## 软件测试概述

### 软件的概念

计算机软件：程序与文档（解释和指导）。

计算机程序：源程序-----用高级语言或汇编语言编写的程序。

目标程序-----源程序经编译或解释加工后可以由计算机直接运行的程序。

### 软件的特点

计算机软件既是作品，又是工具，是作品性和工具性紧密结合的治理成果。

计算机软件开发工作量大，成本高，但复制容易，成本低。

计算机软件具有无形性，可以多次使用，但商业寿命低。

### 软件危机

软件危机：是计算机软件在开发和维护过程中所遇到的一系列严重问题。

主要是两方面的问题：

1. 如何开发，怎样满足软件日益增长的需求。
2. 如何维护数量不断膨胀的已有软件

在大型软件的开发过程中出现了复杂度高、研制周期长，正确性难以保证的三大难题。 遇到问题找不到解决方法，致使问题堆积起来，形成人们难以控制的局面，出现了所谓 的“软件危机”。

软件危机的表现：

1. 对软件的开发和进度的估计不正确
2. 用户对“已完成的”软件系统不满意的现象经常发生
3. 软件产品的质量常常靠不住
4. 软件常常是不可维护的
5. 软件通常没有适当的文档资料
6. 软件成本在计算机系统总成本中所占比列逐年升高

软件危机的原因：

1. 软件生产本身存在复杂性
2. 软件开发所用的技术和方法

软件工程正是为克服软件危机而提出的一种概念，并在实践中不断探索它的原理、技术 和方法。

### 软件测试

软件测试的产生：

1. 软件的规模越来越大
2. 软件开发与用户之间的矛盾

软件测试的定义：

1、1979年，《软件测试艺术》：为了发现错误而执行程序或者系统的过程。

2、1983年，IEEE软件工程标准术语：使用人工或自动手段，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弄清预期结果和实际结果之间的差别。

软件测试的测试对象：

软件测试不等同于程序测试，软件测试贯穿于软件定义和开发的整个期间。需求分析，概要设计，详细设计以及程序编码等各个阶段所得到的文档，包括需求规格说明书、概要设计规格说明，详细设计规格说明以及源程序，都是软件测试的对象。（涵盖方方面）

1.软件需求（ps:理解软件需求，对于用户[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库)人员TE比较重要）

2.软件概要设计

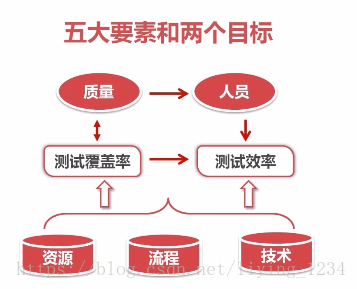
3.软件源代码

4.软件详细设计

5.可运行程序

6.软件运行环境（ps:测试提交Bug时，应该注明当前环境。例如window7+64 IE10 1920\*1080之类的（兼容性问题））

五大要素和两个目标：



软件测试所遵循的原则：

一、测试显示缺陷的存在，但不能证明系统不存在缺陷。

二、穷尽测试是不可能的，应设定及时终止的条件。

三、软件测试应尽早进行。

四、缺陷具备群集特性

五、测试的杀虫剂悖论

       我们在测试当中如果采用同样的测试用例，同样的测试方法，多次同样的测试同一模块，最后我们就不能发现新的缺陷，所以，测试用例和测试方法应不定期的更新，来测试不同部分，从而发现更多的缺陷。

六、测试的二八原则

       将80%的时间和资源用在20%的重要模块上。

七、测试活动依赖于测试背景

       针对不同的测试背景测试活动是不同的。

**软件测试的分类：**

按照测试阶段：

单元测试：对软件中最小的可测试单元进行检查和验证

原则：尽可能保证各个测试用例是相互独立的；

          一般单元测试是有开发人员来实施，用来检验开发的代码是否符合设计要 求；

益处：能尽早的发现缺陷；

           有利于重构；

           简化集成；

           文档（代码即文档）

           用于设计（体现测试思路）

限制：不可能覆盖所有的执行路径，所以不可能保证捕捉到所有路径的错误

       每一行的代码，一般需要3~5行的测试代码才能完成单元测试

框架：Xunit

集成测试：在单元测试的基础上，测试在将所有的软件单元按照概要设计规格说明的要求组装成模块、子系统或系统的过程中各部分工作是否达到或者实现相应技术指标以及要求的活动；

