玉
文字编辑：姜 影
印 刷：北京天竺颖华印刷厂
装 订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编：100036
北京市海淀区翠微东里甲2号 邮编：100036
开 本：787×1092 1/16 印张：18.75 字数：475千字
印 次：2010年7月第1次印刷
定 价：36.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

笔者从事软件测试已近十年，从跟着导师编测试程序的研究生，到完成了关于软件测试研究论文的博士，再到目前成为一个软件测评中心的技术负责人，基本上走过了入门、摸索、提高的路程。在软件测试领域摸爬滚打了这些年以后，总想把自己的一些感触和领悟写出来，希望能够对想要加入这个行业或者已经在这个行业中的人们有些作用。

“从入门到精通”，这个思路很好，因为我们都是这样走过来的（虽然路很长，我们还在途中）。但是这样的主题写起来并不容易，因为走过这个历程的人很少还能够记得自己入门时的情景了。

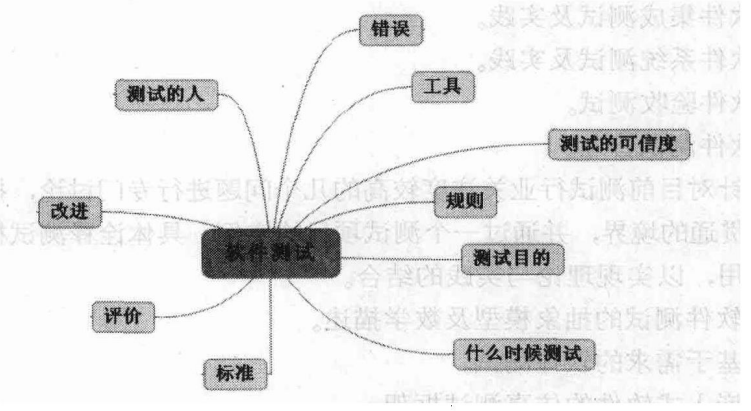
我在试图重新构建从入门到精通的这段路程时，使用了以下的方法：

1. 从我们测评中心的档案库中找到了我为不同阶段做过的软件测试项目所编写的全部测试文档，以现在的眼光去重新审视；
2. 对测评中心内不同职称的测试人员进行访谈，将结果用思维导图的形式表示出来，并进行分析；
3. 参考了一些市面上常用的测试教科书。

经过一番探讨，我整理出测试历程的三个阶段：入门期、提高期、精通期。

入门期

从入门期的思维导图中我们可以看出，一个不懂软件测试的人所想到的内容几乎能够涵盖软件测试的方方面面。



入门期的思维导图

入门时期的测试者一般具有以下特征：

1. 具有空前的求知欲，希望短期内**掌握所有测试知识**；
2. 大脑极具创造力，往往能够提出许多**新奇的想法**，企图一下子发现一块新的大陆。

提高期

经过入门期后，测试者进入一种真正的工作状态，这个时期的测试者一般会有以下

两个特征：

1. 踏踏实实地开始干活，遇到许多实际的问题，希望了解具有**指导作用的问题解决方案**，以便能高效率，高质量地完成工作；
2. 发现测试过程中有许多工作重复而且枯燥，希望**找到合适的工具**，使工作自动化完成。

精通期

在工作中摸爬滚打了一段时间后，忽然发现自己对软件测试有了许多感悟，开始深入思考这个行业，由此进入另一个阶段：

1. 发现软件测试领域有**许多问题还没有得到根本解决**，目前的方法只是权宜之计；
2. 发现软件测试的“下面”原来有许多**通往其他领域的通道**；
3. 发现许多测试问题又变得模糊和陌生起来，于是重新进入**另一个“入门期”**。

基于这样的理解，对全书内容结构做如下安排。

入门篇：了解软件测试的全貌，但适可而止，提出一些问题，有待后续深入探讨，让读者对软件测试有一个全面认识，并做好参与实际工作的准备。

第1章，软件测试的基本概念。

第2章，软件测试的基本方法。

第3章，软件测试的框架表示。

提高篇：了解各类软件测试的执行过程与方法，以及软件测试的自动化工具使用，突出工程实践，使读者能够解决具体问题。

第4章，软件测试过程。

第5章，软件单元测试及实践。

第6章，软件集成测试及实践。

第7章，软件系统测试及实践。

第8章，软件验收测试。

第9章，软件测试管理。

精通篇：针对目前测试行业关注度较高的几个问题进行专门讨论，拓宽测试者的视野，达到融会贯通的境界，并通过一个测试项目的实例，具体诠释测试框架的理论在典型工程中的应用，以实现理论与实践的结合。

