Day15\_测试用例

# 测试用例的概念和作用

如何以最少的人力、资源投入，在最短的时间内完成测试，发现软件系统的缺陷，保证软件的优良品质，则是软件公司探索和追求的目标。

测试用例是测试工作的指导，是软件测试的必须遵守的准则，更是软件测试质量稳定的根本保障。

对一个测试工程师来说,测试用例的设计编写是一项必须掌握的能力，但有效的设计和熟练的编写测试用例却是一个十分复杂的技术，测试用例编写者不仅要掌握软件测试技术和流程，而且要对整个软件不管从业务，还是对软件的设计、程序模块的结构、功能规格说明等都要有透彻的理解。

测试的设计方法不是单独存在的，具体到每个测试项目里都有很多种方法，每种类型都有各自的特点。

测试用例是执行测试的一个实体(包含执行步骤,预期结果,输入参数等操作)。

## 测试用例的定义：

（1）测试用例是为达到最佳的测试效果或高效的揭露隐藏的错误而精心设计的少量(例如一千一万条是没有意义的)测试数据，包括测试输入、执行条件和预期的结果。

（2）测试用例是执行的最小实体。

测试用例的特征：

（1）最有可能抓住错误的；

（2）不是重复的、多余的；

（3）一组相似测试用例中最有效的；

（4）既不是太简单，也不是太复杂。

测试用例的步骤一般不超过7步。

## 编写测试用例的好处：

在开始实施测试之前设计好测试用例，可以避免盲目测试并提高测试效率。

测试用例的使用令软件测试的实施重点突出、目的明确。在软件版本更新后只需修正少部分的测试用例(可以把上一版完成的用例直接拿过来修改)便可展开测试工作，降低工作强度、缩短项目周期。

功能模块的通用化和复用化使软件易于开发，而相对于功能模块的测试用例的通用化和复用化则会使软件测试易于开展，并随着测试用例的不断精化其效率也不断攀升。

## 测试用例的代表性

能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。(一个测试用例应该覆盖合理和不合理的数据)

测试结果的可判定性

即测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

测试结果的可再现性

即对同样的测试用例，系统的执行结果应当是相同的。

-🡪

# 设计测试用例

## 如何设计测试用例？

根据产品规格，测试基本功能；

考虑设计一般用户（非专业人员）的使用方案；

考虑设计稀有或特殊的使用方案；

与系统其他组成部分的配合（如FAX和上网可能要用到MODEM，测试中考虑对设备的共享）；

考虑特殊情况（如内存和硬件的冲突等）；…..其他软件的兼容情况

设计极端情况（如内存泄漏java、破坏性测试等）；

好的测试用例集能花费最小的代价（人力、物力、财力、时间）做最好的测试。

## 测试用例的4个特性

代表性：能够代表并覆盖各种合理的和不合理、合法的和不合法的、边界的和越界的以及极限的输入数据、操作等。

针对性：对程序中的可能存在的错误有针对性地测试

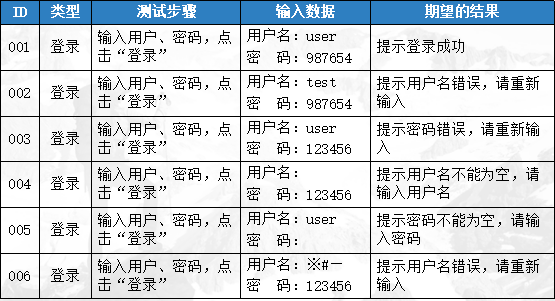
可判定性：测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果

可重现性：对同样的测试用例，系统的执行结果应当是相同的。

## 测试用例通常包括以下几个组成元素：

测试用例编号 测试用例名称(测试注册用例) 测试用例设计者 软件版本号 测试目的 参考信息 测试环境 输入数据(页码) 操作步骤(打开网站,输入信息,点击搜索…等) 预期结果 测试结果 测试模块

## 测试用例示例



-🡪:

# 编写测试用例的基本方法

实际公司写法是靠经验及逻辑思维完成

## 等价类划分法

### 概念

等价类划分是指分步骤地把海量（无限）的测试用例集减得很小，但过程同样有效。

等价类 ：何为等价类，某个输入域的集合，在这个集合中每个输入条件都是等效的。

### 示例

计算两个1～100之间整数的和。34….1=<x<=100的整数 有效等价类

>0 <=10

|  |  |  |
| --- | --- | --- |
| 输入框1 | 输入框2 | 求和  求和求和 |

如果要进行完全测试，一共要设计多少个测试用例呢？

加数1有1～100共计100个取值，加数2也有1～100共计100个取值，所以他们之间的组合就有100\*100=10000种组合可能，但这只是测试了正常范围内的取值。如果用户输入的数据不在1～100之间呢，穷举测试肯定不可能的。由此引入了等价类划分思想。

等价类划分为：

有效等价类：指符合《需求规格说明书》，输入合理的数据集合

无效等价类：指不符合《需求规格说明书》，输入不合理的数据集合



我们将输入域分成了一个有效等价类（1～100）和两个无效等价类（<1,>100），并为每一个等价类进行编号，然后我们就可以从每一个等价类中选取一个代表性的数据来测试，设计如下表所示的测试用例

🡪

### 练习案例:

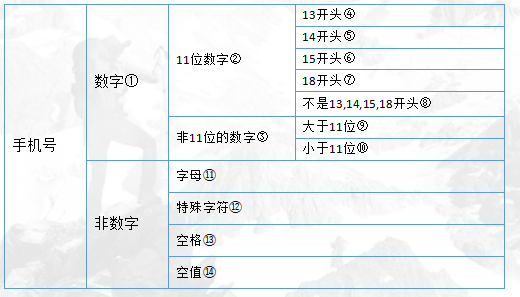
无效等价类<1

无效等价类>100

有效等价类1～100



划分等价类并编号,下表为等价类划分的结果



常用在:有输入参数的时候,划分有效和无效等价类

🡪

## 边界值法

### 边界值法

对数据进行软件测试，就是在检查用户输入的信息、返回的结果以及中间计算结果是否正确。即使最简单的程序要处理的数据量也可能极大，使这些数据得以测试的技巧是，根据一些关键的原则进行等价类的划分，以合理减少测试用例，这些关键的原则是：边界条件，次边界条件、空值和无效数据。

