目 录

测试女巫_找到高效工作流程篇

作者：妞妞（笔名）

摘要：将统计学_6 sigma 应用到测试技术中，需要使用的 6 sigma 工具如下：鱼骨图，Xbar Chart，One Way ANOVA，Two Way ANOVA

关键字：测试技术，统计学，6 sigma

一、 前言

测试女巫又来啦，大家还记的吗？上次我们主要以找到 bug 产生的原因为例介绍 DOE，柏拉图，主效应，交互作用，相关性这些方法，并着重介绍如何将这些方法应用到我们软件测试中。

那这次我们介绍什么呢？回想一下对于我们软件测试来说，从项目经理，软件开发主管那里听到抱怨最多的事情是什么？项目经理：测试时间太多了，会 delay 我们项目的 schedule！软件主管：严重的 bug 能不能早点发现啊？到项目后期发现严重 bug，让我怎么解？！哦，想想就觉得头疼：这两位大哥啊，你们不觉得你们的需求是矛盾的吗？不给我足够的测试时间，我怎么能把 bug 尽早的发现出来呢？每一轮的测试还没有进行全面深入的测试，新的测试需求又进来了，这些状况难道你们都看不到吗？

OK，没关系，淡定~~~，姐早已经不是只会撇着嘴委屈哭泣的职场小白兔，相信办法总比困难多！对了，多年的测试，让我积累了不同项目的大量数据，现在也有了女巫的魔法棒和水晶球（具体魔法棒和水晶球的介绍请参考 34 期的杂志），嗯！这次双剑合璧，让工作经验和 6 sigma 工具联手，来尝试解决这个又棘手又矛盾的问题！各位看客，有没觉得热血沸腾？有没觉得多学一些工具真的是件很开心和幸福的事？有没觉得工作中那些看似不合理的要求，其实都是促使我们进步的机会？好吧，闲话不多说，我们开始这次兴奋之旅吧！

这次我们主要用到的工具为：鱼骨图，Xbar Chart，One Way ANOVA，Two Way ANOVA。在第三十四期的杂志中已经介绍了柏拉图，主效应图，交互作用图的原理和用法，此份文档中也会用到这些工具，因为之前已经介绍过了，这里就不再赘述。

二、 上述这些方法的基础知识介绍

1. 鱼骨图

佛教讲究因果，在我们的统计科学中同样也是讲究因果，统计学中的因果，

是可以通过鱼骨图形象的表达出来。鱼骨图通常分为 3 种：整理问题型鱼骨图(即没有非常明显的因果关系，只是结构构成关系)；原因型鱼骨图(即鱼头在右，鱼头就是果，鱼的身体就是因)；对策型鱼骨图(即鱼头在左，鱼身就是改善鱼头的对策)。此次我们使用的第 2 种鱼骨图：即原因型鱼骨图

对于“原因型鱼骨图”，原因是从以下 6 方面来研究的：

1) 人：人员的原因
2) 机：使用机器的原因
3) 料：操作对象的原因
4) 法：使用方法的原因
5) 环：环境的原因
6) 测：测量方法的原因

2. Xbar Chart

它是一个判断一段时间内数据绘制成一个折线图，观察此折线图是否存在时间效应，如存在时间效应，就说明此流程并不是平稳的过程。所谓的时间效应指的是“是否有超限值”和“有无特殊的规律”

3. 正态分布

又称为高斯分布，有时也称为常态分布，是连续随机变量概率分布的一种，自然界，人类科学，心理和教育中的大量现象，可以通过记录数据来表现其特征，通过研究，这些数据大部分都是符合正态分布的。例如能力高低，学生成绩的好坏等。

我们后续研究的 ANOVA 分析方法，首先需要确认所研究的数据符合正态分布。

4. 变异数分析 ANOVA

1) 基本概念

我们进行统计分析的目的是希望知道一组数值的“变异原因”，对于数值的变化一般分成两部分：随机变异和某些因素导致的变异。

即：总变异=随机变异+某些因素导致的变异，将此理念引申到统计学中，即为：总变异=组内变异+组间变异。

组内变异是此组数据由于分布状况导致资料内部的变异，不同的资料分布状况是不一样的，所以此变异是随机的。

组间变异是由于某些因素导致的变异，这个变异才是我们希望深入研究的。

如组间变异远远大于组内变异，说明此因素是影响此组数值发生变化的原因。如果两者相差无几，则说明此因素对此组数值没有影响。

2) ANOVA 方法分析步骤

a. 先做出虚无假设

即我们所研究的数值在不同的因子的条件下，并没有差异。

b. 判断所研究的数值是否符合正态分布

因为使用 ANOVA 分析的前提条件，必须是符合正态分布，否则此组数值是无法使用此分析方法进行分析。

c. 给定如果接受虚无假设，可以承担的风险是多少

此数值一般是一个固定值，一般为 a=0.05

d. 开始使用工具进行 ANOVA 检定分析

f. 分析完毕后，会有一个残值出现

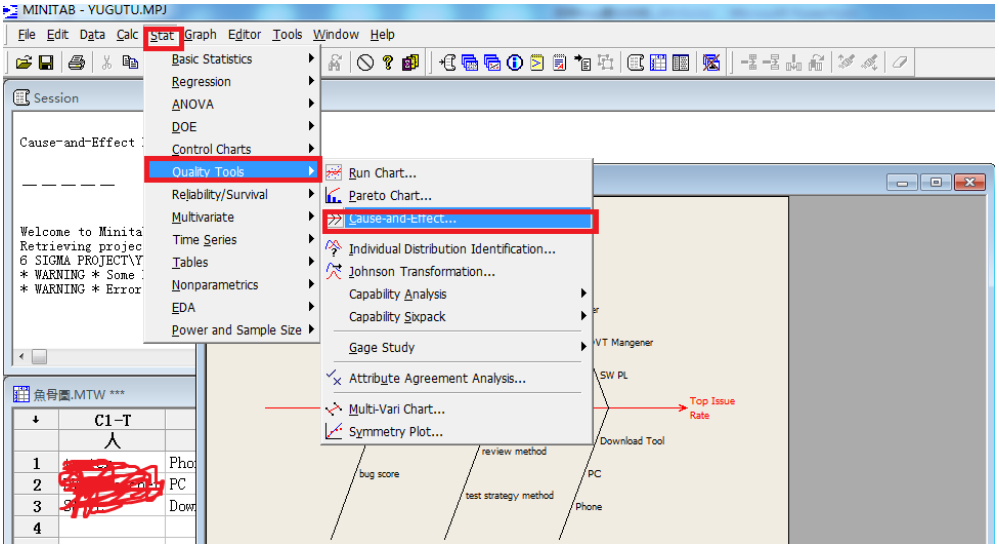
需要对此残值进行分析，如果 P- value 值小于 0.05，说明虚无假设不成立，即此因子是造成此组数值变异的显著因子。

三、 6 Sigma 常用工具介绍

1. 鱼骨图

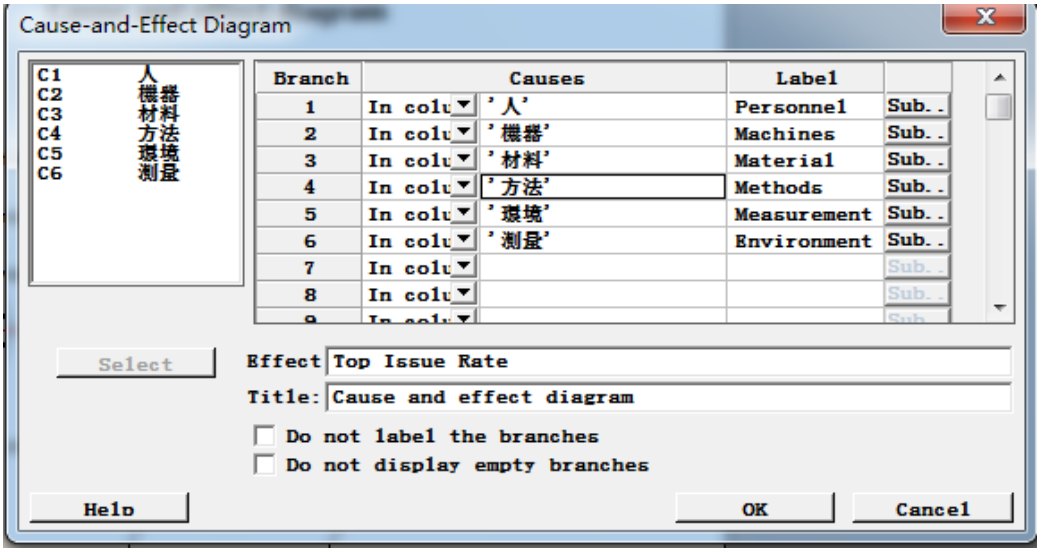
我们主要使用的是“原因型鱼骨图”，使用方法如下

1) 进入 Minab → 统计 (Stat) → 品质工具 (Qualy Tools) → 鱼骨图 (Cause-and-Effect)，如【图 1】



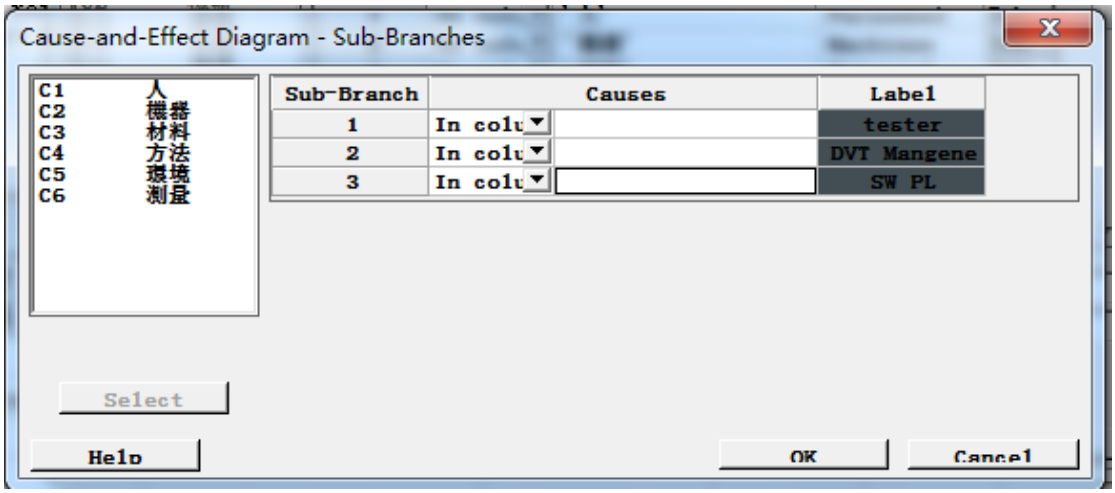
【图 1】

2) 进入如【图 2】界面，此界面是将鱼的“骨架”建立起来



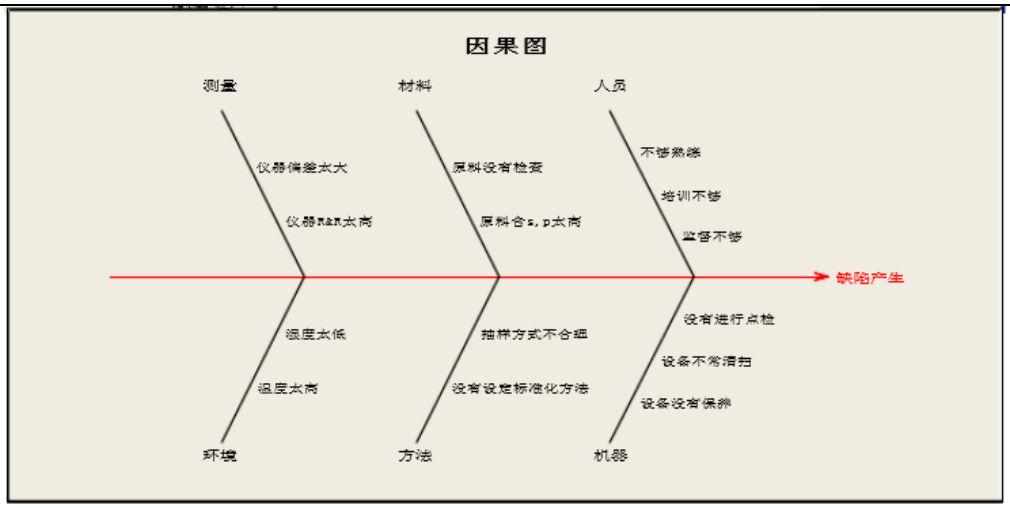
【图 2】

3) 对于每个 Cause,点击相应的子菜单，例如点击“人”的子菜单，出现如【图 3】，在此界面输入与人有关的因素



【图 3】

4) 对于 6 项均输入相关的原因后，点击 OK 就出现【图 4】界面

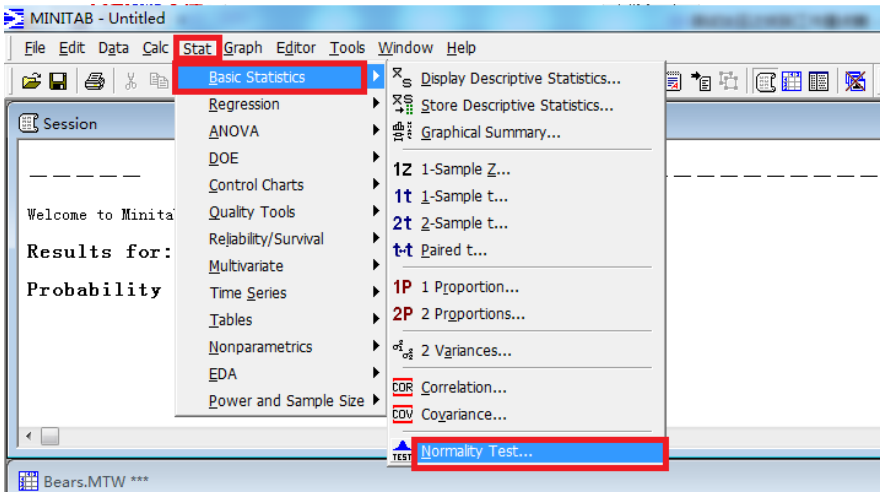


【图 4】

鱼骨图适用与大家开会，集思广益，找出导致问题产生的原因，鱼骨图只是一个帮助我们更有逻辑的分析的工具而已。一般鱼骨图用于问题分析的初级阶段，即从一堆比较杂乱无章的因子中，粗略找出比较重要的因子。

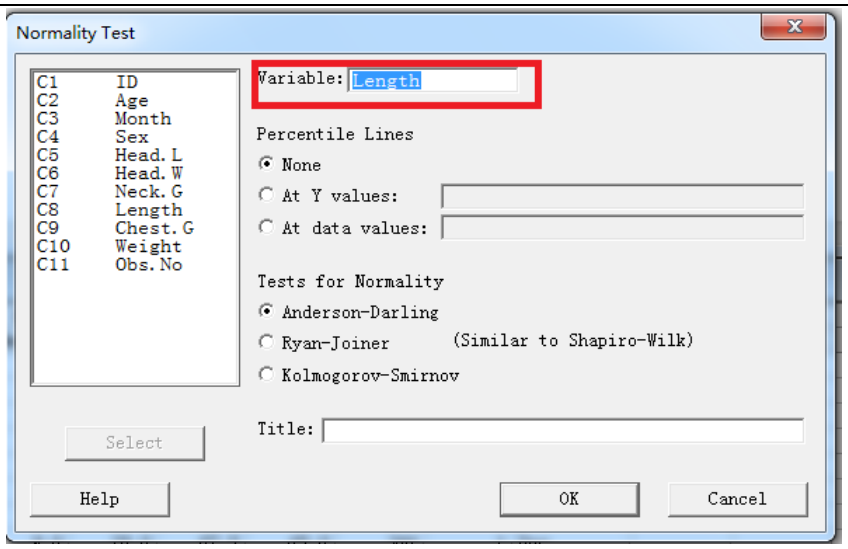
2. 正态分布

1) 进入统计(Stat)→基本统计(Basic Statistics)→正态测试(Normaly Test) 如【图 5】



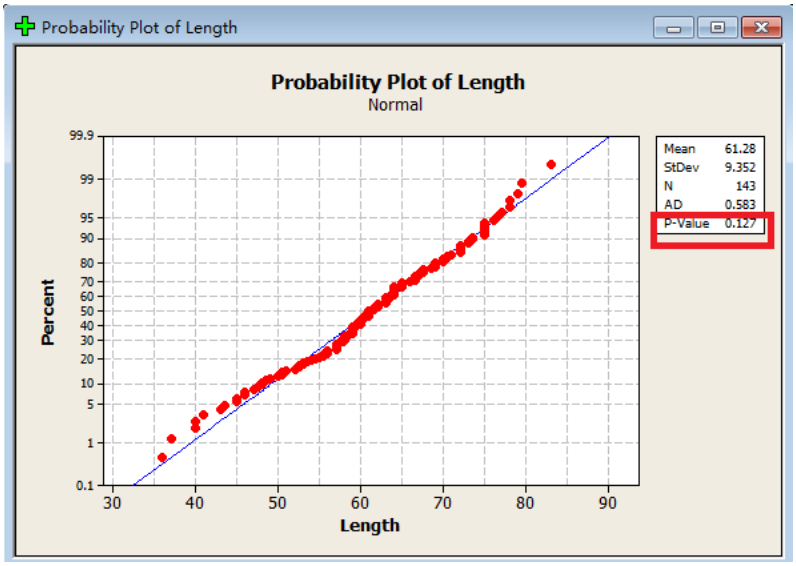
【图 5】

2) 出现如【图 6】的界面，在变量(Variable)界面，输入你希望检定的变量(是从左边的数据列中选择，不是手动输入哦)，点击 OK 按键。



【图 6】

3) 点击 OK 后出现【图 7】，可与看出大部分点都在蓝色的直线上，且 P-Value=0.127，注意：当 P>0.05 时说明此数据是符合正态分布。



【图 10】

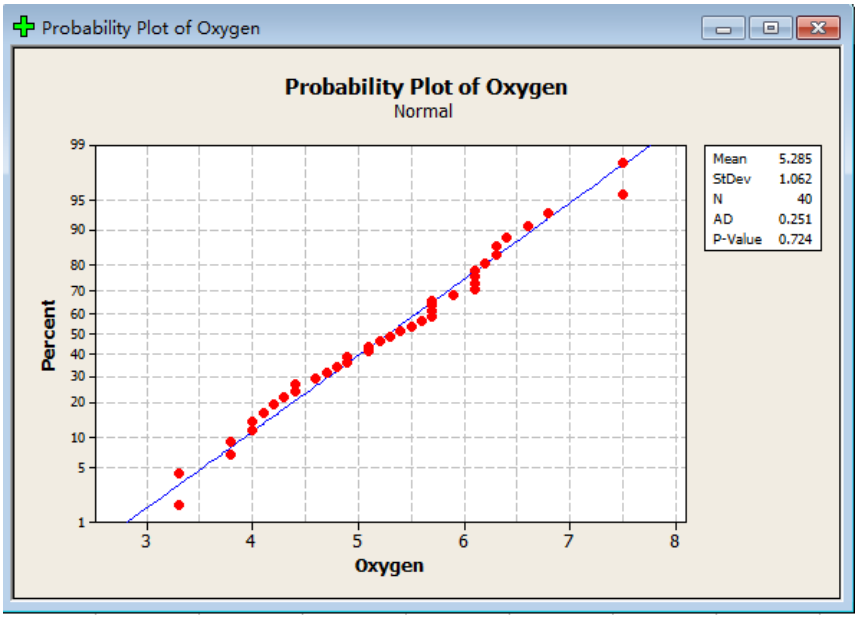
3. 单因素方差分析 One Way ANOVA

1) 例如我们需要研究氧气的含量与不同天是否有关系，如【图 11】

Day	Oxygen
1	5.6
1	5.9
1	6.8
1	7.5
1	4.7
1	4.0
1	4.4
2	6.6
2	4.9
2	5.5
2	4.9
2	6.3
2	4.2
2	3.3
2	4.8
3	6.1
3	5.3
3	5.2
3	4.3
3	4.0
3	6.2
3	5.1
3	3.3
4	5.4
4	6.1
4	5.7
4	6.1

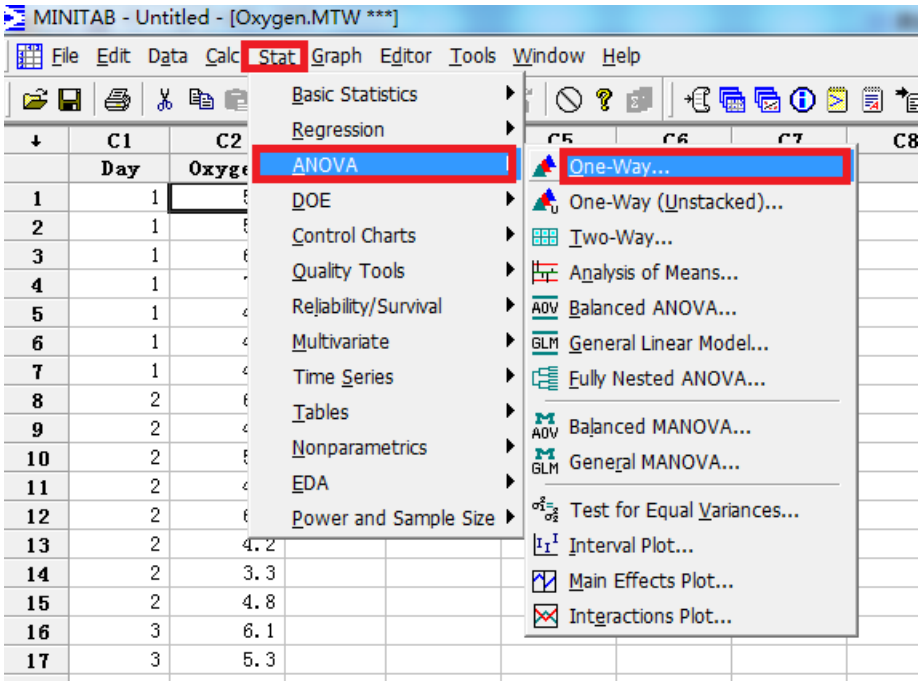
【图 11】

2) 首先明确 Y 是 Oxygen，X 是 Day,需要首先确认 Y 是否符合正态分布如【图 12】，结果是符合正态分布的，因为 P-Value>0.05



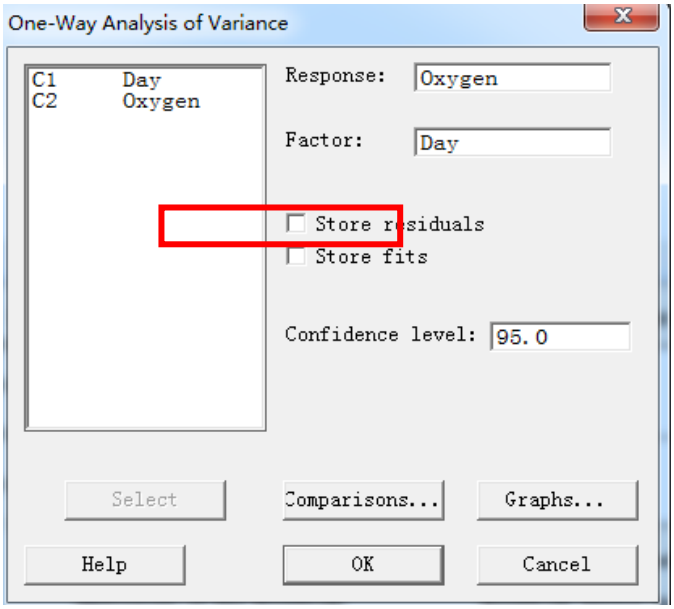
【图 12】

3) 进入统计(Stat)→变异分析(ANOVA)→One-Way 如【图 13】



【图 13】

4) 点击 Ok 后出现如【图 14】，Response 输入 Y，Factor 输入 X，如【图 14】



【图 14】

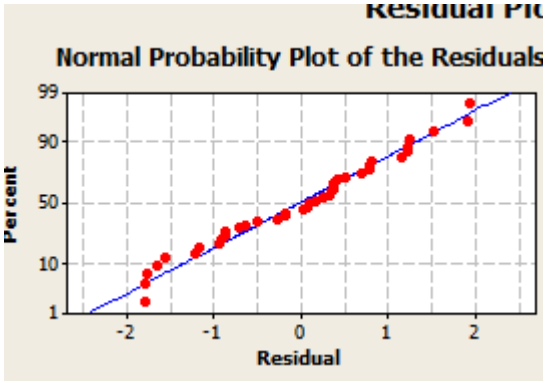
5) 点击 OK 后出现如【图 15】以及【图 16】

对于【图 15】，Day 的 P-value=0.698 大于 0.05，说明此 X 因子不是 Y 的显著因子。

对于【图 16】，是分析残值的，步骤如下

(1) 分析残值是否为正态分布（注意在做 ANOVA 分析时，需要勾选显示残值此选项，如【图 14】红框标出的选项）此测试结果对应残值分析的第一张小

图，如【图 17】



【图 17】

(2)分析残值的平均值是否为 0,此时需要对残值进行 1-sample T 检定，检定结果如【图 18】，它对应的结果为【图 19】

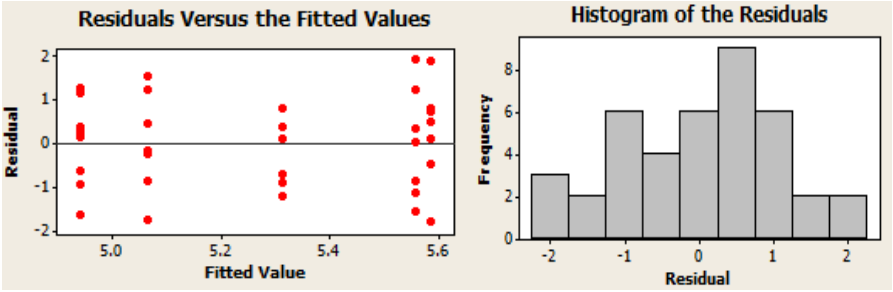
Residual Plots for Oxygen

One-Sample T: RESI1

Test of mu = 0 vs not = 0

Variable	N	Mean	StDev	SE Mean	95% CI	T	P
RESI1	40	0.000000	1.029965	0.162852	(-0.329399, 0.329399)	0.00	1.000

