

# 大话 软件测试

不扭曲·不变形·不晦涩·不忽悠  
明晰软件测试·从哪里来·到哪里去

大鸟和小白为您洞见软件架构底层，  
诠释软件测试的设计哲学。

欧立奇 何金池等编著



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
http://www.phei.com.cn

---

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

**备用QQ:2404062482**

我们真能测好一个杯子么？

为了测一个杯子，我们需要从功能性、性能性、易用性、可操作性等方面进行测试。

#### 功能测试

能否装水（冷水、热水）？能装多少？漏不漏？

杯子是否能够容纳果汁、白水、酒精、汽油等液体？

#### 界面测试

外观什么颜色？外观是否赏心悦目？能否吸引人？

杯子的形状是怎样的？杯子的图案是否合理？

#### 性能测试

能否装100℃的开水（泡茶）？能否装0℃的冰水？

装满水（汽油），放24小时后，是否会漏？

#### 安全性测试

制作杯子的材料是否有毒？

放进微波炉加热时，杯子是否会爆炸？是否会熔化？

从桌子上掉到水泥地上是否会摔碎？杯子破碎后，是否会对使用者造成伤害？

杯子是否容易长细菌？

杯子是否有缺口？会划伤嘴巴吗？

杯子内壁上的材料是否会溶解到水中？

#### 易用性测试

杯子是否烫手？是否容易喝到水？是否有防滑措施？

#### 可移植性

杯子在不同的地方（车里甚至太空）、不同的温度等环境下是否可以正常使用？

#### 文档测试

使用手册是否对杯子的用法、限制、使用条件等有详细描述？

软件测试看似简单，却需要深厚的基本功才能做好。

我们觉得自己能测好，但我们写出的测试计划只能得到10分中的2分。对于测试新手来说，学好测试的理论知识是必须的，因为这些是你测试的基础。

本书将用完整严密的知识体系和诙谐幽默的语言，为您在软件测试的道路上打好坚实的基础；培养读者敏锐的洞察力以及优秀的测试素养，提高自身功力，从容面对软件开发/测试。



博文视点Broadview



@博文视点Broadview



策划编辑：符隆美  
责任编辑：徐津平  
封面设计：吴海燕

上架建议：软件工程-测试

ISBN 978-7-121-24097-3



9 787121 240973 >

定价：45.00元

# 大话 软件测试

欧立奇 何金池 等 编著



电子工业出版社  
Publishing House of Electronics Industry  
北京•BEIJING



## 内 容 简 介

本书通过小白与大鸟的趣味情景对话形式，用多个小故事、案例、漫画来组织讲解软件测试的方方面面，包括测试需求、测试分类、测试计划以及测试管理等。

本书表现形式虽为“大话”，但内容结构实为严谨。在讲解软件测试的过程中，通过问句式结构，把学习门槛降低，让读者可以更加容易地理解测试的目的、策略、方法以及管理，澄清有关软件测试的常见误解，用一种不扭曲、不变形、不晦涩、不忽悠的表达方式表现测试的真谛，以达到不但授之以“鱼”，还授之以“渔”的目的，引导读者体会软件测试过程中蕴藏的大智慧。

本书适合软件测试、软件开发和软件管理人员以及其他计算机爱好者阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有，侵权必究。

### 图书在版编目（CIP）数据

大话软件测试 / 欧立奇等编著. —北京：电子工业出版社，2014.9

ISBN 978-7-121-24097-3

I. ①大... II. ①欧... III. ①软件—测试 IV. ①TP311.5

中国版本图书馆 CIP 数据核字（2014）第 187934 号

策划编辑：符隆美

责任编辑：徐津平

印 刷：北京季蜂印刷有限公司

装 订：北京季蜂印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：720 × 1000 1/16 印张：17.5 字数：336 千字

版 次：2014 年 9 月第 1 版

印 次：2014 年 9 月第 1 次印刷

印 数：4500 册

定 价：45.00 元

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

质量投诉请发邮件至 [zlt@phei.com.cn](mailto:zlt@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线：（010）88258888。

# 前言

本书通篇都是以情景对话的形式，用多个小故事或案例、漫画来组织讲解软件测试。从软件需求设计说起，在对软件测试做了妥善分类后，将本书的内容分为如下几大方面。

在测试需求方面：如何测试需求，如何审核需求，如何设计文档。

在测试分类方面：详解软件质量模型的6大特性27个子特性的各个检查点，并提供经验和案例，从而使读者能够容易地运用到实际项目环境中；让读者能够清楚地得知，软件测试究竟是测什么？

在测试计划方面：如何写用户故事、测试用例、测试计划，如何进行测试建模，如何制定人力资源的分配计划。

在测试管理方面：如何预知风险，如何写日报，如何与DEV（研发）、PM（项目经理）进行交流，如何进行测试项目的管理，如何利用自动化技术来提高测试的效率等。

本书表现形式虽为“大话”，但内容结构实为严谨。在讲解软件测试的过程中，通过问询式结构，把学习门槛降低，让读者可以更加容易地理解测试的目的、策略、方法以及管理，澄清有关软件测试的常见误解，用一种不扭曲、不变形、不晦涩、不忽悠的表达方式表现测试的真谛，以达到不但授之以“鱼”，还授之以“渔”的目的，引导读者体会软件测试过程中蕴藏的大智慧。

本书不同于其他软件测试书籍的主要特点如下。

## 细

软件测试由于工作的特殊性，软件测试人员更要具有认真、耐心、细致、敏感等个性元素，涉及的方面比较多，且比较基础，也比较细。以软件安装为例，一个完整的软件安装检查点就包括：1.安装环境检查；2.中断安装的情况；3.回溯的检查；4.可定制化安装；5.安装特定参数/端口依赖；6.安装路径；7.安装介质；

8.安装语言; 9.安装 Shell; 10.安装组件; 11.操作系统兼容性; 12.操作系统语言包; 13.硬件系统兼容性; 14.逻辑安装次序; 15.安装安全性; 16.安装接口; 17.安装结果检查, 等等。

所以本书把这些细小的知识点和检查点做了汇总, 让读者有章可循, 按图索骥就可以轻松测试。本书使用了四级标题, 每一条都是一个实际的案例, 切切实实地解决读者遇到的实际问题。

## 深

说到测试, 人们首先想到的是: 测试是一种技术。然而事实上, 测试是一种哲学, 一种思想, 思想的背后是一个人的眼界和世界观。一个测试也许能从多方面揭示测试者的素质和看待问题、思考问题的能力。市面上流行的测试书籍在此专业性的分类方面做得不够, 正因为如此, 本书追求的是循循善诱, 讲深讲透, 侧重于软件测试技术的本质理解, 而不仅限于对测试的单纯讲解。

## 广

市面上流行的软件测试书籍仅对软件功能性测试本身比较侧重, 而忽略软件测试外延的东西: 比如非功能性测试中的易用性测试、性能测试、文档测试。而随着用户对易用性、可靠性要求的与日俱增, 本书对以上诸多方面都给出了详细分析, 并结合大量案例制定出测试方案, 以满足读者需求。

## 乐

苦逼的团队做不出有爱的产品, 愉悦编程、快乐测试才能使你的职场之路长久。所以, 本书语言幽默诙谐, 并夹杂了一些漫画来摆事实、讲道理, 力求为众多严谨的软件测试书籍添加一抹亮色。