### 确定边界值的方法

确定边界情况（输入或输出等价类的边界）

选取正好等于、刚刚大于或刚刚小于边界值作为测试数据

输入要求是1 ～ 100之间的整数，因此自然产生了1和100两个边界，我们在设计测试用例的时，要重点考虑这两个边界问题。

-🡪



注明:边界值不是从每个等价类中挑一个作为代表，而是把每个等价类的边界都进行测试。

边界值应用场景:

1. 比如，某商品页面展示一页最多能够展示10条

要一页能够展示10条以内，使能够查询出10条，11条，0条，9条信息。 <=

1. 姓名要求1-20个字符，需要测试0,1,2个字符和19,20,21个字符。
2. 分页，-1 0 1 最大页码

### 练习案例

设置某App登录密码，要求密码必须是数字和字母组合，长度为8-18位。

-🡪

## 因果图法

### 概念:

因果图法比较适合输入条件比较多的情况，测试所有的输入条件的排列组合。所谓的原因就是输入，所谓的结果就是输出。

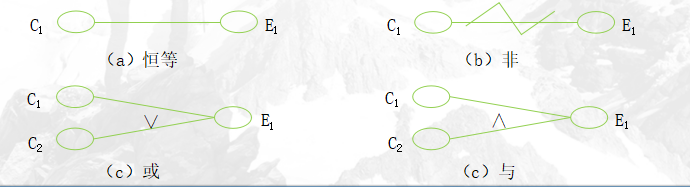
### 因果图基本图形符号

恒等：若原因出现，则结果出现；若原因不出现，则结果不出现。

非（～）：若原因出现，则结果不出现；若原因不出现，则结果出现。

或（∨）：若几个原因中有一个出现，则结果出现；若几个原因都不出现，则结果不出现。

与（∧）：若几个原因都出现，结果才出现；若其中有一个原因不出现，则结果不出现。



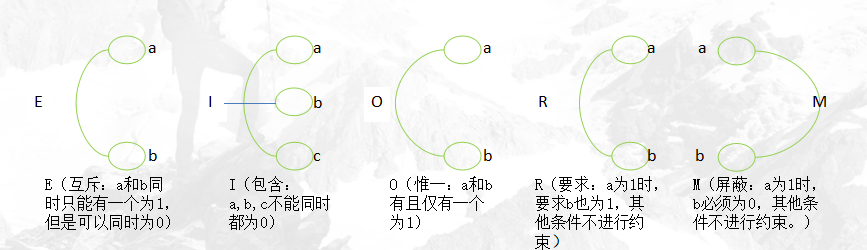
### 因果图的约束符号

E（互斥）：表示两个原因不会同时成立，两个中最多有一个可能成立

I（包含）：表示三个原因中至少有一个必须成立

O（惟一）：表示两个原因中必须有一个，且仅有一个成立

R（要求）：表示两个原因，a出现时，b也必须出现，a出现时，b不可能不出现

M（屏蔽）：两个结果，a为1时，b必须是0，当a为0时，b值不定  


### 因果图测试用例

例如：有一个处理单价为0.5元的盒装饮料的自动售货机软件。若投入0.5元硬币，按“可乐”、“啤酒”、或“奶茶”按钮，相应的饮料就送出来。若投入的是1元硬币，在送出饮料的同时退还5角硬币。