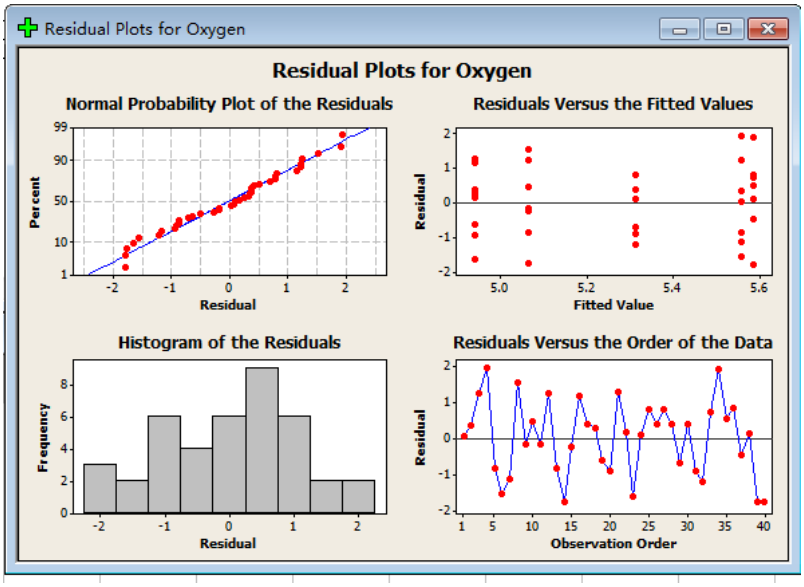
【图 18】



【图 19】

```
One-way ANOVA: Oxygen versus Day

Source  DF      SS    MS     F      P
Day      4    2.62  0.65  0.55  0.698
Error   35   41.37  1.18
Total   39   43.99

S = 1.087   R-Sq = 5.95%   R-Sq(adj) = 0.00%

                                Individual 95% CIs For Mean Based on
                                Pooled StDev
Level  N   Mean  StDev  -+---------+---------+---------+--------
1      7  5.557  1.287            (-------------*-------------)
2      8  5.063  1.073      (-------------*-------------)
3      8  4.938  1.010   (-------------*-------------)
4      9  5.311  0.751          (-------------*-------------)
5      8  5.588  1.294             (-------------*-------------)
                          -+---------+---------+---------+--------
                         4.20      4.80      5.40      6.00

Pooled StDev = 1.087

Residual Plots for Oxygen
```

【图 15】



【图 16】

4. 双因子变异方法分析 Two Way ANOVA

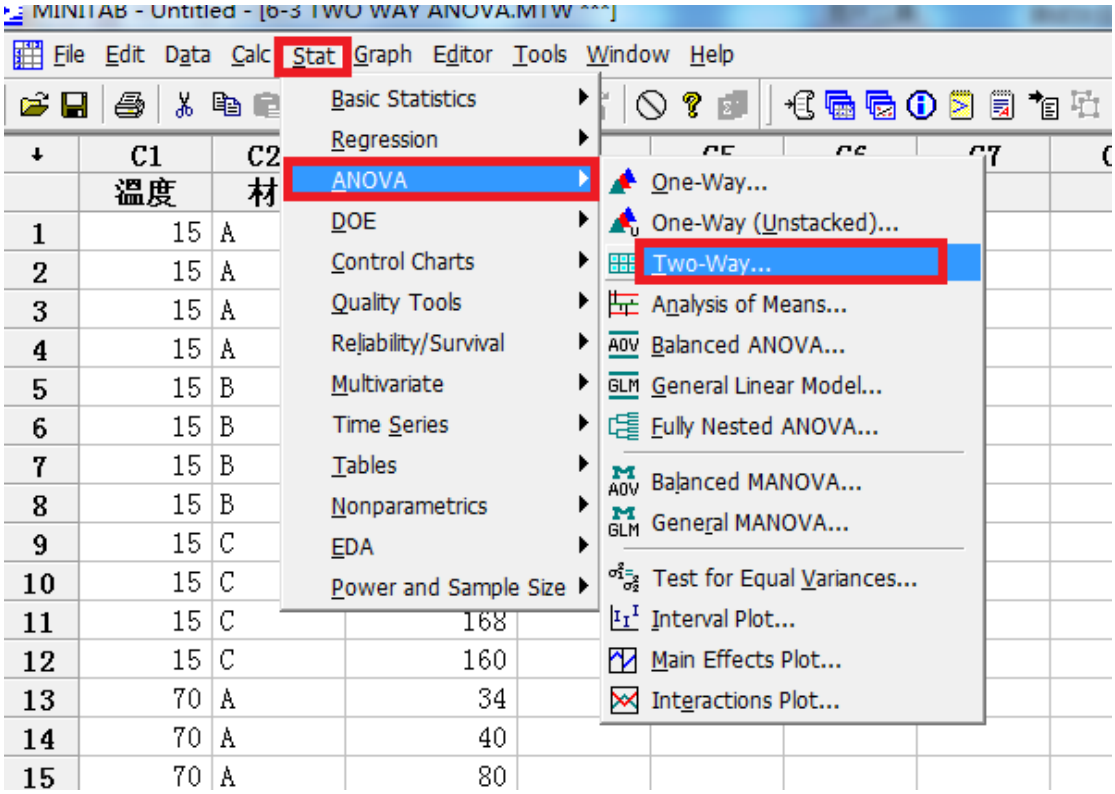
例如我们研究温度和材质会不会影响接收器的寿命

1) 搜集如【图 20】的资料

温度	材質	接收器壽命
15	A	130
15	A	74
15	A	155
15	A	180
15	B	150
15	B	188
15	B	159
15	B	126
15	C	138
15	C	110
15	C	168
15	C	160
70	A	34
70	A	40
70	A	80
70	A	70
70	B	136
70	B	122
70	B	106
70	B	115
70	C	174
70	C	120
70	C	150

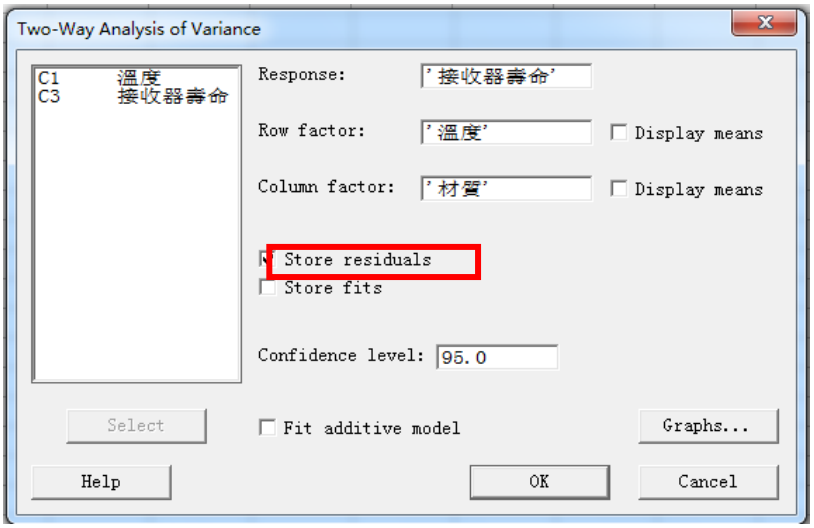
【图 20】

2) 选择“统计(Stat)→ANOVA（变异分析）→Two-Way”如【图 21】



【图 21】

3) 出现如【图 22.1】界面，注意一定要选择 Store residuals 此选项



【图 22.1】

4) 点击 OK 键即出现分析结果，如【图 22.2】以及残值分析如【图 23】：分析方法如 ONE WAY ANOVA，这里就不再赘述。

Residual Plots for 接收器壽命

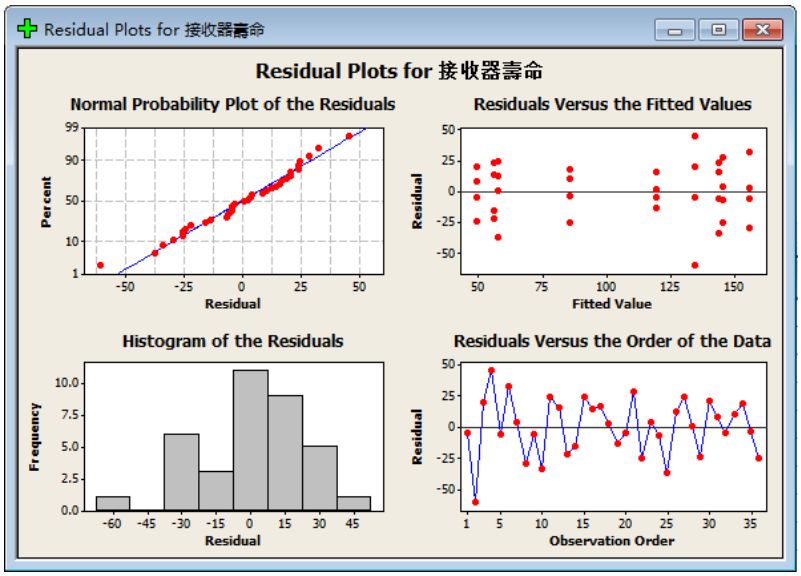
Two-way ANOVA: 接收器壽命 versus 温度, 材質

Source	DF	SS	MS	F	P
温度	2	39099.6	19549.8	29.21	0.000
材質	2	10908.7	5454.4	8.15	0.002
Interaction	4	9896.3	2474.1	3.70	0.016
Error	27	18072.0	669.3		
Total	35	77976.6			

S = 25.87 R-Sq = 76.82% R-Sq(adj) = 69.96%

Residual Plots for 接收器壽命

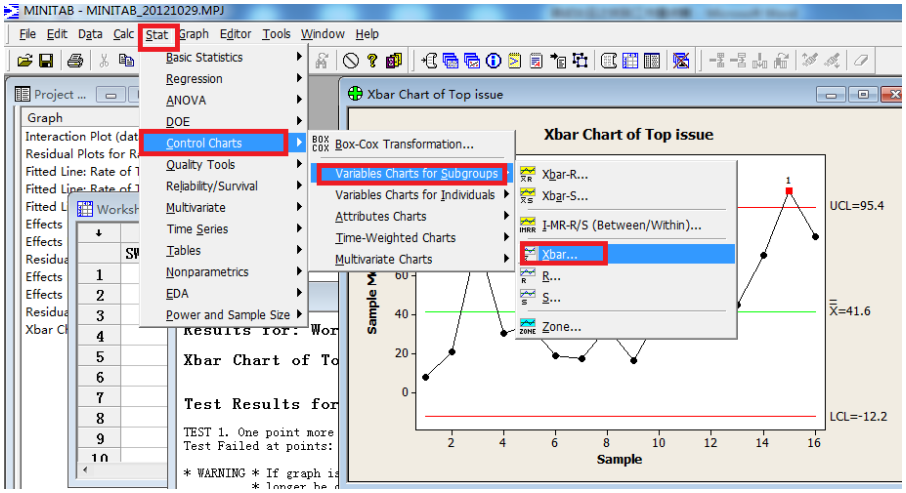
【图 22.2】



【图 23】

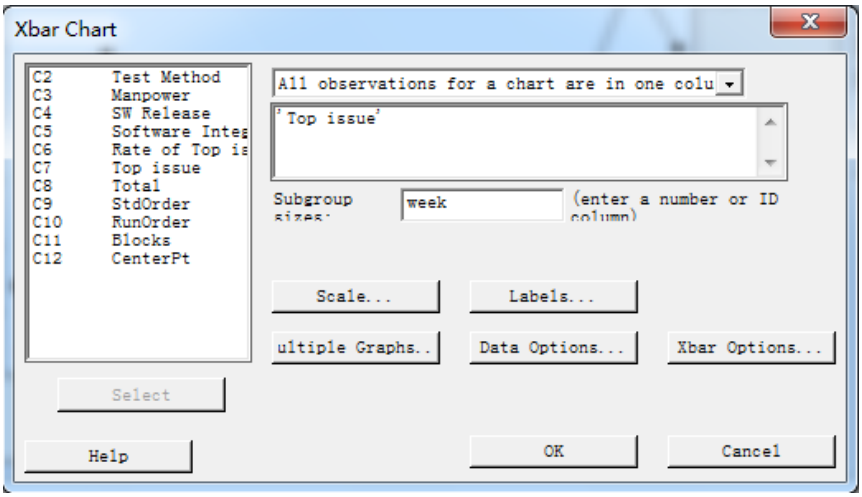
5. X-Bar

1) 选择统计(Stat)→控制图(Control Charts)→Xbar 如【图 24】



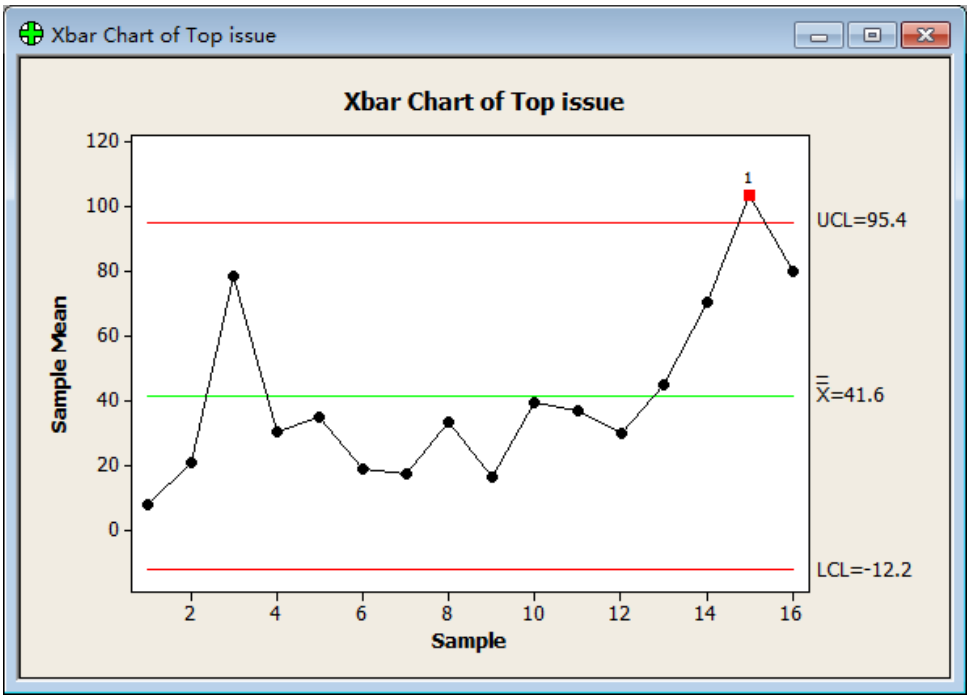
【图 24】

2) 将需要研究的数据选到如下【图 25】中



【图 25】

3) 点击 OK 后出现如【图 26】



【图 26】

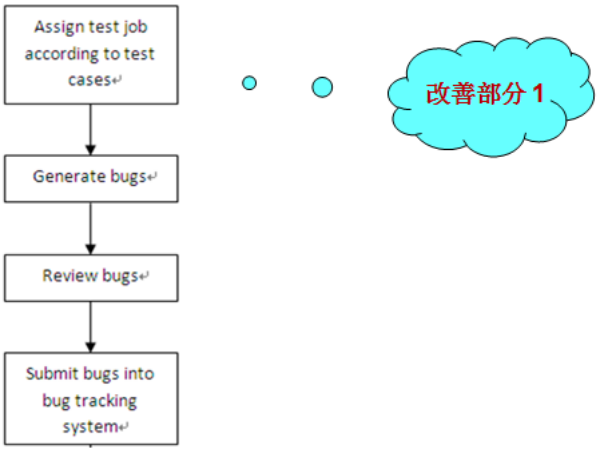
四、 应用到实际工作中_找到高效工作方法

1. 确定目标

在以往做的项目中，发现人力以及测试时间的不足一直客观存在，这就需要我们需要将有限的人力优先投入到正确的测试方向上，所以我们需要根据已有的数据研究严重 bug(我们定义为 Top issue)与什么有关系，进而帮助我们找到更为高效的工作方法

2. 分析现有流程

如【图 27】此流程为目前测试流程的描述



【图 27】

测试流程描述：

在安排测试任务时未有流程强制要求必须制定合适的测试策略，只是要求保证尽量全面的测试，缺乏测试的针对性。

3. 收集数据并筛选因子

1) 搜集 1 个项目的在 16 周发现的严重 bug 数量(Top issue)和所有 bug 的数量(Total)如【图 28】

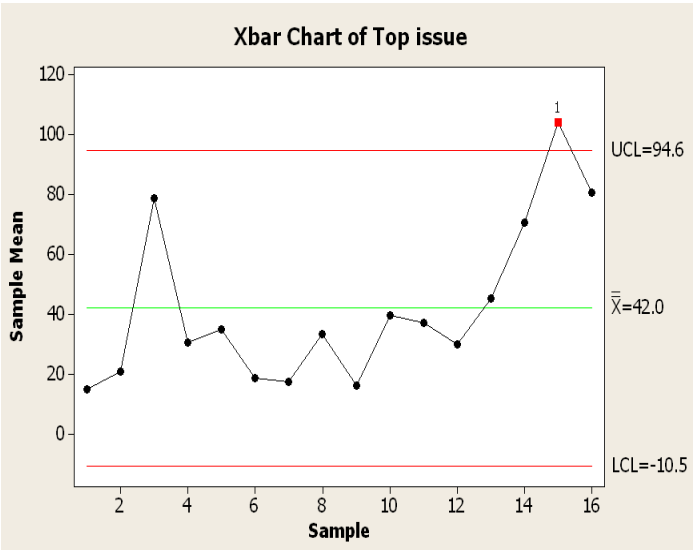
week	Top issue	Total
week39	15	53
week40	21	131
wk41	79	167
wk42	31	133
wk43	35	109
wk44	19	72
wk45	18	103
wk46	34	104
wk47	17	116
wk48	40	159
wk49	38	155
wk50	30	150
wk51	46	181
wk52	71	144
wk53	104	196
wk54	80	164

【图 28】

2) 对于 TOP Issue 查看其 X-Bar chart，如【图 29】可以看出 Top issue 在此 16 周的时间内并不稳定，甚至测试的时间越往后，Top issue 越多。

对于测试人员看到此 chart，首先考虑的就是：如此不稳定的 Top issue 到底是我们前面的漏测导致，还是因为软件品质导致的？

如此看来，对于 Top Issue 的研究，真的势在必行了！



【图 29】

3) 我们使用因果类型鱼骨图，并召集团队开会，根据鱼骨图的逻辑：

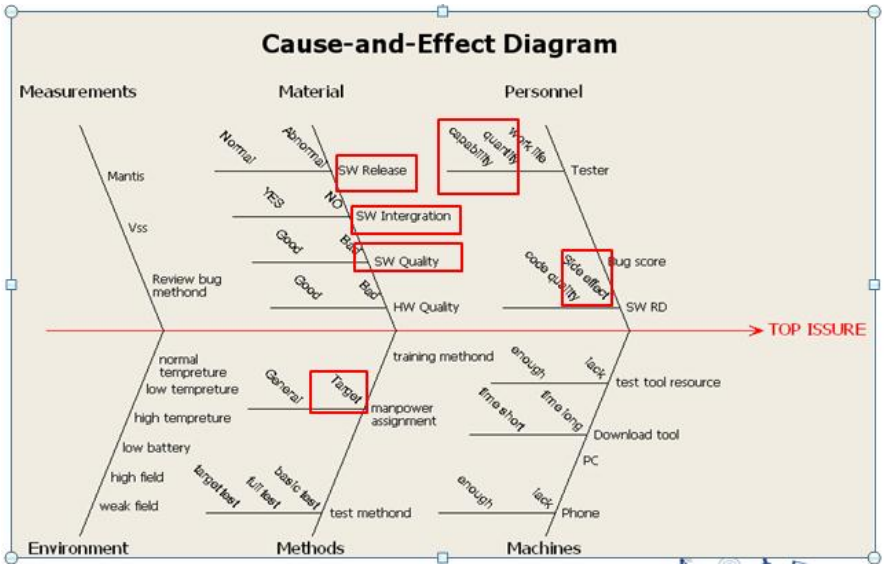
人，机，料，法，环，机，找出关键因子如【图 30】中红框标出的因子。

对于软件方面的主要因素： SW Release（软件版本 release 是否正常，最多一周发布一个版本视为正常，如多于一个版本视为异常）和 Software Integration (此软件版本是否为大改版)

对于 tester 的主要因素： capabily(测试人员的能力)，Quanty（测试人员的数量）

对于 SW RD 主要因素： side effect(更改 code 时，是否会引起其它的 bug)，code qualy（编写代码的品质）

对于 test method 的主要因素：Target test(有目标的测试)



【图 30】

4. 对于筛选出来的关键因子进行深入的研究

1) 根据上述筛选出来的关键因子对于一个项目搜集资料如【图 31】:

week	Test Method	Manpower	SW Release	Software Integration	Top issue	Total
week39	0	0	0	0	15	53
week40	0	0	0	0	21	131
wk41	0	0	1	0	79	167
wk42	0	0	0	0	31	133
wk43	0	0	0	0	35	109
wk44	0	0	0	0	19	72
wk45	0	0	0	0	18	103
wk46	1	1	0	0	34	104
wk47	1	1	1	0	17	116
wk48	1	1	1	0	40	159
wk49	1	1	1	0	38	155
wk50	1	1	0	0	30	150
wk51	1	1	0	1	46	181
wk52	1	1	1	1	71	144
wk53	1	1	1	1	104	196
wk54	1	1	0	1	80	164

【图 31】

说明：

Test method:1→target test　0→routine test

Software integration:0→此软件版本不是大改版

1→此软件版本是大改版

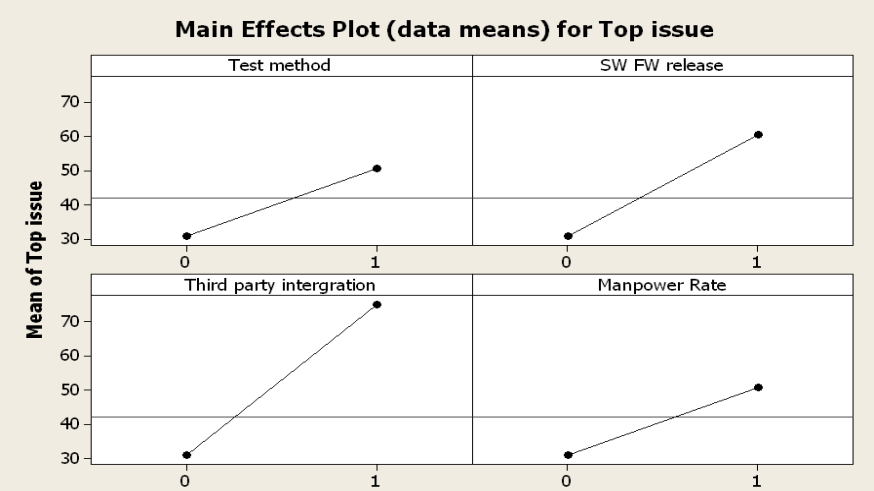
Manpower:0→人力少于 80 人/天　1→人力多于 80 人/天

Top issue→Crical　and Major bug 的个数

Total→本周发现的所有的 bug 的个数

SW Release:1→异常状态 release, 0→正常状态 release

2) 对于上述如 4 个原因针对 Top issue 做主效应图如【图 32】



【图 32】

说明：

Test method:有针对性的测试，会提高 Top issue 的产出率。

Manpower:大于 80 人天相对小于 80 人天，会提高 Top issue 的产出率 。

SW Release：在版本非正常 release 时，会提高 Top issue 的产出率。

Software integration:制定一些有针对性的测试策略会提高 Top issue 的产出率。

3) 对于【图 31】的数据做 ANOVA 分析如【图 36】

```
Results for: BUG SCORE.MTW

General Linear Model: Top issue versus Test method, SW FW release, ...

Factor                    Type   Levels  Values
Test method               fixed       2  0, 1
SW FW release             fixed       2  0, 1
Third party intergration  fixed       2  0, 1
Manpower Rate             fixed       2  0, 1


Analysis of Variance for Top issue, using Adjusted SS for Tests

                          Model
Source                       DF  Reduced DF     Seq SS
Test method                   1           1     1550.8
SW FW release                 1           1     2009.3
Third party intergration      1           1     4792.8
Manpower Rate                 1          0+        0.0
Error                        11          12     2511.5
Total                        15          15    10864.4

+ Rank deficiency due to empty cells, unbalanced nesting, collinearity, or an
    undeclared covariate. No storage of results or further analysis will be
    done.


S = 14.4669   R-Sq = 76.88%   R-Sq(adj) = 71.10%
```

【图 36】

根据 Sep SS 的分析得到：

影响最大因子依次是：

Software integration>SW release>Test method

Manpower 对 Top issue 几乎无影响

4) 根据 DOE 设计的试验进行实际复现 bug 得出结果如【图 38】

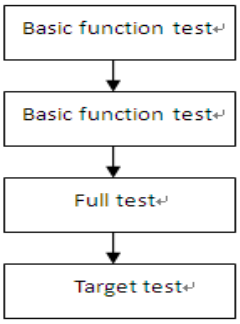
关键因素	问题来源
Test Method	为什么不是只要制定有针对性的测试就是一定影响 top issue 产出,是否是测试方法制定的有问题？
SW Release	1.如是异常状态 release(一周大于 2 个版本)，Top issue 会有较大的增长。 2.测试方法与 SW Release 有较大的相关性，即应该根据不同的 FW release 状况制定不同的测试方法。 此部分已经与 SW RD 进行了沟通，希望他们能尽量保证正常状态 Release
Human resource quanty	为什么它不是 Top issue 的关键因子?如何使用人力才能让 Top issue 更有效率的产出？
SW integration	1.如 code 有较大的改变,则 Top Issue 会爆发性的增长 2.SW Release 与 SW integration 有较明显的相关性，即

	如代码有重大改变，SW Release 的规律性就较容易打破。 此部分已经与 SW RD 进行沟通，如代码有重大改变，则代码的不可控性会大大增加

【图 38】

对于测试人员而言，能控制的就是 Test method(测试方法)和 Human resource quanty(测试人员的数量)，所以接下来我们就重点研究这两个因子。

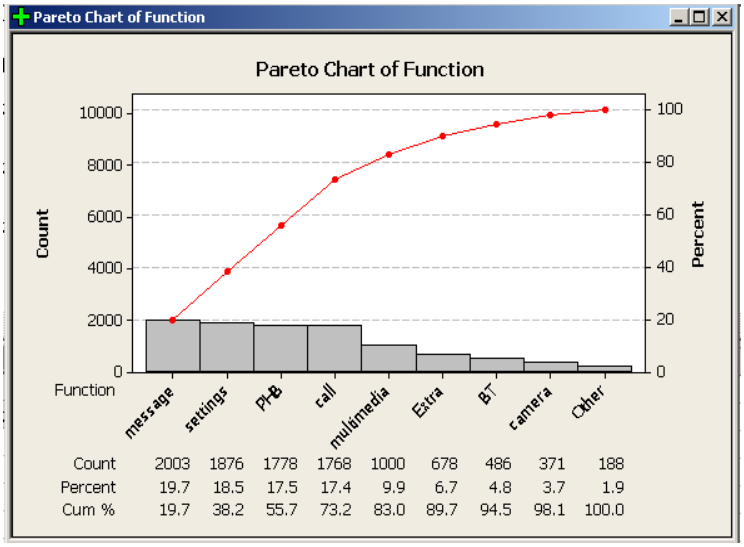
5) 首先研究 Test method 此方法，我们先看目前的测试方法如【图 39】