本书不是一本万能书籍, 但肯定是您软件测试/开发/管理工作的好助手、好伙伴。

本书主要由欧立奇、何金池编著, 其他参与编写的人员有刘洋、秦晓东、李启高、马雪、马煜、胥虎军、李富星、牛永洁等。

最后, 感谢本书编写过程中的几位重要人士的支持, 衷心感谢明总、峰总、大胖、小四的鼎力协助。

# 目录

第 1 部分 软件需求与设计 .....	1
第 1 章 软件需求 .....	2
1.1 从需求的含混性说到软件测试的目的 .....	2
1.2 需求的定义与分类 .....	5
第 2 章 PRD 审核 .....	8
2.1 PRD 分类 .....	8
2.2 软件产品定位 .....	9
2.3 软件产品需求 .....	14
2.4 审核软件产品需求 .....	17
2.5 范围约束 .....	26
第 3 章 用户故事 .....	29
3.1 什么用户故事 .....	29
3.2 用户故事特点 .....	30
3.3 用户故事分解、细化、合并 .....	32
第 4 章 审核 FS .....	35
4.1 实现的含混性 .....	35
4.2 交付目标 .....	36
4.3 范围约束 .....	37
4.4 假设和依赖 .....	37
4.5 功能描述 .....	38



4.6 审核功能描述 ..... 38

4.7 非功能描述 ..... 41

第 2 部分 软件功能性测试 ..... 43

第 5 章 功能性测试的准确性和合适性 ..... 44

5.1 功能性测试概念 ..... 44

5.2 功能性测试分类 ..... 45

5.3 适合性测试 ..... 45

5.4 准确性测试 ..... 46

第 6 章 软件功能性测试用户故事 ..... 47

6.1 软件功能性测试故事表 ..... 47

6.2 执行者/行为/状态/预期结果/检查点 ..... 48

第 7 章 软件互操作性测试 ..... 72

7.1 互操作性测试定义 ..... 72

7.2 兼容性和互操作性的区别 ..... 73

7.3 不可互操作的原因 ..... 74

7.4 互操作性测试分类 ..... 75

第 8 章 安全性测试 ..... 81

8.1 软件安全性测试概念 ..... 81

8.2 软件安全性测试策略 ..... 82

8.3 用户认证安全 ..... 84

8.4 系统网络安全性测试 ..... 89

8.5 数据库安全性测试 ..... 94

第 9 章 全球化测试 ..... 99

9.1 全球化测试分类 ..... 99

9.2 日期 ..... 101

9.3 字符格式 ..... 103

9.4 数字格式 ..... 104

9.5 输入法编辑器测试 ..... 106

9.6 语言敏感信息测试 ..... 107

第 3 部分	软件非功能性测试	109
第 10 章	易用性测试	110
10.1	易用性测试分类	110
10.2	易理解性测试	111
10.3	易学习性测试	113
10.4	易操作性测试	114
10.5	UI 测试	121
第 11 章	可靠性测试	126
11.1	容错性测试	126
11.2	可恢复性测试	127
11.3	故障转移测试	130
第 12 章	可移植性测试	131
12.1	可移植性测试定义与分类	131
12.2	用户故事列表	131
12.3	行为/状态/预期结果/检查点	133
第 13 章	性能测试	143
13.1	功能与性能的区别	143
13.2	性能测试指标	144
13.3	获取性能需求	148
13.4	性能测试分类	149
13.5	如何进行性能测试	151
13.6	分析性能瓶颈	152
第 14 章	文档测试	156
14.1	文档测试重要性	156
14.2	文档种类	156
14.3	文档测试检查点	157
第 4 部分	软件项目流程与风险	159
第 15 章	软件项目开发流程	160
15.1	Project Milestone 定义	160
15.2	软件项目的主要阶段	161

15.3	研发周期制定 .....	170
15.4	工作量估计 .....	171
第 16 章	项目风险分析 .....	177
16.1	风险、问题、缺陷的区别 .....	177
16.2	风险分类 .....	177
16.3	风险分析 .....	183
16.4	风险缓解 .....	184
16.5	常见的风险处理措施 .....	186
第 5 部分	测试策略与测试计划 .....	190
第 17 章	测试策略 .....	191
17.1	测试策略的定义和分类 .....	191
17.2	测试重点/测试优先级分析 .....	192
17.3	各时间阶段对应的测试策略 .....	193
17.4	多平台/操作系统/浏览器的测试策略 .....	198
17.5	测试开始和结束的标准 .....	198
17.6	测试环境策略 .....	201
17.7	测试人员指定策略/任务分配策略 .....	203
17.8	测试工具的使用策略 .....	205
17.9	测试报告/进度策略 .....	205
第 18 章	测试计划 .....	206
18.1	测试计划的定义 .....	206
18.2	测试计划的范围 .....	207
18.3	测试用例设计方法 .....	207
18.4	测试用例优先级划分 .....	217
第 6 部分	软件测试管理 .....	219
第 19 章	如何预防 Bug .....	220
19.1	Bug 和 Defect 的区别 .....	220
19.2	预防 Bug 的意义 .....	221
19.3	干净的代码 .....	222
19.4	代码可复用 .....	224

19.5	代码审核管理机制	227
19.6	做好单元测试	227
19.7	代码重构	228
第 20 章	如何 Log 高质量的 Bug	232
20.1	Bug 报告写给谁看	232
20.2	Bug 模板介绍	232
20.3	常见的 Bug 问题	246
20.4	如何分析 Root Cause	247
20.5	Bug 生命周期	251
20.6	测试报告分析	253
第 21 章	其他 QA 日常工作	255
21.1	日报	255
21.2	如何开会	258
后记	QA 的自我修养	261
第 1 课	QA 到底是做什么的?	261
第 2 课	质疑和思考	262
第 3 课	QA 要具备的技能	264
第 4 课	QA 和 DEV 的关系	265
第 5 课	QA 的主要贡献	266
第 6 课	自动化测试和常规测试的关系	267



# 第 1 部分 软件需求与设计

---

若要建一幢数百万元的房子，房主一定会与建房者详细讨论各种细节，他们都明白完工以后的修改会造成损失，都知道变更细节的危害性。软件开发也一样，软件项目中百分之四十至百分之六十的问题都是在需求分析阶段埋下的“祸根”，导致的后果便是开发者开发的软件与用户所想得到的软件存在着巨大期望差异。在大多数的软件系统中，最终用户可能都不清楚他的需求是什么，这是千真万确的。所以，在我们没有问清楚所有的需求之前，先别忙着进行设计和测试。

# 第 1 章 软件需求

**测**试的目的应该是验证需求，Bug（预期结果与实际结果之间的差别，即缺陷）是这个过程中的产品而非目标。测试人员应该像工兵一样，在大部队（客户）预期前进的方向上探雷、扫雷，而不需要去关心那些根本没有人会去碰的地雷。衡量一个测试人员应该去衡量他测试了多少需求（测试工作量）。许多测试团队在需求不清晰的情况下，就进行测试，这简直是事倍功半。往往到最后还要返工重来。所以对于测试工程师来说，测试需求是永远放在第一位的。