分析这一段说明，我们可列出原因和结果

原因(输入)：

投入0.5元硬币；

投入1元；

按“可乐”按钮；

按“啤酒”按钮；

按“奶茶”按钮。

中间状态： ① 已投币；②已按钮

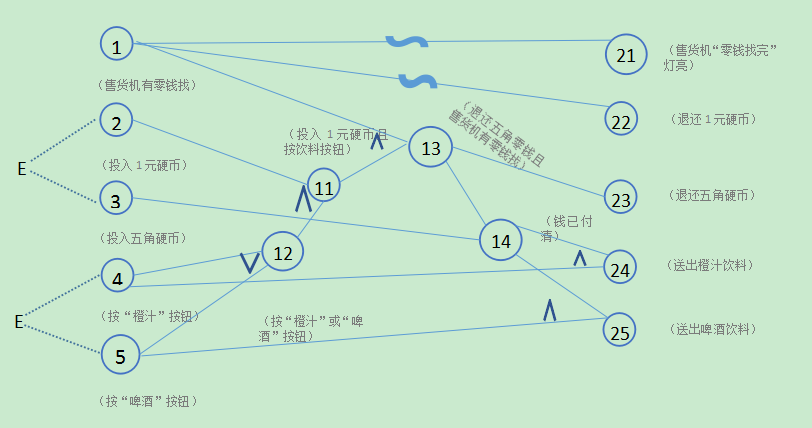
结果(输出)：

退还5角硬币；

送出“可乐”饮料；

送出“啤酒”饮料；

送出“奶茶”饮料；



## 判定表法

由不同的输入原因 得到不同的输入结果



以上不是测试用例,只是判定表法



🡪

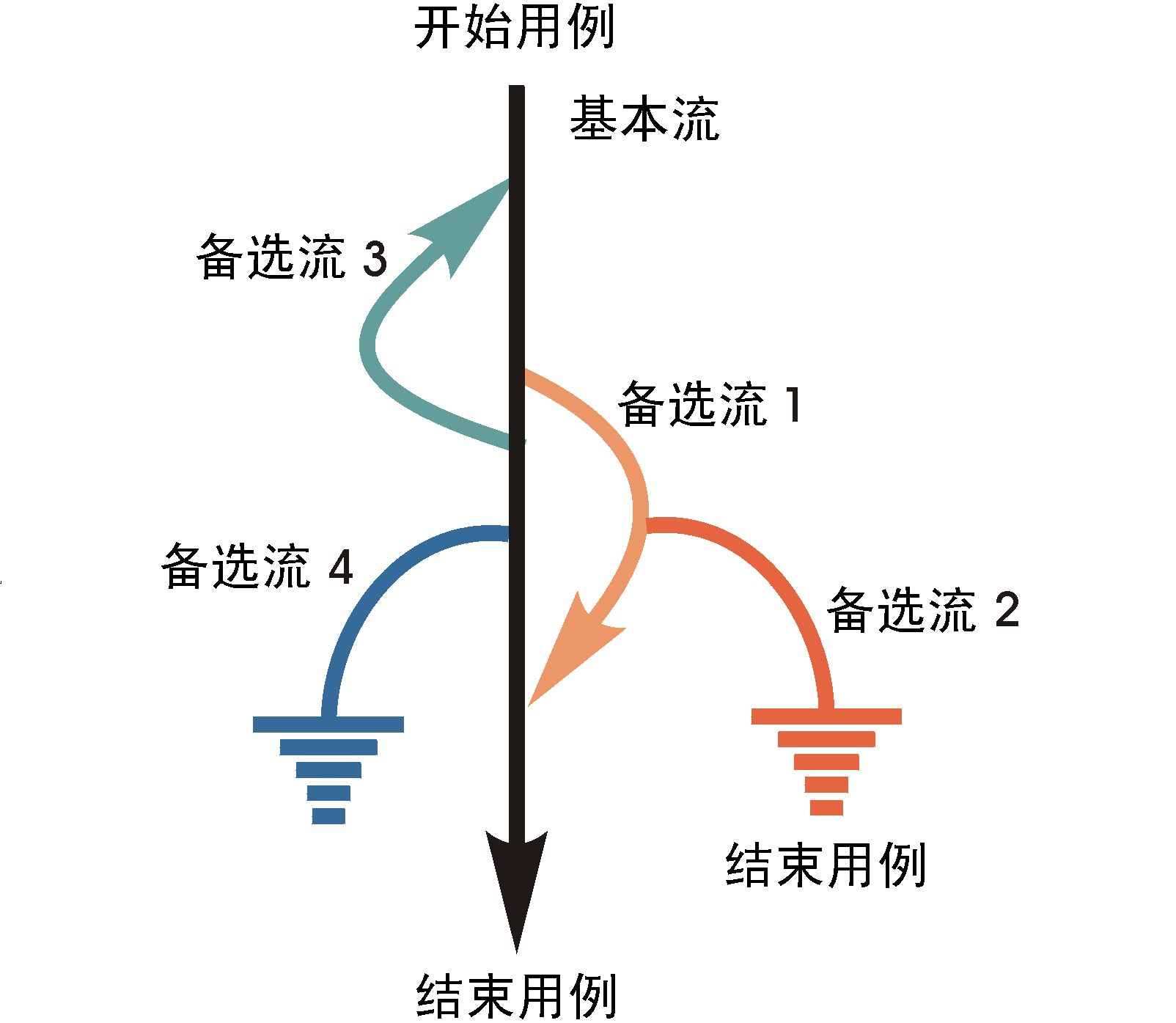
## 场景法

### 测试用例设计的思想

现在的软件几乎都是用事件触发来控制流程的，事件触发时的情景便形成了场景，而同一事件不同的触发顺序和处理结果就形成事件流。这种在软件设计方面的思想也可以引入到软件测试中，可以比较生动地描绘出事件触发时的情景，有利于测试设计者设计测试用例，同时使测试用例更容易理解和执行。

用例场景是通过描述流经用例的路径来确定的过程，

这个流经过程要从用例开始到结束遍历其中所有基本流和备选流。



基本流和备选流

如图所示，图中经过用例的每条路径都用基本流和备选流来表示，直黑线表示基本流，是经过用例的最简单的路径。备选流用不同的色彩表示，一个备选流可能从基本流开始，在某个特定条件下执行，然后重新加入基本流中（如备选流1和3）；也可能起源于另一个备选流（如备选流2），或者终止用例而不再重新加入到某个流（如备选流2和4）。