【图 39】

从目前的研究结果看此测试方法对于 TOP Issue 的产出没有很大的影响，要改变思路找出一个更有效率的测试方法。

6) 根据 5 个项目的研究找出 Top issue 的分布状况，深入研究 top issue 产生的原因使用柏拉图分析如下，如【图 40】



【图 40】

从 5 个项目分析：

80%的 Top issues 出自：Message, Setting, Call, PHB, Multimedia,

7) 对于 Top issue 的产生状况进行进一步的数据搜集如【图 41】

表格的内容说明：Top issue 数量处于前 5 名的 bug 分类

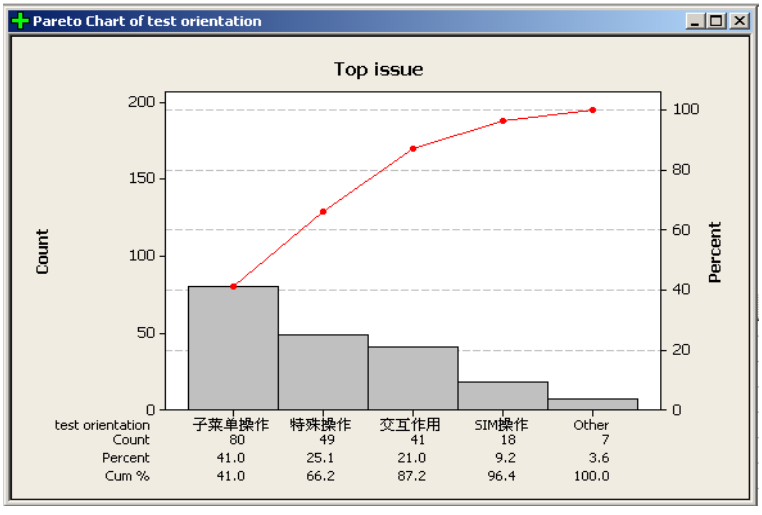
function	test orientation	the quantity of bug
Message	SIM操作	11
	子菜单操作	14
	特殊操作	12
	交互作用	11
Setting	子菜单操作	28
	特殊操作	15
Call	交互作用	14
	SIM操作	7
	特殊操作	9
	子菜单操作	6
	压力测试	4
PHB	交互作用	6
	特殊操作	8
	子菜单操作	18
	压力测试	3
Multimedia	交互作用	10
	特殊操作	5
	子菜单操作	14

【图 41】

8) 对于图【41】的数据进行柏拉图分析，结果如【图 42】

根据柏拉图的分析：

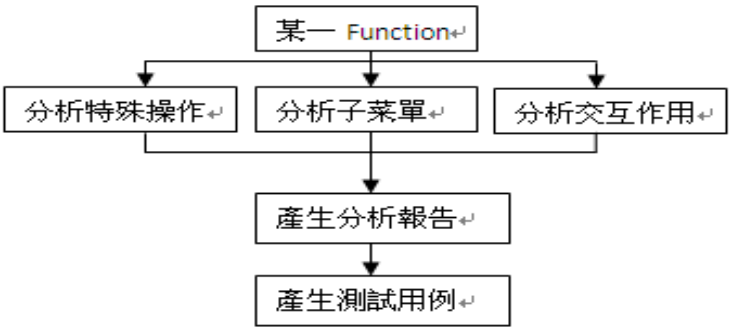
对于此 5 个 function,子菜单操作+特殊操作+交互作用,占所有 Top issue 的 80%以上



【图 42】

9) 根据 Top issue 细化分布状况，产生 test case 的流程：

即此 test case 是根据 Top issue 的分布状况产生，如【图 43】



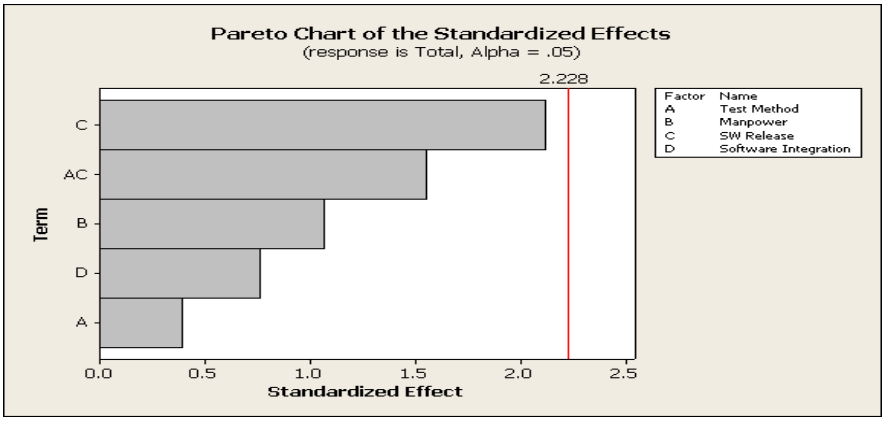
【图 43】

10) 从目前的研究结果如【图 38】看此人力的多少对于 TOP Issue 的产出没有很大的影响。 需要进一步研究

1.分析人力数量的多少，对我们工作有什么具体的影响

2.如何科学的使用人力才能提高 Top Issue 的产出（之前并未有人力如何分配的流程）

11) 对于图【31】中的 Total issue 与人力的关系做一个分析，如【图 44】



【图 41】

12) 进一步做 ANOVA 分析如图【42】

Analysis of Variance for Total, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
SW Release	1	4905.1	3654.8	3654.8	4.70	0.053
Manpower	1	7245.1	3180.3	3180.3	4.09	0.068
Test Method	1	14.1	126.1	126.1	0.16	0.695
SW Release*Test Method	1	1962.4	1962.4	1962.4	2.52	0.141
Error	11	8559.3	8559.3	778.1		
Total	15	22685.9				

Source	SS	Adj	%SS
SW Release	4905.1	3654.8	21.62%
Manpower	7245.1	3180.3	31.94%
Test Method	14.1	126.1	0.06%
SW Release*Test Met	1962.4	1962.4	8.65%
Error	8559.3	8550.3	37.73%

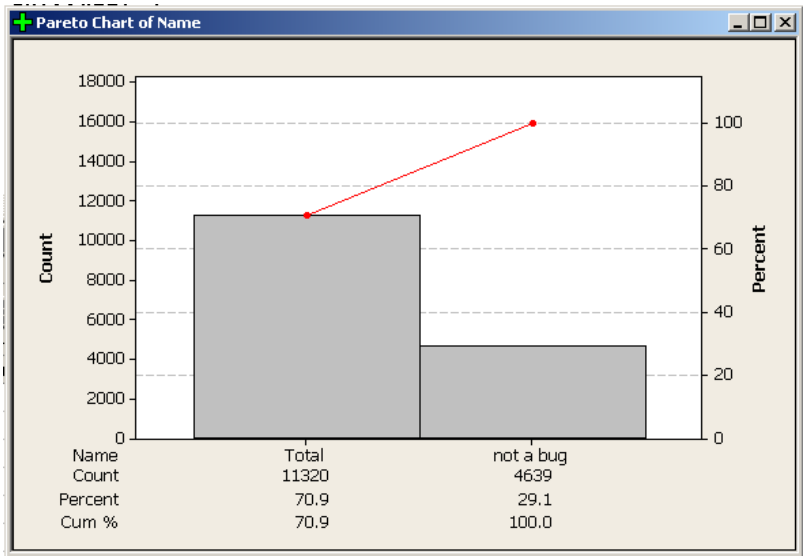
【图 42】

根据 P-Value 和%SS 的综合分析得到：

影响最大因子依次是 Manpower>SW release>SW Release*Test method>Test method。

从上述研究可以看出发现 bug 的总数与测试人员的数量有很大的关系：即测试人员越多发现 bug 的数量越多。

13) 对于为什么 Human resource 的数量为什么不是 Top issue 产出的关键因子却是 Total issue 产出的关键因子进行研究:首先研究 Approved bug 的状况， 使用柏拉图分析结果如【图 43】



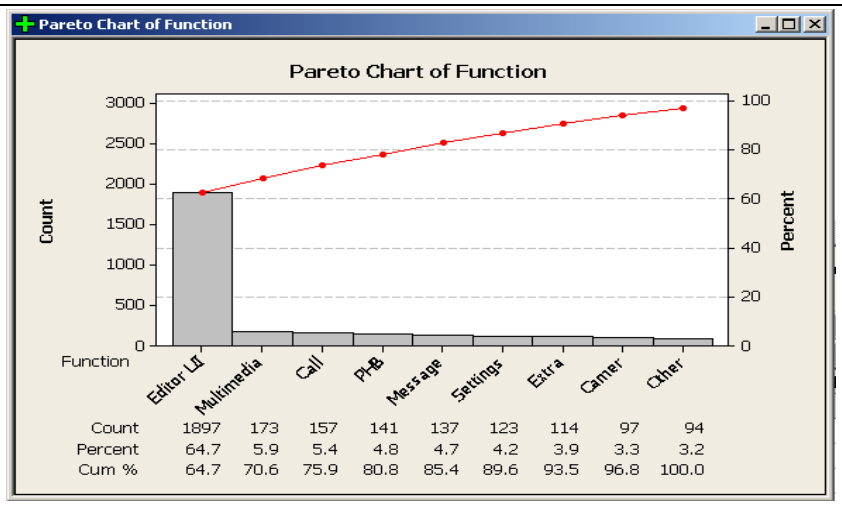
【图 43】

14) 对于无效 bug 的分布状况进行细化研究： Minor 且处于无效 bug 状态的资料搜集 如【图 44】：即 5 个项目的无效 bug 分布状况

Minor bugs	Call	message	PHB	Settings	File manage	Extra	Mutimedia	camera	BT	Editor&UI display	total
[redacted]	28	20	20	18	23	21	48	25	11	303	517
[redacted]	23	23	24	25	15	14	31	24	15	344	538
[redacted]	67	60	40	35	25	35	32	15	20	684	1057
[redacted]	14	15	27	23	12	22	32	20	10	302	477
[redacted]	25	18	30	20	27	16	30	28	14	261	469

【图 44】

15) 对于【图 44】资料进行柏拉图分析，结果如【图 45】



【图 45】

从以上 5 个项目柏拉图可以看出：无效 bug 主要集中在 Edor & UI display 即显示问题。

测试的策略应该引导测试人员： 应该尽量少关注显示和编辑方面的的问题。

16) 搜集每个测试人员对于一个项目的 bug 分数的数据，目的是可以产出更有效率的分配人力的方法搜集了 33 个 tester 在一个项目中的 bug 分数数据如【图 46】

Member	Average score	Staff	work life
[redacted]	4.81	1	2.00
[redacted]	2.03	1	2.00
[redacted]	1.53	1	2.00
[redacted]	2.88	1	2.00
[redacted]	1.27	1	1.00
[redacted]	1.36	1	1.00
[redacted]	1.28	1	1.00
[redacted]	0.89	1	0.00
[redacted]	0.25	1	1.00
[redacted]	0.56	1	0.00
[redacted]	0.14	0	0.00
[redacted]	0.42	0	0.00
[redacted]	0.09	0	0.00
[redacted]	0.93	0	0.00
[redacted]	0.14	0	0.00
[redacted]	0.30	0	0.00
[redacted]	0.06	0	0.00
[redacted]	0.86	0	0.00

Member	Average score	Staff	work life
[redacted]	0.34	0	0.00
[redacted]	0.60	0	0.00
[redacted]	0.96	0	0.00
[redacted]	0.25	0	0.00
[redacted]	0.26	0	0.00
[redacted]	0.76	0	0.00
[redacted]	0.17	0	0.00
[redacted]	0.19	0	0.00
[redacted]	0.49	0	0.00
[redacted]	0.08	0	0.00
[redacted]	0.08	0	0.00
[redacted]	0.10	0	0.00
[redacted]	0.30	0	0.00
[redacted]	0.30	0	0.00
[redacted]	0.29	0	0.00

【图 46】

对于 Average score 做一个简单的解释：我们对于每个项目每周都会统计发现 bug 的分数，即根据每个人发现 bug，根据 bug 的严重程度分配一个权

重分数，例如 Blocker 最严重分数为 8，Enhancement 最轻微分数为 0.2。如果 Blocker 的 bug 本周发现 2 个，则 bug score=8*2。如【图 47】

其它参数说明：

Staff:1→正式员工　　0→外包人员

Work life: 2→工作时间大于 3 年

1→工作时间大于 1 年

0→工作时间小于 1 年

Member	Blocker (8)	Critical (6)	Major (5)	Normal (3)	Minor (2)	Trivial (0.5)	Enhancement (0.2)	Weekly Scroe	Testing Day	Ratio By Average
[redacted]	0	0	0	1	3	0	0	4.5	1	2.2755
[redacted]	0	0	2	6	1	0	0	15	3.5	2.1671
[redacted]	1	0	1	4.5	0	0	0	13.25	4	1.675
[redacted]	0	0	1.5	5	0	0	0	11.25	2	2.8443
[redacted]	0	0	0	5.5	0	0	0	8.25	4	1.0429
[redacted]	0	0	0	0	0	0	0	0	3	0
[redacted]	0	0	1.5	2	0	1	0	7	4	0.8849
[redacted]	0	1	0	1	1	0	0	5.5	3	0.927
[redacted]	0	0	0	1	0	0	0	1.5	3	0.2528
[redacted]	0	0	0	0	0	0	0	0	3	0
[redacted]	0	0	0	0	0	0	0	0	3	0
								0		#DIV/0!
Total	1	1	6	26	5	1	0	66.25	33.5	

【图 47】

17) 对于 staff 和 work life 对于 bug 分数的影响，运用了主效应图，如【图 48】

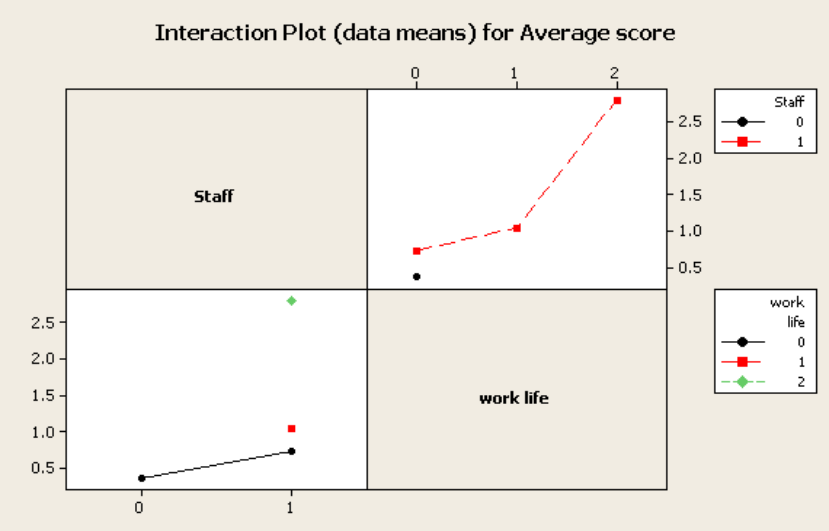
Staff: 1.正式员工的贡献度远高于的外包人员贡献度

2.工作时间长的员工的贡献度要远高于工作年限低的员工的贡献度。

【图 48】

18) Staff 和 Work life 的交互作用研究如【图 49】

work life 和 staff 交互作用不明显



【图 49】

19) 使用 One way ANOVA 分别分析 Staff 和 work life 对于 bug score 的影响，如【图 50】

Staff: 1、staff 对于 bug score 来说是关键因子

2、正式员工 bug score:在 1.2→2.2 之间

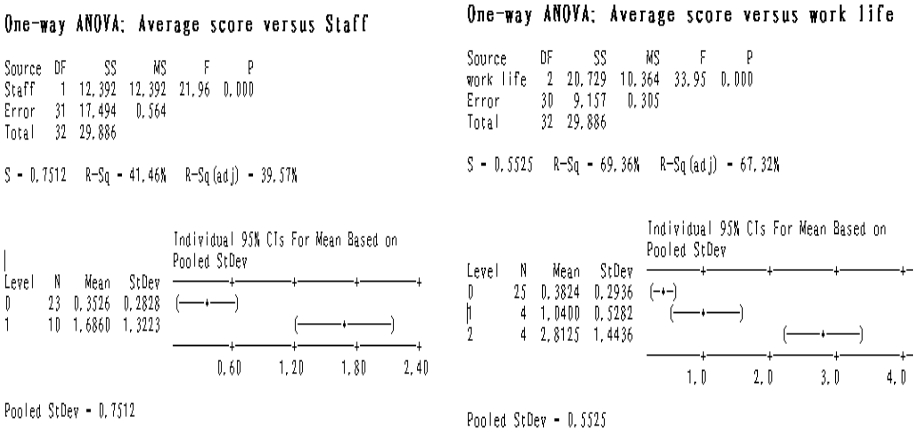
外包人员 bug score 0→0.6 之间

Work life: 1、work life 无交互作用对于 bug score 来说是关键因子

2、工作年限小于 1 年:bug score 在 0→0.5 之间

1 年<工作年限<3 年 bug score 在 0.5→1.6 之间

工作年限>3 年 bug score 在 2.2→3.5 之间



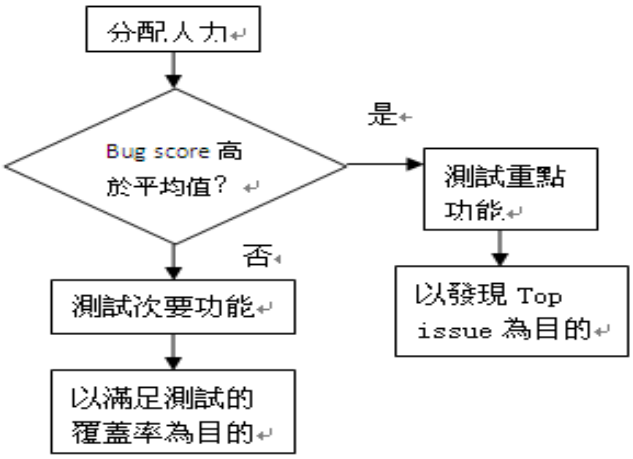
【图 50】

20) 根据上述研究的结果得到流程图如【图 51】

人力分配的流程： 优先将资深人力分配到 5 大 Function 的子菜单操

作，特殊操作，交互作用方面的测试

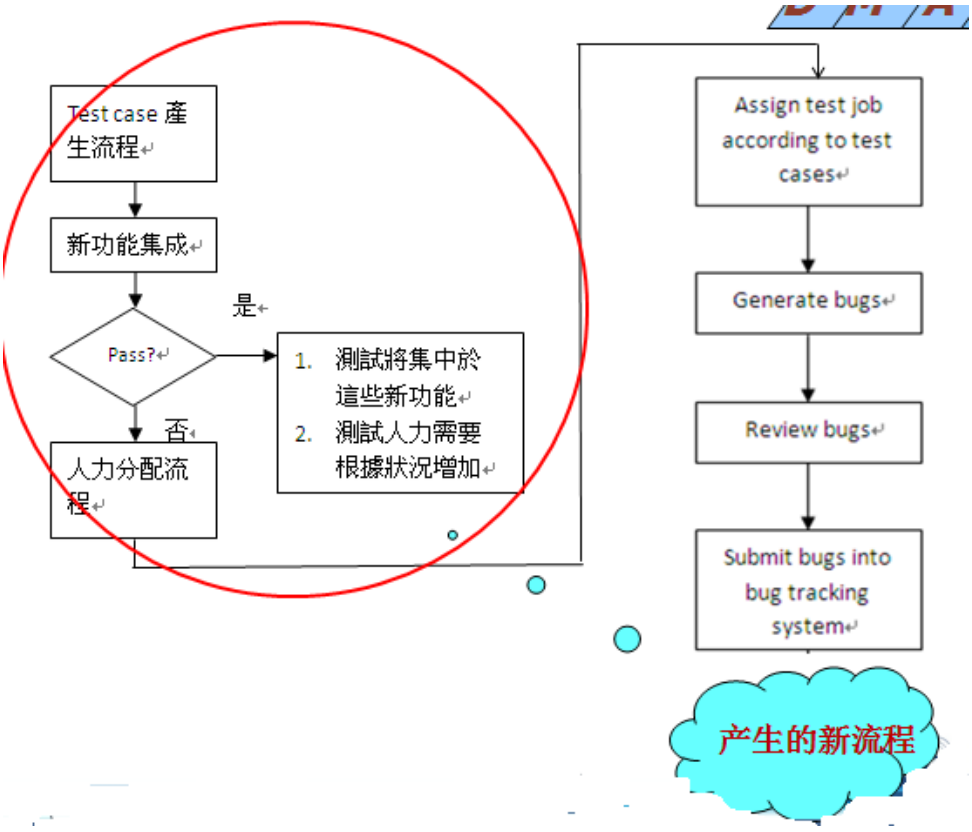
之前的流程：安排人力并未考虑 bug 的分数



【图 51】

5. 产生的最终流程

如【图 55】



【图 55】

1) 人力分配流程， 请查看【图 51】
2) Test case 产生流程，请查看【图 43】

五、 总结

我们再一次给出了一个典型的例子来说明统计学如何应用到实际工作中，如果通过多年累积的工作经验以及数据，来改善将来的工作，这是一件非常有意义的工作，我们经常讲我们有多少年的工作经验，如果我们把这些经验能够以数据的形式总结出来，并使用 6 sigma 此专业工具进行分析并得出结论，那这个经验就是我们的最可贵的财富了

所以在工作中，注意统计数据，并学习专业的统计工具对我们的职场增值有非常重要的意义！

总结一下这次学的理论和工具：其实是可以分成三大类：筛选因子工具，ANOVA 的分析工具以及过程监控工具。筛选因子工具分别为：“因果型的鱼骨图”，此工具主要针对在初期，我们对于比较多的，杂乱无章的因子进行初步的筛选，它只是给我们一共一个逻辑思考的模板，在此模板上，充分发挥大家的力量，开会讨论出一些重要因子。我们还用到了 X-Bar 此工具（即过程监控工具），此工具主要监控某一个因子在一段时间内，是否处于稳定的状态。接下来，我们用到的是分析工具：ANOVA。对于此工具主要是先假定因子对 result 没有影响，然后根据分析结果，来判断我们的假设是否正确，如不正确，说明此因子是关键因子。

事实上 6 sigma 还有很多其它的工具还未介绍，后续我就打算按照这样的流程：根据我们软件测试所需要研究的问题为出发点，再此过程中遇到需要使用的工具和相关理论，对于新的工具和理论，就一一进行介绍。

参考文献：

有关 ANOVA 基本概念介绍的网站：http://wiki.mbalib.com/zh-tw/

有关“X-Bar”的介绍的网站：http://tech.casd.cn/wzym/0157/g10157/g1sxn930.htm

基于 iOS 平台的 GUI 程序设计

作者：王佩多

摘要：2010 年，由我国研制的天河-1A 超级计算机系统获得 TOP500 第一名。超级计算机系统庞大，设计复杂。为使得系统监控更加便捷，开发直观、可操作性强的监控系统是必要的。

随着 iPhone 与 iPad 等设备的普及，iOS 平台以其简洁美观的界面和方便的操控方式被越来越多的用户接受。iOS 的用户界面的概念基础是多点触控（Multi-Touch），即采用手指对显示屏直接操作，因为其优雅直观的界面设计，降低了用户初次使用所需要的学习周期。iOS 可以直接与硬件底层通信，效率要高于 Android 等其它嵌入式操作系统，运行软件更加流畅。它的优化程度和用户体验受到用户的广泛认同。

iPad 平台上的远程监控系统的优点在于可以把信息直观的显示在用户界面上，并且操作方便，人机交互友好。所以用 iOS 编写程序实现可视化监控，可以为天河系统提供一个方便的状态查询诊断工具。可以让客户更加迅捷的了解整个计算集群的实时状态。从而更好地为天河工程服务。

本课题从实际应用需求出发，以人机交互原理为指导，结合面向对象的设计模式，在掌握基于 Xcode IDE 开发环境以及 iPhone SDK 的程序设计方法以及 Cocoa 框架、Objective-C 语言等 iOS 开发设计工具的前提下，完成一个基于 iOS 的远程 3D 监控可视化系统的界面设计。该通信系统具有对错误列表的维护、诊断信息的发送与接收、机柜状态确认等功能。

本系统具有以下特点：1、采用 Cocoa Touch 框架在设计 GUI 界面实例，使系统美观、实用，人机交互更为出色。2、运用面向对象的设计模式、Objective C 语言和 iOS 平台进行编写，保证系统的可靠性和稳定性。3、系统运行在 iPad 手持设备上运行，移动方便，具有高效性。

关键词：iOS；iPad；界面设计；远程监控系统

第 1 章　绪　论

1.1　研究背景

当今社会信息化发展迅速，采用可视化监控较大的工程，已经成为了时代的潮流。由于可视化监控的清晰明了，降低了用户的使用门槛，越来越多的用户提出了对监控可视化的需求。

作为苹果公司推出的嵌入式操作系统，iOS（原名：iPhone OS）的用户界面的概念基础是多点触控（Multi-Touch），即采用手指对显示屏直接操作。随着

iPhone 与 iPad 设备的普及，iOS 平台以其简洁美观的界面和方便的操控方式被越来越多的用户接受。

本课题基于以上考虑，在 Xcode 环境下，采用 iPhone SDK 完成一个基于 Cocoa Touch 的远程监视系统，在 iOS 设备上实现可视化监控。通过本课题的程序设计，可以为天河系统提供一个直观简单的状态查询诊断工具，可以让客户更加迅捷的了解整个计算集群的实时状态。从而更好地为天河工程服务。

1.1.1 iOS 介绍

iOS 是由苹果公司以开源的类 Unix（Unix-like）操作系统 Darwin 为基础开发的操作系统，支持的设备包括 iPhone、iPod touch、iPad、Apple TV。苹果公司最早于 2007 年 1 月 9 日由美国国际数据集团创办，专门面向苹果 Macintosh 平台的 Macworld 大会上公布这个系统，最初是设计给 iPhone 使用的，后来陆续套用到 iPod touch、iPad 以及 Apple TV 等苹果产品上。[1]