第10章，软件测试的抽象模型及数学描述。

第11章，基于需求的软件测试。

第12章，嵌入式软件的仿真测试框架。

第13章，典型工程应用。

另外，全书主要以C/C++test, Insure++以及仿真测试平台三种测试工具作为主要实践工具进行方法和工具的介绍。

下面，让我们一起开始软件测试的旅程吧！

目　录

第1篇　入 门 篇

第2篇 提 高 篇

第3篇 精 通 篇

第1篇 入门篇

第1章 软件测试的基本概念

——掌握软件测试者之间的语言

1.1 软件测试的定义

1.1.1 软件测试定义的发展

关于软件测试的定义及其目标的研究是一个不断发展的过程，下面选取了不同时期具有代表性的几个定义加以分析。

1979年，Myers在软件测试的经典著作《软件测试艺术》（The Art of Software Testing）一书中写道："测试是为发现错误而运行一个程序或者系统的过程。"显然，该定义认为软件测试发生在软件开发周期的末期，其主要目的是发现错误。

到了1983年，软件测试的定义已经发生了改变，测试活动包含了对软件质量进行评估的内容，软件测试不只是一个发现缺陷的过程。Bill Hetzel在《软件测试完全指南》（Complete Guide of Software Testing）一书中指出："测试是以评价一个程序或者系统的属性为目标的任何一种活动。测试是对软件质量的度量。"

同时，1983年IEEE提出的软件工程术语中软件测试的定义是："使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验软件是否满足规定的需求或弄清预期结果与实际结果之间的差别"。

1997年，朱鸿在《软件质量保障与测试》一书中将软件测试视为软件生存期内的一个重要阶段，是软件质量保证的关键步骤，是软件投入运行前，对软件需求分析、设计规格说明和编码进行最终复审的活动。显然，此时的软件测试范围已经不仅限于对程序代码的考察，已经扩展为对整个软件产品的考察，且软件测试的目标已经与软件质量有了密切的关联。

2002年，Rick D.C在《系统的软件测试》（Systematic Software Testing）一书中提供了如下的测试定义："测试是为了度量和提高被测试软件的质量，对测试软件进行工程设计、使用和维护的并发生命周期活动。"在这个定义中，不仅包括度量，还包括了提高软件的质量，这种测试又被称为"预防性测试（preventive testing）"。

测试定义的发展不仅反映了测试目标的发展，实质上也反映了人们对于软件质量内涵认识的发展。软件测试从一开始就与软件质量有着不可分割的联系，Bill Hetzel和Rick D.C的测试定义中都提到了软件质量（前者是度量质量，后者是度量并提高质量），Myers虽然没有明确提出质量的概念，但是发现软件错误并改正错误的过程实质上就是在提高软件质量，只不过当时人们对于软件质量的认识仍然停留在软件运行不出故障的阶段。

那么，经过测试的软件是否就可以达到很高的质量水平，甚至没有缺陷了呢？显然，答案是否定的。关于“软件测试无法充分”的论断随着软件测试的发展一直存在，其困难主要来自于以下三点：

- 软件可能的输入范围太大，无法穷尽测试；
- 软件中可能的运行路径太多，无法全部覆盖；
- 软件测试中使用的规格说明没有固定的客观标准，从不同角度看，软件缺陷的标准不同。

另外，不完全的测试更加无法证明一个软件的正确性。测试中发现一个缺陷就能说明软件不完全正确，即使测试中没有发现缺陷，也会因为测试的不充分而不能说明软件的正确性。所以，软件测试是保证软件质量的一个重要手段，但它的能力又是有限的，经过测试的软件也不是完全可靠的。

软件测试的定义随着软件测试的目标而不断变化，但有一点可以确定，软件测试与软件的质量不可分割，而且始终与发现软件缺陷的过程密切相关。

1.1.2 对软件测试的正确认识

软件测试既是一项技术性工作，同时也涉及经济学和心理学的一些重要因素，甚至涉及管理学的若干内容。软件测试是一门实践性极强的综合性学科。

在理想情况下，我们会测试程序的所有可能执行情况。然而，在大多数情况下，这几乎是不可能的。即使一个看起来非常简单的程序，其可能的输入与输出组合可达到数百种甚至上千种，对所有的可能情况都设计测试用例是不切实际的。对一个复杂的应用程序进行完全的测试，将耗费大量的时间和人力资源，这在经济上是不可行的。