集成测试的主要实施方案：

Bigbang   组装所有单元，再做测试；

自顶向下   主程序按层级向下；

自底向上   最底层模块，逐层组装测试；

核心系统集成

高频集成（持续集成）

集成测试与单元测试的区别：

对象不同、测试依据不同、测试方法

系统测试：是将经过集成测试的软件，作为计算机系统的一个部分，与系统中其他部分结合起来，在实际运行环境下对计算机系统进行的一系列严格有效的测试，以发现软件潜在的问题，保证系统的正常运行；

关注点：

关注系统本身的使用；

关注系统与其他相关系统的联通；

关注系统在不同使用压力下的表现；

关注系统在真是使用环境下的表现；

系统测试和集成测试的不同：

对象不同、测试时间、测试内容、测试角度（系统偏向业务方面）

验收测试：（交付测试）针对用户需求、业务流程的正式的测试，确定系统是否满足验收标准，由用户、客户或者其他授权机构决定是否接收系统；

细分：用户验收测试、运行验收测试、合同规范验收测试

  Alpha 开发提供环境，用户测试

  Beta  在用户环境测试

**按照测试手段来分类**：黑盒测试、白盒测试、静态测试、动态测试、手工测试、自动化测试

黑盒测试：对内部不可见

优点：1、容易实施，不需要关注内部的实现。

          2、更贴近用户的使用角度。

缺点：1、测试覆盖率较低，一般只能覆盖到代码量的不到40%。

          2、针对黑盒测试的自动化测试，复用率较低，维护成本较高。

黑盒测试主要测试什么？

1、是否有不正确或遗漏的功能？

2、在接口上，输入是否能正确的接受？能否输出正确的结果？

3、是否有数据结构错误或外部信息（例如数据文件）访问错误？

4、性能上是否能够满足要求？

一般系统测试使用较多。

黑盒测试的主要设计方法

等价类划分法：针对程序有很多的输入条件，等价类划分就是把所有的输入当中等价的归成一类，这样最后形成若干典型的代表性的输入，通过这些典型的数据来进行测试用例的设计。

边界值分析法：是一种特殊的等价类划分，关注的是各种各样的边界条件。

错误推测法：是基于经验、或者直觉来判断程序中可能出现错误的地方，从而针对性的设计用例的方法。

因果图法：指拿到程序的需求规格说明书，针对每种输入和输出在因果图里，会把输入和输出看做是原因和结果，对输入和输出赋以特殊的标识符，然后将这些情况形成一个叫因果图，最终根据规格的语义说明形成一个判定表，根据判定表编写测试用例。