遵循上图中每个经过用例的可能路径，可以确定不同的用例场景。从基本流开始，再将基本流和备选流结合起来，可以确定以下用例场景：

场景 1 基本流

场景 2 基本流 备选流 1

场景 3 基本流 备选流 1 备选流 2

场景 4 基本流 备选流 3

场景 5 基本流 备选流 3 备选流 1

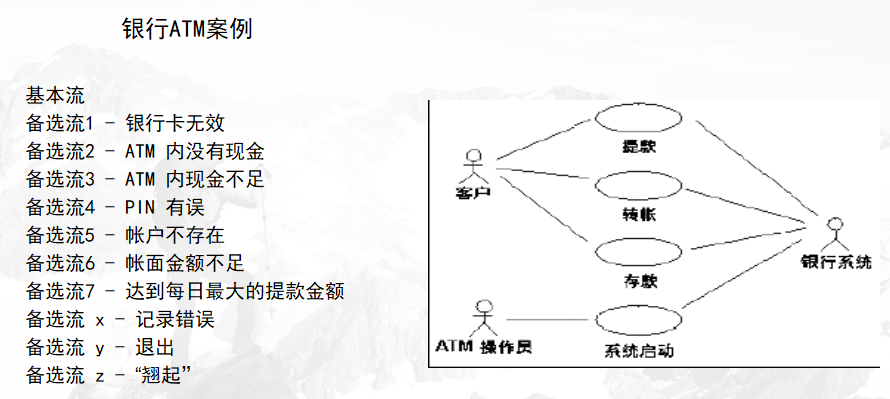
场景 6 基本流 备选流 3 备选流 1 备选流 2

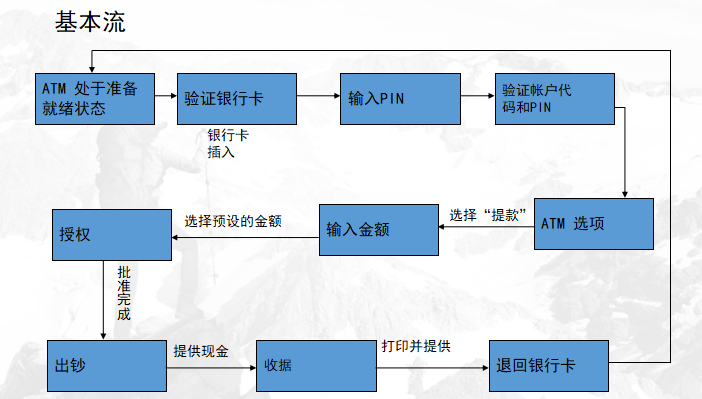
场景 7 基本流 备选流 4

场景 8 基本流 备选流 3 备选流 4

注：为方便起见，场景 5、6 和 8 只描述了备选流 3 指示的循环执行一次的情况。

### 银行案例ATM:





第一次测试中，根据测试计划，我们需要核实提款用例已经正确地实施。此时尚未实施整个用例，只实施了下面的事件流：

基本流-提取预设金额（100 元、200元、500元、1000元）

备选流2 - ATM 内没有现金

备选流3 - ATM 内现金不足

备选流4 - PIN 有误

备选流5 - 帐户不存在/帐户类型有误

备选流6 - 帐面金额不足



对于这7个场景中的每一个场景都需要确定测试用例。可以采用矩阵或决策表来确定和管理测试用例。

从确定执行用例场景所需的数据元素入手构建矩阵。然后，对于每个场景，至少要确定包含执行场景所需的适当条件的测试用例。

下面显示了一种通用格式，其中各行代表各个测试用例，而各列则代表测试用例的信息。

本示例中，对于每个测试用例，存在一个测试用例ID、条件（或说明）、测试用例中涉及的所有数据元素（作为输入或已经存在于数据库中）以及预期结果。





基于常见应用场景及使用场景的分析

## 错误推测法

### 基本思想

主观+经验

错误推测方法的基本思想: 列举出程序中所有可能有的错误和容易发生错误的特殊情况,根据他们选择测试用例. 例如, 在单元测试时曾列出的许多在模块中常见的错误. 以前产品测试中曾经发现的错误等, 这些就是经验的总结。还有, 输入数据和输出数据为0的情况。输入表格为空格或输入表格只有一行. 这些都是容易发生错误的情况。可选择这些情况下的例子作为测试用例.

总之，就是进行错误的操作。

### 测试示例

例如，测试手机终端的通话功能，可以设计各种通话失败的情况来补充测试用 例：

1) 无SIM 卡插入时进行呼出（非紧急呼叫）

2) 插入已欠费SIM卡进行呼出

3) 射频器件损坏或无信号区域插入有效SIM卡呼出

4) 网络正常，插入有效SIM卡，呼出无效号码（如1、888、333333、不输入任何号码等）

5) 网络正常，插入有效SIM卡，使用“快速拨号”功能呼出设置无效号码的数字

## 正交表

### 一个有名的案例研究

1992年AT&T发表了一篇讲述在测试过程中使用正交表一个案例研究。

它描述了对PC (IBM格式)和StarMai(基于局城网的电子邮件软件)

做回归测试:

* 最初制定的测试计划是用18周的时间执行1500个测试用例,但是,开发推迟了,测试时间被压缩到仅8周时间!
* 测试负责人采取另外一个测试方案和计划,即2个人8周的时间测试1000 个测试用例,但是他不敢保证测试的质量,对这些用例检测缺陷的能力不放心.
* 为了减轻这种不确定性的问题,他用正交表法重新设计了测试用例，此时测试用例只有422个,用这422个测试用例去测试发现了41个缺陷,开发人员修复缺陷,然后软件就发布了.
* 在使用的两年时间内,凡被测试到的领域都没有再发现缺陷,因此在发现缺陷这方面,此测试计划是100%有效.
* 据测试负责人估计,如果ATST采用1000个测试用例的测试计划,可能仅仅只发现这些缺陷中的32个,

与最初的计划相比,用正交表设计测试用例执行工作量不到50%,但却多发现28%的缺陷,而且测试人员个人的效率也增加了。

### 正交排列法

正交排列法概述

正交排列法能够使用最小的测试过程集合获得最大的测试覆盖率,当可能的输入数据或者输入数据的组合数量很大时,由于不可能为每个输入组合 都创建测试用例,可以采用这种方法.

案例:字符属性设置程序

窗体中有多个控件(字体,字符样式,颜色,字号),每个控件有多个取值

字体：仿宋、楷体，华文彩云

字符样式:粗体、斜体，下划线

颜色:红色，绿色、蓝色

字号: 20号, 30号,40号

有多少个排列组合呢？ 3^4 = 81

图：



### 案例：字符属性设置程序

在测试时，要考虑这些控件的组合情况，组合量非常大（3^4 = 81种组合情况）

由于组合量太大，不可能为每一种组合都创建测试用例。如何采用最少的测试用例集合获得最大的测试覆盖率-------采用正交排列法

### 正交排列表重要概念

- 正交试验设计

是研究多因素多水平的一种设计方法,它是根据正交性从全面试验中挑选出部分有代表性的点进行试验，这些有代表性的点具备了“均匀分散,齐整可比"的特点,正交实验设计是一种基于正交表的,高效率、快速、经济的实验设计方法。

### 正交表的概念

-正交表:一种特制的表,一般的正交表记为,Ln,(m^K)

- n是表的行数,也就是需要测试组合的次数

- K是表的列数,表示控件的个数 （因素的个数,或因子个数)

- m是每个控件包含的取值个数 （各因素的水平数,即各因素的状态数）

- 如:L9(3^4)

·有4个控件

·每个控件有3个取值

.9为需要测试的组合个数

.叫4因素3水平

图：



[..\4.资料\常见正交表.txt](../4.资料/常见正交表.txt)