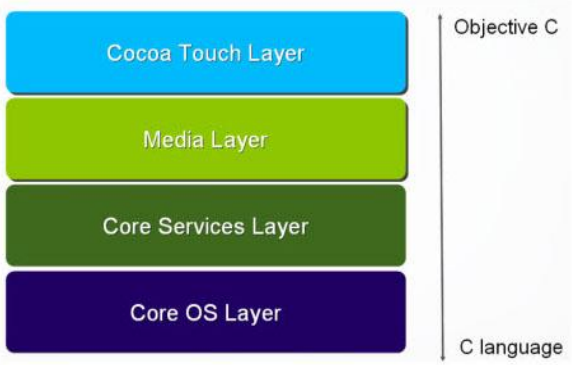


图 1-1　iOS 系统结构

如图 1-1 所示，iOS 的系统结构分为以下四个层次：核心操作系统（the Core OS layer），核心服务层（the Core Services layer），媒体层（the Media layer），Cocoa 触摸层（the Cocoa Touch layer）。

最底层是核心操作系统层，它包括内存管理、文件系统、电源管理以及一些其他的系统任务，它可以直接和硬件设备进行交互。核心操作系统层的主要任务是管理系统的虚拟内存，线程，网络。这一层包括驱动程序和 iOS 的基本接口。在此层的驱动程序还提供了硬件和系统框架，硬件接口。

第二层是核心服务层，通过它来访问 iOS 服务，这些服务包括访问文件，低级别数据类型等。

第三层是媒体层，通过它可以在应用程序中使用各种媒体文件，进行音频与视频的录制，图形的绘制，以及制作基础的动画效果。

最上层是 Cocoa 触摸层，这一层为应用程序开发提供了各种有关用户界面的框架，负责用户在 iOS 触摸交互操作。Cocoa 触摸层由多个框架构成，他们为应用程序提供核心功能。这些框架包括：

1) UIK

UIK 提供了大量的功能。它负责启动和结束应用程序、控制界面和多点触摸时间，并能够访问常见的数据视图。它还负责 iOS 内部众多继承功能。访问多媒体库，照片库和加速计也是使用 UIK 中的类和方法来实现的。

2) Map K

Map K 框架允许开发人员在应用程序中添加 Google 地图视图，这包括标注、定位和事件处理功能。

3) Game K

Game K 框架进一步提高了 iOS 应用程序的网络交互性。它提供了创建并使用对等网络的机制，这包括会话、仲裁和语音聊天。

4) Message UI/Address Book UI/Event K UI

框架 Message UI、Address Book UI 和 Event K UI 允许应用程序访问电子邮件、联系人和日历事件。

5) iAd

iAd 框架支持在应用程序中加入广告。iAd 是交互式广告组件，使用简单的拖放操作就可将其加入软件中。

iOS 界面具有以下四种特点：

1) 设计特点

iOS 用户界面通过轻触屏幕对设备进行控制，同时支持按键以及多点触控，此外透过其内建的加速器，可以旋转装置以使屏幕改变方向，这样的设计使设备更便于使用。

2) 实体按键

包括屏幕下方的 Home 按键（用于退出应用程序、回到主界面、长按开启语音控制），顶部的 Power 按键（用于锁定屏幕、关机），侧面的音量控制按键。

3) 多点触控

包括滑动（Swiping）、轻按（Tapping）、挤压（Pinching）、反向挤压（Reverse Pinching）。例如向左向右滑动时，屏幕显示上一张或下一张视图；当两个手指同时向相反方向开合时，手机屏幕上显示的视图可以放大或缩小。苹果公司为 iOS 系统设置了一系列手势，开发者也可以为自己的应用程序自定义各种手势。

4) 屏幕界面

以应用程序方格的形式呈现，最底部一栏为 Dock，可以有最多四个程序图标被固定在 Dock 上，其他可以通过“滑动”的方式进行变换显示的应用程序。状态栏处于屏幕上方，能显示时间、电池电量和讯号强度等信息。[2]

下表为 iOS 与 Android，windows phone 等主流嵌入式操作系统详细对比：

表 1-1　主流嵌入式操作系统对比

	iOS	Android	Windows phone
系统内核	Hybrid（XNU）	宏内核（Linux 核心）	Windows CE 7.0
编程语言	C，Objective-C	C（核心），Java（用户界面）	C#，Visual Basic
开发者	Apple Inc.	Google ，Open Handset Alliance	微软公司
最新版本	iOS6	4.2.2（Jelly Bean）	Windows Phone 8
源码模式	不开源，含有开源组件	自由及开源软件	封闭式系统
发行时间	2007.7.29	2008.10.21	2010.10
支持平台	ARM	ARM、MIPS,Power Archecture、X86	ARM 及其他

图 1-2　主流嵌入式操作系统界面对比（左：iOS，中：android，右：windows phone 8）

与这两种嵌入式操作系统界面相比，iOS 通过采用同样尺寸和样式的图标整齐平铺在桌面，简约而方便用户使用。相比下 Android 屏幕下方为 3 个虚拟按键，界面略显简单。

1.1.2 Objective-C 介绍

Objective-C 是 ANSI C 的一个超集，由 Brand Jacob 设计，在 C 语言的基础上增加了面向对象的特性，并将 C 语言构造与源自 Smalltalk-80 对象和消息传递的理念结合在一起，创建了他的新语言。Objective-C 允许开发人员继续使用熟悉的 C 语言开发风格，同时在访问该语言中基于对象的特性，在 20 世纪 80 年代末期。Steve Jobs 刚起步的计算机公司 NeXT 使用 Objective-C 作为 NeXTStep 操作系统的主要开发语言。Objective-C 目前的版本是 2.0。

Objective-C 对面向对象编程的支持填补了标准 C 语言的空白。对象是指与一个预设的函数调用列表相关的数掘结构。Objective-C 中每个对象都具有实例变量和方法，前者是数据结构的字段，后者是对象可以执行的函数调用。面向对象的代码使用这些对象和方法引入了编程抽象，从而提高了代码的可读性和可靠性。面向对象的编程支持构建可重用代码单元，这些代码单元可以与过程开发的常规流程分离，Cocoa Touch 和 Mac OS X 上的 Cocoa,，提供这些智能对象的库，使开发人员能够用最少的工作量和代码创建出高效的应用程序。

Objective-C 语言虽然实现了对 C 语言的支持，但是与 C++仍有如下几点区别：

1) C++类不能从 Objective-C 类继承。
2) Objective-C 定义内部不能定义 C++命名空间。
3) Objective-C 类的成员变量不能包括不含默认构造函数或含有虚方法的 C++类对象，但使用 C++类指针并无此限制。
4) C++“传递值”的特性不能用在 Objective-C 对象上，而只能传递其指针。
5) Objective-C 声明不能存在于 C++模板声明中。[3]

1.1.3 Cocoa Touch 介绍

Cocoa Touch 是由苹果公司提供的软件开发 API，用于开发 iPhone、iPod、iPad 上的软件。也是苹果公司针对 iPhone 应用程序快速开发提供的一个类库。此库以一系列框架库的形式存在，支持开发人员使用用户界面元素构建图像化的事件驱动的应用程序。

Cocoa Touch 框架专注于基于触摸的开发接口和性能优化。其中，UIK 提供了开发 iOS 上的图形化事件驱动程序所需的基本工具。UIK 基于 Foundation 框架，该框架同样存在于 Mac OS X 系统中，提供了文件处理、网络、字符串处理以及其他基础架构。通过 UIK，可以访问 iOS 上特殊的 GUI 控制、按钮和全屏

幕视图。还可以通过加速计和 Multi-Touch 手势控制应用程序。

UIK 用于为 iOS 应用程序提供界面对象和控制器。UIK 采用事件（通过 UIEvent 类实现）机制。另外，iOS 上的应用程序都可以看作一个 UIApplication 实例。表 1-2 为 UIK 上的一些常用对象类。

除了 UIK 之外，Cocoa Touch 包含创建 iOS 应用程序所需的所有框架，从 3D 图形、专业音频到网络，甚至提供特殊设备访问 API 以控制摄像机或从 GPS 硬件获取位置。Cocoa Touch 既包含强大的 Objective-C 框架，也在需要时提供基础的 C 语言 API 来直接访问系统。这些框架示例包括：

1) Core Animation

Core Animation，可以通过基于组合独立图层的简单编程模型来创建丰富的用户体验。

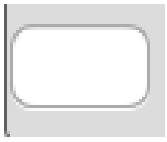
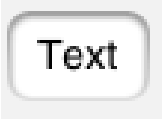
2) Core Audio

Core Audio 是播放、处理和录制音频的专业级技术，能够轻松为应用程序添加强大的音频功能。

3) Core Data

Core Data 提供面向对象的数据管理解决方案，该方案易于使用和理解，甚至可处理任何应用或大或小的数据模型。

表 1-2 UIKit 常用类举例

类名	图标	功能
UIButton		按钮，可以设置按钮上的文字、图像等属性，这个对象侦听触摸事件。当用户触摸按钮时，这个对象调用时间关联的目标对象上的某一个方法。
UILabel		标签，用于显示不可以更改的文本，并会随着文本的大小改变自身的大小。
UIDatePicker		日期选择器，显示一个多栏旋转的轮子，用于让用户选择日期和时间。
UextField		文本输入框。用户单击这个输入框时，键盘出现，从而用户输入文本。当用户单击回车键时，键盘消失。
UableView		表示图，可以按照多种风格（如 plain、sectioned 和 grouped 风格）来显示数据。

UIScrollView		滚动视图，提供一种机制显示比应用程序窗口更多的内容。
UIView		视图，是窗口（或父视图）上的一个矩形区域，用于显示 UI 对象和接受事件。
UISearchBar		搜索栏，它显示一个可以编辑的搜索栏，其中包括搜索图标。用户单击这个搜索栏时，键盘出现，从而用户输入文本，当用户单击回车时，键盘消失。

1.1.4 iOS 开发集成环境介绍

iOS 的开发工具为由 Apple 公司开发的 iOS 软件开发工具包（iPhone SDK）。首个版本 iPhone SDK1.2b1 于 2008 年 2 月发布。软件开发工具包需要在 Mac OS X Leopard（v10.5）及以上系统并拥有英特尔处理器才能运行。开发组件主要由 Xcode，iPhone SDK， Interface Builder，Simulator，Instruments 等等组成。

Xcode 前身继承自 NeXT 的 Project Builder。它可以创建由 C、C++、Objective-C 和 Java 编写的源代码组成的工程，可以生成 Mac OS X 支持的所有类型的执行代码，包括命令行工具、框架、插件、内核扩展、程序包、和应用程序。

Interface Builder（IB）提供了快速的原型工具，用于以图形化的方式布局用户界面以及从 Xcode 源代码链接到这些预构建的界面。借助 IB，可以使用可视设计工具绘制界面，然后将这些屏幕元素连接到应用程序中的对象或方法上。

iPad 的模拟器（Simulator）运行于 Mac OS 系统之上。开发者无需连接到实际的 iPhone、iPad 等手持设备，即可在 Mac OS 上测试应用程序。因为模拟器提供的 API 和 iPhone 上的 API 相同。真机测试与模拟器测试毕竟存在许多不同之处。模拟器存在一定的局限性。比如由于手持设备的局限性，iPad 的内存远小于台式机的内存。因此在模拟器上运行流畅的程序，在真机上运行时可能产生意想不到的状况。此外模拟器不支持摄像头或加速器等硬件特性，所以与摄像头相关的工作，必须在真机上进行运行测试。模拟器也不支持触摸输入和振动，所以测试时使用鼠标点击模拟用户的触摸输入。

通过 Instruments，开发者可以观测到内存管理等问题。Instruments 可以对内存的使用进行采样并监视其性能。开发者还可以根据 Instruments 观察到应用程序中占有资源最多的地方，进而优化程序设计。[4]

1.2 课题的研究内容及意义

从计算机的出现到现在计算机应用的普及，图形用户界面从只有 x-y 点图像显示到今天 Windows 等系统中具有多媒体视图的人机交互界面，不断完善，逐步成熟，监控软件也从简单的文字界面向图形化的人机交互平台发展。本课题的目的是研究设计的是一个能够实现远程监控、具有基于 iOS 开发图形界面的远程监控控件。

随着 iPhone 与 iPad 等设备的普及，iOS 平台以其简洁美观的界面和方便的操控方式被越来越多的用户接受。iOS 的用户界面采用多点触控（Multi-Touch）方式操作，因为其优雅直观的界面设计，使用户乐于使用。iOS 可以直接与硬件底层通信，效率要高于 Android 等嵌入式操作系统，运行软件更加流畅。因此受到用户的广泛认同。

iPad 平台上的远程监控系统的优点在于可以把信息直观的显示在用户界面上，并且操作方便，人机交互友好。所以用 iOS 编写程序实现可视化监控，可以为天河系统提供一个方便的状态查询诊断工具。可以让客户更加迅捷的了解整个计算集群的实时状态。

本课题从实际应用需求出发，以人机交互原理为指导，结合面向对象的设计模式，在掌握基于 Xcode IDE 开发环境以及 iPhone SDK 的程序设计方法以及 Cocoa 框架、Objective-C 语言等 iOS 开发设计工具的前提下，完成一个基于 iOS 的远程 3D 监控可视化系统的界面设计。该通信系统具有对错误列表的维护、诊断信息的发送与接收、机柜状态确认等功能。

本课题实现的远程监控系统具有以下特点：1、采用 Cocoa Touch 框架在设计 GUI 界面实例，使系统美观、实用，人机交互更为出色。2、运用面向对象的设计模式、Objective C 语言在 iOS 平台上进行编写，保证了系统的可靠性和稳定性。3、系统运行在 iPad 手持设备上运行，移动方便，具有高效性。

研究本课题既有如下意义：

1) 本课题为天河系统提供一个方便的状态查询诊断工具，可以让客户迅捷的了解整个计算机群的实时状态，有利于天和系统的使用与维护。
2) 通过完成课题，使我对人机交互基本原理有了更进一步的理解，通过完成课题，学习了 iOS 程序设计，更深入的了解了关于人机界面的只是，并加深了对面向对象编程思想的理解。也积累了一些自己十分缺乏的工程经验。为下一步分配到部队工作，打下了基础，做好了准备。

1.3 本课题国内外研究现状及发展趋势

1.3.1 国内外研究历史

iOS 发展历史：

iOS 最早于 2007 年 1 月 9 日的苹果 Macworld 展览会上公布，随后于同年的 6 月发布的第一版 iOS 操作系统，当初的名称为 iPhone runs OS X。

2007 年 10 月 17 日，苹果公司发布了第一个本地化 iPhone 应用程序开发包（SDK）。

2008 年 3 月 6 日，苹果发布了第一个测试版开发包，并且将 iPhone runs OS X 改名为 iPhone OS。

2010 年 2 月 27 日，苹果公司发布 iPad，iPad 同样使用了 iPhone OS。这年，苹果公司重新设计了 iPhone OS 的系统结构和自带程序。

2010 年 6 月，苹果公司将 iPhone OS 改名为 iOS。

2012 年 6 月，苹果公司在 WWDC 2012 上宣布了 iOS6。iOS 6 拥有更完善的文本输入法，并内置了对热门中文互联网服务的支持等。

Objective-C 历史：

1980 年代初，Brad Cox 与 Tom Love 发明 Objective-C。《Object-oriented Programming, An Evolutionary Approach》，是对 Objective-C 最权威的描述。

1992 年，自由软件基金会的 GNU 开发环境增加了对 Objective-C 的支持。

1996 年 12 月 20 日，苹果公司宣布收购 NeXT Software 公司，NEXTSTEP/OPENSTEP 环境成为苹果操作系统下一个主要发行版本 OS X 的基础。这个开发环境的该版本被苹果公司称为 Cocoa。

2006 年 7 月苹果全球开发者会议中，Apple 宣布了“Objective-C 2.0”的发布，其增加了现代的垃圾收集，语法改进，运行时性能改进，以及 64 位支持。

2007 年 10 月发布的 Mac OS X v10.5 中包含了 Objective-C2.0 的编译器。[5]

1.3.2 发展趋势

Objective-C 发布 2.0 版本，让 Objective-C 语言更加的高效，方便程序员使用，2.0 版本增加了垃圾收集功能，对语法进行了改进，提升了运行性能。在近期统计的编程语言使用排行中，Objective-C 跃居第二名，仅次于 C 语言。苹果公司推出 iOS 6 后，提供了超过 200 项新功能，如完善了文本输入法，并且内置了对热门中文互联网服务的支持等。iOS 6 提供了更为友好的人机交互界面，让人机界面更加的人性化，以及可操控性增强。iOS 设备也不断更新，下一代 iPad 将具有更快的 CPU 与 GPU，将具备更加强大的图形处理能力。内存也将加大，可以运行更庞大的软件。

目前，较大的工程，采用可视化监控已成为一种趋势。监控可视化让以往复杂的监控工作变得简单，问题一旦出现，信息能够及时的反映在界面上，让系统的管理及维护工作更加的方便。

1.4 本文结构

全文共分为四章，研究了基于 iOS 的 GUI 程序设计的实现原理以及实现方法。具体的章节安排如下：

第一章，绪论。介绍了课题背景，研究历史与现状，展望课题的发展趋势。并简述了课题的研究内容及意义。

第二章，iOS 程序设计研究。介绍了 iOS 基本的编程原理。并且进一步研究了 Cocoa，介绍了 foundation 框架与 Application K 框架。

第三章，实现运行在 iPad 上的 iOS 远程监控系统。首先介绍总体功能设计规划，并对系统进行了用例分析，接着从文件结构与系统窗体结构两个方面对系统结构进行了介绍，然后对系统中所使用的 GUI 主要控件进行了逐一分析，最后介绍了系统主要界面与系统关键代码。

第四章，基于 OpenGL ES 的 3d 程序设计，介绍了 OpenGL ES 原理，并且从 OBJ 模型介绍了 3D 机柜文件显示技术。

第 2 章 iOS 程序设计研究

2.1 基本编程原理

2.1.1 核心应用程序

所有的 iPad 应用程序都是基于 UIK 框架构建而成的，因此，它们在本质上具有相同的核心架构。UIK 负责提供运行应用程序和协调用户输入及屏幕显示所需要的关键对象。应用程序之间不同的地方在于如何配置缺省对象，以及如何通过定制对象来添加用户界面和行为。

从应用程序启动到退出的过程中，UIK 框架负责管理大部分关键的基础设施。iPad 应用程序不断地从系统接收事件，而且必须响应那些事件。接收事件是 UIApplication 对象的工作，但是，响应事件则需要的定制代码来处理。应用程序的生命周期是由发生在程序启动到终止期间的一系类事件构成的。

在 iOS 中，用户可以通过轻点 Home 屏幕上的图标来启动应用程序。在轻点图标之后的不久，系统就会显示一个过渡图形，然后调用相应的 main 函数来启动应用程序。从这个点之后，大量的初始化工作就会交给 UIK，由它装载应用程序的用户界面和准备事件循环。在事件循环过程中，UIK 会将事件分发给定制对象及响应应用程序发出的命令。当用户进行退出应用程序的操作时，UIK 会通知应用程序，并开始应用程序的终止过程。[6]

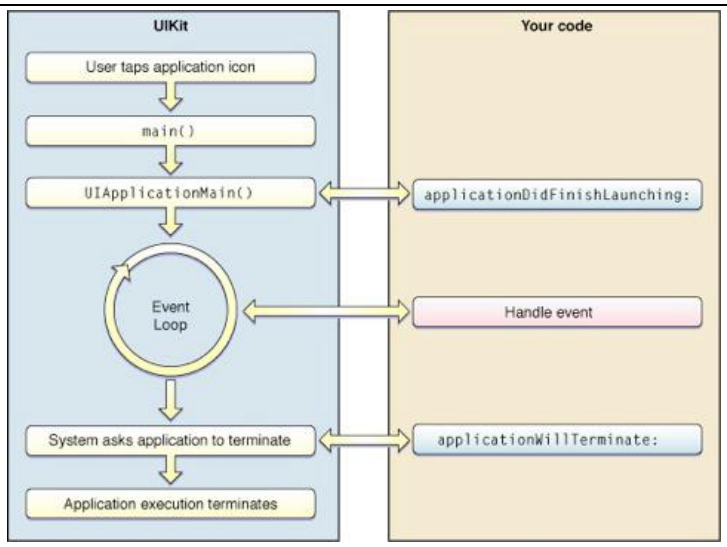


图 2-1　iPad 应用程序生命周期

图 2-1 显示了一个简化了的 iPad 应用程序生命周期。这个框图展示了发生在应用程序启动到退出过程中的事件序列。在应用程序初始化和终止的时候，UIK 会向应用程序委托对象发送特定的消息，使其知道正在发生的事件。在事件循环中，UIK 将事件派发给应用程序的定制事件处理器。

在 iPad 的应用程序中，main 函数仅在最小程度上被使用，应用程序运行所需的大多数实际工作由 UIApplicationMain 函数来处理。因此，当在 Xcode 中开始一个新的应用程序工程时，每个工程模板都会提供一个 main 函数的标准实现。以下是 iOS 程序中的 main 函数代码：

```
#import <UIK/UIK.h>

int main(int argc, char *argv[])

{

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] in];

    int retVal = UIApplicationMain(argc, argv, nil, nil);

    [pool release];

    return retVal;

}
```

Main 函数用来完成以下几项工作：创建一个自动释放池，调用 UIApplicationMain 函数，以及使用自动释放池。自动释放池用于内存管理，它是 Cocoa 的一种机制，用于延缓释放具有一定功能的代码块中创建的对象。

程序的核心代码是 UIApplicationMain 函数，它接收四个参数，并将它们用于初始化应用程序。除了传给 main 函数的 argc 和 argv 之外，该函数还需要两个

字符串参数，用于标识应用程序的首要类（即应用程序对象所属的类）和应用程序委托类。如果首要类字符串的值为 nil，UIK 就缺省使用 UIApplication 类；如果应用程序委托类为 nil，UIK 就会将应用程序主 nib 文件（针对通过 Xcode 模板创建的应用程序）中的某个对象假定为应用程序的委托对象。如果将这些参数设置为非 nil 值，则在应用程序启动时，UIApplicationMain 函数会创建一个与传入值相对应的类实例，并将它用于既定的目的。因此，如果应用程序使用了 UIApplication 类的定制子类，就需要在第三个参数指定该定制类的类名。委托是一种避免对复杂的 UIK 对象（比如缺省的 UIApplication 对象）进行子类化的机制。在这种机制下，可以不进行子类化和方法重载，而是将自己的定制代码放到委托对象中，从而避免对复杂对象进行修改。Nib 文件是基于磁盘的资源文件，用于存储单或多个对象的快照。iPad 应用程序的主 nib 文件通常包含一个窗口对象和一个应用程序委托对象，还可能包含一个或多个管理窗口的其它重要对象。装载一个 nib 文件会使该文件中的对象被重新构造，从而将每个对象的磁盘表示转化为应用程序可以操作的内存对象。从 nib 文件中装载的对象和通过编程方式创建的对象之间没有区别。然而，对于用户界面而言，以图形的方式（使用 Interface Builder 程序）创建与用户界面相关联的对象并将它们存储在 nib 文件中通常比以编程的方式进行创建更加方便。[7]

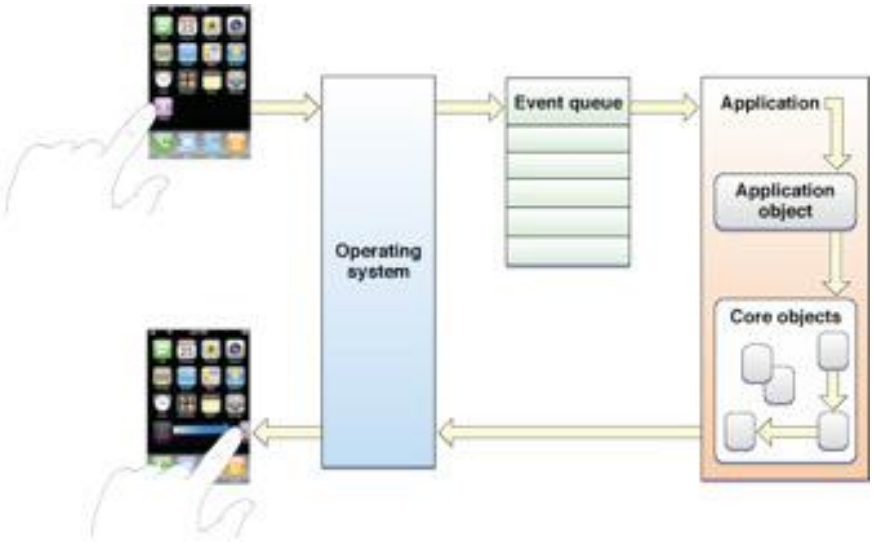


图 2-2　事件处理周期

在应用程序初始化之后，UIApplicationMain 函数就会启动管理应用程序事件和描画周期的基础组件，如图 2-2 所示。在用户和设备进行交互的时候，iOS 会检测触摸事件，并将事件放入应用程序的事件队列。然后，UIApplication 对象的事件处理设施会从队列的上部逐个取出事件，将它分发到最适合对其进行处理的对象。举例来说，在一个按键上发生的触摸事件会被分发到对应的按键对象。事件也可以被分发给控制器对象和应用程序中不直接负责处理触摸事件的其它

对象。

在 iOS 的多点触摸事件模型中，触摸数据被封装在事件对象（UIEvent）中。为了跟踪触摸动作，事件对象中包含一些触摸对象（Uouch），每个触摸对象都对应于一个正在触摸屏幕的手指。当用户把手指放在屏幕上，然后四处移动，并最终离开屏幕的时候，系统通过对应的触摸对象报告每个手指的变化。在启动一个应用程序时，系统会为该程序创建一个进程和一个单一的线程。这个初始线程成为应用程序的主线程，UIApplication 对象正是在这个线程中建立主运行循环及配置应用程序的事件处理代码。图 2-3 显示了事件处理代码和主运行循环的关系。系统发送的触摸事件会在队列中等待，直到被应用程序的主运行循环处理。[8]

2.1.2 运行环境

iOS 的运行环境被设计为快速而安全的程序执行环境。它具有以下几个特点：

1) 启动过程快，使用时间短

iOS 设备的优势是它们的便捷性。用户通常从口袋里掏出设备，用上几秒或几分钟，就又放回口袋中了。在这个过程中，用户可能会打电话、查找联系人、改变正在播放的歌曲、或者取得一片信息。在 iOS 中，每次只能有一个前台应用程序。这意味着每次用户在 Home 屏幕上轻点应用程序图标时，程序必须快速启动和初始化，以尽可能减少延迟。

除了快速启动，应用程序还必须做好快速退出的准备。每次用户离开应用程序时，无论是按下 Home 键还是通过软件提供的功能打开了另一个应用程序，iOS 会通知应用程序退出。在那个时候，需要尽快将未保存的修改保存到磁盘上。如果应用程序退出的时间超过 5 秒，系统可能会立刻终止它的运行。