正交试验分析法：是通过正交性，从一组数据当中筛选出典型的代表性的数据的设计方法，主要用于筛选输入数据，然后再来设计测试用例的输入输出。

状态迁移图法：通过梳理软件功能点里面的状态迁移关系来设计测试用例。

流程分析法：通过梳理程序的逻辑执行的路径来设计测试用例。

白盒测试：机构化测试或透明化测试，内部结构对测试人员是透明的。

主要的逻辑单位

优点：1、迫使测试人员去仔细思考软件的实现，理解原理。

          2、可以检测代码中的每条分支和路径。

          3、揭示隐藏在代码中的错误。

          4、对代码的测试比较彻底。

缺点：1、昂贵。（较高覆盖，工作量大）

          2、无法检测代码中遗漏的路径和数据敏感性错误。

          3、不能直接验证需求的正确性。

白盒测试的主要测试方法：

代码检测法：主要包括多面检查、代码审查和走查，主要检查代码和设计的一致性。

静态结构分析法：测试者通过采用测试工具来分析源代码的系统结构、数据结构、内部的控制逻辑，通过内部结构的分析来设计测试用例。

静态质量度量法：根据标准的质量模型作为基准，然后来构造质量的度量模型，用来评估软件的各个方面的要素。

逻辑覆盖法：语句覆盖、条件覆盖、条件组合覆盖、分支覆盖||判定覆盖、路径覆盖、条件&判定组合覆盖

基本路径测试法：在程序控制流图的基础上，通过分析控制构造的圈复杂度，导出基本可执行的路径的集合，进而设计测试用例的方法。

灰盒测试：介于黑、白测试之间的，关注输出对于输入的正确性，同时也关注内部表现

更多的在系统组件来评价软件设计符合要求的情况

静态测试：是指无须执行被测程序，而是通过评审软件文档或代码，度量程序静态复杂度，检查软件是否符合编程标准，借以发现编写的程序不足之处，减少错误出现的概率。

动态测试：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、正确性和健壮性等。

手工测试：由专门的测试人员从用户视角来验证软件是否满足设计要求的行为。更适用针对深度的测试和强调主观判断的测试。

众包测试、探索式测试

自动化测试：使用单独的测试工具软件控制测试的自动化执行以及对预期和结果进行自动检查。

   单元测试、接口测试、性能测试