软件测试作为一项由人来完成的工程活动，其中必然包含了许多人为因素，其中心理学作用至关重要。下面我们重点谈论一些比较著名的关于测试的观点。

1. 没有不存在缺陷的软件

这是软件测试活动存在的理由，也是软件测试人员应该恪守的最重要的信念。这一点对于程序员来说有些残酷，即使是一个功能十分简单的程序，也可能会有缺陷，至少有许多可以改进和完善的地方。

小故事：讨厌的二分查找

事实上，“是否存在没有缺陷的软件”一直是程序员们关注的一个话题。在此我们借用Andy Oram和Greg Wilson所著的《Beautiful Code》中的一段二分查找程序进行讨论。

二分查找是一个简单而又高效的算法，这个算法用来确定一个预先排好顺序的数组x[0..n-1]中是否含有某个目标元素t。如果数组包含t，程序返回它在数组中的位置，否则返回-1。算法的描述如下：

在一个包含t的数组内，二分查找通过对范围的跟踪来解决问题。开始时，范围就是整个数组。通过将范围中间的元素与t比较并丢弃一半范围，范围就被缩小。这个过程一直持续，直到t被发现，或者那个可能包含t的范围已成为空。

大多数程序员认为，有了上面的描述，写出代码是很简单的事。他们错了，能使你相信这一点的唯一方法是现在就合上书，去亲手写写代码试试看。

二分查找是一个非常好的例子，因为它是如此简单，却又如此容易被写错。在《Programming pearls》一书中，Jon Bentley记述了他是怎样在多年时间里先后让上百个专业程序员实现二分查找的，每次他都先给出一个算法的基本描述。他很慷慨，每次给他们两个小时的时间来实现它，而且允许他们使用他们自己选择的高级语言（包括伪代码）。令人惊讶的是，大约只有10%的专业程序员正确地实现了二分查找。更让人惊讶的是，Donald Knuth在他的《Sorting and Searching》（见《计算机程序设计艺术，第3卷：排序和查找》（第二版）一书中指出，虽然第一个二分查找算法早在1946年就被发表，但第一个没有bug的二分查找算法却是在12年后才被发表。

然而，最让人惊讶的是，Jon Bentley正式发表的并被证明过的算法，也就是被实现或改编过成千上万次的那个，最终还是有问题的，问题发生在数组足够大，而实现算法的语言采用固定精度算术运算时。在Java语言中，这个bug导致一个 ArrayIndexOutOfBoundsException异常被抛出，而在C语言中，会得到一个无法预测的越界的数组下标。以下就是带有这个著名bug的Java实现。

```
public static int buggyBinarySearch(int[]a,int target){
int low=0;
int high=a.length-1;
while (low<=high){
int mid =(low+high)/2;
int midVal=a[mid];
 if(midVal<target)
low=mid+1;
else if (midVal>target)
high=mid-1;
else
return mid;
}
return-1;
}
```

bug位于这一行：

```
int mid =(low+high)/2;
```

如果low和high的和大于Integer.MAX_VALUE（在Java中是2^{31}-1），计算就会发生溢出，使它成为一个负数，被2除时结果当然仍是负数。

推荐的解决方案是修改计算中间值的方法来防止整数溢出。方法之一是用减法——而不是加法——来实现：

```
int mid =low+((high -low)/2);
```

对于那个bug，建议使用无符号位移运算，这种方法或许更快，但对大多数Java程序员来说，可能也比较晦涩。

```
int mid =(low+high) >>>1;
```

想一下，二分查找算法的思想是多么简单，而这么多年又有多少智力花在它上面，这就充分说明了即使是最简单的代码也需要测试。这充分说明了若想编写出一段完全没有缺陷的代码几乎是不可能的。

从理论上讲，对中间值的计算方法的修正，应该是解决这段令人讨厌的代码的最后一个bug，一个在好几十年的时间里，连最好的程序员都抓不到的bug。

```
public static int binarySearch(int[]a,int target){
int low=0;
int high=a.length-1;
while (low<=high){
int mid =(low+high)>>>1;
int midVal=a[mid];
 if(midVal<target)
low=mid+1;
else if (midVal>target)
high=mid-1;
else
return mid;
}
return-1;
}
```