当用户切换到另一个应用程序时，虽然程序不是运行在后台，但是使它看起来好像是在后台运行。当程序退出时，除了对未保存的数据进行保存之外，还应该保存当前的状态信息；而在启动时，则应该寻找这些状态信息，并将程序恢复到最后一次使用时的状态。这样可以使用户回到最后一次使用时的状态，使用户体验更加一致。以这种方式保存用户的当前位置还可以避免每次启动都需要经过多个屏幕才能找到需要的信息，从而节省使用的时间。

2) 应用程序沙箱（sandbox）

由于安全的原因，iOS 将每个应用程序（包括其偏好设置信息和数据）限制在文件系统的特定位置上。这个限制是安全特性的一部分，称为应用程序的“沙箱”。沙箱是一组细粒度的控制，用于限制应用程序对文件、偏好设置、网络资源、和硬件等的访问。在 iOS 中，应用程序和它的数据驻留在一个安全的地方，其它应用程序都不能进行访问。在应用程序安装之后，系统就通过计算得到一个

不透明的标识，然后基于应用程序的根目录和这个标识构建一个指向应用程序家目录的路径。

在安装过程中，系统会创建应用程序的家目录和几个关键的子目录，配置应用程序沙箱，以及将应用程序的程序包拷贝到家目录上。将应用程序及其数据放在一个特定的地方可以简化备份恢复操作，还可以简化应用程序的更新及卸载操作。

沙箱可以限制攻击者对其它程序和系统造成的破坏，但是不能防止攻击的发生。换句话说，沙箱不能使程序避免恶意的直接攻击。举例来说，如果在输入处理代码中有一个可利用的缓冲区溢出，而又没有对用户输入进行正当性检查，则攻击者可能仍然可以使应用程序崩溃，或者通过这种漏洞来执行攻击者的代码。[9]

3) 虚拟内存系统

在本质上，iOS 使用与 Mac OS X 同样的虚存系统。在 iOS 中，每个程序都仍然有自己的虚拟地址空间，但其可用的虚拟内存受限于现有的物理内存的数量（这和 Mac OS X 不同）。这是因为当内存用满的时候，iOS 并不将非永久内存页面（volatile pages）写入到磁盘。相反，虚拟内存系统会根据需要释放永久内存（nonvolatile memory），确保为正在运行的应用程序提供所需的空间。内存的释放是通过删除当前没有正在使用或包含只读内容（比如代码页面）的内存页面来实现的，这样的页面可以在稍后需要使用的时候重新装载到内存中。如果内存还是不够，系统也可能向正在运行的应用程序发出通告，要求它们释放额外的内存。所有的应用程序都应该响应这种通告，并尽自己所能减轻系统的内存压力。

4) 自动休眠定时器

iOS 试图省电的一个方法是使用自动休眠定时器。如果在一定的时间内没有检测到触摸事件，系统最初会使屏幕变暗，并最终完全关闭屏幕。大多数开发者都应该让这个定时器打开，但是，游戏和不使用触摸输入的应用程序开发者可以禁用这个定时器，使屏幕在应用程序运行时不会变暗。将共享的 UIApplication 对象的 idleTimerDisabled 属性设置为 YES，就可以禁用自动休眠定时器。由于禁用休眠定时器会导致更大的电能消耗，所以开发者应该尽一切可能避免这样做。只有地图程序、游戏、以及不依赖于触摸输入而又需要在设备屏幕上显示内容的应用程序才应该考虑禁用休眠定时器。音频应用程序不需要禁用这个定时器，因为在屏幕变暗之后，音频内容可以继续播放。如果禁用了定时器，请务必尽快重新激活它，使系统可以更省电。

2.1.3 窗口和视图

和 Mac OS X 一样，iOS 通过窗口和视图在屏幕上展现图形内容。视图是

UIView 类的实例，负责在屏幕上定义一个矩形区域。在 iPad 的应用程序中，视图在展示用户界面及响应用户界面交互方面发挥关键作用。每个视图对象都要负责渲染视图矩形区域中的内容，并响应该区域中发生的触碰事件。这一双重行为意味着视图是应用程序与用户交互的重要机制。在一个基于模型-视图-控制器的应用程序中，视图对象明显属于视图部分。除了显示内容和处理事件之外，视图还可以用于管理单或多个子视图。子视图是指嵌入到另一视图对象边框内部的视图对象，而被嵌入的视图则被称为父视图或超视图。视图的这种布局方式被称为视图层，一个视图可以包含任意数量的子视图，通过为子视图添加子视图的方式，视图可以实现任意深度的嵌套。视图在视图层次中的组织方式决定了在屏幕上显示的内容，原因是子视图总是被显示在其父视图的上方，这个组织方法还决定了视图如何响应事件和变化。每个父视图都负责管理其直接的子视图，即根据需要调整它们的位置和尺寸，以及响应它们没有处理的事件。视图在程序中起到以下几点作用：

1) 描画和动画
   a) 视图负责对其所属的矩形区域进行描画。
   b) 某些视图属性变量可以以动画的形式过渡到新的值。
2) 视图管理子视图列表。
   a) 视图定义了自身相对于其父视图的尺寸调整行为。
   b) 必要时，视图可以通过代码调整其子视图的尺寸和位置。
   c) 视图可以将其坐标系统下的点转换为其它视图或窗口坐标系统下的点。
3) 事件处理
   a) 视图可以接收触摸事件。
   b) 视图是响应链的参与者。

UIView 类定义了视图的基本行为，但并不定义其视觉表示。相反，UIK 通过其子类来为像文本框、按键及工具条这样的标准界面元素定义具体的外观和行为。

下图显示了所有 UIK 视图类的层次框图：

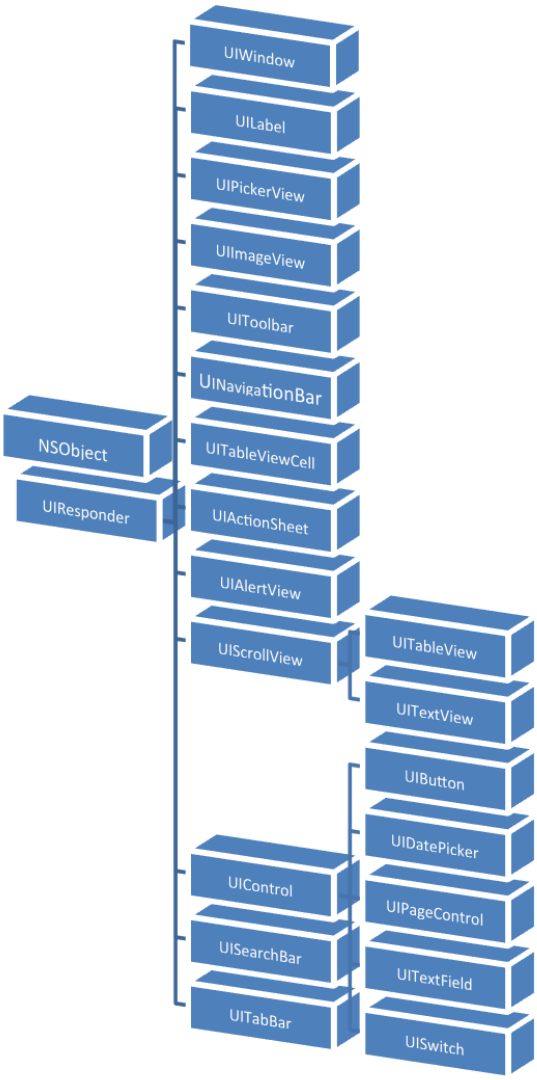


图 2-5　UIk 视图类

这个视图层次可以分为如下几个大类：

1) 容器

容器视图用于增强其它视图的功能，或者为视图内容提供额外的视觉分隔。比如，UIScrollView 类可以用于显示因内容太大而无法显示在一个屏幕上的视图。UableView 类是 UIScrollView 类的子类，用于管理数据列表。表格的行可以支持选择，所以通常也用于层次数据的导航—比如用于挖掘一组有层次结构的对象。Uoolbar 对象则是一个特殊类型的容器，用于为一或多个类似于按键的项提供视觉分组。工具条通常出现在屏幕的底部。Safari、Mail 和 Photos 程序都使用工具条显示一些按键，这些按键代表经常使用的命令。工具条可以一直显示，也可以根据应用程序的需要进行显示。

2) 控件

控件用于创建大多数应用程序的用户界面。控件是一种特殊类型的视图，继

承自 UIControl 超类，通常用于显示一个具体的值，并处理修改这个值所需要的所有用户交互。控件通常使用标准的系统范式（比如目标-动作模式和委托模式）来通知应用程序发生了用户交互。控件包括按键、文本框、滑块和切换开关。

3) 显示视图

控件和很多其它类型的视图都提供了交互行为，而另外一些视图则只是用于简单地显示信息。具有这种行为的 UIK 类包括 UIImageView、UILabel、UIProgressView、UIActivyIndicatorView。

4) 文本和 web 视图

文本和 web 视图为应用程序提供更为高级的显示多行文本的方法。UextView 类支持在滚动区域内显示和编辑多行文本，而 UIWebView 类则提供了显示 HTML 内容的方法，通过这个类，可以将图形和高级的文本格式选项集成到应用程序中，并以定制的方式对内容进行布局。

2.2 Cocoa 研究

2.2.1 概述

Cocoa 包括两个方面：即运行环境方面和开发方面。在运行环境方面，Cocoa 应用程序呈现 Aqua 用户界面，且和操作系统的其它可视部分紧密集成，这些部分包括 Finder、Dock 和基于所有环境的其它应用程序。开发方面，Cocoa 是一个面向对象的软件组件的集成套件，它使开发者可以快速创建强壮和全功能的 Mac OS X 应用程序。这些类是可复用和可支配的软件积木，开发者可以直接使用，或者根据具体需求对其进行扩展。[11]

Cocoa 可以在应用程序中起到以下作用：

1) 基本应用程序框架

Cocoa 为事件驱动的行为和应用程序、窗口和工作空间（workspace）的管理提供了基础设施。在大多数情况下，不必直接处理事件或发送任何描画命令给渲染库。

2) 用户界面对象

Cocoa 为应用程序的用户界面提供了丰富而又现成的对象。这些对象的大部分都在 Interface Builder（创建用户界面的开发工具）的选盘上，此外 Cocoa 还有一些支持用户界面的技术，包括提高可访问性、执行正当性检查以及连接用户界面对象和定制对象需要的技术。

3) 描画和图像处理

Cocoa 带有一个可以锁定图形焦点并将视图（或视图的一部分）标识为“变脏”的框架，从而支持高效的定制视图描画。Cocoa 中还有一些描绘贝齐尔（Bezier）路径、执行远交变换、合成图像以及创建不同图像表示的编程工具类。

4) 系统交互

Cocoa 使的应用程序可以和文件系统、工作空间以及其它应用程序进行（或使用它们提供的服务）。

5) 数据交换

Cocoa 通过拷贝-粘贴、拖拽模型以及 Services 菜单简化了应用程序内部和应用程序之间的数据交换。

6) 性能

为了增强应用程序的性能，Cocoa 提供了多线程、空闲时间处理、资源的迟缓加载、内存管理和运行环操作方面的编程支持。

2.2.2 Cocoa 框架

Mac OS X 包含多个 Cocoa 框架，其中 Foundation 和 Application K 框架，它们是核心的 Cocoa 框架。其它框架是辅助和可选的，而 Foundation 和 Application K 框架在 Cocoa 开发中是必要的。

Foundation 框架定义了一些基础类，可以用于各种类型的 Cocoa 程序。Foundation 框架和 Application K 框架的区分标准在于用户界面。如果一个对象既不出现在用户界面上，也不是专门用于支持用户界面，那么它就属于 Foundation 框架。Cocoa 应用程序定义为需要连接 Application K 框架，同时也总是必须连接 Foundation 框架的程序。这两个类层次都共用同一个根类，即 NSObject 类，很多）Application K 的方法和函数都将 Foundation 对象作为参数或返回值。

Foundation 类层次的根是 NSObject 类，它和 NSObject 及 NSCopying 协议一起定义了基本的对象属性和行为。Foundation 框架的剩余部分由几组相互关联的类和一些独立的类组成。有一些代表基本数据类型的类，如字符串、字节数组、用于存储其它对象的集合类，一些代表系统信息的类，如日期类，还有一些代表系统实体的类，比如端口、线程和进程。

Application K 框架包含实现图形的事件驱动的用户界面需要的所有对象：窗口、对话框、按键、菜单、滚动条、文本输入框，这个列表还在不断增加。Application K 处理所有的细节，它可以高效地进行屏幕描画和营建设备及屏幕缓冲区进行通讯，在描画之前清除屏幕上的区域，以及对视图进行裁剪。

Application K 由超过 125 个类和协议组成。所有的类最终都从 Foundation 框架的 NSObject 类继承而来。

Application K 中最大分支的根是 NSResponder 类，它负责定义响应者链，即对用户事件进行响应的有序对象列表。当用户进行按键或鼠标点击时，系统就会产生一个事件，并沿着响应链向上传递，寻找可以响应该事件的对象。任何处理事件的对象都必须继承自 NSResponder 类。核心的 Application K 类

NSApplication、NSWindow、NSView 都继承自 NSResponder。[12]

Application K 类的第二大分支继承自 NSCell 类。这组类和 NSControl 类的派生类有大体上的映像关系。对于负责响应用户动作的用户界面对象，Application K 采用的架构将它们的工作分为控件（control）对象和单元（cell）对象。NSControl 和 NSCell 类以及它们的子类定义了一组常见的用户界面对象，比如按键（button）、滑块（slider）和浏览器（browser），用户可以通过图形化的操作控制应用程序的某些方面。大多数的控件对象和一个或多个单元对象相关联，单元对象负责实现描画细节和事件的处理。举例来说，一个按键是由一个 NSButton 对象和一个 NSButtonCell 对象构成的。

2.2.3 Cocoa 对象的生命周期

Cocoa 对象的生命周期跨越不同的阶段。它需要被创建、初始化和使用（就是其它对象向它发送消息），它可能被保持、拷贝或者归档，并最终被释放和销毁。

从最后，即从清理对象的方式开始讨论。和其它编程语言不同，Objective-C 没有自动释放不再使用的对象的“垃圾收集”功能。垃圾收集开销大而且不灵活，Cocoa 和 Objective-C 选择一种主动的策略驱动的例程来保持对象，并在不再需要的时候进行清理。

这种例程和策略建立在引用计数的基础上。每个 Cocoa 对象都带有一个整数，用于指示对其持久性感兴趣的其它对象（甚至是例程代码的现场）的数目。这个整数被称为对象的保持数（“保持”是为了避免和“引用”重复）。当通过 alloc 或者 allocWhZone:类方法创建对象的时候，Cocoa 起到了以下作用：

1) 它将对象的 isa 指针，NSObject 类中唯一的公共实例变量，指向对象的类，因此将对象集成到类层次的运行视图中。
2) 它将对象的保持数，一个所有对象都有的隐藏的实例变量，设置为 1。

当释放一个对象，向对象发送一个 release 消息时，NSObject 会减少它的保持数。如果保持数从 1 下降到 0，对象就会被解除分配。对象的解除分配有两个步骤：首先是对象的 dealloc 方法被调用，以释放实例变量和动态分配的内存，然后是操作系统将对象的本身销毁，并回收对象占用的内存。如果不希望对象很快消失，在创建对象之后向它发送一个 retain 消息，对象的保持数就会增加到 2。这样，就需要两个 release 消息才能导致对象的释放。[13]

第 3 章 基于 iPad 的 iOS 远程监控系统

3.1 总体功能设计规划

3.1.1 项目规划

天河远程监控系统使用 Cocoa 框架开发，系统维护机柜状态列表、由界面部分，网络通信模块和 3D 模块组成。通信模块负责与通信服务器交互，获取系统状态信息并将得到的信息传给界面，界面负责实时显示系统状态，并与用户交互。3D 模块负责将机柜等可能出错的硬件可视化。用户进入系统后，可以查看系统中任意一个机柜的状态。如果发生错误或异常，将在 iPad 上以 3D 的形式显示出来。

3.1.2 系统功能结构图

iOS 上的监控系统从设在机柜的服务器程序获取机柜当前的状态，将机柜状态动态的反映在 iOS 系统中的 3D 模型中，并通过列表给出详细的说明。用户通过对 iPad 程序的手势操作，可以了解整个计算机群的实时动态。远程监控系统功能结构如图 3-1 所示。

图 3-1 系统功能结构图

3.2 系统结构

3.2.1 文件结构

系统的文件结构如图 3-2 所示。

图 3-2　系统文件结构图

系统的文件结构中 Classes 为代码，分为三部分，第一部分为 Picker 控件的代码实现，第二部分为 OpenGL 3D 模型实现，第三部分为分栏控件与表格的实现。Resources 中包含了系统中所用的图片和界面文件。Frameworks 中包含系统所调用的库，Products 中为系统的应用程序。

3.2.2　系统窗体结构

远程监控系统中的窗体结构如图 3-3。

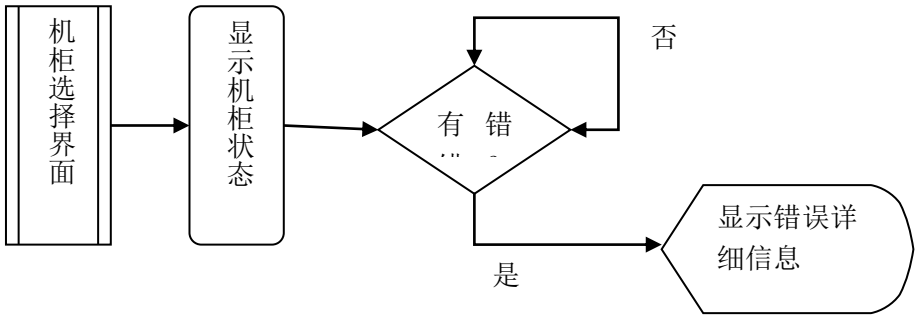


图 3-3　窗体结构图

3.3 系统用例分析

系统用例图如图 3-4 所示，完整用例如下：

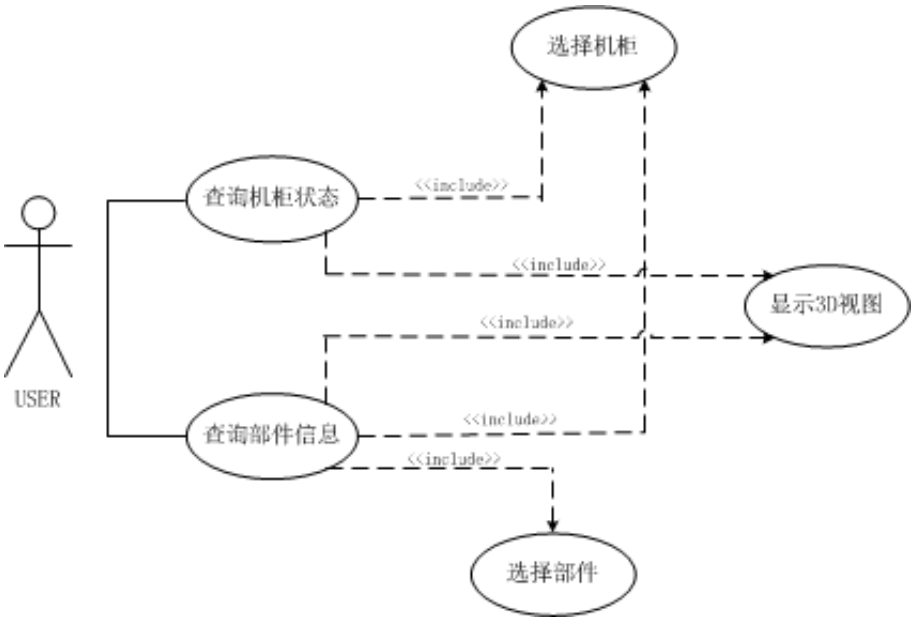


图 3-4 系统用例图

1) 用例名：查询机柜状态

主要执行者：用户

基本交互动作：

1.用户选择机柜。

2.系统显示当前机柜实时状态。

3.系统显示 3D 视图。

2) 用例名：查询部件信息

主要执行者：用户

基本交互动作：

1.用户选择机柜。

2.系统显示当前机柜实时状态。

3.在部件列表中选择部件。

4.系统显示当前部件信息。

5.系统显示 3D 视图。

3) 用例名：选择机柜

主要执行者：用户

基本交互动作：

1.选择机柜类型。

2.转动第一个 Picker 选择器选择机柜所在行数。

3.转动第二个 Picker 选择器选择机柜所在列数。

4.转动第三个 Picker 选择器选择机柜的正反面。

5.确认选择。

4) 用例名：选择部件

主要执行者：用户

基本交互动作：

1. 在部件列表中选择部件。

1a 所选部件发生故障。

1a1 在故障列表中选择部件。

5) 用例名：显示 3D 视图

主要执行者：远程监控系统

基本交互动作：

1. 显示 3D 视图。

1a 部件发生故障。

1a1 故障部件标红显示。

3.4 程序中实现的 GUI 控件

3.4.1 表视图（Table View）

图 3-5 表格控件效果

表示图是 iOS 常用的数据显示控件，用途广泛，可以以列表的形式显示各种信息，每个列表单元都是一个可以单独编程的视图，因此表示图控件可以被极大程度的定制，显示各种编程风格。表示图实现如图 3-5 所示。

如要在程序中实现表格控件，必须实现两个协议，即 UableViewDelegate 和 UableViewDataSource。

```
@implementation SimpleTableViewController
{
    NSArray *tableData;
}
- （void）viewDidLoad
{
    [super viewDidLoad];
    // Do any addional setup after loading the view, typically from a nib.
    tableData = [NSArray arrayWhObjects:@"Egg Benedict",
                    @"Mushroom Risotto", @"Full Breakfast", nil];
}
```

上面代码声明了一个数组 NSArray *tableData，并且给数组进行赋值，用来存储表格内容。注意字符用 NSString 定义。

TableView 的显示内容可分为两级，第一级为 Section，第 2 级为 Row，每个 Section 中可以包含由 Row 组成的列表，并支持多层嵌套，在程序中，Row 对应的类为 TableViewCell。TableViewCell 可以被用户定制，本身甚至可以为一个列表控件。也支持图形等选项。

图 3-6　表视图举例

如图 3-6 中有两个 Section，Section 0 中有 2 个 Row，Section 1 中能也看到 2 个 Row。

UableView 是表视图幕后的实现类，主要用于初始化表格数据并且提供控制。UableViewDataSource 定义了连接列表控件与其数据提供者的标准接口。

UableViewDataSource 协议定了 2 个必须要求实现的方法：

tableView:cellForRowAtIndexPath：在每个 row 应当显示什么数据

tableView:numberOfRowsInSection：告诉列表控件 Section 的数目

```
- （NSInteger）tableView:（UableView *）tableView
 numberOfRowsInSection:（NSInteger）section
 {
     return [tableData count];
}
- （UableViewCell *）tableView:（UableView *）tableView cellForRowAtIndexPath:
（NSIndexPath *）indexPath
{
     static NSString *simpleTableIdentifier = @"SimpleTableem";
     UableViewCell *cell = [tableView dequeueReusableCellWhIdentifier:simpleTableIdentifier];
     if （cell == nil） {
          cell = [[UableViewCell alloc]
                    inWhStyle:UableViewCellStyleDefault reuseIdentifier:simpleTableIdentifier];
     }
     cell.textLabel.text = [tableData objectAtIndex:indexPath.row];
     cell.imageView.image = [UIImage imageNamed:@"test.jpeg"];
     return cell;
}
```

tableView:cellForRowAtIndexPath 方法绘制了 cell，并且可以给表格中每行添加图片。这个方法返回的就是某个 cell，可以自定义各种样式渲染，也可以用几种默认样式去渲染。indexPath 标示的是你要渲染的这个 cell 的位置，indexPath 对象里有两个属性，section 和 row。

UableViewDelegate 负责处理 UableView 的显示。协议中具有很多可选方法，这些方法包括管理表行的高度，配置节点头部和底部，对表单元重新排序等等。

3.4.2 选择器视图（Picker View）

图 3-7 选择器控件效果

选择器在界面中给用户提供选择列表，选择器实现如图 3-7 所示。选择器通过 UIPickerViewDatasource 以及 UIPickerViewDelegate 协议实现。

Picker 控件会向数据源请求所需数据，需要 UIPickerViewDatasource 数据源，picker 控件转动到某一个值，会告知代理，需要 UIPickerViewDelegate 协议。

```
@interface mrcuriosyViewController : UIViewController
<UIPickerViewDataSource, UIPickerViewDelegate>
{
    NSArray* activies;
    NSArray* feelings;
}
```

在 mrcuriosyViewController.h 声明遵循<UIPickerViewDataSource，UIPickerViewDelegate>两个协议，并且建立两个数组，用来存放 picker 内容。

```
@implementation mrcuriosyViewController
- （void）viewDidLoad
{
    [super viewDidLoad];
     // Do any addional setup after loading the view, typically from a nib.
    activies = [[NSArray alloc] inWhObjects:@"机柜 1 ",@"机柜 2 ",@"机柜 3 ", nil];
    feelings = [[NSArray alloc] inWhObjects:@" 处理器 1 ",@"处理器 2 ", nil];
}
```

这个方法属于视图控制器，会在视图从.xib 文件载入后被调用，协议 UIPickerViewDataSource 中规定了两个必须实现的方法:

umberOfComponentsInPickerView：每个选择器有多少部件

pickerViewnumberOfRowsInComponent：每个部件里有多少行

另外协议 UIPickerViewDelegate 规定必须实现方法

pickerViewtleForRow: forComponent，这个方法必须返回一个 NSString，其中包含有给定部件的标题。

3.4.3 分栏控件视图（Spl View）

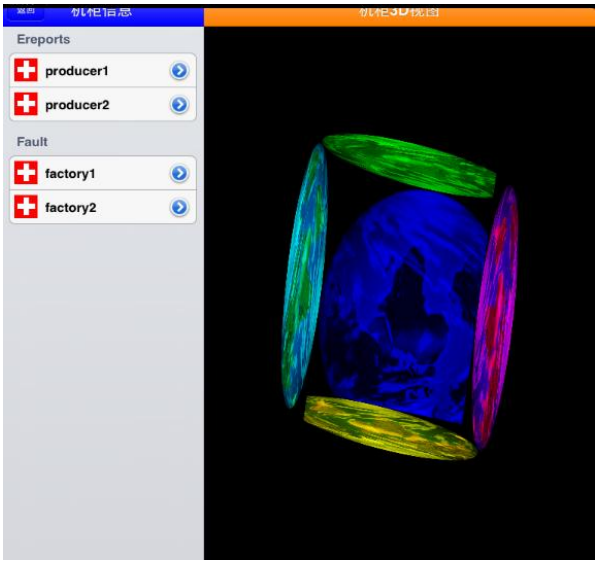


图 3-8 分栏控件效果

分栏控件是 iPad 专有的一个复合控件。分栏框架实现如图 3-8 所示。创建一个 UISplViewController 的方法是在 Xcode 里将其拖出来，只能拖动 Spl View 到一个 iPad 类型的 storyboard。Spl View 通常只是一个基本元素，它填满整个屏幕，不可能把 Spl View 放到其他什么的内部，一般情况下是提供给整个 app 的。Spl View 有两个 ViewControllers，一个左侧一个右侧，左侧叫 Master，右侧叫 Detail。SplViewController 有一个 property 叫做 ViewController，它是一个数组，这个数组有两个元素，左侧和右侧，左侧是元素 0，右侧是元素 1，可以在代码中设置你的两个 ViewController。