## 1.1 从需求的含混性说到软件测试的目的

### 1.1.1 需求的含混性

**小白** 大鸟啊，我郁闷得很啊。

**大鸟** 咋啦？

**小白** 我们公司参与了一个大项目。当时市场人员把项目签下来的时候，项目组是欢天喜地。项目做了一年多，到了交付的时候，用户却很不满意，当初说好的需求，好多都变了卦。

用户是上帝，最关键的是如果收不到后面的钱，那就算白干了。公司要求项目组加班加点进行修改。我们测试部门也不得不一遍一遍地做着重复劳动，搞得大家是怨声载道。做市场的和做开发的相互指责，然后，大家又一起骂客户刻薄。公司里面弥漫着灰心丧气的气氛。

**大鸟** 你们是怎么和客户谈需求的？

**小白** 市场人员先谈单，估计人家需要做什么。然后，我们这边派一个搞技术的人员过去了解需求，拿一些对方的表格和笔记回来。

**大鸟** 你们去询问需求，然后做了一个文档。你们头脑里的东西，跟客户要的东西，其实是不一样的。但是，大家都认为这样白纸黑字，基本上是一样的。但这里面其实是有差异的。这种差异，有时影响不大，但有时，是致命的。毕竟，文档不是最终的实物。客户永远认为，他是把需求给你讲清楚了，如果你做不到，

不是他的责任。而且，你要记住一点，用户只有在见到或使用过实物的时候，他才知道他其实要的是什么东西。否则，如果研发人员没搞清需求就闷头做事，迟早项目会变成“包工队的故事”。

### 场景漫画：包工队的故事

甲：说一个包工队接一好活儿，盖一个 70 多米高的大烟囱！

乙：还真不错！

甲：起早贪黑把活干完了，人家来一验收，死活不给工钱，还叮当五四被打了一顿！

乙：为啥啊，质量不行？

甲：把图纸拿倒了，人家让挖口井！

小白 “用户只有在见到或使用过实物的时候，他才知道他其实要的是什么东西。”这句话，我严重同意。可是，要按你的说法，那用户和我们之间，就永远不可能存在真正意义上的沟通！那不成了虚无主义了吗？”

大鸟 如果你不了解客户的业务需求，或者不熟悉其行业规则的时候，你需要更严谨的方式来询问和确定需求。否则，那些落在纸上的文字和文字之间，埋藏着数也数不完的陷阱。需求阶段过于匆忙，就会出问题。比如说，客户忙，随便给你找一个表格，到最后就是无尽的麻烦。

小白 客户不都是这样吗？有何问题呢？

大鸟 客户没有问题，而是去问的人要格外小心了。他要搞清楚客户的背景、立场——客户关注自己想要的结果，但不同人关注的东西不同。有些客户希望不要给他们增加过多的工作任务，越傻瓜越省事越好；有些用户关心新系统的性能是否可靠，速度够不够快；而系统管理员，关心的技术和安全。总之，每个人都各怀心事。需求是多种多样的，所以对于用户所提出来的需求都要仔细记录，认真掂量。

小白 如何掂量？

大鸟 多问自己几个问题。

(1) 这是谁的问题？谁是顾客？谁必须被取悦？（Who 的问题）

(2) 他们的问题是什么？来自于哪些方面？基于什么？有什么特殊性？



(What 和 Why 的问题)

(3) 哪些问题是一定要解决的? 哪些可以不解决? 哪些用户根本不在乎?

(Scope 的问题)

(4) 哪些问题是比较简单的? 哪些是比较复杂的? 简单的要不要合并? 复杂的如何分解? (拆分和合并的问题)

很多人去做需求, 以为拿着本子记下来就好了。其实, 很多时候, 都没那么简单。觉得简单, 是因为很多人认为从技术上设计这类的软件, 简直是小菜一碟。但是, 他没有想到的是, 很多问题只是冰山一角, 有些问题暗流汹涌, 有些问题前后矛盾, 有些问题不可实现。用户需求暗藏着一个个陷阱和地雷。

### 场景: 两个圆的故事

甲用铅笔在纸上画了两个圈。问乙: “你说, 这是啥?”

乙说: “两个圆。”

甲摇摇头: “是两个鸡蛋。”

乙觉得甲很无聊, 不耐烦地说: “好吧, 就算两个鸡蛋。”

甲摇摇头: “不是, 这不是鸡蛋, 这是两个乒乓球。”

乙不知甲在暗示什么, 他在听。

甲说: “我看到的東西, 和你看到的東西, 不一樣。但是, 在紙上, 畫的是同樣的兩個圓。”

## 1.1.2 软件测试的目的

**小白** 你上面所说的这些东西, 不是市场人员或者需求分析人员应该考虑的问题吗? 和测试 Team 有什么关系呢?

**大鸟** 那我问你, 你现在做软件测试, 测试的目的是什么?

**小白** 让我想想。软件测试的目的是尽可能发现并改正被测试软件中的 Bug (缺陷), 提高软件的可靠性。

**大鸟** 这个定义听起来很正确, 但用它来指导测试会带来很多问题。比如有的公司, 用发现的 Bug 的数量来衡量测试人员的业绩, 其结果如何呢? 其一, 有一些不够敬业的测试人员会找出一些无关痛痒的 Bug 来充数, 结果许多时间会被浪费在这些无关痛痒的 Bug 上; 其二, 测试人员会花很大力气设计一些复杂的测试用力去发现一些迄今尚未发现的缺陷, 而不关心这些缺陷在实际用户的使用过程当中是否会发生, 从而浪费了大量的宝贵时间。究其根源, 就是因为对测试目的的这种错误理解造成的。为什么这么说呢? 因为软件里 Bug 的数量是无法估计的, 所以如果测试的目的是为了找 Bug, 那么测试工作将变成一项无法完成也无法衡



量进度而且部分无效的工作（因为有些 Bug 在实际的运行过程当中根本不会发生）。

**小白** 唔……那你看这样说对不对：测试的目的就是为了保证软件质量？

**大鸟** 这个定义也是看似正确，但实际上，混淆了测试和质量保证工作的边界。软件质量要素有很多，包括可理解性（Understandability）、简洁性（Conciseness）、可移植性（Portability）、一致性（Consistency）、可维护性（Maintainability）、可测量性（Testability）、易用性（Usability）、结构（Structure）、效率（Efficiency）、安全性（Security）等，所以，软件质量保证和测试的关注点其实是不同的。假如说，你保证了软件安全性的高质量，但是用户压根就不关心安全性，那你不是白测吗？

**小白** 那么测试的目的应该是什么呢？

**大鸟** IEEE 对软件测试的定义是：“使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。”所以，简言之，测试的目的应该是验证需求，Bug（预期结果与实际结果之间的差别，即缺陷）是这个过程中的产品而非目标。测试人员应该像工兵一样，在大部队（客户）预期前进的方向上探雷、扫雷，而不需要去关心那些根本没有人会去碰的地雷。衡量一个测试人员应该去衡量他/她测试了多少需求（测试工作量）。我看到许多测试团队在需求不清晰的情况下，就进行测试，这简直是事倍功半。往往到最后还要返工重来。所以对于我们 QA（测试工程师）来说，测试需求是永远放在第一位的。

**小白** 我一直以为程序完成后了，才开始软件测试。

**大鸟** 我一直反对软件测试仅存在于程序完成之后。如果只将程序设计阶段后的阶段称之为软件测试的话，需求阶段和设计阶段的缺陷产生的放大效应会加大。这非常不利于保证软件质量。需求缺陷、设计缺陷也是软件缺陷，记住“软件缺陷具有生育能力”。软件测试应该跨越整个软件开发流程。需求测试和设计测试也可以算作软件测试的一种。软件测试应该是一个泛型概念，涵盖整个软件生命周期，这样才能确保周期中的每个阶段经得起考验。