这个binarySearch看上去是正确的，但它仍可能有问题。或许不是bug，但至少是可以而且应该被修改的地方。这些修改不但可以使代码更加健壮，而且可读性、可维护性和可测试性都比原来更好。

对于这个漂亮的二分查找，作为一个专门从事单元测试的team leader，看到之后的第一个反应就是用工具来充分测试这段代码，看看它是否如传说中那么完美。我们采用的工具是Parasoft公司出品的Jtest。这个业界口碑最好的Java语言单元测试工具对这段代码测试后得出的测试报告非常令人吃惊，这是一个没有任何bug的代码。

因此，我认为：尽管还可以更好，该代码已经可以用“完美”来形容了。

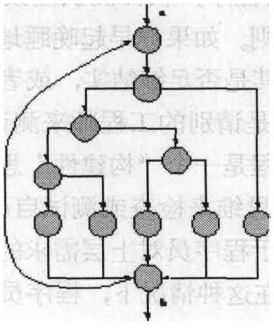
---

2. 不存在完全充分的软件测试，测试需要终止

无论是从测试方法上还是从测试实践中的测试资源考虑，充分的测试都是不可能的任务。

实例：没有完全充分的测试

程序中不同逻辑路径的数量可能达到天文数字。下图中的小程序显示了这一点。该图是一个控制流图，每一个结点或圆圈都代表一个按顺序执行的语句段，通常以一个分支语句结束。每一条边或弧线表示语句段之间的控制（分支）的转换。下图描述的是一个有着0～20行语句的程序，包含一个迭代20次的DO循环。在DO循环体中，包含一系列嵌套的IF语句。要确定不同逻辑路径的数量，也相当于要确定从点a到点b之间所有不同路径的数量（假定程序中所有的判断语句都是相互独立的）。这个数量大约是10^{14}即100万亿，是从$5^{20}+5^{19}+...+5^{1}$计算而来，5是循环体内的路径数量。大多数人难以对这个数字有一个直观的概念，不妨设想一下：如果在每五分钟内可以编写、执行和确认一个测试用例，那么大约需要10亿年才能测试完所有的路径。假如可以快上300倍，每一秒就能完成一次测试，也得用漫长的320万年才能完成这项工作。



小经验

在实际测试项目中，我们见到过路径覆盖只能做到1%的案例。并不是测试用例设计得太少，不能覆盖更多的软件执行路径，而是代码中存在的迭代和循环使路径数目达到了天文数字。这样的测试结果是需要分析和说明的。有时用户希望使用覆盖率作为衡量测试充分性的指标，这本无可厚非，但是仅使用覆盖率还是非常有局限性的。

3. 软件测试的过程具有一定的破坏性

不要只为了证明程序能够正确运行而测试程序，相反，应该一开始就假设程序中存在错误（事实也确实如此），然后测试程序，发现尽可能多的错误。

虽然Myers对测试的定义：“测试是为发现错误而运行一个程序或者系统的过程。”随着时间的推移已经被不断完善了，但是它所体现出来的“测试具有破坏性”的基本思想还是正确的。

4. 测试用例中的一个必需部分是对预期输出或结果进行定义

这是软件测试活动最基本也是最容易被忽视的一条准则，如果在测试之前没有对预期输出或结果进行严格的定义，那么就会出现“测什么，就是什么”的情况。换句话说，尽管“软件测试是一个破坏性的思维过程”是合理的，人们在潜意思中仍然渴望看到正确的结果，而克服这种倾向的一种方法就是事前精确定义软件的预期输出，严格检查软件的输出结果，因此，一个测试用例必须包括两个主要部分：

（1）对软件的输入数据的详细描述；

（2）对软件在上述输入数据下的正确输出结果的精确描述。

小经验

预期输出是程序编写人员对自己程序的期待值。有些时候这些预期输出可以根据经验获得，比如两个数的乘积；有些时候预期输出必须来自需求文档，也就是说程序必须实现预定的功能；还有些时候，我们不能十分确定预期输出是什么，因为我们编写程序的目的就是为了得到确切的，比手工计算更加精确的结果，比如矩阵运算。

5. 程序员应该避免测试自己的程序，软件测试应该保持一定的独立性

这是软件测试中的一条重要准则。如果你早起晚睡地盖好了一座漂亮的小木屋，你能够亲手拿着大铁锤去砸木屋的门以测试其是否足够结实，或者爬到屋顶上往房子上倒水以测试其是否漏水吗？如果你下不去手，那还是请别的工程师来测试你的代码吧。