首先在文件 MasterViewController.h 中声明属性。

```
@property （nonatomic, copy） NSArray *flowerData;
@property （nonatomic, copy） NSArray *flowerSections;
```

接下来，打开文件 ViewController.m，在@implementation 代码行下方添加配套的编译指令@synthesize。

```
@synthesize flowerData;
@synthesize flowerSections;
```

在文件 ViewController.m 的方法 viewDidUnload 中，执行清理工作，将两个属性设置为 nil。

```
[self setFlowerData:nil];

[self setFlowerSection:nil];
```

添加两个 NSArray：flowerData 和 flowerSections，分别用于存储花朵信息和分区信息，还需要声明方法 createFlowerData，它将数据加入到数据组中。

方法 createFlowerData 创建两个数组: flowerData，flowerSections。声明了两个 NSMutableArrays，并使用@name ，@picture 初始化数组，并将数组合并创建 NSArray flowerData 。分别用 [flowerData objectAtIndex:0] 和 [flowerData objectAtIndex:1]来分别代表两个数组。

在创建表示图数据源协议方法中，与表示图类似，需要实现以下四种方法：

numberOfSectionsInTableView：返回表示图将包含的分区数

tableView:numberOFrowsInSection：返回分区包含的行数

tableView:tleForHeaderInSection：返回表示分区标题的字符串

tableView:cellForRowAtIndexPath：返回单元格对象，供表视图显示

分栏视图需要让 DetailViewController 更新，为了与 DetailViewController 通信，需要使用其属性 detailem，由于 detailem 可以指向任何对象，需要把它设置为 NSDictionary 。需要在文件 MasterViewController.m 实现方法 tableView:didSelectRowAtIndexPath。

```
- （void）tableView:（UableView *）aTableView
    didSelectRowAtIndexPath:（NSIndexPath *）indexPath
  {
      self.detailViewController.detailem=[[flowerData
                                          objectAtIndex:indexPath.section]
                                          objectAtIndex: indexPath.row];
    }
```

为了在详细视图控制器中捕获事件并更新视图，还需在文件 DetailViewController.m 中修改方法 configureView。[14]

```
- （void）configureView
{
    if （self.detailem）
    {
        NSURL *detailURL;
        detailURL=[[NSURL alloc] inWhString:
                                    [self.detailem objectForKey:@”url”]];
        [self.detailWebView loadRequest:[NSURLRequest requestWhURL:detailURL]];
        self.navigationem.tle = [self.detailem objectForKey:@”name”];
        self.detailDescriptionLabel.hidden=YES;
    }
}
```

3.5 系统主要功能

3.5.1 系统主要界面

机柜选择界面中包括一个 picker 控件用于选择机柜号，一个按钮控件，选定后进入下一页。以及两个按钮控件，用于选择机柜的类型。

图 3-9 机柜选择界面

机柜信息界面，用分栏控件实现，左侧表格中显示机柜部件，Section 1 为正常的部件，Section 2 为故障部件，可以选择进入详细部件信息界面。右侧显示机柜 3D 视图。

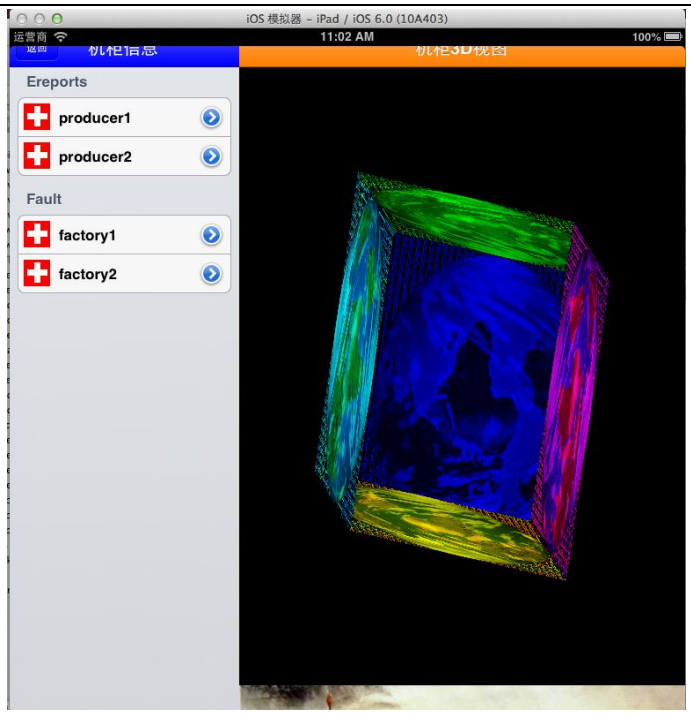


图 3-10　机柜信息界面

部件详细信息界面，用分栏控件实现。左侧显示部件的名称、温度、负载等，右侧为上为部件 3D 视图，下为部件详细信息。

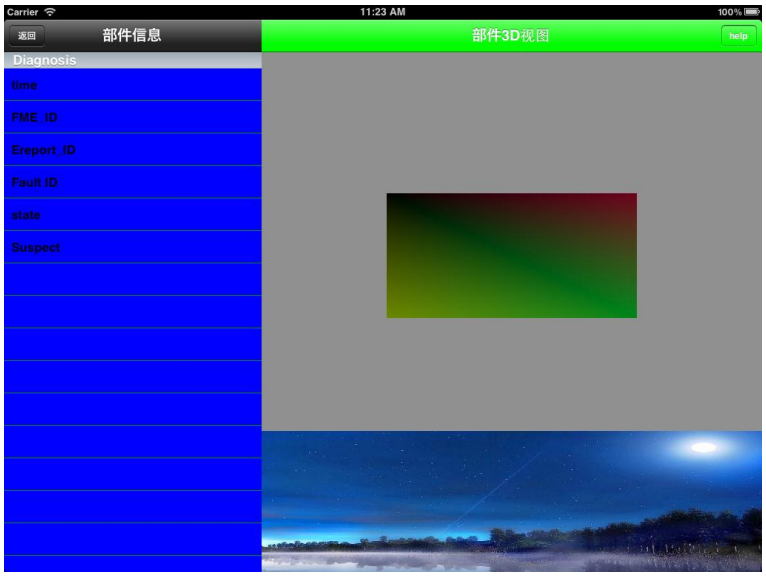


图 3-11　部件详细信息界面

3.5.2　系统关键代码

以下代码实现了 TableView：其中-（UableViewCell *）tableView:（UableView *）tableView cellForRowAtIndexPath:（NSIndexPath *）indexPath 方法用于实现 table 控件每一行的内容。

```
cell = [[[UableViewCell alloc] inWhStyle:UableViewCellStyleValue1
                                reuseIdentifier:CellIdentifier] autorelease];
cell.accessoryType = UableViewCellAccessoryDetailDisclosureButton;
```

这两行代码定义了控件的风格，并在单元格右边定义了展开按钮，以进入下一级视图。

```
- （NSInteger）numberOfSectionsInTableView:（UableView *）aTableView {
    return [machineParts count];
}
-（NSString *）tableView:（UableView *）tableView tleForHeaderInSection:（NSInteger）section{
    return [machineParts objectAtIndex:section];
}
- （NSInteger）tableView:（UableView *）aTableView numberOfRowsInSection:（NSInteger）
section {
    return [[machineData objectAtIndex:section] count];
}
-（UableViewCell *）tableView:（UableView *）tableView cellForRowAtIndexPath:（NSIndexPath
*）indexPath {
    static NSString *CellIdentifier = @"CellIdentifier";
  UableViewCell*cell=[tableViewdequeueReusableCellWhIdentifier:CellIdentifier];
    if （cell == nil） {
        cell = [[[UableViewCell alloc] inWhStyle:UableViewCellStyleValue1
reuseIdentifier:CellIdentifier] autorelease];
        cell.accessoryType = UableViewCellAccessoryDetailDisclosureButton;
}
```

第四章　基于 OpenGL ES 的 3d 程序设计

4.1　OpenGL ES 简介

OpenGL ES 是 OpenGL 三维图形 API 的子集，主要是针对手机、PDA 和游戏主机等嵌入式设备而设计。由于它的跨平台性和易移植性等特点，目前广泛应

用于嵌入式系统的图形学编程中。随着嵌入式系统的飞速发展，用户在图形学界面方面的要求也越来越高，传统的二维图形学早已经不能满足用户的审美和使用需求，人们在使用电子设备时更加强调用户的真实体验，这在图形学技术方面体现为对现实生活中的图像的三维表示以及动态交互。而 OpenGL ES 在处理、生成三维图像的卓越表现使得它成为开发人员的优先选择。由此可见，基于 OpenGL ES 的程序设计的重要性也将日益凸显。

OpenGL ES 是从 OpenGL 裁剪的定制而来的，它去除了 glBegin/glEnd，四边形（GL_QUADS）、多边形（GL_POLYGONS）等复杂图元等许多非绝对必要的特性。经过多年发展，现在主要有两个版本：OpenGL ES 1.x 和 OpenGL ES 2.x，其中 OpenGL ES 1.x 主要针对固定管线硬件（即指令内置），而 OpenGL ES 2.x 主要针对可编程管线硬件，即通过使用可编程管线可以直接编写显卡指令，这样不但能实现所有的固定管线的功能，还能编写更丰富的特效，充分发挥 GPU 性能。这两个版本的差别是很大的，尤其体现在编程上，它们区别主要体现在函数名命名方式、渲染方式以及着色器的使用流程等方面。

目前 OpenGL ES 最常用版本 OpenGL ES 2.0 是 Khronos Group 在 2007 年 3 月份制定的一种业界标准应用程序编程接口(API)，这个接口可以大大提高不同消费电子设备的 3D 图形渲染速度，在嵌入式系统上实现了全面可编程的 3D 图形。

4.2 OpenGL ES 原理

4.2.1 客户端-服务器模型

OpenGL ES 使用客户端-服务器模型，如图 4.1。当应用程序呼叫 OpenGL ES 函数时，它会与 OpenGL ES 客户端交谈。客户端程序到函数调用完毕，必要时转化为 OpenGL 服务器命令。客户端-服务器模型允许工作处理被分为客户端与服务器端的函数调用。客户端，服务器，及之间的沟通路径的本质，OpenGL ES 各别具体实践；在 iOS 终至上，工作是介于 CPU 与专用的图形处理器之间。

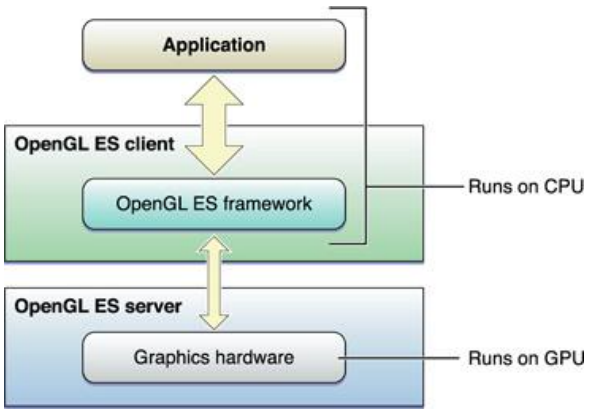


图 4-1 OpenGL ES 客户端-服务器模型

4.2.2 帧缓存对象

帧缓存(Framebuffer)对象是图型管线的最终目的，是 OpenGL ES 中最重要的对象类型。帧缓存对象只是个容器，内含纹理(texture)与渲染缓存(renderbuffer)，这个容器把材质和渲染缓存在自己上以创建一个完整的渲染需要的设置。

帧缓存对象是渲染命令的目标。OpenGL ES2.0 提供的帧缓存对象是核心规范中的一部分；作为 iOS 里唯一的渲染目标，在 OpenGLES1.1 中 OES_framebuffer_object 拓展提供。借助把图片附在帧缓存上，帧缓存对象提供对颜色、深度和模板数据的存储，如图 4-2 所示。最通常的图片附着是一个渲染缓存对象，但是一个 OpenGL ES 材质可以被附在帧缓存的颜色附着点上，取而代之，允许图片被直接渲染进材质。再后来，材质可以作为一个对未来渲染命令的输入起作用。

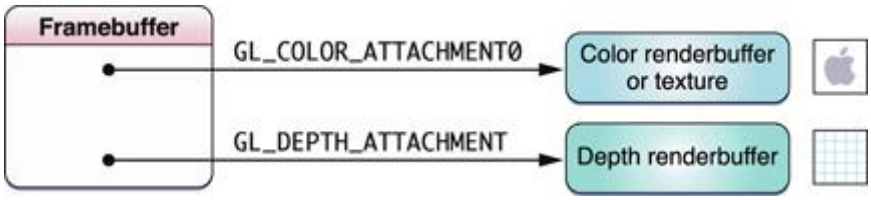


图 4-2 带颜色和深度渲染缓存的帧缓存

4.3 3D 机柜文件显示技术

4.3.1 OBJ 模型

在计算机平台上，常用的 3D 模型有 3DS、OBJ、MD2、MD3、MDL 等多种格式。这些格式在存储方式存在很大的差别，但基本思想大同小异。3D 模型又分为动态模型和静态模型。在动态模型中除了保存模型的定点和表面数据之外，还有与动画相关的信息。这些动态模型在渲染起来较为复杂。本课题采用简单的静态模型格式—OBJ 模型。OBJ 以纯文本的形式存储了模型的顶点、法线和纹理坐标和材质使用信息。在 OBJ 文件中，每行的格式如下：

前缀 参数 1 参数 2 参数 3 ...

其中，前缀标识了这一行所存储的信息类型。参数则是具体的数据。如表 4-1 所示 OBJ 文件的前缀有

表 4-1 OBJ 文件中的前缀

前缀	说明
v 表示本行指定一个顶点。	此前缀后跟着3个单精度浮点数，分别表示该定点的X、Y、Z坐标值
vt 表示本行指定一个纹理坐标。	此前缀后跟着两个单精度浮点数。分别表示此纹理坐标的U、V值

vn 表示本行指定一个法线向量。	此前缀后跟着3个单精度浮点数，分别表示该法向量的X、Y、Z坐标值
f 表示本行指定一个表面(Face)。	一个表面实际上就是一个三角形图元。此前缀行的参数格式后面将详细介绍。
usemtl	此前缀后只跟着一个参数。该参数指定了从此行之后到下一个以usemtl开头的行之间的所有表面所使用的材质名称。该材质可以在此OBJ文件所附属的MTL文件中找到具体信息。
mtllib	此前缀后只跟着一个参数。该参数指定了此OBJ文件所使用的材质库文件

4.3.2 长方体的 OBJ 实现

如下是长方体的 OBJ 实现：

```
mtllib ./Box.mtl
# object (null) to come ...
v -46.508743 -45.052959 50.796341
# 8 vertices
vt 0.000000 0.000000 0.000000
# 12 texture vertices
g (null)
s 2
f 1/10 3/12 4/11
f 4/11 2/9 1/10
s 4
f 5/9 6/10 8/12
f 8/12 7/11 5/9
s 8
f 1/5 2/6 6/8
```

```
f 6/8 5/7 1/5
s 16
f 2/1 4/2 8/4
f 8/4 6/3 2/1
s 32
f 4/5 3/6 7/8
f 7/8 8/7 4/5
s 64
f 3/1 1/2 5/4
f 5/4 7/3 3/1
# 12 faces
g
```

其中以“#”开头的表示注释，以 g 开头的表示组的前缀。这些前缀并不影响模型的外观。在一个 OBJ 文件中，首先有一些以 v、vt 或 vn 前缀开头的行指定了所有的顶点、纹理坐标、法线的坐标。然后再由一些以 f 开头的行指定每一个三角形所对应的顶点、纹理坐标和法线的索引。在顶点、纹理坐标和法线的索引之间，使用符号“/”隔开的。一个 f 行可以以下面几种格式出现：

1) f 1 2 3

这样的行表示以第 1、2、3 号顶点组成一个三角形。

2) f 1/3 2/5 3/4

这样的行表示以第 1、2、3 号顶点组成一个三角形，其中第一个顶点的纹理坐标的索引值为 3，第二个顶点的纹理坐标的索引值为 5，第三个顶点的纹理坐标的索引值为 4。

3) f 1/3/4 2/5/6 3/4/2

这样的行表示以第 1、2、3 号顶点组成一个三角形，其中第一个顶点的纹理坐标的索引值为 3，其法线的索引值是 4；第二个顶点的纹理坐标的索引值为 5，其法线的索引值是 6；第三个顶点的纹理坐标的索引值为 6，其法线的索引值是 2。

4) f 1//4 2//6 3//2

这样的行表示以第 1、2、3 号顶点组成一个三角形，且忽略纹理坐标。其中第一个顶点的法线的索引值是 4；第二个顶点的法线的索引值是 6；第三个顶点的法线的索引值是 2。

4.3.3 系统中的机柜 3D 显示

在系统中显示 3D 模型的主要代码如下：首先将要显示的 3D 模型转为浮点数组。然后调用 OpenGL 函数 glVertexPointer 将数组定义为顶点序列，最后用 glDrawArrays 命令绘出数组内容，用 glRotatef 实现图像的旋转。

```
[EAGLContext setCurrentContext:context];
    glBindFramebufferOES（GL_FRAMEBUFFER_OES, defaultFramebuffer）;
    glViewport（0, 0, backingWidth*3, backingHeight*3）;
        glClear（GL_COLOR_BUFFER_B|GL_DEPTH_BUFFER_B）;
    glMatrixMode（GL_MODELVIEW）;
    glBindRenderbufferOES（GL_RENDERBUFFER_OES, colorRenderbuffer）;
        glLoadIdenty（）;
        glColor4f（1.0,0.5, 0.1,   0）;
    rota+=0.5;
    glTranslatef（-0.6+24, 1.0+15.5, 10.0）;
    glRotatef（rota , -1.6, 0, -1.0）;
    glScalef（0.06138225542208073, 0.06138225542208073, 0.06138225542208073）;
        glVertexPointer（3, GL_FLOAT, 0, zallVerts）;
        glEnableClientState（GL_VERTEX_ARRAY）;
        for （i=0; i<zallNumVerts/4; i+=3） {
            glDrawArrays（GL_LINE_STRIP, i,3）;
        }
    glBindRenderbufferOES（GL_RENDERBUFFER_OES, colorRenderbuffer）;
    [context presentRenderbuffer:GL_RENDERBUFFER_OES];
}
```

其中函数 glVertexPointer(int size,int type,int stride,Buffer pointer) 用一个数组指定了每个顶点的坐标，其中参数意义如下：

size：指定每个顶点对应的坐标个数，只能是 2,3,4 中的一个，默认值是 4

type：指定了数组中每个顶点坐标的数据类型，可取常量:GL_BYTE, GL_SHORT,GL_FIXED,GL_FLOAT;

stride：指定了连续顶点间的字节排列方式，如果为 0，数组中的顶点就会被

认为是按照紧凑方式排列的，默认值为 0

pointer：指定了数组中第一个顶点的首地址，默认值为 0。

在 OpenGl 中所有的图形都是通过分解成三角形的方式进行绘制，函数 glDrawArrays (GLenum mode, GLint first, GLsizei count)提供绘制功能。当采用顶点数组方式绘制图形时，使用该函数。该函数根据顶点数组中的坐标数据和指定的模式，进行绘制。其中参数意义如下：

Mode：指定绘制方式， 可取三种值

1.GL_TRIANGLES：每三个顶之间绘制三角形，之间不连接

2.GL_TRIANGLE_FAN：以 V0V1V2,V0V2V3,V0V3V4……的形式绘制三角形

3.GL_TRIANGLE_STRIP：顺序在每三个顶点之间均绘制三角形。这个方法可以保证从相同的方向上所有三角形均被绘制。

以 V0V1V2,V1V2V3,V2V3V4……的形式绘制三角形

first：指定从数组缓存中的哪一位开始绘制，一般为 0。

count：指定数组中顶点的数量。

函数 glRotatef(float angle, float x, float y, float z)实现图像旋转，参数分别代表角度 x，y 和 z 的旋转。

4.3.4 OpenGL 3D 显示效果（模拟器）

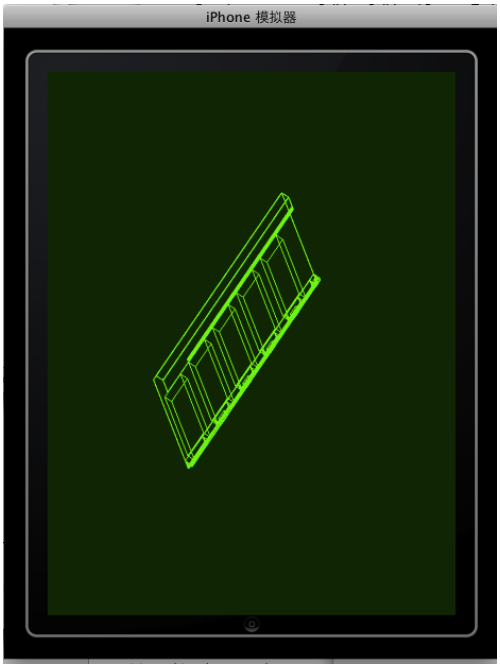


图 4-3 左侧板 3D 视图

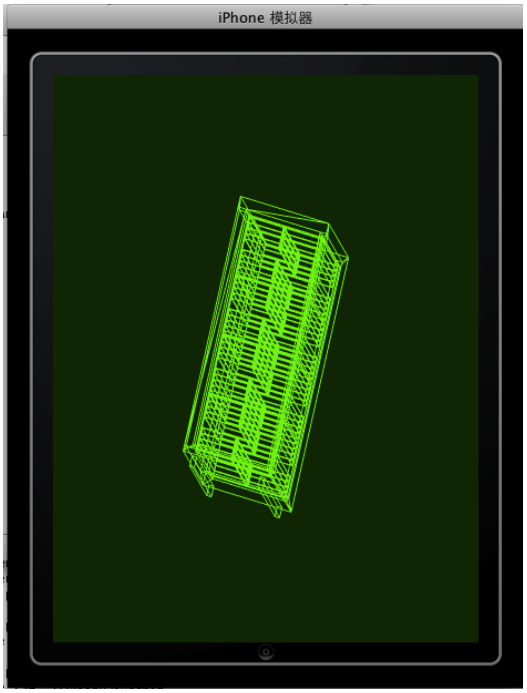


图 4-4　机柜 3D 视图

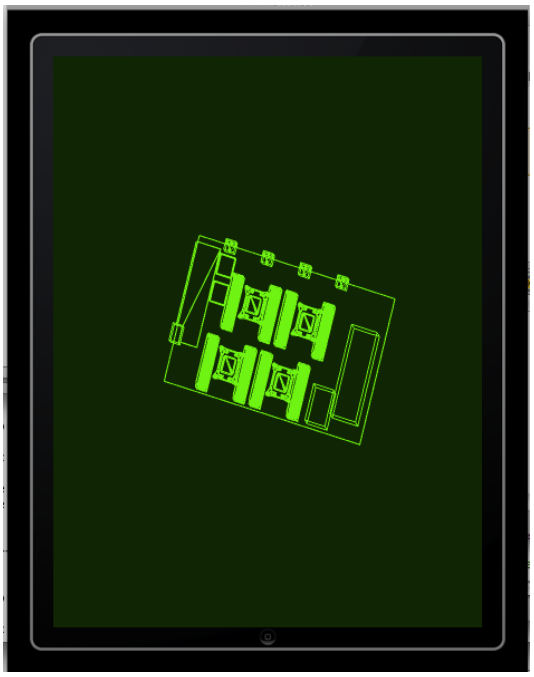


图 4-5　右侧板 3D 视图

结　论

本课题从实际应用需求出发，以人机交互原理为指导，结合面向对象的设计模式，基于 Xcode IDE 开发环境以及 iPhone SDK 的程序设计方法，实现了一个基于 iOS 的远程 3D 监控可视化系统的界面设计。

天河远程监控系统操作使用简单，实现了对机柜错误信息列表的维护、信息的发送与接收、实时接收系统状态等功能，系统主要特点如下：

1．系统运行稳定、安全可靠，最大限度的实现了易安装性、易维护性和易操作性。Objective-C 语言的可靠性使得本软件不仅可以在 iOS 上安全稳定的运行，而且安装方便维护简单操作容易。

2．界面设计美观，人机交互界面友好。本系统面向普通操作水平的用户，界面要求简洁实用，遵循常用的布局规则。本系统首次使用 Cocoa Touch 图形库实现图形界面的设计，用户可通过手势与系统交互，保证了系统的实用性和美观性。

3．可视化的 3D 显示。本系统利用 OpenGL 实现实时 3D 画面，用户可以从 3D 窗口直接看出系统的状态，提高了使用效率。

本文实现了远程 3D 监控可视化系统的界面设计，但是系统功能还可进一步加强。未来工作有如下几点：1、完善界面内容，优化人机交互。调整颜色搭配，让系统拥有更好的用户体验。2、将软件移植到 iPhone 平台。

参考文献

[1] Wikipedia. iOS introduction[DB/OL].http://en.wikipedia.org/wiki/ios,2012

[2] Apple Inc. Develop for iOS[EB/OL].http://developer.apple.cn/technology/ios,2012

[3] Shawn Welch. Form Idea to App: Creating iOS UI, Animations, and Gestures [M]. New Riders, 2011. 171-204.

[4] Joe Conway Aaron Hillegass. iOS Programming: The Big Nerd Ranch Guide 2nd Edion[M]. Big Nerd Ranch, 2011. 223-241.

[5] Weimeng Lee. Beginning iPhone SDK Programming wh Objective-C[M]. Wiley Publishing Inc, 2010. 6-16, 369-377.

[6] Carlo Chung, Pro Objective-C Design Patterns for iOS[M]. Apress, 2011. 55-74.

[7] John Ray, Sams Teach Yourself iOS5 Application Development in 24 Hours[M]. SAMS, 2011. 256-322.