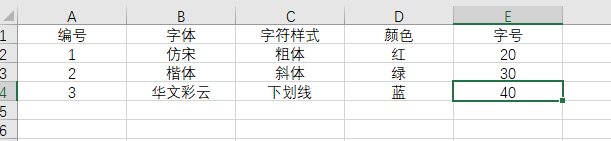
### 正交排列法的使用步骤

1. 根据所测程序中控件的个数（因素）以及每个控件的取值个数(水平) ,选取一个合适的正交排列表

因素：（控件/列）4 水平：（空间中可选取的值）3

3^4 找到对应的正交表

2. 把控件及其取值列举出来，并对其进行编号



3. 把控件及其取值映射到正交排列表中

.把正交排列表中的ABCD (因子)分别替换成4个控件

.把每列中的1.2.3 (状态)分别换成这个控件的3个取值(水平),排列顺序要按照表中给出的顺序

4. 根据映射好的正交排列表编写测试用例

### 完成：案例字符属性设置程序

步骤1：根据所测程序中空间的个数以及每个控件的取值个数，选取一个合适的证件排列表

- 4个控件（因素）：字体，字符样式，颜色，字号

- 每个控件有3个取值（水平）

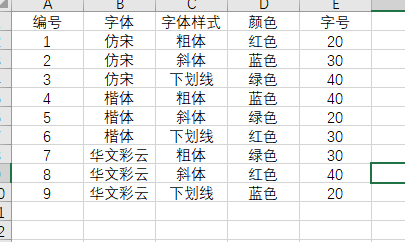
- 选择L9（3^4）正交表

图：



步骤2：把控件及其取值列举出来，并对取值进行编号

图：

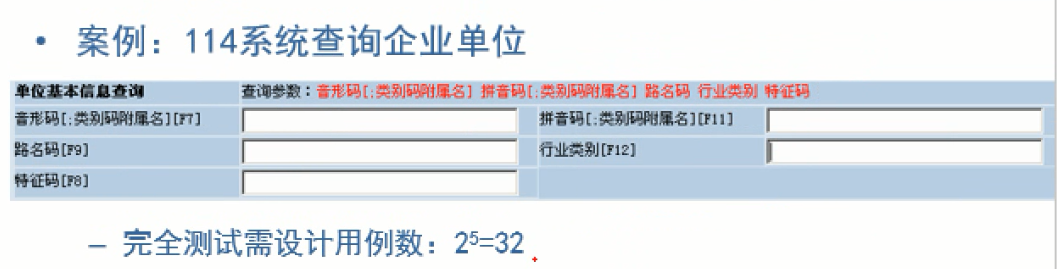


小结：

将81条测试用例降到了9条，这是进行测试的最少组合量，但是，在测试中有72种（81-9=72）组合没有测试到。当然如果时间允许，应该再补充一些测试用例。因为测试用例的组合越多，存在缺陷的可能性就越大。还是竟可能多的去测试。但是九条是我们得到科学的测试方法。（时间问题单指我们使用的还是正交表的形式，把绝大数的情况找到。）

### 114查询系统案例

案例：



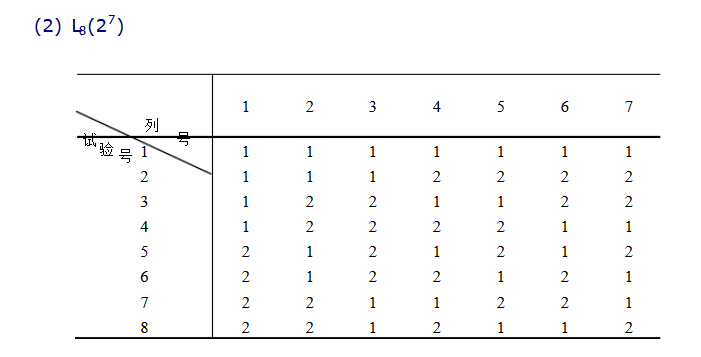
一共有五个控件，每个控件中只有两种情况（空，不为空）

在挑选正交表的时候我们会发现没有2^5的正交表，那我们就选择2^7的正交表，原因就是尽可能多的去测试。

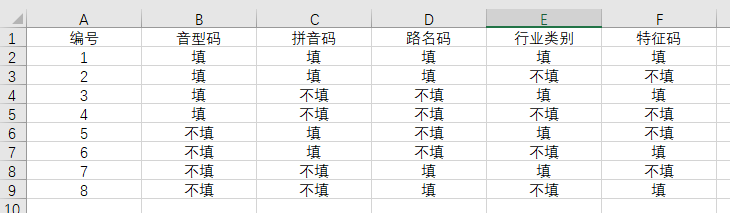
五个控件分别是：音型码，拼音码，路名码，行业类别，特征码

两种取值：填，不填

图：



图：



### 使用正交排列法的局限性

- 目前常见的正交排列表只有前面附录文件中给出的几种

- 即使是已有的正交排列表,基本都要求每个控件中取值的个数要相等,这在实际软件中很少遇到。

没有现成的正交排列表怎么办?

* 通过正交排列法的学习,我们更多的应该学习到一种测试思想,也就是在从所有组合集合中选取测试教据时,应该均匀的选取其中的组合作为测试用例,而不要只在某个局部选取数据。

例如：

水平数不同

* 因素（变量）的水平数（变量的取值）不相同

例：

* 体重 年龄段 性别

胖 老人 男

适中 青年 女

瘦 儿童

-找不到现成的正交表，我们就需要使用混合正交表，{能使用工具来生成}。

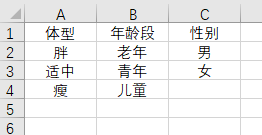
### 混合正交表的使用 （工作中绝大部分使用）

正交表生成工具allpairs

. 很多情况下无法找到合适的正交表,就要使用正交表生成工具。

使用步骤:

1. 制作取值表 【只列出数据即可，不用编号】

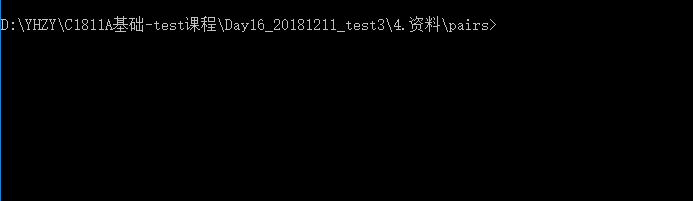


2、复制取值表的数据,放到文本文档中保存(注意不要更改任何格式,例如文件叫a.txt)

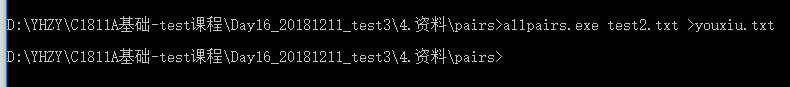
3、把文本文档放在allpairs文件夹中

4、win + r后输入cmd进入控制台

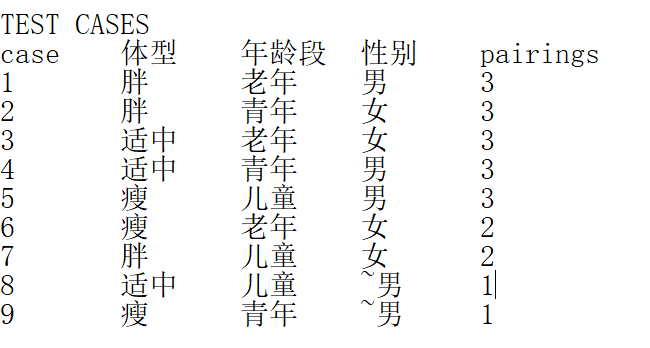
5、进入allpairs文件来



6、在控制台中输入allpairs.exe Test2.txt >youxiu.txt(youxiu是自己起的名字,用来存放生成的组合用例,可以自动生成,不必提前建好)



最终结果：



注意：波浪线后内容随意，男女都可。最后一列多于数据可以删除的。

# 测试用例的评审和变更

测试用例并非一成不变。如果软件修改之后发生变化，或者需求发生变更，那么测试用例便不再满足当前版本软件的测试需求，由此需要进行修改和变更操作。

首先要清楚内部评审的定义，是测试组内部的评审，还是项目组内部的评审。评审的定义不同，内容也不会相同。

如果是测试组内部的评审，应该着重于：

1.测试用例本身的描述是否清晰，是否存在二义性；

2.是否考虑到测试用例的执行效率.往往测试用例中步骤不断重复执行，验证点却不同，而且测试设计的冗余性，都造成了效率的低下；

3.是否针对需求跟踪矩阵，覆盖了所有的软件需求；

4.是否完全遵守了软件需求的规定。这并不一定的，因为即使再严格的评审，也会出现错误，应具体情况具体对待。

如果是项目组内部的评审，也就需要评审委员会来做了，角度不同，评审的标准也不同。比如收集客户需求的人员注重你的业务逻辑是否正确，分析软件需求规格的人注重你的用例是否跟规格要求一致，开发负责人会注重你的用例中对程序的要求是否合理。

测试用例的评审能够使用例的结构更清晰，覆盖的用户场景更全面对于测试工程师来说也是一个快速提高用例设计能力的过程。

1、需要评审的原因

测试用例是软件测试的准则，但它并不是一经编制完成就成为准则。由于用例开发人员的设计经验和对需求理解的深度各不相同，所以用例的质量难免会有不同程度的差异。

2、进行评审的时机

一般会有两个时间点。第一，是在用例的初步设计完成之后进行评审第二是在整个详细用例全部完成之后进行二次评审。如果项目时间比较紧张，尽可能保证对用例设计进行评审，提前发现其中的不足之处。

3、参与评审人员

这里会分为多个级别进行评审。

1)部门评审，测试部门全体成员参与的评审。

2)公司评审，这里包括了项目经理、需求分析人员、架构设计人员、开发人员和测试人员。

3)客户评审，包括了客户方的开发人员和测试人员。这种情况在外包公司比较常见。

4、评审内容

评审的内容有以下几个方面

1)用例设计的结构安排是否清晰、合理，是否利于高效对需求进行覆盖。

2)优先极安排是否合理。

3)是否覆盖测试需求上的所有功能点。

4)用例是否具有很好可执行性。例如用例的前提条件、执行步骤、输入数据和期待结果是否清晰、正确期待结果是否有明显的验证方法。

5)是否已经删除了冗余的用例。

6)是否包含充分的负面测试用例。充分的定义，如果在这里使用2&8法则，那就是4倍于正面用例的数量，毕竟一个健壮的软件，其中80%的代码都是在"保护"20%的功能实现。

7)是否从用户层面来设计用户使用场景和使用流程的测试用例。

8)是否简洁，复用性强。例如，可将重复度高的步骤或过程抽取出来定义为一些可复用标准步骤。

个人认为，一个"健康"的测试用例至少要通过前5个标准。

5、评审的方式

1)召开评审会议。与会者在设计人员讲解之后给出意见和建议，同时进行详细的评审记录。

2)通用邮件与相关人员沟通

3)通用IM工具直接与相关人员交流

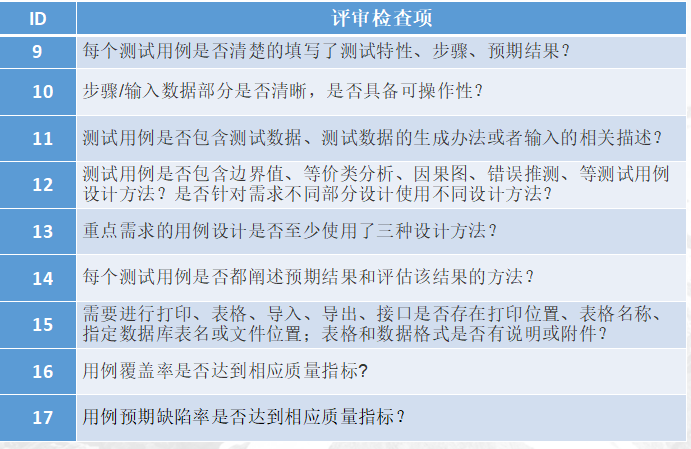
方式只是手段，得到其它人员对于用例的反馈信息才是目的。

无论采用那种方式，都应该在沟通之前把用例设计的相关文档发送给对方进行前期的学习和了解，以节省沟通成本。

6、评审结束标准

在评审活动中会收集到用例的反馈信息，在此基础上进行用例更新，直到通过评审。





# 测试计划及缺陷管理

## 测试计划

测试计划，应该包括:

|  |  |  |  |
| --- | --- | --- | --- |
| 测试背景 | 测试目标 | 测试范围 | 测试输出文档(测试报告,bug文档) |
|  |  |  |  |
| 测试工具 | 测试规模工作量分析(覆盖率,哪些模块用例比较多) |  | 测试进度及时间安排(每天具体的测试任务) |
|  |  |  |  |
| 测试轮次安排 | 测试方法(测试分类中的方法) |  |  |



🡪

## BUG分类

bug 臭虫

### 按照严重程度分:

一般分为5个等级:

系统崩溃，严重，一般，次要，建议

### 按优先级分:

1. Low --时间和资源允许的情况下修复

2. Medium --不会延迟发布,会在以后修复

3. High --会制约开发和测试的进行,需要在发布之前修复

4. veryHigh --影响系统,产生严重影响

5. Urgent --导致系统几乎不可用

### Bug定级示例

| **缺陷级别定义** |
| --- |
| **1级，系统崩溃** **定义：**严重阻碍测试和开发工作 **对应优先级：最高** **具体可分为：** 1.功能完全没有实现 2.应用闪退/崩溃无法运行 3.其他导致功能无法测试的问题 |
| **2级，至关重要** **定义：**非阻碍用例执行的严重问题 **对应优先级：高** **具体可分为：** 1.简单操作应用闪退/崩溃，卡死  2.数据丢失 3.严重影响系统自身功能无法运行 4.严重数值计算错误 5.数据库损坏或无法保存配置 6.安全性问题（包括数据加密等） |
| **3级，主要** **定义：**功能存在缺陷，但不影响应用和系统的稳定性 **对应优先级：中** **具体可分为：** 1.内存泄露(长生命周期的对象持有短生命周期的引用 )  2.功能实现逻辑覆盖不全面(搜索->搜索页面->商品列表->搜索页面) 3.非必现，但复现概率超过50%的闪退/崩溃 |
| **4级，一般** **定义：**对应用熟悉度高才能感知到的问题，对应用基本功能实现无影响 **对应优先级：中** **具体可分为：** 1.轻微数值计算错误(999.0 999.00) 2.功能实现有误，与产品文档不完全贴切 3.用户简单操作，即可明显感知的UI问题 |
| **5级，较小** **定义：**界面，性能缺陷 **对应优先级：低** **具体可分为：**  1.操作界面错误（提示显示规则，刷新规则是否与文档一致,例如登陆成功和登录成功）  2.边界条件显示错误  　 　 　 3.提示信息和界面效果展示错误（包括未给出信息、信息提示错误等）  4.复现率低于5%的闪退/崩溃和安全模式  　 　 　 5.插件兼容和性能未优化问题  　 　 　 6.非正常操作导致UI显示异常 |
|  |
| **6级，建议** **定义：**对于产品的意见或者建议 **对应优先级：低** **具体可分为：** 1.对于产品设计方面的意见和建议 2.对于产品界面优化方面的意见和建议 3.对于产品需要优化增强用户体验方面的意见和建议 |

🡪

### 按照测试种类分:

逻辑功能类，性能类，界面类，易用性类，兼容性类

### 按照功能模块分:

登录注册，购物车，分类，订单，个人信息

### 软件缺陷分类（缺陷>BUG）

|  |  |  |
| --- | --- | --- |
| 缺陷类型 | 内容说明 | 备注 |
| 系统缺陷 | 1 由于程序所引起的死机，异常退出  2 程序死循环  3 程序错误，不能执行正常工作或重要功能，使系统崩溃或资源不足 | 不能执行正常工作或重要功能，使系统崩溃或资源不足 |
| 数据缺陷 | 1 数据计算错误  2 数据约束错误  3 数据输入、输出错误 | 严重的影响系统要求或基本功能的实现 |
| 数据库缺陷 | 1 数据库发生死锁  2 数据库的表、缺省值未加约束条件  3 数据库连接错误  4 数据库的表有过多的空字段 |  |
| 接口缺陷 | 1 数据通信错误  2 程序接口错误 |  |
| 功能缺陷 | 1 功能无法实现  2 功能实现错误 |  |
| 安全性缺陷 | 1 用户权限无法实现  2 超出限制错误  3 访问控制错误  4 加密错误 |  |
| 兼容性缺陷 | 与需求规定配置兼容性不配合 |  |
| 性能缺陷 | 1 未达到预期的性能目标  2 性能测试中出错，导致无法进行测试 |  |
| 界面缺陷 | 1 操作界面错误  2 打印内容，格式错误  3 删除操作未给出提示  4 长时间操作未给出提示  5 界面不规范 |  |
| 建议 | 1 功能建议  2 操作建议 | 建议性的改进 |

### 常见缺陷的查找

**U1用户界面**

色彩

色彩的搭配无序、混乱是软件图形界面设计的大忌,图形界面应尽量设计得温和些。这类缺陷主观类强,个人主观占据主动，所提交的缺陷一般严重程度不可定得太高。

**例如:**

整体界面色彩单调,无变化,仅使用一种色彩,且篇幅较大，可以提交建议性的缺陷,即使是简单的界面,宁可采用无色，不可使用鲜艳的单色，如紅色,黄色、绿色等.