程序员完成设计或者代码的过程是一个“构建性”思维过程，在完成设计或编写代码的工作之后，他很难以一种“破坏性”思维来检查或测试自己的程序。

另外，软件中的大量缺陷是由于程序员对上层需求的理解有误，或者干脆对于某些程序所涉及的知识根本不知道所导致的。在这种情况下，程序员自己测试自己的程序基本上是徒劳无功的。

6. 应该彻底检查每个测试的执行结果

这个看似显而易见的原则，在多数的软件测试实践中却经常被忽视。出于下面的两个主要原因，这条原则在执行过程中存在一定的困难：

（1）软件的执行结果形式多样，可能是界面的显示或者数据文件，也可能是软件的执行时间（例如性能测试），不同形式的结果对判断测试是否通过具有不同程度的影响；

（2）软件的预期结果描述并不精确，导致软件的实际结果与预期结果存在细微的差别，很难界定是否属于软件缺陷范畴。

全面、认真地检查每个测试的执行结果是发现软件缺陷的最后环节，否则前期周密的测试设计工作就付之东流了。

小实例

在使用测试工具进行测试时，由于测试用例是批量运行的，测试结束后的第一个工作就是验证测试结果。工具的默认验证是不通过，也就是说，测试工程师需要手工确认每一个测试结果，验证其是否应该是通过的。我们曾经与测试工具厂商的研发工程师交流过，他们迫于压力，已经在工具中增添了一个功能项，叫“批量验证”，也就是说可以一次性地将所有的“不通过”更改为“通过”。这是进步还是偷懒呢？

7. 测试用例的编写不仅应该根据有效和预料到的输入情况，还应该根据无效和未预料到的输入情况（这个就必须依靠测试工程师的经验了。如果测试人员有开发经验，就能够根据曾经遇到的问题，设计出能够发现深层缺陷的输入数据）。

软件测试过程中往往有这样一种倾向：测试人员习惯于按照软件文档中规定的软件功能或性能要求对软件进行测试，因此常常会出现将重点集中在有效和预期的输入情况上，而忽略了无效和未预料的情况。

8. 检查程序是否“没有做它应该做的”仅仅是测试的一半，测试的另一半是检查程序是否“做了它不应该做的”。

这条原则是在上一条原则的基础上衍生的可执行操作。这个原则看上去有点“不知好歹”。难道程序实现了更多的功能不应该得到鼓励吗？要记住，程序只能做被规定的那些工作，任何多余的功能都应视为安全隐患。全面的测试应该包括两个目标：

（1）验证程序是不是做了它应该做的事情，确保软件的可靠性；

（2）验证程序是不是没有做它不应该做的事情，确保软件的安全性。

9. 应该避免测试用例用后即弃，除非软件本身就是一个一次性的软件

这条原则在进行交互式软件测试的过程中尤其适用。人们通常会坐在电脑前，匆忙地编写测试用例，然后将这些用例交由程序执行。这样做会产生这样的问题，精心设计的用例在测试结束后就消失了。一旦软件需要重新测试，又必须重新设计这些用例。由于重新设计用例需要投入大量的工作，人们又总是避免这种重复工作，因此程序的重新测试就不会同上次一样严格，这意味着，如果对程序的更改导致了程序某个先前可以执行的部分发生了故障，这个故障往往是不会被发现的。保留这些测试用例，当程序对某些部分进行了修改之后需要重新测试。

测试用例的管理与复用是回归测试中的关键所在。测试工具的支持此时显得尤为重要。

10. 程序某部分存在更多错误的可能性，与该部分已发现错误的数量成正比

这就是软件测试中的“冰山原理”：两座冰山，露出水面部分较大的那一座，其水下部分必然也较大。该原则的另一个说法是，错误总是倾向于群聚存在，而在一个具体的程序中，某些部分比其他部分更容易存在错误。这条原则与“二八原理”有着密切联系。

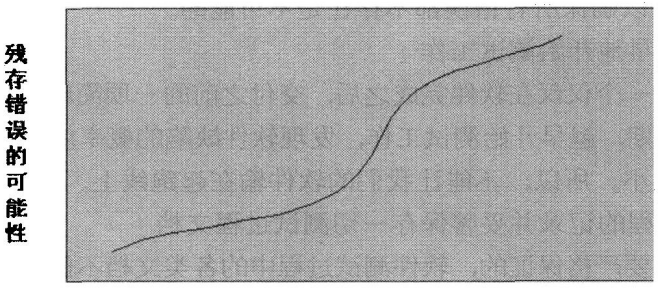


图1-1 软件错误的冰山原理

这条原则的具体指导体现在软件测试中的探针测试法。对软件进行全面的探针测试后，再集中优势对已经发现错误的部分进行额外的重点测试。

小知识：软件工程领域的“二八原理”