## 1.2 需求的定义与分类

**小白** 我明白了，正因为测试的目的是为了验证需求，所以测试部门就该和研发、市场、技术等支持部门通力合作，搞清楚需求到底是什么？大鸟，到底需求的定义是什么呢？

**大鸟** 我认为需求是这样的：需求是指明了必须实现什么的规格说明。它描述了系统的行为、特性或属性，以及在开发过程中对系统的约束。用户（提出）的需求是用户个人认为在对自身目标的解决方案中，自己不能或者不愿完成的部分。

因为不愿且不能，所以用户的目标可能是含混的、模糊的。好多产品只是有一个潜在的客户群，产品做出来后用户才有反馈。

**小白** 如何减少需求含混性？

**大鸟** 当我们发现存在比较大的歧义时，自己通过交流的方式来降低这种歧义。开始的时候，我们会有一个初步的文档。但是在合同签下来后，会有一个相对时间较长的需求的再确认过程，我们会和客户一起来走一个流程。然后，我们会把大家商业需求的结果，转换成最终的设计，用 PPT 把用户故事（包括操作界面和业务流程）都模拟出来，让客户有身临其境的感觉。在正式开工以前，给客户汇报。到此为止，我们并不做任何真正的编码工作。

**小白** 这样做，还是很花时间，效果如何呢？

**大鸟** 相对于搞错了需求，重新开发，这是最合算的了。很多人都不愿意这样做，最后人都跑光了。知道项目经理最痛苦的事情是什么吗？项目没完人跑光了（你知道项目经理最最痛苦的事情又是什么吗？人在，项目不让你做了！这种事情，也有不少啊）。

### 1.2.1 业务需求

**小白** 那么，如何获得、采集需求？

**大鸟** 搞清楚用户需求是一个长期不断地获取和澄清的过程。对项目不同阶段所要求的需求粒度（需求分解细化的程度）也各不相同。在项目初始的立项阶段，需求采集的主体是市场人员，市场人员从客户那里得到一些直接的或者潜在的需求，这些需求或为全新的，或为对现有产品的改进期望。市场人员得到此类需求后，及时做初步分析，并书面反馈给 PM（项目经理）。这时市场人员是需求的主要来源，但并不是唯一的来源，技术支持人员、中间商、潜在客户群都会做出贡献。我们把立项阶段得到的需求叫业务需求。

**小白** 业务需求？

**大鸟** 业务需求反映的是企业/组织对软件系统的高层次目标。业务需求确定后会得到一个叫做**业务愿景**的东西，其实就是项目成功后会是个什么样子，并在涉众范围内达成一致。而业务需求的确定对之后的用户需求和软件需求起了限定作用，业务需求就是需求过程的宪法，任何需求不得与之相违背。比如说：研发一台电视机，为用户提供看电视的服务。请注意，是服务，是以外部的眼光，全局性地

看这一需要实现的东西。

**小白** 高层次目标？

**大鸟** 对，就是高层次的。咱们中国有一个成语“纲举目张”，纲是鱼网上的总绳，比喻事物的主干部分；目是指网眼，比喻事物的从属部分。纲举目张是指抓住事物的关键，就可以带动其他环节。在这里面，业务需求就是那个纲，指用户需求的大方向、大框架，它的输出结果就是 MRD，即市场需求书（Market Requirement Document），或者有的公司称其为商业需求书。

### 1.2.2 用户需求

**小白** 懂了，那然后呢？

**大鸟** 然后我们要在业务需求的基础上进行用户访谈、调查，对用户使用的场景进行整理，从而建立用户角度的需求。用户需求必须能够体现软件系统将给用户带来的业务价值，或用户要求系统必须能完成的任务，即用户能使用系统来做什么。比如说，为了让用户能够看到电视节目，要做到：接受电视信号、显示到屏幕、可以换台等。这些内容都是为业务需求服务的，也是为了能够看到电视节目所必不可少的，即用户需求是业务需求的深入和细化。在这个阶段，我们需要将 MRD 中笼统的“概念化”需求转化为可实现的“专业化”细则，将 MRD 中的内容进行指标化和技术化。此外我们还要挖掘出其中的隐性需求，以便后续引导用户的兴趣和投资点。

此阶段的输出结果是 PRD，即产品需求文档（Product Requirement Document），着重在论述“要做什么”的问题。此文档重点体现产品的功能和各项指标的说明，分别是功能要求、开发要求、兼容性要求、性能要求、扩展要求、产品文档要求、产品外观要求、产品发布要求、产品支持和培训要求、产品其他要求等。其中，在功能要求当中，应该重视主要 Feature（特性）的重点描述。

## 第 2 章 PRD 审核

**苏** 格拉底早在 2400 年前就提倡并描述了对事务的认知和推理，直到今天，哲学家、科学家和心理学家都还在继续研究证据和推理，这是科学实践的基础。研究认识论的人有科学家、教育家、哲学家，当然还有精英级的软件测试员。学习认识论的学生研究科学、哲学和心理学，目标是了解怎样才能改进我们的思维，以便能够更多地利用批评性思维的最新成果，将认识论运用于软件测试。实际上我们对软件需求审核（review）的过程，也就是一个对软件需求推理和认知的过程。

### 2.1 PRD 分类

**小白** PRD 对于测试工程师来说一定是非常重要的吧。

**大鸟** 没错，PRD 来自于需求分析阶段的开始，是测试执行的源头，是测试生命的源泉，只有源泉干净清晰，才能保证身体的健康。同样，只有 PRD 业务描述清晰、完整、正确才能保证整个测试流程的“生命”，否则整个流程都会紊乱。所以在 PRD 出来后做审核工作是非常有必要的，每个测试人员都应该负责地去审核需求人员编写的每 1 个字、每 1 张图，哪怕是每 1 个标点。但是这种审核不是做文字查错，更多的应该是去关注业务、流程、逻辑和查漏补缺。

**小白** 我最早还以为 PRD 仅是 PM 和市场人员的事呢？

**大鸟** 的确，现在很多公司或者组织在需求分析和架构阶段，仅有 PM 和架构师参与，而 DEV（研发工程师）和 QA 在一开始并不被“卷入”。这样操作将会导致两方面的负面结果，一是很容易导致 DEV 和 QA 因为不清楚或者误解原始需求和设计背景，而在实现的过程中出现或大或小的功能或者性能的偏差；二是上游的设计没有完全挖掘出一些隐性的风险和漏洞（具体实现层面的细节），而这些隐性的不可实现便会导致研发的不可进行或者延迟交付。因此项目架构应该由架构师牵头，整个开发团队各抒己见，共同完成。

**小白** 我知道了，那么，PRD 包括哪些方面的内容呢？

**大鸟** PRD 主要包括软件产品定位、软件产品需求以及范围约束这 3 大项。每一



个大项，又分为若干小项，如下表所示。

- a 【软件产品定位】
  - a.1 产品背景/定位
  - a.2 目标潜在客户群
  - a.3 用户问题
  - a.4 产品特点/特性
  - a.5 产品结构图
- b 【软件产品需求】
  - b.1 功能性需求
  - b.2 非功能性需求
- c 【范围约束】
  - c.1 用户的期盼超出了实现的能力
  - c.2 非技术因素决定的技术选型
  - c.3 预期的使用环境