手工测试 VS 自动化测试

手工测试

·易发现缺陷

·容易实施

·创造性、灵活性

·覆盖量化难

·重复测试效率低

·不一致性、可靠性低

·人力资源依赖

自动化测试

·高效率、速度快

·高复用性

·覆盖率容易度量

·准确、可靠

·不知疲劳

·机械、发现缺陷率低

·一次性投入较大

按测试手段来分类：

黑盒测试：把测试的对象看成一个黑盒，不了解内部的逻辑和结构，从用户的角度来对软件进行测试

白盒测试：测试人员是完全的了解内部的结构和设计逻辑，通过逻辑的覆盖来保证测试的完健性

静态测试：不运行被测软件，通过静态的检查代码文档来进行测试

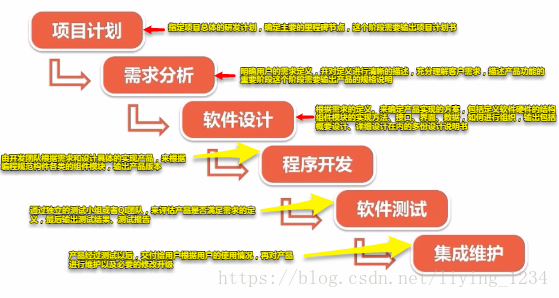
动态测试：运行被测软件，通过软件运行的表现来判断软件运行的功能是否正常

手工测试：由专门的测试人员根据测试用例来实施的测试

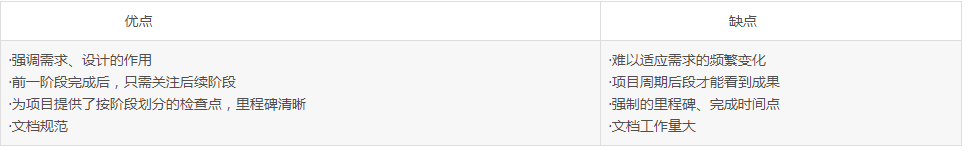
自动化测试：指借助第三方测试工具来自动化的运行检查的测试

**按测试模式来分类：**瀑布模型、敏捷测试、基于脚本的测试、基于风险的测试、探索式测试等

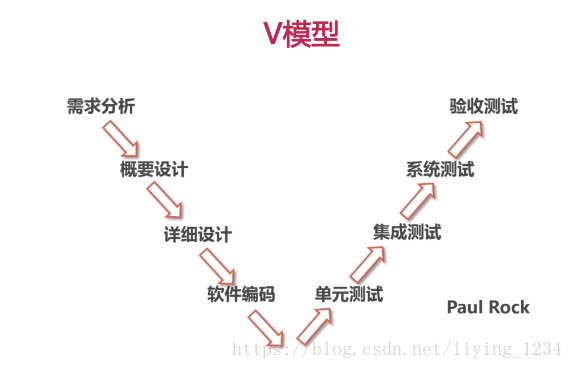
传统的瀑布模型：



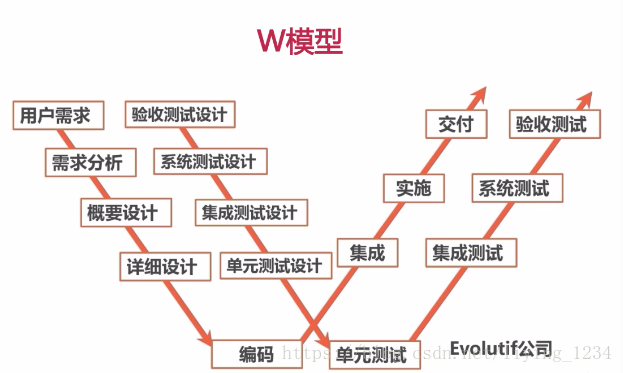
以上一个阶段的输出作为下一阶段的输入



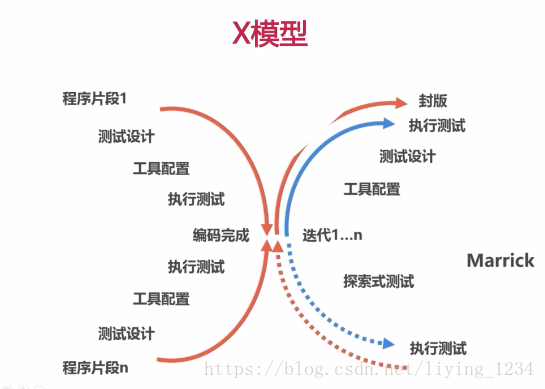
V模型：后期发现错误



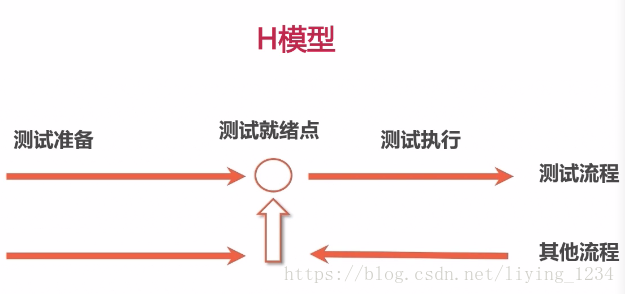
W模型：尽早的发现错误

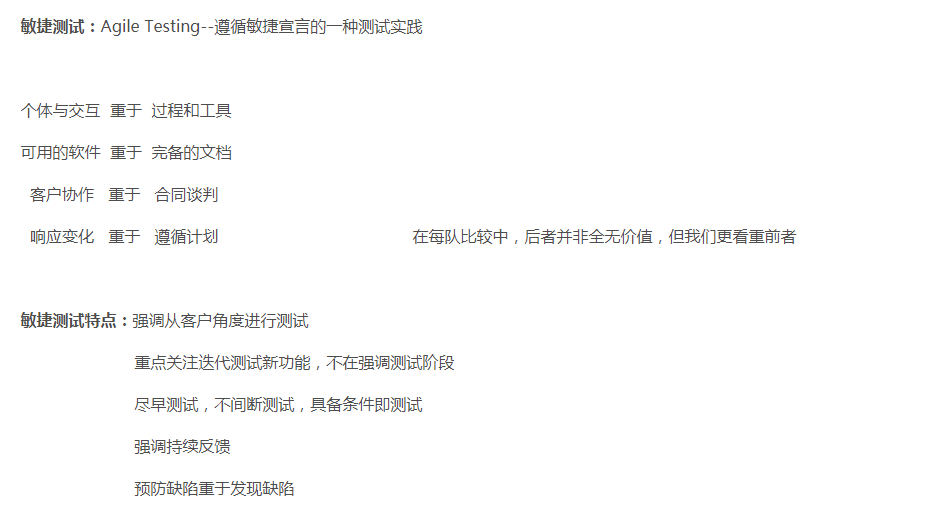


X模型



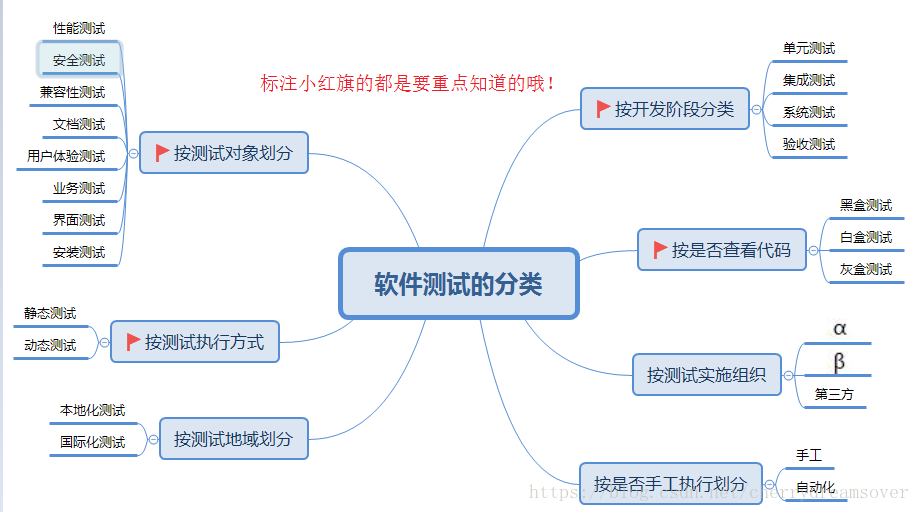
H模型







* ****α测试是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。****
* ****β测试是软件的多个用户在一个或多个用户的实际使用环境下进行的测试。开发者通常不在测试现场，Beta测试不能由程序员或测试员完成。****