背景色与将界面字体色相近,不能清晰区分,色彩搭配理乱、复杂,且不符合软件标准。

**功能结构布局**

功能结构布局主要从界面的功能区域划分来考虑.相同的、类似的功能应该放在邻近的区域。

例如:

记录添加功能界面,添加按钮未放在醒目的位置

导航功能位于界面的右则

整体功能区域分布混乱

图片

图片选用不合理,与当前软件类型不符,无法正确体现当前界面功能性含义。图片不规范，不清晰

例如：

图片色彩过于艳丽或黯淡,機湖不清;

图片变形,

图片不符合当前界面的主题,图片与描述性文字不符

页面大小

在B/S结构的软件系统中,当一个页面元素太多，未做精简时,在打开该页面时可能需要较长的加载时间,这对于软件性能是一个不小的影响,既增加了服务器的压力,又容易引起用户的反感.

例如:

图片未经压缩、格式不正确,比如采用BMP

代码冗余,存在太多无用代码

页面元素太多,太过复杂

**字体：**

字体在软件界面中尤其重要，字体的使用要符合软件开发的规范

**例如：**

字体过大，与其他页面信息脱节，无法形成主体

字体过小，无法看清内容

字体不符合当前界面风格

**窗体大小**

窗体的设计要有层次感,父窗口,子奋口应该有所区别，窗口不应该有太多空白处,功能区城充实。

**例如:**

窗口太大,功能按钮分散,间隔太大;

窗口太小,功能按钮过于集中,间隔太小,或空间会显示不全

弹出窗口未能定于屏幕居中位置,

界面文字

页面信息描述不清楚，有语病，错别字，简单语言复杂化，描述不正确，不符合当前页面，错误的帮助信息，乱码等。

容错处理(功能缺陷)

容错处理在软件系统中占据十分重要的地位,所谓容错,就是容忍错误的能力.当用户在使用软件过程中发生错误后:软件应该能给出引导信息,指引用户进行正确的操作。

例如:

用户输入错误,系统无提示,无影响，用户不能清楚知道系统不处理的原因;

给出信息提示,用户接受后无法继续操作:不给用户一个改过自新的机会

用户输入不合法的信息后,系统进行提示, 用户确定后,系统仍能处理错误的信息。

取消功能不能取消,比如删除,系统给出提示，是否确定删除，用户否认后仍执行了删除.

**数据转换**

软件中的功能主体一般由增加,修改,删除,查询等组成。

例如：

无法增加记录,比如点击新增,页面自动关闭

增加记录后无显示,但提示增加成功

增加记录后显示不正确,显示为乱码,信息显示不全

增加记录后多出记录

无法修改记录

修改后不能自动更新,需手工更新

无法删除记录,无法全部期除

删除不成动,但相应的记录已被删除

无法查询,查询的结果错误

性能缺陷

这里所说的性能问题不需专业的工具就能发现的问题,这类问题在平常做黑盒测试的时候就能发现。

例如:

打开文档, 10秒应该可以完成的,却用了3分钟

启动软件, CPU长时间100%,内存消耗过多.

5个用户可以正常使用,20个用户使用时系统崩溃了

打开一个登录贡面花了1分钟;

完成一个查询功能,花了2分钟,

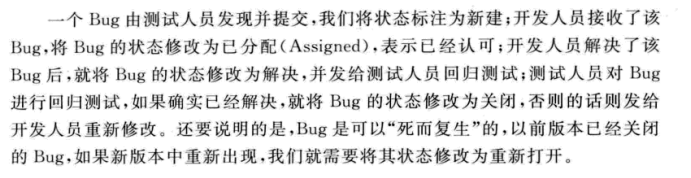
### 按照解决方案分



### 按照Bug生命周期..面试

新建(测试)，确认(程序员)，解决，重新验证，关闭，重新打开

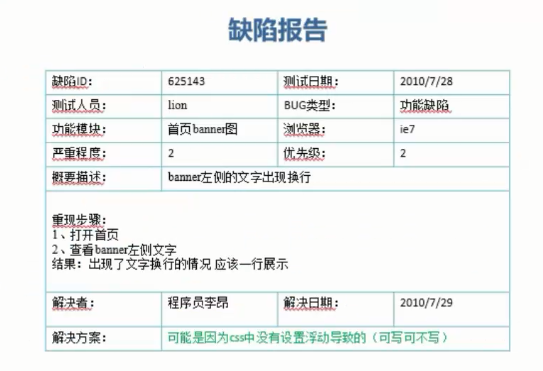
公司处理bug的流程:



## 缺陷报告(Bug报告，提的Bug)

需要包含下面15项的部分就可以,,,每一个bug都要有一个bug报告

|  |  |  |
| --- | --- | --- |
| 编号 | 属性名称 | 描述 |
| 1 | 缺陷ID | 唯一的id，确保根据ID追踪缺陷 |
| 2 | 缺陷状态 | 进展情况，关闭，打开，修复 |
| 3 | 缺陷标题 | 描述缺陷的标题 |
| 4 | 缺陷的严重程度 | 致命，较严重，严重，一般，低 |
| 5 | 缺陷的优先级 | 缺陷修复的先后顺序，哪些优先修正，哪些稍后修正 |
| 6 | 缺陷的所属模块 | 缺陷所属的项目模块 |
| 7 | 缺陷记录者 | 提交缺陷的测试人员 |
| 8 | 缺陷提交时间 | 缺陷的提交时间 |
| 9 | 缺陷处理人 | 处理缺陷的处理人 |
| 10 | 处理结果描述 | 对处理结果的描述，描述处理情况和代码修改说明 |
| 11 | 缺陷验证人 | 对被处理缺陷验证的验证人（回测者）1 |
| 12 | 验证结果描述 | 对验证结果的描述（通过，不通过） |
| 13 | 缺陷详细描述 | 缺陷的重现步骤 |
| 14 | 缺陷环境说明 | 对测试环境的描述 |
| 15 | 必要的附件 | 如涉及到附件或者错误现象的图片等 |



## Bug的处理🡪如生命周期