## 2.2 软件产品定位

### 2.2.1 产品背景/定位

**小白** 产品背景及定位是指什么呢？

**大鸟** 产品背景及定位描述了软件需求规格说明中所定义的产品的背景和起源。说明了该产品是否是成熟产品所改进的下一代产品、是否是现有应用程序的替代品，或者是否是一个新型的、自主型产品。如果 PRD 定义了大系统的一个组成部分，那么就要说明这部分软件是怎样与整个系统相关联的，并且要定义出两者之间的接口。

背景分析包括市场状况及该产品的用途、未来增长趋势、市场的竞争能力、需求规模、发展前景。比如说，要开发一款基于 HPC（High Performance Compute 高性能计算）的系统管理软件，就要先讲一下 HPC 行业背景分析。

#### 案例 HPC2.0 产品背景/定位

HPC2.0 是在 HPC1.0 基础上的改进版本。

传统的 HPC 应用程序通常内置了 MPI（并行计算），它的常见市场用途包括：现代地质、建筑结构、计算流体力学、天气建模、其他科学模型仿真。

此外，潜在的市场用途还包括在同一个应用程序中同时运行大量的独立作业，例如电子模拟、蒙特卡罗模拟、基因组测序和业务分析，以及基于使用 Hadoop 基础设施的

大数据分析，如内存分析等。

最后，HPC 应用程序还包括业务分析及企业级应用。如利用多个服务器应用程序以增加容量和吞吐量，进一步提高了集群性能和可靠性。

## 2.2.2 目标潜在客户群

**小白** 讲完了产品背景及定位，下面就该讲目标潜在客户群了。所谓的目标客户，就是我们打算把产品卖给谁是吧？

**大鸟** 对的，所谓目标客户，是指企业的产品或者服务的所针对对象，是产品能满足需求的客户群。一般来说，目标客户具备以下几个特征。

(1) **购买能力**：具备一定数量及支付能力，特别是具备发展的潜力。

(2) **购买需求**：对你所销售的产品的某一个或多个功能有迫切需求，而这一需求是目前市场上其他种类产品所不能完美提供的。

**小白** 目标客户是潜在客户吗？

**大鸟** 目标客户包括潜在客户和价值客户。用户产生购买行为后，就从潜在客户变成了价值客户。然后，我们要做的是继续深挖用户的需求，保持产品的先进性，从而使价值客户成为忠实客户。

### 案例 HPC 产品目标潜在客户群

HPC 管理软件针对的一些客户类型如下：

(1) 使用集群软件管理商业领域向外扩展应用程序的客户，集群是由系统管理员作为单一操作系统进行统一管理的。

例如，波音公司、埃克森美孚公司、英国网络公司等。

(2) 需要工具来部署和管理集群操作系统和软件堆栈应用开发的客户。

例如，富士通公司等。

(3) 进行超级计算和研究并寻找可扩展的集群管理解决方案的客户。

例如，红牛公司、德意志银行等。

(4) 组织并利用超级计算机的能力租赁给内部或外部客户的客户。

例如，日本北海道超级计算中心等。

## 2.2.3 用户问题

**小白** 下面是关于用户问题的内容。我有一个问题，为什么不直接获得用户需求，而是还要费力搞清楚用户问题、用户的麻烦是什么？

**大鸟** 用户遇到的麻烦、痛点、顾虑到底是什么？面临的问题是什么？这是我们获取用户需求的基础。如果不知道为什么（Why）就直接问用户要什么（What），

那么有可能把事情做偏。下面的例子是高性能计算领域用户常见的问题。

#### 案例 组织部署和管理基础设施时经常面临的问题

- 昂贵的 IT 成本（包括硬件管理、不同操作系统的配置、不相容的预警系统等）；
- 故障排除时间太久；
- 终端用户使用效率太低；
- 从采购到生产环节成本高，部署时间长；
- 没有简单的方法可以快速部署和有效地管理多个集群；
- 技术的可扩展性限制：现有的企业管理解决方案无法向外扩展；

企业云解决方案如 VMware、OpenStack 等目前不支持物理服务器环境中的高性能计算应用程序。

**小白** 是啊，的确如此。那怎样才是一个好的问题定义呢？

**大鸟** “问题定义”只定义了“问题是什么”，而不涉及任何可能的解决方案。它是一个很简单的陈述，并且听起来应该像一个问题。像“机器宕机了，以至于客户不能运行相关的 Job 了”这样的句子是一个很好的问题定义，而“我们需要 HA 系统，使得在机器宕机的情况下还能运行用户的 Job”这种句子就是糟糕的问题定义，因为它听起来不像是问题，倒像是解决方案。

**小白** 只搜集问题而先不做猜想？

**大鸟** 对，先不做猜想。福尔摩斯说过：猜想是很不好的习惯，它有害于做逻辑推理。没有掌握全部问题之前，先做出假设来，这是很大的错误，那样就会使判断产生误差。

**小白** 如何获得问题的来源？

**大鸟** 问题的来源是指要考虑“这是谁的问题？”，其目的可能是：

- （1）确定谁是顾客，就是说，谁必须被取悦？
- （2）搜集一些有用的线索，以找到合适的解决方案。

**小白** 举个例子，就说刚才“机器宕机了”这个问题。我们怎么来寻找是谁的问题？

**大鸟** 如果“机器宕机了”这个问题只是来自于一个小型 PC 机用户，那么从影响面上考虑，问题不那么大。我们可以采取如下解决方案。

- （1）远程电话协助。
- （2）派专业人员上门服务，探查是否存在任何软件问题（如病毒、木马系统崩溃等）或者硬件问题。
- （3）让用户自己过来报修。

**小白** 毕竟是小型客户，影响范围小，优先级和严重性也就小。

**大鸟** 但是大型用户就不一样了。2013年7月，有大量用户反映微信发生全面故障，故障包括微信信息无法发出、无法刷新朋友圈、无法登录公众账号平台、无法连接微信网页版、页面显示为“服务器暂时不可用”。这可就是一个非常严重的问题了。

**小白** 对，当问题来自于大型客户，解决方案就迥然不同了。

**大鸟** 如果我们把工商银行或者腾讯公司当作我们服务器的顾客，我们可能会得到一个迥然不同的列表，比如说：

- (1) 服务器是否需要事务级的管理。
- (2) 是否要求电源备机控制系统。
- (3) 是否需要高易用性的远程分布服务系统。
- (4) 技术支持人员是否需要常驻现场。

这两个客户列表，尽管不一定完全排斥，但是我们确实能够看出一些倾向上的差别。为了避免因为这些倾向而做出草率的决定，我们要在提出解决方案之前仔细考虑，并把问题想得尽量全面些：考虑一类问题而不仅是一个问题。

**小白** 怎么去寻找一个问题的解决方案？

**大鸟** 四句话：把自己当成别人；把别人当成自己；把别人当成别人；把自己当成自己。

- (1) 在动手去解决问题之前，好好想想问题的来源。
- (2) 站在各个角度来看待面临的问题，以便能够了解其真正所在。
- (3) 不要把解决方法误认为是问题的定义，哪怕是你自己常用的解决方法。