# **[软件测试的生命周期及测试流程](https://www.cnblogs.com/xiaomo123/p/7568459.html)**

**一、软件的生命周期**

定义：软件从产生到报废的生命周期。

生命周期包括：问题的定义及规划（开发方与需求方讨论）、需求分析、软件设计、软件编码、软件测试（单元测试、集成测试、系统测试、验收测试）、运营维护阶段。（行业性概念）

**什么是软件测试生命周期（STLC）**

它是指测试流程，这个流程是按照一定顺序执行的一系列特定的步骤，去保证产品质量符合需求。在STLC流程中，每个活动都按照计划的系统的执行。每个阶段有不同的目标和交付产物。在STLC中每个组织有不同的阶段；但是基本的是一样的。

下面是STLC的8个阶段：

1.需求阶段（Requirements phase）

2.计划阶段（Planning Phase）

3.分析阶段（Analysis phase）

4.设计阶段（Design Phase）

5.实施阶段（Implementation Phase）

6.执行阶段（Execution Phase）

7.总结阶段（Conclusion Phase）

8.结束阶段（Closure Phase）

#1.需求阶段：

在这个阶段，是分析和学习需求的阶段。和其他团队一起头脑风暴，努力去查找需求是不是可测的。这个阶段帮助去辨认测试的范围。如果任何功能是不可测试的，及时沟通，做出一些减轻策略（减小风险）的计划。

#2.计划阶段：

在实际场景中，测试计划是测试流程的第一步。在这个阶段我们辨别出哪些活动和资源能匹配测试目标。我们也努力去辨别测试指标，测试方法以及如何追踪这些指标。

计划的基础是什么？只有需求吗？

答案是NO。需求只是一种基础，然而还有其他2方面的因素影响测试计划。他们是：

-组织的测试策略

-风险分析/风险管理和减轻

#3.分析阶段：

STLC定义“WHAT”被测试。一般地，我们通过需求文档、产品风险和其他测试依据去辨别测试条件。测试条件应该可以被追溯到需求。有很多因素可以影响测试条件的辨别：

-测试的层次和深度

-产品的复杂性

-产品和项目风险

-软件开发生命周期被涉及

-测试管理

-团队的知识和技能

-相关干系人的可用性

我们应该努力用一种很详细的方式把测试条件写下来。例如，一个电子商务网站，你有一个测试条件为“用户应该可以支付”。或者你可以详细的描述为“用户应该可以通过信用卡，微信、支付宝等支付”。把详细的测试条件写下来最大的好处是可以提高测试覆盖率，因为测试用例就是通过这些测试条件写的，这些详情触发写出更多的测试用例。同时，也可以分辨出退出测试的标准，比如哪些条件决定可以终止测试。

#4.设计阶段：

这个阶段都有“HOW”去测试。包括以下的任务：

-详述测试条件。拆分测试条件为多个子条件去提供覆盖率。

-辨别和获取测试数据

-辨别和搭建测试环境

-创建需求跟踪指标

-创建测试覆盖指标

#5.实施阶段：

这个阶段最主要的任务是创建详细的测试用例。测试用例的优先级以及哪些用例会成为回归测试的一部分。在最终决定测试用例之前，审核测试用例的正确性是非常重要的。同时在真正执行开始之前也不要忘了sign off(署名，比如新版本发布前的最终测试报告需要发送sign off报告)测试用例。如果你的项目设计自动化，分辨哪些用例适合自动化以及准备测试脚本。不要忘记审核。

#6. 执行阶段：

从名字可以知道，这个阶段是STLC的真正执行阶段。但在你执行之前，确保你的标准是和需求匹配的。执行测试用例，报bug如果有任何的不匹配。同时填写追踪指标去跟踪你的进度。