[8] Mark Dalrymple, Scott Knaster. Objective-C 基础教程[M].高朝勤,杨越译.北京：北京邮电出版社,2009.32-56.

[9] Jack Nutting, Dave Wooldridge. iPad 开发基础教程[M].盛海燕,曾少宁,李光杰等译.北京:

人民邮电出版社,2010.55-154.

[10] Erica Sadun. iPhone 开发秘籍[M].第 2 版.张彩霞,高颖,易磊译.北京:人民邮电出版社,2010.154-253.

[11] Erik M Buck, Donald A. Yacktman. Cocoa 设计模式[M].陈宗斌,孔祥波译.北京:机械工业出版社,2011.98-104.

[12] Dave Mark, Jack Nutting, Jeff Lamarche.iPhone4 与 iPad 开发基础教程[M]. 漆 振,杨越,孙文磊译.北京:人民邮电出版社,2011.254-308.

[13] James A. Brannan, Blake Ward. iOS SDK Programming: A Beginner's Guide[M]. Mc Graw hill, 2011.159-184.

[14] Maher Ali. Advanced Mobile Development for Apple iPhone and iPod Touch[J]. A John Wiley and Sons, 2009(2):30-45

软件测评机构项目的配置管理

姓名：田辉

摘要：随着 CNAS CL45-2013 的发布，第三方软件测评实验室将配置管理正式纳入管理体系。相对于软件开发过程的配置管理，软件测试过程的配置管理有其自身的特点。本文从软件测评机构的立场出发，阐述了测试项目的配置管理过程及其活动。

关键字：软件测评机构，测试项目，配置管理

1、适用范围

本文所指的配置管理适用于软件测评机构的所有测试项目。

2、术语及定义

（1）配置管理：Configuration Management，简称 CM，是采用技术手段和行政手段进行管理和监督的一套规范化方法。它包括对配置项的功能特性和物理特性加以标识，并将其文件化；控制这些配置项特性的变更；记录和报告配置项的状态和变更实施情况；保证配置项的完整性、正确性和与需求的一致性；控制配置项的储存、装载和交付。

（2）配置项：Configuration Item，简称 CI，指配置管理中受控制的对象，是软件生命周期中创建的信息，包括各类管理文档、评审记录和文档、软件文档、源码及其可执行码、运行所需的系统软件和支持软件以及各种有关数据等。配置项是配置管理的最小单元。

（3）配置项标识：Configuration Item Identification，能保证配置项的可追溯性和唯一性的名称。

（4）开发库：Development Library，用于存放项目期间处于开发状态的相关文档和代码，以及存放项目组工作期间的相关沟通记录等，并为变更实施提供工作空间。项目配置管理员不对其中的内容进行控制，其中存放的内容由项目负责人负责。

（5）受控库：Controlled Library，建立以用来存放项目中通过评审形成基线的配置项的库。项目配置管理员对其中存放的内容负责。

（6）产品库：Product Library，建立以用来存放最终发布产品的库。组织级配置管理员对其中存放的内容负责。

（7）基线（Baseline），也称为里程碑(milestone)，是已经正式通过审核批准的某规约或产品，可作为进一步开发的基础，并且只能通过正式的变更控制过程

改变。

（8）变更管理：Change Management，是软件配置管理的一个要素，由评估、协调、批准或不批准以及对正式构造配置标识的配置项实施变更等活动组成。

（9）配置审计：Configuration Audit，指在配置标识、配置控制、配置状态记录的基础上对所有配置项的功能及内容进行审查，以保证软件配置项的可跟踪性。

3、配置管理基础

3.1 配置管理工具

软件测评机构根据自身的实际需要选择配置管理工具，商用软件、开源软件均可，所选择的配置管理工具须在《配置管理计划》中进行声明。必要时，制定配置管理工具的使用规程。

3.2 配置项标识规则

软件测评机构须规定配置项标识的命名规则，机构开展的所有测试项目的配置项命名均须遵守规则，配置管理员和配置库的使用者不得擅自变更配置项标识。

3.3 配置项版本号规则

版本控制的目的是按照一定的规则保存配置项的所有版本，避免发生版本丢失或混乱等现象。配置项的状态有三种："草稿"（Draft）、"正式发布"(Released)、"正在修改"(Changing)。

配置项的版本号规则与配置项的状态相关。

（1）处于"草稿"状态的配置项的版本号格式为 0.YZ，YZ 的数字范围为 01～99。随着草稿的修正，YZ 的取值应递增。YZ 的初值和增幅由用户自己把握。

（2）处于"正式"状态的配置项的版本号格式为 X.Y，X 为主版本号，取值范围为 1～9。Y 为次版本号，取值范围为 0～9。配置项第一次成为"正式"文件时，版本号为 1.0。如果配置项升级幅度比较小，可以将变动部分制作成配置项的附件，附件版本依次为 1.0，1.1，……。当附件的变动积累到一定程度时，配置项豹 Y 值可适量增加，Y 值增加一定程度时，X 值将适量增加。当配置项升级幅度比较大时，才允许直接增大 X 值。一般，控制库中配置项变更时，X 不便，Y 值适量增加；产品库中的配置项变更时，直接增大 X 值。

（3）处于"修改"状态的配置项的版本号格式为 X.YZ。配置项正在修改时，一般只增大 Z 值，X.Y 值保持不变。当配置项修改完毕，状态成为"正式"时，将

Z 值设置为 0，增加 X.Y 值。参见上述规则（2）。

3.4 配置库目录

配置库一般分为开发库、受控库、产品库三个物理配置库。开发库、受控库、产品库之间的关系如图 1 所示。其中，开发库存放测试产生的相关数据，包括测试数据和测试记录等；发布只针对产品库而言；产品库为组织级别的，由组织级的配置管理员管理，项目的配置管理员不参与管理。

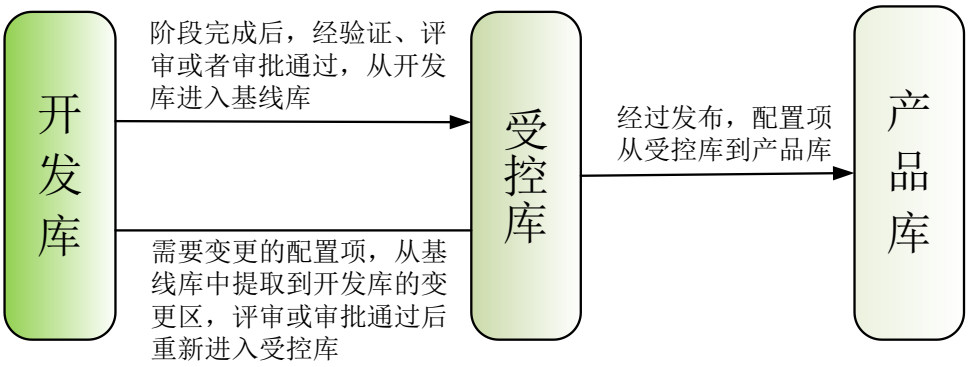


图 1 配置库间的关系

为更好的进行配置管理活动，软件测评机构可以规定使用统一的配置库结构、基线名称、配置项。项目实施过程中，允许根据项目实际定义的基线进行裁剪。项目的配置库目录结构须在《配置管理计划》中说明。

3.5 配置库权限管理

配置管理员使用统一的权限制定原则为配置库使用人员分配具体的权限。配置库的访问控制权限须在《配置管理计划》中说明。权限一般分为四种，如表 1 所示。

表 1 权限概要

Right	内容
R	Read，可以读取文件夹下的内容，不可以 checkin。
W	Write，可以读取文件夹下的内容，可以 checkin。
*	拥有文件夹得全部权限
	对于文件夹没有任何权限

权限制定原则如下：

规则 1：开发库完全向项目组开放读写权限，配置管理员不做管理。

规则 2：受控库纳入基线的配置项的写权限只能分配给项目组的配置管理

员。

规则 3：项目完全结束后，由技术主管授权配置管理员将项目受控库中的数据发布到产品库。

规则 4：产品库中的项目，由于某些原因需要进行修改，则将该项目的数据全部复制提交给项目的配置管理，并作为新项目建立配置库。

规则 5：一般，配置库的权限设置的原则如下：

① 开发库：项目组全员——R/W。

② 受控库：配置管理员——R/W，项目组全员——R。

③ 产品库：配置管理员——R/W，技术主管、测试部门主管——R，其他人员根据需要申请 R 的权限，测试部门主管审核，技术主管批准。

3.6 配置库安全机制

（1）项目组成员在获得配置库的访问权限后，应当第一时间修改自己的访问密码，并维护密码的安全性。

（2）各项目组所有配置库的管理员密码，在创建、修改后必须立即报告技术主管，报告内容包括密码的明文以及创建、修改生效的日期。

（3）每季度的最后一天，必须更换所有配置管理员密码，如果这四天恰逢假日，则顺延到假日结束后第一天内进行。

4、配置管理过程

4.1 选择配置管理员

组建项目组时，项目负责人指定项目的配置管理员。

在软件测评机构开展的测试项目，由机构任命的配置管理员负责配置管理工作。对于在客户现场开展的项目的配置管理，当客户方工作场所在软件测评机构所在地时，由配置管理员跟随项目组进驻客户现场，进行配置管理工作；当客户工作场所不在软件测评机构所在地时，由配置管理员授权该项目负责人进行配置管理工作。

4.2 识别基线与配置项

在项目策划阶段，项目负责人组织项目组成员确定项目基线；配置管理员依据项目的测试方案，结合软件测评机构对测试活动的规定识别项目的配置项、根据配置项命名规则的规定确定配置项标识、明确哪个基线下有哪些配置项。

4.3 制定配置管理计划

在项目策划阶段，配置管理员根据项目的基线和识别的配置项等，制定项目的《配置管理计划》。配置管理员将《配置管理计划》提交项目负责人审核批准。审核批准后，项目负责人将《配置管理计划》纳入测试方案，作为项目配置管理活动的指导性文件。

配置管理员为给定项目制订《配置管理计划》时，应该与软件测评机构的规定、可应用的约束、普遍接受的指南、项目的本质（例如规模和关键性）等保持一致。另外，一般还要考虑一些问题，例如组织与责任、资源与进度、工具选择与实现等，制订计划活动的结果记录在配置管理计划中。

《配置管理计划》覆盖的主要活动包括配置管理的软硬件资源、配置项标识、配置库规划、权限管理、配置控制、配置状态报告、配置审计、配置库备份、基线发布管理等。

4.4 建立配置库

配置管理员负责依据审核通过的《配置管理计划》，参考软件测评机构统一的配置库结构、基线名称、配置项，建立各级配置库目录结构。

4.5 分配配置库权限

配置管理员在建立完配置库后，依据相关人员在项目中所担当的角色，《配置管理计划》中人员权限的规定分配管理权限。

4.6 配置管理培训

如有必要，在项目策划阶段由配置管理员为项目组成员进行项目配置管理规范、配置管理工具的使用等配置管理培训。

4.7 配置状态报告

配置状态报告是配置管理中一个很重要的活动，该活动贯穿整个项目周期。配置管理员使用《配置项状态记录表》对配置项的状态进行跟踪记录。《配置项状态记录表》的内容包括但不限于：配置项所属基线、配置项编号、配置项名称、最新版本号、作者、存放位置、配置项状态及状态标识日期、变更次数、预计纳入基线时间、实际纳入基线时间等。

4.8 发布基线

（1）配置管理员及项目组成员根据《配置管理计划》判断项目是否达到里程碑点，如果达到里程碑点，配置管理员填写《配置项入库申请表》。

（2）项目负责人组织相关人员对《配置项入库申请表》中需要纳入基线的配置项进行评审或审核，判断是否达到发布标准。

（3）如果达到标准，项目负责人授权配置管理员发布基线。

（4）配置管理员从开发库里提出待发布的配置项，并将其纳入受控库中的相关基线。

（5）配置管理员通知所有项目组成员基线发布信息。

（6）配置管理员更新《配置项状态记录表》中配置项的状态。

4.9 配置审计

4.9.1 配置审计的时机

（1）一般基线发布前对要进入基线的配置项进行功能审计。

（2）如果两条基线的发布时长超过一个月时，应该在两条基线发布中间适当安排物理配置审计，建议是 1 个月至少 1 次。

（3）项目结束发布到产品库前，必须要进行物理配置审计。

4.9.2 配置审计的分类

配置审计类型一般包括功能配置审计和物理配置审计两种。

（1）功能配置审计是验证一个配置项的实际内容是否符合它的需求的一项审查，以便建立一个基线。即通过对配置项内容的评价，鉴定配置项是否达到了所规定的要求。功能审计由项目负责人审核或组织项目组成员评审的方式进行。

（2）物理配置审计是对照《配置管理计划》检查已纳入基线的某个配置项，即通过检查纳入基线的配置项的版本、基线与配置项的一致性，保证配置项更改的完整性。物理审计由配置管理员执行，审计的内容包括配置项是否已经提交、配置项的命名规则版本是否正确、配置项是否已经进入正确的库和目录、配置项的访问控制权限是否正确、变更后的配置项是否归并等。物理配置审计的结果记录到《配置审计检查表》中。

4.10 配置项变更

4.10.1 变更申请

配置项如果需要发生变更时，项目组成员填写《配置项变更登记表》，描述变更来源、变更依据、变更的配置项标识及版本号、变更原因、变更内容、变更的影响分析、申请人、申请日期等。

4.10.2 变更审批

（1）项目组成员将《配置项变更登记表》提交项目负责人审核，技术主管批准。审核批准的过程要确保这些变更与所有技术需求和项目需求一致；对变更影响的评价要紧密结合项目或合同需求。

（2）项目负责人如果同意变更，需在《配置项变更登记表》中注明同意并签字，还需注明批准变更的配置项及版本、变更的执行人、变更的完成时限；如果不同意变更，则需在《配置项变更登记表》注明不同意及不同意的理由，并签字。

（3）项目负责人同意变更后，将《配置项变更登记表》提交技术主管批准。技术主管如果同意变更，需在《配置项变更登记表》中注明同意并签字；如果不同意变更，则需注明不同意及不同意的理由，并签字。

（4）项目负责人和技术主管均同意变更后，项目负责人持《配置项变更登记表》，授权配置管理员将需要变更的配置项检出到开发库。

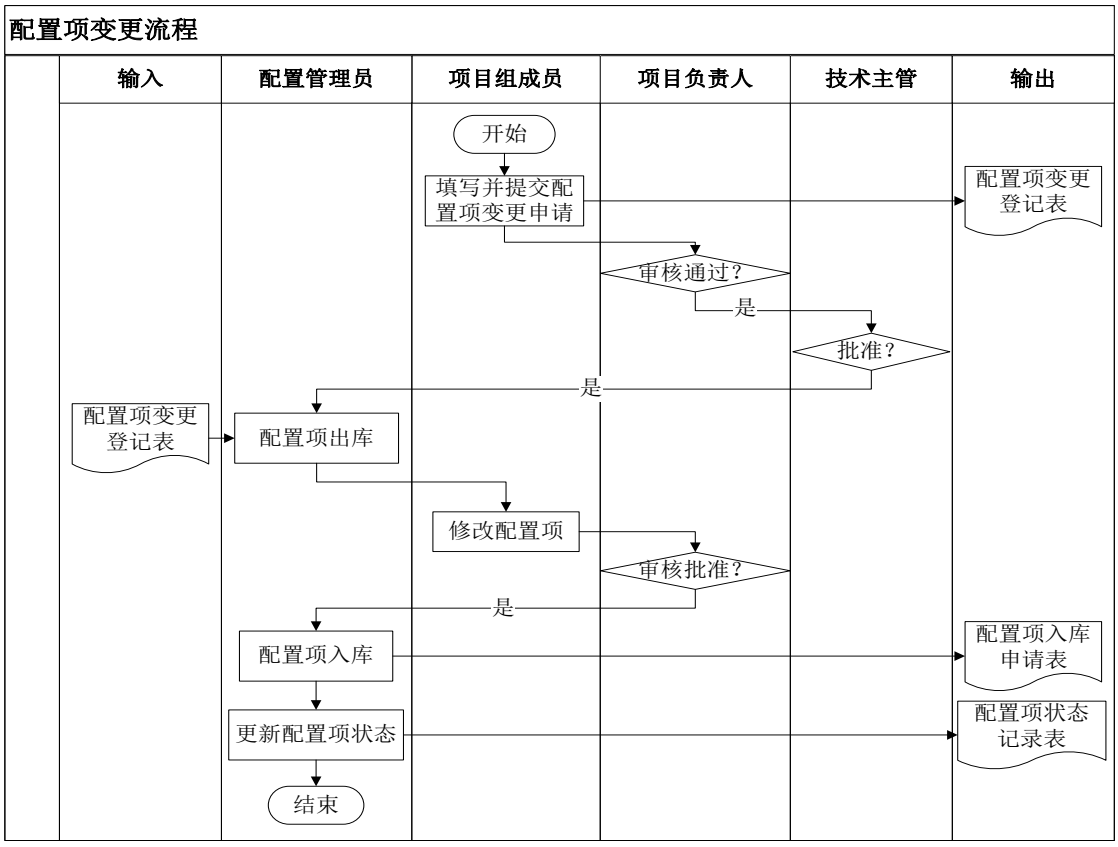


图 2 配置项变更流程图

4.10.3 变更执行

（1）项目负责人根据审核意见中的安排，组织相关人员执行配置项变更。变更执行人根据安排执行变更，并向受影响的各方报告情况。

（2）配置项变更后，其版本号须按照本文“3.3 配置项版本号规则”变更。

（3）配置项变更完成后，变更执行人在《配置项变更登记表》中记录变更信息，包括变更后的配置项标识及版本号、配置项变更的内容、变更执行人、变更完成日期。

（4）变更的配置项重新纳入基线前，由项目负责人组织人员对其进行审批，

判断变更的配置项是否符合标准，并在《配置项变更登记表》中填写审批意见。

（5）变更的配置项审批通过后，项目组负责人授权配置管理员重新纳入基线。

（6）配置管理员从开发库里提出变更后的配置项，并纳入基线。

（7）配置管理员将配置项变更情况通知所有项目组成员。

（8）配置管理员更新《配置项状态记录表》中配置项的状态。

4.11 配置库备份与清理

（1）配置管理员定期清除配置库里的垃圾文件。在清除配置库中的过时版本时，必须确保历史上对其进行的完整备份的有效性。

（2）每周五下班后，配置管理员对配置库进行全目录备份。

（3）项目结项以后，配置管理员将项目配置库压缩后的备份文件刻录成光盘，备份光盘上须注明以下信息：项目编号、项目名称、光盘所含具体配置库名及其完整备份日期、此次备份共几张光盘、以及每张光盘在本次备份中的序号。光盘交项目负责人，由项目负责人将所有项目资料交档案管理员存档。

（4）配置管理员整理受控库中的配置项，将整个项目作为产品发布到产品库。

（5）完成项目的产品发布后，配置管理员清空配置环境，将项目的配置库删除。

5、相关记录

（1）《配置管理计划》

（2）《配置项状态记录表》

（3）《配置项入库申请表》

（4）《配置项变更登记表》

（5）《配置审计检查表》

参考文献

（1）《GBT 11457-2006 信息技术 软件工程术语》

（2）《配置项版本号规则》http://www.cnitpm.com/pm/7363.html

（3）《配置库管理规程》百度文库

作者简介：田辉，软件工程硕士，现担任国家网络软件产品质量监督检验中心（济南）的质量助理、配置管理员，具有多年软件研发与软件测试工作经验。

软件测试：哲学视角——关于测试我们知道什么

作者：殷亮

博观而约取，厚积而薄发

从直观上说所有真正的检验都是试图给理论“下绊子”：它不但是一场严格的考试，而且是一场不近人情的考试——其目的在于使考生们落败考场，而不是给考生们以机会去展示他们所知道的东西。展示所知的态度是那些打算去确证或“证实”其理论的人的态度。

——卡尔·波普尔(Karl Popper)[①]

证实与证伪

人们通常把软件测试视为一种归纳并推论过程，尤其是在功能性测试（functional testing）方面，我们借助边界值、等价类、决策表等测试设计方法，通过设计与执行部分用例来推断整体。比如，对于一个只接受 1～255 的输入，如果按照边界值的方法，我们选择检验 1、2、128、254、255，即可推断程序是否运行正确。也就是说在这里说我们通过观察 1/51 场景的测试结果，来推论 50/51 未经测试场景的结果。

这样的测试必然是不完备的，从逻辑的观点来看，显然不能证明从单称陈述（某个场景没问题）中推论出全称陈述（所有场景都没问题）是正确的。就像我们不能从单称陈述“昨天太阳从东方升起”或“今天太阳从东方升起”来推论全称陈述“每一天太阳都从东方升起”一样。在上例中，实际上我们是可以对 255 个场景进行完备的测试的，但对于绝大多数需要测试的软件系统，测试工作者面对的往往是无穷的场景。现在让我们考虑一个很简单的功能：用鼠标点击屏幕上的一个按钮，鼠标按键松开后系统弹出 OK 的提示。在不检查实现代码的情况下，我们并不能保证在按钮的不同坐标位置点击后会出现同样的结果，也不能保证从按下到松开的不同延时会出现同样的结果，另外，即使是同样的坐标、同样的延时，要预期第一次点击与第二次点击会出现同样的结果似乎也是武断的。尽管这些条件看上去有些不切实、甚至带有诡辩的味道，但从实现的角度来讲，这些都是有可能的：因为编程人员完全可以在代码中人为地设置一个让人意想不到的复杂条件，来触发一个后门、或者一个非常规事件——但它却是再多的测试都不一定能发现的。因此，不管是操作步骤与操作环境的复杂组合，还是道德风险的可能性，都使得能证明软件有缺陷的潜在场景已近乎无限大。这也就意味着：即使只是单纯地针对软件中某一个功能做完全验证，也是几乎不可能的。

而在早先的测试文献中我们也不难找到类似的论述，比如在 Weinberg 1986 中，杰拉德·温伯格（Gerald Weinberg）就指出我们无法测试所有的可能：“首

先，人类的大脑不仅会犯错误，而且它的容量也是有限的。其次，没有人的生命可以无限长。所以无论我们多想进行所有可能的测试，也无法全部考虑到；即使全部考虑到了，也无法活得足够长来将它们进行完”[②]。而在 Patton 2006 中，罗恩·巴顿（Ron Patton）也强调了完全测试程序是不可能的，主要原因是“输入量太大；输出结果太多；软件执行路径太多；软件说明书是主观的”[③]。在这些早先的文献中，都如上文一样，强调了因为操作组合与环境组合的难以穷尽给质量证明所带来的困难。然而从哲学的视角看，导致潜在场景难以穷尽的主要因素还并非海量的操作组合与环境组合，而是时空。因为从时空的角度看，即使是对一特定操作之结果的论断也是一个全称陈述。也就是说诸如“在设备 A 上运行程序，并在按钮 B 的坐标（15，15）上点击鼠标，系统显示结果 C”这样一个陈述实际上就已包含了无穷多个可能的场景，因为它断定该陈述在一切时空中都为真。但这显然是无法证实的，因为即使我们昨天观察到它为真、今天观察到它为真，甚至过去数百次观察都是如此，我们也仍不能断言在下次观察中必然为真。因此我们可以说，近乎无穷的操作组合与环境组合使得我们事实上无法证明软件的完美，但真正无穷的时空因素却是使得这类证明在理论上就已不可能的根本原因。

软件测试并不能证明软件是完美的，不管受测的软件是多么简单都是如此，当我们从时空的维度来重新审视软件时，我们感受到了归纳的无力，在实证领域，不存在所谓的归纳。就像观察到数以亿计的人说汉语，也无法确保所有的人都说汉语一样——在时空的无限论域中，一种假设哪怕是得到了再多的经验证据支持，其逻辑概率也永远趋于零。所以，我们不可能通过测试来宣称或者断言软件的完美。当然，这并不是说世界上不可能存在完美的软件，而是只说——我们有可能开发出了完美的软件而不知。

与证实全称陈述的复杂性和不可能相比，要证伪一个全称陈述却是相对的简单，因为尽管有再多的观察也不能推论出全称陈述为真，但反过来，只要存在一例观察不为真我们便推倒了整个全称陈述。这就是哲学家卡乐·波普尔（Karl Popper）所说的证实与证伪之间存在的“令人震惊的不对称”：“单一的观察陈述集合有时可以证伪或反驳一个全称定律，但它不可能在确立定律的意义上证实定律。这个事实的另一个表述为：单一的观察陈述集合可以证实一个存在陈述（这意味着证伪一个全称定律），但无法证伪该存在陈述。这是基本的逻辑情境，它表明了一种令人震惊的不对称”[④]。当我们证伪一个全称陈述时，实际上是证实了“存在着某一例观察结果与全称陈述与推论不符”。在软件测试中，如果我们拒绝“某软件是完美”这一论述，那么必然是我们证实了在该软件中至少存在着一例缺陷。我们将诸如“软件存在缺陷”、“上帝是存在的”与“世间有黑天鹅”等

称之为存在陈述，这些陈述在形式上都是存在主义的。与全称陈述的不可证实但可被证伪相反，存在陈述是可证实但不可被证伪的。在理论上，我们要证实这些陈述是有可能的（至少是潜在可能），但却无法否定它们。也就是说，即使某人终此一生都未目睹过上帝，又或者我们之前所见到的一万只天鹅全都不是黑色的，我们也不能反驳它们的存在；与之相反，哪怕是只要有一例观察与陈述相符，这些陈述就得到了证实。因此，在关于软件缺陷的论述中，我们可以证实某个软件确实存在缺陷，但却无法证伪它：即证明其“不存在”缺陷、或者认为它是完美的。事实上，我们甚至不需要进行任何的测试，也能大胆地去评价任何软件、断定它们“存在缺陷”，而不用担心会遭到何种反驳——因为存在陈述是不可证伪的。

这是一个有趣的结论。如果软件测试工作者的任务是判断软件是否有缺陷的话，那么可以说这份工作既是最容易的，也是最困难的。它的容易之处在于我们总是可以无任何后顾之忧地给出存在缺陷的判断，而困难则是我们永远也无法通过努力来证明软件无缺陷。在现实世界中，人们往往会寄望测试能证明软件的完美，或者让测试工作者确保此时此地检验了的测试场景，在彼时彼地也能按预期执行，但这却恰恰是测试最不可能完成的任务。