## 2.2.4 产品特点 and 特性

**小白** 产品的特点和特性是指什么呢？

**大鸟** 简单点说，就是阐述将要开发的这款软件的特性和有必要去开发的理由：创新点是哪些？优越性是哪些？有没有什么独特的价值？研发出来后对相关领域有多大影响？再和市场已经存在的那些软件做个对比，自己产品的优点、改进点是哪些？

**小白** 是啊，如果没有的话，公司为什么要投资进行研发呢。

**大鸟** 下面以 ETT 公司开发的 HPC（高性能计算）系列软件产品做为一个例子。

### 案例 ETT 公司的 HPC 产品特性

ETT 公司的 HPC 软件是一个商业产品，用于管理大规模硬件和软件。此外，它还是一个可以管理多集群和多租户的计算基础架构的软件产品。

ETT 公司 HPC 软件相对市场其他产品的独特价值是:

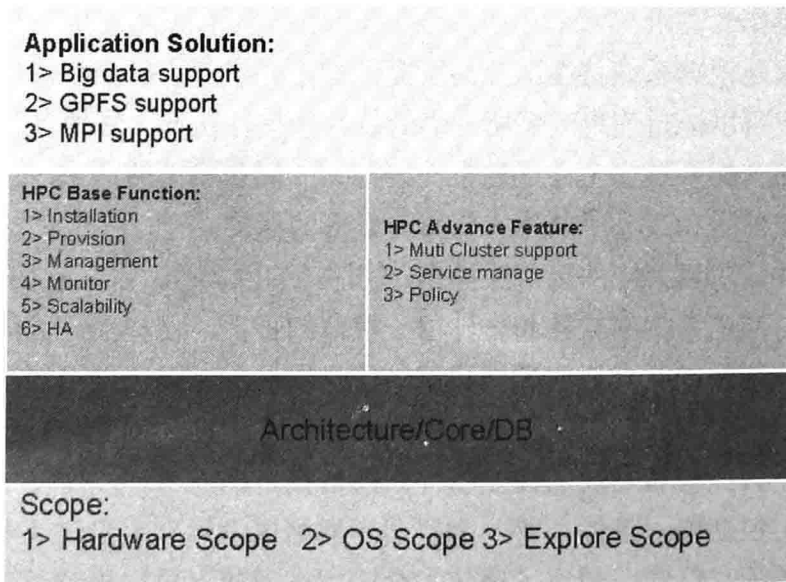
- (1) 综合管理和监控硬件组件、操作系统和中间件。
- (2) 支持多个操作系统和软件配置。
- (3) 支持多集群、多租户管理。
- (4) 高可扩展性——最多可以扩展到 50000 个节点。
- (5) 能够提供各种规模的集群环境, 如高性能计算、Hadoop 和 OpenStack 的等。

(6) ETT 的 HPC 软件不仅能在有 OEM (Original Equipment Manufacturer, 贴牌代工厂商) 合作伙伴的硬件 (如 Dell、HP 等) 上运行, 而且支持在非 OEM 合作伙伴的硬件上运行。

### 2.2.5 产品结构图

**小白** 产品的结构图指什么呢?

**大鸟** 产品的结构图指产品的架构和组成方式。这里, 以 HPC 产品为例简单解释一下。



(1) 在这个图中, 最下层是 Scope (范围) 支持, 是指这个产品在哪个硬件范围、操作系统范围以及浏览器范围获得支持。

(2) 上面一层是产品的架构、内核、数据库。

(3) 再上面一层是 HPC 产品的基本功能 (安装、部署、管理、监控、扩展性、高易用性) 和高级功能 (多集群、服务管理、策略管理)。

(4) 最上面一层是应用层, 指应用上支持哪些服务 (大数据、GPFS 等)。

## 2.3 软件产品需求

**小白** 软件产品的需求包括什么呢？

**大鸟** 包括如下几个方面。

- (1) 功能性需求。
- (2) 非功能性需求。
- (3) 文档需求。
- (4) 项目需求优先级定义。

### 2.3.1 功能性需求

**小白** 功能性需求是指什么？

**大鸟** 功能性需求包括如下几个方面。

- (1) 特性需求。
- (2) 安全性需求。
- (3) 互操作性需求。

特性需求是根据系统特性即产品所提供的主要服务来提交给用户使用的软件功能，使用户可以通过使用产品所提供的特性来执行任务，达到自己的期望。

安全性需求指系统安全性、完整性或与私人问题相关的需求，这些问题将会影响到产品的使用和产品所创建或使用的数据的保护。

互操作性需求包括硬件接口互操作、软件接口互操作，以及通信接口互操作。

硬件接口描述系统中软件和硬件每一接口的特征。这种描述可能包括支持的硬件类型、软硬件之间交流的数据和控制信息的性质以及所使用的通信协议。软件接口描述该产品与其他外部组件（由名字和版本识别）的连接，包括数据库、操作系统、工具、库和集成的商业组件。通信接口描述与产品所使用的通信功能相关的需求，包括电子邮件、Web 浏览器、网络通信标准或协议及电子表格等，定义了相关的消息格式，规定了通信安全或加密问题、数据传输速率和同步通信机制。

以 ETT 公司的 HPC 产品为例，我们可以看一下一个大型软件的功能性需求都包括什么。

#### 案例 ETT 公司的 HPC 产品功能性需求

软件安装/卸载/升级/更新需求：

- 交互式安装

- 静默安装
- 工厂安装
- 重新安装 (P2)
- 卸载 (P2)
- 安装中断后继续安装 (P3)
- 升级
- 打补丁

节点/网络/模块管理包括:

- 添加
- 删除
- 修改
- 替换
- 重新部署
- 导入
- 导出
- 显示
- 锁定
- 激活
- 查询
- 生成报表
- 预警
- 配置
- 同步
- 过滤
- 状态监控

应用层模板管理:

- 模板实例化
- 发布
- 拷贝

用户安全管理:

- License 控制
- 用户认证安全
- 系统网络安全性
- 数据库安全性

全球化支持:

- 中文支持



- 德文支持

- 日文支持

其他需求:

.....

## 2.3.2 非功能性需求

**小白** 非功能性需求是指什么?

**大鸟** 非功能性需求是指软件产品为满足用户业务需求而必须具有且除功能需求以外的特性,包括系统的性能、易用性、可靠性、可维护性、可支持性、可扩充性、可移植性以及对技术和对业务的适应性等。例如:

- 性能 (Performance) ——响应时间、吞吐量、准确性、有效性、资源利用率 (如要求系统能满足 100 个人同时使用,页面反应时间不能超过 6 秒);
- 易用性 (Usability) ——人性化因素、帮助、文档;
- 可靠性 (Reliability) ——故障频率 (如系统能  $7 \times 24$  小时连续运行,年非计划宕机时间不能高于 8 小时)、可恢复性 (系统出现故障时,能够快速切换到备用机)、可预测性;
- 可支持性 (Supportability) ——适应性、可维护性、国际化、可配置性。
- 可移植性 (Portability) ——是否有对原来产品进行软硬件迁移的支持? 迁移过程中是否有兼容性的要求?