#7.总结阶段：

这个阶段聚焦在检验标准和报告。依赖你的项目和干系人选择，你能决定是发日报还是周报等等。有不同的报告类型（日报，周报）你可以发送，但重点是，报告的内容是根据你发送对象的不同而变化的。如果项目经理属于测试背景的，那么他们对技术方面更感兴趣，因此在报告中包含技术方面的内容（用例的pass个数，fail个数，bug个数，严重bug等等）。但是如果你向更高层的干系人报告，他们可能对技术方面不感兴趣，可以给他们发送一些风险相关的，比如通过测试减轻风险的发生。

#8.结束阶段：

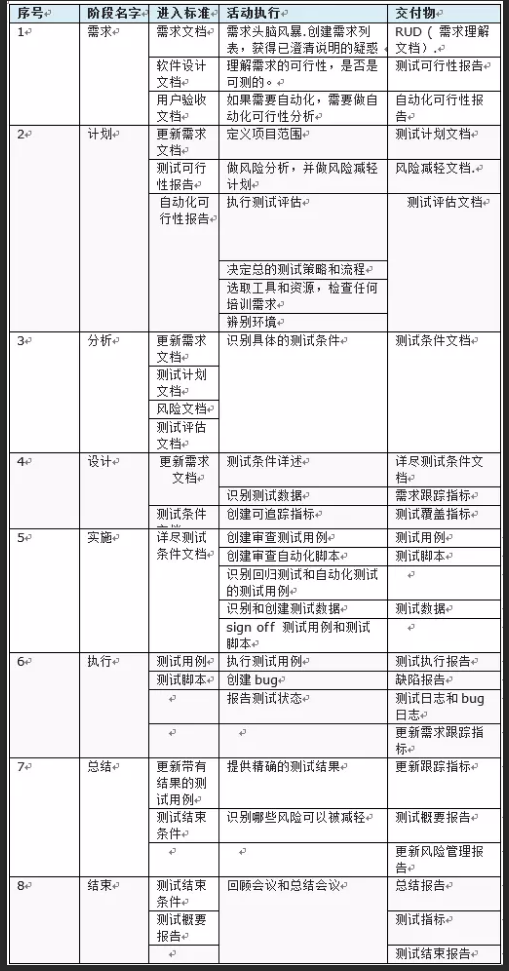
这个阶段的任务包括：

-检查测试的完成度。是否所有的用例被执行或是有意减轻一些。检查是否还有S1的bug是激活状态。

-经验总结会议及书写相关文档。包括哪些做得好，哪些需要提高和如何提高

**总结：**

一起来总结一下STLC。



**二、软件测试的基本流程**

开发流程：需求分析--得知功能组成和具体逻辑--编写代码--单元测试--打包提交测试--测试提交bug--修复bug--测试再进行测试--...N轮--版本上线--面向用户使用

测试流程：需求分析+原型图--编写测试用例--评审测试用例--走开发流程，等待开发研发完成，提交测试（酱油期）--测试提交bug-修复bug--测试再进行测试--...N轮--版本上线--面向用户

1）测试需求分析阶段：阅读需求，理解需求，主要就是对业务的学习，分析需求点，参与需求评审会议。

2）测试计划阶段：主要任务是编写测试计划，参考软件需求规格说明书，项目总体计划，内容包括测试范围（来自需求文档），进度的安排，人力物力的分配，整体测试策略的制定，和风险的评估与规避措施有一个制定，一般由测试负责人编写，当然我们可能也会参与相关的评审工作。

3）测试设计阶段：主要任务是编写测试用例，会参考需求文档（原型图），概要设计，详细设计等文档，有不明确的也会及时和开发，产品经理沟通，用例编写完成后会进行评审。

4）测试执行阶段：首先搭建测试环境，执行预测（冒烟测试），以判定当前版本可测与否，如果预测通过，正式进入系统测试，遇到问题提交bug到缺陷管理平台，并对bug进行跟踪，直到被测软件达到测试需求要求，没有重大bug，测试结束----（完善测试用例）

5）测试评估阶段：出测试报告，对整个测试的过程和版本质量做一个详细的评估，确认是否可以上线。

软件开发流程、测试流程梳理

6）开发人员的工作流程：需求分析-得知功能组成及设计软件结构、数据结构（概要设计、详细设计）-编写代码单元测试-代码审查-打包提交测试部-等待测试提交bug-修复bug-等待测试回归bug-……N轮-版本上线-面向用户使用