Pareto是意大利的经济学家，他在分析社会财富分配状况时得出了“20%的富人，掌握了80%的社会财富，其他80%的人只占有20%的社会财富”。这一原理随后被广泛地应用于其他领域，如系统工程中的“二八原理”，硬件研制中对硬件系统关键件的分析等。

“二八原理”在软件开发中也被证明是正确的，Barry Boehm给出了软件质量和软件现象所遵循的“二八原理”：

(七)20%的软件模块包含了软件中80%的缺陷；

(八)20%的软件缺陷，消耗了80%的更改维护费用；

(九)20%的软件改进，花费80%的适应性维护费用；

(十)20%的模块占用了80%的执行时间；

Ⅰ20%的模块消耗了80%的资源（人力、经费等）；

Ⅱ20%的软件工具占用了80%的全部工具使用时间。

本书作者再补充二条重要的软件质量的Pareto原理：

(七)20%的软件缺陷，造成了80%的软件故障；

(八)20%的软件开发和管理人员（骨干），决定了80%的软件开发质量。

根据Pareto原理，质量管理之父Juran提出了质量管理中存在着“少数的重要事情（Vital few）”和“多数的琐碎事情（trivial many）”的观念。这就给软件质量管理指明了方向：在有限的资源条件（有限的人员、时间、经费）下，抓软件质量要抓“少数的重要事情”，要抓20%的关键模块，对这些关键模块的开发必须从文档、评审、配置管理、测试等方面严加控制，以有效地提高软件质量。

11. 软件测试是一项极富创造性、极富智力挑战性的工作

人的天性中肯定是有破坏欲望的。软件测试就是一个合理合法地“破坏”一件事物的良机。发现一个隐藏很深的错误，避免软件因此错误而死机或出现严重失效，其带来的满足感和成就感是难以想象的。测试一个大型软件所需要的创造性可能超过了开发该软件所需要的创造性。要充分测试一个软件以确保所有错误都不存在是不可能的。

12. 应该尽可能早地开始测试工作

软件测试绝不是一个仅仅在软件完成之后，交付之前的一项简单活动。软件测试活动要贯穿于整个软件生存周期，越早开始测试工作，发现软件缺陷的概率就越大，修改缺陷造成的质量和成本影响就会愈小。所以，不能让我们的软件输在起跑线上。

13. 重视测试过程的记录并妥善保存一切测试过程文档

测试质量也是需要严格保证的，软件测试过程中的各类文档不仅涉及软件测试质量，同时也涉及软件质量本身。将测试过程中的测试记录妥善保管是评价测试质量和进行回归测试的重要依据。

另外，让所有的测试过程和结果都有据可查，也是工作态度的体现。一个合格的测试工程师，必须是一个有责任心，工作有条理的技术人员。在项目组中配备专门的文档工程师，是个省时省力，事半功倍的好办法。

14. 测试可以证明软件缺陷的存在，但不能证明缺陷不存在

测试可以证明软件产品是失败的，也就是说软件中存在缺陷，但测试不能证明软件中没有缺陷。适当的测试可以减少测试对象中的隐藏缺陷。即使在测试中没有发现失效，也不能证明软件没有缺陷。

这条规则再次提醒我们，不是说经过测试的软件就是万无一失的。虽然看上去有点让人失望，但是测试是让软件通向完美的最佳路径，而且测试越多，我们对软件的信心就会越强。

15. 杀虫剂悖论

如果同样的测试用例被重复执行多次，会减少其有效性。先前没有发现的缺陷也不会被发现。因此，为了维持测试的有效性，应当对测试用例进行不断的修改和更新。这样软件中未被测试过的部分或者先前没有被使用过的输入组合会重新被执行，从而发现更多的缺陷。测试用例也要与时俱进方能百战不殆。

16. 测试依赖于测试内容

测试必须与应用程序的运行环境及使用中固定的风险相适应。因此，没有两个系统可以以完全相同的方式进行测试。对于每个软件系统，测试出口准则等应该根据它们使用的环境分别量体定制。飞行器上面的安全关键软件与电子商务系统软件的测试是明显不同的。一把钥匙开