以 ETT 公司的 HPC 产品为例,我们可以看一个软件的非功能性需求包括什么。

### 案例 ETT 公司的 HPC 产品非功能性需求

- 可靠性测试 (P1)

我们的目标是支持自动化高可用性计算 (High Availability)。

- 可扩展性需求 (P1)

我们的目标是支持集群节点从目前的 2000 个扩展到 5000 个。

集群的报表、预警系统的图表系统必须能够支持扩展到 5000 个节点。

- 性能需求

同时部署 500 个节点的时间  $< 4$  个小时。

同时部署 2000 个节点的时间  $< 8$  个小时。

同时更新 2000 个节点的时间  $< 8$  个小时。

### 2.3.3 文档需求

**小白** 大鸟，文档需求是指什么？

**大鸟** 文档需求列举出将与软件一同发行的用户文档部分，例如用户手册、在线帮助和教程，并明确所有已知的用户文档的交付格式或标准。

**案例 ETT 公司的 HPC 产品非功能性需求**

(1) Online Help (在线帮助)。

(2) Man Page (功能帮助以及参数帮助)。

### 2.3.4 项目优先级的定义

**小白** 对于一个项目优先级的定义是什么样的呢？

**大鸟** 一般来说，项目定义为 3 个优先级，分别是 P1、P2、P3。

- P1——是强制性的一定要完成的模块，如果出现问题，将会影响项目发布。
- P2——对项目发布非常重要，但是如果出现问题也不会影响项目发布（具体由项目经理决定）。
- P3——重要，但不是强制性的或预期的一定要发布。

## 2.4 审核软件产品需求

**小白** 大鸟，上面我们已经介绍了功能性需求和非功能性需求。那么从 QA 角度来讲，我们怎么来审核 PRD，以确保它是个好的需求文档呢？

**大鸟** 一般来说，用户需求存在如下两个特点。

(1) 零散：用户会提出不同角度、不同层面、不同粒度的需求，所以是零散的。

(2) 存在矛盾：由于用户处在企业/组织的不同层面，因此难免会出现盲人摸象的现象，从而导致需求的片面性甚至自相矛盾。

一个合理的需求文档有如下特性：

- (1) 完整性。
- (2) 正确性。
- (3) 一致性。

**小白** 你总说审核软件需求。先问一个问题，审核到底是什么意思？

**大鸟** “审核”在很多人的脑海中就是得出一个通过与否的结论。其实不是那么

简单。顾名思义，审核（review）实际意思就是再（re）看（view）一遍的意思，其本质含义是通过再次的审读，尽早地暴露出错误。QA 通过审核需求文档，消除歧义、错误、不完整性，以确保沟通没有失真。搞清楚用户想要什么？用户要这干什么？他为什么这么想？他会不会漏了什么？

**小白** 那你也说了，正是因为需求文档存在歧义、错误、不完整性，所以我们需要不断地审核。

**大鸟** 是的。

## 2.4.1 如何保证软件需求的完整性

**小白** 软件需求的完整性是指什么呢？

**大鸟** 完整性是指：不应该遗漏要求和必需的信息。不完整性是指：完整性涉及的东西可能根本不存在，或者只说了一部分，没说完整，有遗漏。

**小白** 别漏了。可是如何保证不漏呢？

**大鸟** 我们一般用如下这几种方法。

- (1) 演绎法。
- (2) 比较法。
- (3) 分解法。
- (4) 打标签。

### 2.4.1.1 演绎法

**小白** 演绎法如何解决需求的不完整性？

**大鸟** 我们知道用户的需求可能是零散的，我们将这些零散、孤立的需求认为是一个个需求孤岛，需要把这些需求孤岛连接起来，最终形成闭环。

**小白** 需求孤岛？

**大鸟** 咱们看下面几个例子。

例 1 业务需求——研发一个飞机，这个飞机可以起飞，可以空中平行飞行。

例 2 用户需求——HPC 软件可以正常地安装。

例 3 功能需求——普通用户可以对管理节点进行增加、删除的操作。

你觉得上面这些需求缺点什么吗？

**小白** 第一个例子，貌似没有提降落的事啊。又不是恐怖分子，怎么会研发这种只能起飞和飞行的飞机呢？应该是漏掉了吧。

**大鸟** 这种遗漏可能是一个大大的缺陷。用户所要求的飞行服务需求应该是一个体系，而不是割裂、孤立的需求。例 2 也有这样的问题：HPC 管理软件里面有安

装的需求，但是没有提及卸载、升级、打补丁、重新安装的需求。例3亦然，普通用户对管理节点有增加、删除的需求但是没有提及修改、查看的需求，另外只是普通用户有这个需求吗？管理员、根（root）用户又会怎样呢？进行了增加、删除操作后，成功了如何？失败了又怎样？这些都是需求的不完整性。

**小白** 如何避免呢？

**大鸟** 把它放到体系里。还是刚才那个例子，用户有安装的需求：安装的需求放在软件测试里，是属于“可安装性测试”里面的，这里包括安装、卸载、升级、降级、不完全安装等多方面的需求。而“可安装性测试”又属于“可移植性测试”这个体系的一部分。我们把一个孤点需求放在软件需求的整个体系中，通过零散的需求拼图碎片补全整个需求拼图。

**小白** 有点像达芬奇密码里的拼图。

**大鸟** 准确点说是用演绎法。

**小白** 演绎法是什么呢？

**大鸟** 演绎法是指从普遍性结论或一般性事理推导出个别性结论的论证方法。在演绎论证中，普遍性结论是依据，而个别性结论是论点。演绎推理与归纳推理相反，它反映了论据与论点之间由一般到个别的逻辑关系。

比如说：“天下乌鸦一般黑，西安也有乌鸦，所以西安的乌鸦也是黑的”。这段话中就包含着一个完整的演绎论证。“天下乌鸦一般黑”，是普遍性原理，是论据，是“大前提”；“西安也有乌鸦”，是已知的判断，是“小前提”；而“西安的乌鸦也是黑的”则是结论，也是论点。

同理，大多数软件都有“安装、卸载、升级、降级、不完全安装”需求，大多数用户管理都有“增、删、读、改、导入、导出”等需求。所以，如果开发一个新的软件，那它的需求至少要和体系里面的软件需求保持一致，至少不能丢。

**小白** 嗯，但是会不会有这样的情况，我们研发了一个产品，就是只有安装，不能卸载。

**大鸟** 有可能，但是不需要做的东西，你要把它放在 Scope 外面去，并且从 QA 的角度来说，至少你要想到这些，否则你怎么审核呢。在本书第2部分的测试分类中，我们会详细讲述测试分类的体系来验证这些需求。

#### 2.4.1.2 比较法

**小白** 除了演绎法之外。你说还有比较法？

**大鸟** 是的，比较法包括纵向比较、横向比较。

**小白** 纵向比较是啥？

**大鸟** 现在的版本和以前的版本相比少不少东西，这是纵向的比较。

**例** HPC4.2 版本的产品，与 HPC4.1 版本产品相比，管理节点不能修改 IP。

HPC4.1 版本的产品是可以修改管理节点 IP 的，但是 HPC4.2 版本反而不能这样做了。是用户真的没有这样的需求吗？如果用户没有这样的需求，为什么？

如果用户有这样的需求，而仅仅是实现的局限性造成的，那就要问清楚，下一个版本还要实现吗？并把它明确加入到 Scope 里面去。

此外，上一版本的关键遗留问题（没做的）和上一版本没做好现在要加强（Enhance）的地方，是否已包含在内？这也是检查项之一。

**小白** 横向比较是啥？

**大鸟** 新引入一个东西，和类似产品相比少不少东西，这是横向的比较。

**例** HPC 产品中，系统可以监控 GPFS，监控包括 Alert（预警）。

GPFS 是一个新引入的东西，系统以前监控的是 NFS，而且监控包括 Alert、Report 两种。那么 GPFS 的监控为什么不包括 Report 呢？既然是一个体系，应该包括完整的监控。