7）测试人员的工作流程：需求分析--编写测试用例-评审测试用例-搭建测试环境-等待开发研发完成，提交测试包进行测试（酱油期）-部署测试包-冒烟测试（预测）-执行测试用例-bug跟踪处理（提交及回归bug）……N轮-版本上线-面向客户使用

**测试计划：**

**测试方案：**

测试策略：

测试报告：

软件测试人员的能力要求：兴趣、灵感、经验、态度、沟通能力、开发能力

测试和调试的区别：

1. 完成的任务不同：

测试的任务是发现程序中的缺陷；

调试的任务是定位并且解决程序中的缺陷；

1. 执行的人不同：

测试主要由测试人员和开发人员来执行，黑盒测试主要由测试人员完成，

单元/集成测试主要由开发人员执行。

1. 执行的周期不同：

测试贯穿整个软件生命周期，调试一般在开发阶段。

**软件缺陷定义：**

1. 软件为达到产品说明书中已标明的功能
2. 软件出现了产品说明书中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，或者最终用户认为该软件使用效果不好

Bug的分类：

****致命****：不能完全满足系统要求，系统停止运行，系统的重要部件无法运行，系统崩溃或者挂起等导致系统不能正常运行。  
修改优先级为最高，该级别问题需要立即修改。

1.系统崩溃、冻结  
2.导致程序重启,死机或非法退出  
3.死循环  
4.数据丢失或异常  
5.数据通讯错误。  
6.硬件故障，系统悬挂  
7.严重花屏  
8.功能设计与需求严重不符  
9.内存泄漏  
10.用户权限问题、安全问题  
11.其它导致无法测试的错误

****严重****：严重地影响系统要求或基本功能的实现，且没有更正办法（重新安装或重新启动该软件不属于更正办法）。使系统不稳定、或破坏数据、或产生错误结果，或部分功能无法执行，而且是常规操作中经常发生或非常规操作中不可避免的主要问题，系统无法满足主要的业务要求，性能、功能或可用性严重降低。  
修改优先级为高，该级别需要程序员尽快修改。

1.功能不符合用户需求  
2.数据计算错误  
3.业务流程错误  
4.程序接口错误  
5.因错误操作迫使程序中断；  
6.系统可被执行，但操作功能无法执行（含指令）；  
7.功能项的某些项目（选项）使用无效（对系统非致命的）；  
8.功能实现不完整，如删除时没有考虑数据关联；  
9.系统刷新错误，语音或数据通讯错误；  
10.功能的实现不正确，如在系统实现的界面上，一些可接受输入的控件点击后无作用，对数据库的操作不能正确实现。

****一般****：系统可以满足业务要求，系统性能或响应时间变慢、产生错误的中间结果但不影响最终结果等影响有限的问题。  
修改优先级为中，该级别需要程序员修改。

1.数据长度不一致  
2.内容或格式错误  
3.响应时间较慢  
4.功能性建议  
5.提示信息错误（包括未给出信息、信息提示错误等）  
6.操作界面错误（包括数据窗口内列名定义、含义是否一致）；  
7.简单的输入限制未放在前台进行控制；  
8.系统未优化（性能问题）；  
9.不能定位焦点或定位有误，影响功能实现；  
10.增删改功能，在本界面不能实现，但在另一界面可以补充实现。

****轻微****：使操作者不方便或操作麻烦，但它不影响执行工作功能或重要功能。界面拼写错误或用户使用不方便等小问题或需要完善的问题  
修改优先级为低，该级别需要程序员修改或不修改。

1.界面不规范；  
2.辅助说明描述不清楚；  
3.输入输出不规范；  
4.长时间操作未给用户提示；  
5.提示窗口文字未采用行业术语；  
6.可输入区域和只读区域没有明显的区分标志；  
7.必填项与非必填项应加以区别；  
8.滚动条无效；  
9.键盘支持不好，如在可输入多行的字段中，不支持回车换行  
10.界面不能及时刷新，影响功能实现

****建议****：希望提出的建议以及建议进行但不强制进行的修改。不会给发布的准确性或可用性带来任何严重影响  
修改优先级为低，该级别需要程序员修改或不修改

1.各种提示框信息使用不统一,未采用行业术语  
2.界面显示或描述建议  
3.光标跳转设置不好，鼠标（光标）定位错误；  
4.其他建议性问题。