测试与决定

测试工作者在测试过程中所得到的发现与知识，实际上既不足以证明软件的完美，也不足以决定后续的行动。测试是一种批判、一种实验，如果说实验是一切知识的试金石，那么测试就是我们对软件所宣称的质量的试金石。测试能告知我们的是：软件经受了何种广度与强度的检验，以及它在什么场景下存在着缺陷，但也仅此而已。我们通过测试来获取信息，而非给出预言。相反，测试恰恰是对来自开发人员对软件质量的预言（这个软件没问题）的检验与批判。正如在物理学中的分工那样（理论物理学家进行想象、推演和猜测新的定律，但并不做实验；而实验物理学家则进行实验、想象、推演和猜测），测试工作就是想尽一切办法去寻找问题、去让软件失败，除此无他，与证明扯不上半点关系。努力去维护受测系统、去证明我们的软件是多少的正确，这从来就不是可取的测试精神，测试工作者永远不会说受测软件是无缺陷的，而只会说这软件目前通过了眼下的检验，我们还没有发现否定它的理由。然而这一肯定的判决也只能是暂时的，我们仍对随后的否定判决开放。因此，测试可以看作是一个探索的过程、一个批判的过程，它获取信息，而不做出决定。测试不负责决定软件是否可以发布、也不负责决定我们是否要取消某个项目，甚至都无法决定接下来是否还需要进一步的测试。测试过程中所产生的信息，对于这些决定来说是重要的、也是关键的，但我

们也必须承认，这些信息并不充分。

如果说测试永远是不完备的，只能告知我们当下软件经过了何种程序的检验，那么，对于软件发布后它在客户处的表现、会不会出现问题、以及会出现什么问题，理论上我们都只是推断与猜测。在这一过程中，我们通过已检验的场景来推断未经检验的场景，通过这一时空的检验结果来推断另一时空的执行结果。在做出是发布软件还是继续测试的决定时，实际上我们面临着一种两难，因为这里面存在着两类可能的错误：第一类错误是软件还有未找出的缺陷但我们过早地发布了，而另一类错误则是推迟发布但没并没有找出新的缺陷。但是遗憾的是，在这两类可能的错误之间，我们别无选择，只能选择尽力避免其一，而不能选择两者都避免。因为我们越是试图控制一类错误的发生概率，反过来就越是加大了犯另一类错误的可能性。

在统计概念中，这称之为假设检验。在进行假设检验时，我们将设置一个假设为原假设，用H_0表示，在这里它表示

H_0：软件还存在可能找出的缺陷

第二个假设称为备选假设或研究假设，用H_1表示。在决定是否应该继续测试时，它表示

H_1：软件不存在可能找出的缺陷

如此一来，我们设置了两个互为矛盾的假设，在真实世界里只能有其一为真。然而，就像前文中所提到的，我们有可能开发出完美的软件而不知；因此一般来说，我们并不知道、也不能断定哪个假设是正确的，我们只能猜测。当我们进行假设检验时，实际上会存在两种可能的错误。第一类错误是当原假设正确时，我们却拒绝了它。第二类错误被定义为当原假设有错误时，我们却没有拒绝。在上面的例子中，第一类错误就是进一步测试还能发现更多的问题（H_0为真），但我们却匆忙发布了软件（拒绝了H_0）；而当我们为了保险进行了一些冗余测试（拒绝了H_1），但并没有发现问题时（H_1为真），第二类错误就发生了。我们把发生第一类错误的概率记为α，通常它也被称作显著水平；第二类错误发生的概率记为β。发生错误的概率α和β是相反的关系，这就意味着任何尝试减少某一类错误的方法都会使另外一类错误发生的概率增加。在决定是发布还是继续测试时，如果我们试图减少第一类错误（还能进一步找到缺陷，但没有继续测试），我们就必须进行更多更完备地测试，但这一决定无疑就增加了犯第二类错误（做了更多的测试，但没有进一步发现缺陷）的可能性。反过来也是如此，当我们修复一个局部缺陷而不进行完整测试的时候，实际上我们是希望避免发生第二类错误的可能性，但这样做我们就肯定增加了犯第一类错误的可能性。

第一类错误（匆忙发布软件）的成本是客户故障成本、商誉蒙尘或法律风险

等，而第二类错误（推迟软件的发布）的成本则可能是冗余的测试成本、商机延误或未能尽早地得到市场反馈。有些产品犯第一类错误的成本较高，比如航天软件与医疗软件，因此人们往往会进行更多更完备的测试，宁愿做了许多事后看来无谓的测试、犯第二类错误，也要全力避免第一类错误发生。另外一些软件则相反，比如互联网产品、或是一些市场处于生命周期早期的产品，在这些领域，商机往往是瞬息万变、稍纵即逝的。在此若是因为犯第二类错误而导致商机延误，与因第一类错误引起的软件故障相比可能会严重得多。因此，综合来看，需要做多完备的测试、何时发布软件?这实际上是判断哪一类错误更严重的问题，不同的行业会有不同的选择，不同的竞争者也会有不同的选择，而测试本身并不足以做出这一选择。这里面涉及到的商业价值与成本评估问题，是市场工作者的领域，测试可以提供信息，但绝不能越俎代庖。从上述角度推论，我们可以认为在软件领域里，单从测试角度是不足以决定软件是否可以发布的，同时也不存在通用的软件发布标准。

测试与系统

软件系统无疑是复杂的，作为一个复杂系统，我们无法证明其正确性。这一点与自然界规律是相似的，也正因为自然界的复杂性与不可知，对于自然科学的规律，我们只能证伪不能证实。然而，尽管同是因为复杂性导致了软件行为与自然界行为的不可证实，但这两类系统的复杂性来源与复杂程度却是不同的，认识到这点将有助于我们进一步理解软件测试。经济学家肯尼思·博尔丁（Kenneth Boulding）根据复杂程度，将系统分为九大类型⑤：

1．静态架构	拥有静态结构，如晶体里原子的排列或动物的骨骼。
2．时钟结构	具有预定运动模式的简单动态系统，如钟表和太阳系。
3．自调系统	具有根据某些外给目标或准则自我调节的能力，如恒温器。
4．开放系统	能够通过从环境获取资源进行自我维系，如生物细胞。
5．衍生系统	不是通过复制，而是通过含有成长指令的种子或卵进行繁衍，例如鸡和蛋系统。
6．内像系统	具有获取信息详尽地认识环境的能力，以及把这些认识组织成关于环境整体形象或知识结构的能力，这是动物能力所能达到的层次。

7．抽象系统	具有自我意识，能够运用语言，人类行为处于这一层次。
8．社会系统	由具有第 7 层次能力并遵循共同的社会秩序与文化的行动者组成。
9．绝对和必然不可知的系统	

博尔丁的分类给我们许多启发。首先，它展示了世界上系统存在的广泛性与多样性。其次，我们在上述分类中可以看出软件系统与自然系统之间的不同：它们分别属于第 3 层与最为复杂的第 9 层。

作为第 9 层的自然系统，其复杂性在于，我们对于它内部机制与目标完全不可知。无论科学如何发展，科学都不是一个确定的或即成的陈述系统，也不是一个朝着一个终级状态稳定前进的系统，而仅仅只是关于真实世界的猜想。已故的诺贝尔物理学奖得主理查德·费恩曼（Richard Feynman）有一个关于自然系统的生动比喻：“可以作一想象：组成这个世界的运动物体的复杂排列似乎有点像是天神们所下的一盘伟大的国际象棋，我们则是这盘棋的观众。我们不知道弈棋的规则，所有能做的事就是观看这场棋赛。当然，假如我们观看了足够长的时间，总归能看出几条规则来……除了我们还不知道所有的规则外，我们真正能用已知规则来解释的事情也是非常有限，因为几乎所有的情况都是极其复杂的，我们不能领会这盘棋中应用这些规则的走法，更无法预言下一步将要怎样。” ⑥大自然的复杂性与内在机制的不可知使得我们一切有关这个世界的知识都只是猜测，而证实与证伪之间惊人的不对称又使得我们即使找到了真理的时候也无法知道这一点。因此对于自然科学，波普尔说：“我们不知道，我们只能猜测”。

与自然系统相比，软件系统有着与之截然不同的复杂性来源。作为人类科技与意志的产物，我们并非对其内部机制与目的不可知，相反，是我们设计了它的内部机制与目标，可以说，我们是它的造物主。但我们人作为造物主，与自然系统万能的造物主相比，又是极为渺小和有限的。在构造一个有大量执行路径的软件系统的过程中，我们难免会想当然、会忘记、会迷糊、也会疏忽，甚至还有可能存在道德风险，这些都使得我们不应当将软件视为一个必然会遵从设计与预期的系统，它是复杂的、不确定的，因而也是需要测试与批判的。一方面，是人无与伦比的智慧赋予了它遵从预期行动的能力，另一方面，也是人不可避免的局限性为其埋下了可能偏出预期的种子。也就是说，其行为的不可预知不在于机制的不可知，而是我们并不能保证我们能完全不出错地将设计的机制转换为软件所预期的行为。正是因为这种结果的不可保证性，我们才需要测试——我们设计了内部机制但无人能保证它的实现。

这就是软件系统的两面性，一方面，我们不能因为知其既定的机制，就确信它一定能按照这些机制来实现其行为，因而我们需要怀疑、需要忘掉这些机制去测试它的各种输入与输出，黑盒测试在任何时候都是必不可少；另一方面，我们又不能将其看作更高复杂层次的系统，假装我们对其内部机制一无所知。如此一来，我们实际上是完全放弃了原本可以从内部理解其行为的便利，而这样的测试设计必然是欠缺效率的，因而从白盒的角度分析软件系统也是同样必不可少。

今天，关于软件的复用，我们已经取得了长足的发展。从函数级复用到组件级复用、再到框架级复用，软件复用一方面缩短了软件系统的开发周期，另一方面也减少了缺陷的引入。显然，如果我们不关心软件的内部机制的话，那么测试人员将不能从软件复用中得到任何好处：因为站在黑盒的角度，我们不能对软件的实现作任何假设，理论上我们必须测试一切窗口控件，即使这些窗口控件是复用的，早已经过了千锤百炼；我们也必须测试一切底层 API，即使这些底层 API 都是标准的系统调用。但是，如果我们能正视这些机制、利用这些机制的话，我们的测试将更有针对性，也更有效率。我们不应该认为测试应该从零开始，就像我们不应该认为一切科学理论都应该从零开始一样。在科学的各个领域里，往往都只存在着少数的一些基本的公理，在这些少数公理的基础上，科学家们借助大量的次级理论构建出了巍巍的科学大厦。因此，绝大多数的软件系统与科学理论一样，实际上都不是孤立地存在的，它们都包含了一些暂时经受了批判的部分。当我们检验系统时可以选择将这一部分看作是背景知识，暂时地接受，而将精力更多地集中在检验那些更脆弱、更容易发现缺陷的部分。

小结

在经典著作《人类理解研究》中，哲学家大卫·休谟曾如此设问并回答：“我以前所食的那个面包诚然滋养了我，那就是说，具有那些可感性质的那个物体在那时候，是赋有那些神秘的能力的。但是我们果能由此推断说，别的面包在别的时候也一样可以滋养我、而且相似的可感性质总有相似的神秘能力伴随着它么？这个结论在各方面看来都不是必然的。” ⑦休谟因而指出了归纳的局限性，即我们不能因为曾经看到那样一个物象总有那样一个结果伴随着它，就可以预测相似的物象也会有相似的结果。在大自然中，这种不可预测性是因为自然系统的不可知。而对于软件测试，尽管我们知道不少，但我们也不能因观察到软件系统的某一功能在此时此地没问题，因而断定在客户处也必然是没问题的。在要求对事物做出准确预测的时候，只要我们没有把握到事物的必然性，那么不管是对事物内部有一知半解还是完全一无所知，实际上是没有区别的。因此，在探讨关于测试我们知道什么这一问题时，我们大可将软件视为实现上不可知的系统，这是测

试的本质，也是测试工作的起点。然而，当我们探讨如何去测试与批判一个系统时，则将软件系统视为在一定程度上是可知的又是非常有必要的，因为这是关于效率的问题。因此，我们如何看待受测系统，取决于我们需要回答或解决的是哪类问题。但无论如何，软件测试都是批判而非证明：从不可知的观点出发，我们不能证明；而从有限可知的观点看，又有太多的东西可以不去证明。忙乱地写着冗余的测试用例的测试人员，就像是在路灯下寻找车钥匙的人一样，我们不能说他错了，但他肯定不可能是卓有成效的。优秀的测试工作者是将测试看作是批判、是艺术的人，他们勇敢地承认自己的局限性，卸下了证明的重负，站在巨人的肩上，然后看得更远。

注释

[1]. Karl Popper. 《Realism And The AIM of Science》. 1983.

[2]. Gerald M. Weinberg. 《Perfect Software: And Other Illusions about Testing》. 1986.

[3]. Ron Patton. 《Software Testing》. 2006.

[4]. Karl Popper. 《Realism And The AIM of Science》. 1983.

[5]. Kenneth E. Boulding. 《General System Theory: The Skeleton of Science》. Management Science, 2. 1956.

[6]. Richard Feynman. 《The Feynman Lectures On Physics, Volume I》. 1963.

[7]. David Hume. 《An Enquiry Concerning Human Understanding》. 1927.

提高测试质量，从提高代码质量抓起

作者：郝闯

我们都知道，在测试的过程中，"黑盒测试"仅仅从外围来获得测试对象的质量情况，而这往往是不够的，所以有公司也引入"白盒测试"。但是，说到"白盒测试"这件事情，在业界，恐怕只有大公司对于一些特殊业务去做，而大部分的中小型公司在力度方面还有待加强。关于是否做"白盒测试"这一点我还特意调研过一些圈子里的朋友，从我调研的结果来看，几乎都没有做这个。为什么？互联网日新月异，需要不断适应新的变化，讲究小而美，做白盒测试，需要的人力、时间、资源等成本都很难耗得起。

从"白盒测试"的一些常见的覆盖方法来看，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。从"白盒测试"的方法来看，代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、域测试、符号测试、Z 路径覆盖、程序变异...看看这些，要做起来，需要花费的精力和代码量是开发的好几倍，而项目往往会延期，开发的代码都完成不了，更别提测试代码了。

或许你说没关系我们可以加班来写，但是等你辛辛苦苦写完一大片的白盒测试代码，结果 PM 的一句话，需求又变了。业务大变化，你写的测试代码很多又和业务强相关，复用度不高。眼睁睁看着自己努力的成果没有价值，你又有何感想，怎么办，让产品那边不要调整需求？那是不可能的，产品的一些变更是由市场驱动，尤其是现在的移动互联网时代，一切唯快不破，市场不断变化，产品也一样，快速出产品，快速测试，快速响应市场；

那妥协吧，不做代码级别的测试？但是，那产品的质量方面又怎么有把握？你认为就通过外围黑盒测试一下，你放心吗？不做代码级别的，就测试一下黑盒算了；但是，如果是一个有责任心有点追求的测试人员，你又放心吗？

你说，那好吧，我们可以申请更多的人力资源来做？算了吧，开发人员都不够；申请延期，申请更多的时间？等按照延期的时间，等产品出来了，寿命也就耽误了一段时间，用户需求又变了，该下架了；那到底怎么处理，是不是很纠结？

其实这些烦恼相信大家都会遇到，只是怎么处理。选择一些好的方法，能让效率提升，并起到事半功倍的效果。下面我们之前以及现在的一些实践做一些简单的分享，欢迎各位指正。有人指出来我们的问题，再去优化；在我看来，代码级别的测试，是一定要的，否则，我拿出一份测试报告的时候，我是没有一定信心的。但是代码级别测到什么程度，这个可以根据实际来决定。因为我们的目标

很明确：**通过提高代码的质量，提高测试质量，从而提高产品质量**。

而如何编写高质量的代码？这是不论开发还是测试人员都需要考虑的问题。只要我们是在 Coding，那么代码的质量就很重要。建议的方法是“拿来主义”，借助业界的开源框架、开源工具，来帮助我们的测试工作。**你可以通过持续集成+自动化构建+自动化部署的方式来提升你的效率。把繁杂的事情交给工具，解放人的双手和大脑，把时间重点花在核心业务与测试分析与设计中去。通过 Code Review，或者 CI 工具中的一些插件可以检查到一些代码中的问题，对提高代码质量很有帮助。**而在实际中，不论是否做 CI，都可以从以下角度来考虑提高代码质量。对于没有做过的同学来说，可以通过而下面这些角度去尝试一下。

1.编写规范的代码

遵循规范，是代码优雅起来的第一步。而让代码优雅遵循规范的重要性我就不再这里强调和描述了。编写规范代码一般需要看这几点：

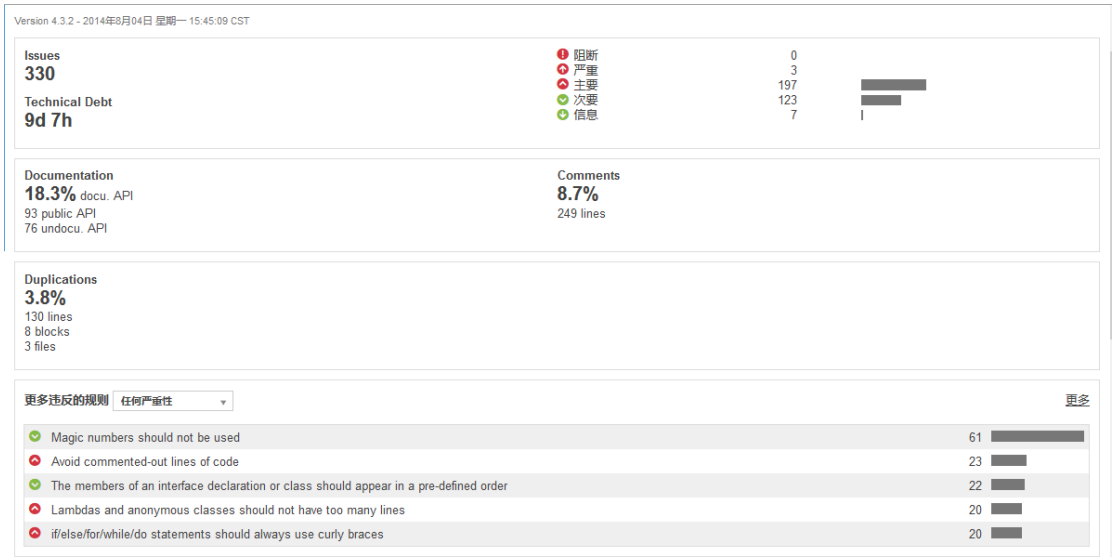
源代码编写规范：这些代码规范其实网上都是开源可以获得的，比较典型的像 Google 的四大语言规范 C++、Java、Python、JavaScript；

Google C++代码规范：
http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

Google Java 代码规范：
http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

代码命名规则：一个好的命名往往会让别人在阅读代码的时候，一看方法、函数、类的名称，就能基本猜测出来意图；比如一般的操作，createXXX、deleteXXX、updateXXX、getXXX；大小写驼峰式命名，不要给一些特殊的简写，或者过长的方法名等；

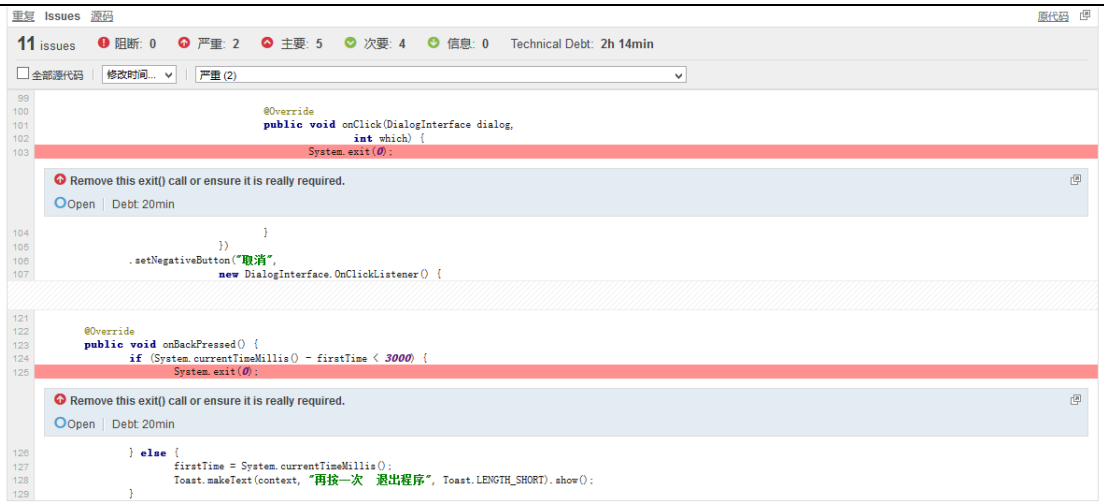
代码编写风格：代码编写风格一致好处不再强调，但可以让多个人相互之间 Debug 时候，根据规律，易于找到那块会有 log，那块会有输出，并且易于维护；



2.编写正确的代码

编写正确的代码实际要包含三点：确定代码功能、确定正确的输入和输出、确定正确的处理过程；

代码功能是否正确，输入输出是否正确，这个和业务相关了，要求开发人员在写代码的时候，一定要了解业务细节，当然这也考验 PRD 或者 SRS 的质量。个人一直认为一个产品经理的水平对团队有很大的影响，而自从"人人都是产品经理"这个概念被抛出来之后，貌似人人都可以做产品经理了，但实际上，还真不是的，一个 NB 的产品经理与一个初级的伪产品经理的区别往往就在把握需求的差异方面表现的淋淋尽致。不过，这里我们不展开这个话题，还是回到 Coding 上来，对细节的把握以及处理过程需要在在写每一个判断或者循环的逻辑时候考虑；**每行代码一定是要做某个事情的，可能实现某个功能，或者是再做一些准备，而输入输出的范围也取决于业务属性，处理过程是否正确，也意味着对业务逻辑把握的准确程度**；从小的角度来讲，处理过程可能是一个函数、一个算法，而算法的优劣决定了产品的质量与性能，从大的角度来讲，处理过程是对业务属性的代码呈现，之前 PRD 以及交互所画的是仅仅是一个 DEMO，而这里的代码则是真实的呈现，完成一系列的用户需要的功能；



3.避免代码错误，对异常处理

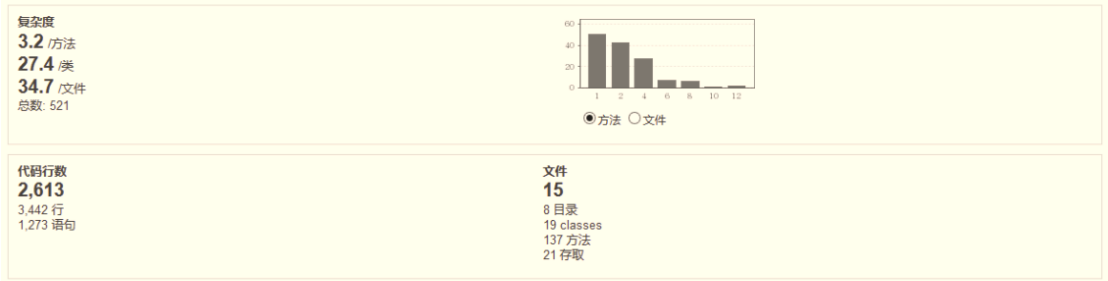
避免代码错误和异常处理，详细来说，就是要处理代码中错误的来源、错误报告机制、为异常分类并提供处理；

代码中错误的来源可能是底层系统，可能是来自用户的不恰当操作，也可能是因为环境导致的，对错误的来源明确的分类和标识有助于定位和分析，而在测试设计的过程中，其实我们往往会设计正常用例，也会设计异常用例，其实和这个是类似的，**当分类之后，对繁杂的场景也就区分开来，便于识别**；

确定错误来源之后，针对不同的错误来源，需要作相应不同的错误报告机制；给用户呈现的错误和给程序员用于 DEBUG 的错误信息必然是不相同的；做到异常分类，进行分类处理，分类角度可能因项目不同而异，定义不同错误码，分不同级别的错误来分别处理；

4.函数设计

在函数设计方面，最能考验开发人员的经验；函数设计的巧妙与否，对算法、对程序、对逻辑，都有影响；高级别的开发人员，会在函数声明设计、函数解耦、代码块的粒度切割、合理使用分支、使用适当的循环类型等各方面做到平衡；



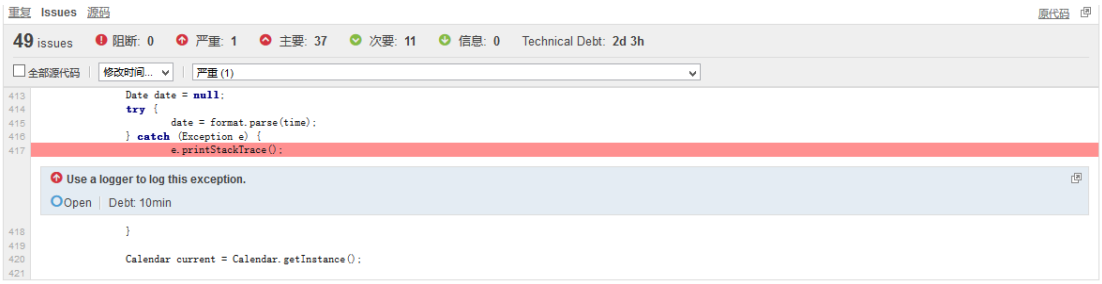
5.数据变量设计

数据变量设计其实是和函数设计有一定关系的，函数的复杂性会影响到数据

变量，选择合适的数据类型和数据粒度、数据容器、数据结构的排序、应对数据结构变化，在数据量小的情况下，可能看不出来，但是在大量访问和请求的时候，就可以看出来产品性能方面的区别了。

6.专项代码质量

对于一些核心服务、底层调度的算法，需要尽可能占用最小的资源，做最多的活儿，这样才能使得效率最大化，当然越是底层的对可移植性，安全性要求也越高，底层与核心代码等也必须要求做单元测试，在代码覆盖率有一定要求，从而来保证对上层服务支撑的可靠性。



7.代码评审

代码评审在以往的传统行业公司，或者日韩企业公司里面会做，而移动互联网或者创业型公司可能做的较少，但是代码评审无疑是有价值的。这个事情可以交给工具，使用一些开源的工具，能够快速地给出代码质量指标，也省去了人为开会进行 Code Review 与结对编程的时间；

8.代码构建与代码质量检查

当代码基本完成时，我们需要进行代码构建，这个时候也可以直接运行单元

测试，可以在代码集成得到测试覆盖率，代码检查结果，其中的注视情况，Issure 情况；

当然，上面只是几种例子，实际还要靠实践，阅读代码，编写代码，阅读开源代码也是个不错的选择，比如 githhub 与 OSChina 上面就有不少开源代码，阅读开源代码，揣摩作者写代码的思路，有助于我们形成好的代码习惯。所谓“熟读唐诗三百首，不会作诗也会吟”，在文学领域如此，代码方面道理相同。在读代码的实践中，可以看到一些大侠巨匠们代码组织的技巧；如果能有一些心得，无疑是极有价值的了。

更多精彩内容请点击 →原创测试文章系列（三十五）（上篇）