### 2.4.1.3 分解法

**小白** 分解法是分解什么呢？

**大鸟** 分解关键字。文字表述中，往往关键字过于庞大，需要分解。像下面这个例子。

**例** 用户不能访问系统管理员级别的数据库里面的数据。

这个就是一个非常模糊的用户需求。它有如下的问题：

- 如果关键字是“用户”，用户是普通用户还是 root 用户，系统管理员级别的用户可以吗？
- 如果关键字是“不能”，那我想知道，验证成功如何？验证失败又该如何？
- 如果关键字是“访问”，访问的方式很多，增、删、读、改、导入、导出都包括吗？
- 如果关键字是“系统管理员级别的”，那么系统管理员级别是指什么？能否继续分解？情况如何？
- 如果关键字是“数据库”，那是什么样的数据库？是 Sql database 还是报表？
- 如果关键字是“数据”，数据包括什么？是数字、图表？

**小白** 哇哦，一句话居然扯出来那么多东西。

**大鸟** 正因为关键字不同，一个含混的需求可以分解成若干个清晰的子需求。

小白 那该咋办?

大鸟 限定、分解、细化。所谓限定,是指在范围约束里就把范围都定义清楚了,这里就不至于那么乱了。比如说:我们在范围约束中就指明数据库仅指 Oracle 10 数据库,那这里的数据库概念就清晰了。

所谓分解就是把主谓宾状都分解:根据什么样的人(Who)在什么样的条件下(Condition)做了什么样的事情(What),结果如何(成功如何?失败如何?)。必要的时候还可以画流程图把它分解成一个细粒度的需求,分解的方法我会在用户故事里面细讲。

小白 需要分解,需要细化。

大鸟 没错,下面的就是一个分解后的软件需求。

例 当请求者输入账户号码时,系统将根据在线用户列表来验证所输入的账号。如果在此列表中查不到该账号,则系统将显示一个出错信息并拒绝订货;否则,进入订货流程。

#### 2.4.1.4 打标签

小白 还有一个问题哦,假如说 QA 确实知道某些需求不完整,缺乏一些信息,但这个东西用户、PM 暂时也不确定,该咋办呢?

大鸟 没有问题,如果知道缺少某项信息,那么用 TBD (To be done, 待确定)作为标准标识来标明这项缺漏,这样将有助于你避免不完整性。在开始开发之前,必须解决需求中所有的 TBD 项。

### 2.4.2 如何保证软件需求的正确性:明确用户动机

小白 完整性说完了,接着我们说一下正确性。软件需求的正确性是指什么呢?

大鸟 正确性是指:没有歧义,逻辑一致,表达清晰。

小白 可是如何保证软件需求的正确性?

大鸟 有下面两种方法。

(1) 明确用户动机。

(2) 正确的表达方式。

小白 既然我们已知道用户要什么,干嘛一定还要追究用户为什么要这么做呢?

大鸟 在福尔摩斯探案集里,每次锁定了犯罪嫌疑人后,除了拿出证据之外,读者更期待知道罪犯的动机是什么。同理,在需求阶段,用户想要什么只是表相。用户的痛点、难点、困难、处境、顾虑、背景等,这才是他们要这样干的原因。许多需求团队在进行需求捕获活动时,经常预期用户能够直接告诉他们要做什么

(What), 而不太关心用户提出需求的真正动机 (Why)。这样就会经常引起下面这样的局面发生:

“你要绳子, 我给你了; 你要木板, 我也给你了; 你为什么说这不是你想要的呢?”。我想程序员也有类似的问题想问自己的客户: “你要文本框, 我提供了; 你要一个表单, 我也有了; 你为什么说这个功能不是你想要的呢?”

**小白** 所以要透过现象看本质。

**大鸟** 是的。因为对于一个特定的问题来说, 可能的解决方案会有很多, 因此用户可能在使用软件的过程中会不断发现其他可选的、更合适的替代方案, 从而导致了不必要的需求变更。而缓解这一现象的关键就在于, 在需求捕获的过程中要多问“为什么”。还有一种情况是多个原因导致一个结果。

**小白** 不过虽然很多书籍里都强调需求是讲做什么 (What) 而不是怎么做 (How)。但是实际上我感觉: 只写 What 会导致开发结果不可控, 所以某些项目管理人员在需求里面写了好多 How。

**大鸟** 其实, 当我们发现 How 没说清楚的时候, 一定要反思 Why 说清楚了没有, 因为需求人员并不是最佳决策者, 他们也缺乏解决方案的知识, 经验缺乏的结果会造成后期变更。如果程序员不知道 Why 就去做, 很可能做拧了, 结果就像欧亨利小说中的老板娘一样。

### 欧亨利小说节选

很久很久以前, 一个男人天天都到一家小面包店里买两个隔夜的陈面包, 新鲜面包是五分钱一个, 陈面包五分钱却可以买两个。除了陈面包以外, 他从来没有买过别的东西。时间一长, 小店女主人开始注意这个男人, 从开始的同情、怜悯, 到后来对他的坚强和坚持产生好感。后来她得知他是一个艺术家, 她想: 他眼镜后面的目光是多么温柔和善啊! ——但却靠陈面包过活! 不过天才在成名之前, 往往要经过一番奋斗。

又过了很长时间, 男人又来买面包, 店主鼓起勇气, 乘男人不注意, 快速地在面包里面塞了一块黄油, 然后男人拿走了。女店主开始不安, 猜想男人发现后会怎样, 会感动? 会接受? 还是自尊被打击, 再也不来了。

第二天, 男人一早就出现在小店, 发疯似地向女店主狂吼: “谁让你在面包里面加黄油的? 一年了, 我每天都加班到半夜, 这幅设计图本来今天就可以完工了, 谁让你在面包里面加黄油的!”

原来, 橡皮能擦掉铅笔字是 1770 年英国科学家普里斯特首先发现的。在这之前, 人们是用陈面包来擦铅笔的。

**小白** 好心办坏事。

**大鸟** 女主人知道用户的需求 (需要陈面包), 但是却搞不知道用户为什么要这样做 (擦画); 她以为是用来吃的 (没搞清楚用户动机), 想要把需求变更一下 (夹



了块黄油), 结果好心做了坏事。同理只有切实知道用户的理由, 你才能提供正确的实现方法。

**小白** 为啥这么做, 先给我一个理由。

**大鸟** 下面是一个我工作中的案例。

**案例** 曾经有一次我们 Team 做一个节点部署的软件。当时用户提出这样的需求: 当节点部署完了之后, 可以自动跑一个用户自定义的脚本 (What)。我们当时部署节点的方式是通过 OS 模板 (Template) 方式来作。比如说你想安装一个操作系统为 Rhel6.4 的节点, 那么你就得指定一个 Rhel6.4 的模板, 然后方能部署。所以我们当时的解决方式是把用户自定义的脚本 (Script) 绑在了模板上, 这样用户部署节点的时候, 就可以自动执行这个脚本了 (How)。从当时来看, 这个解决方案确实解决了用户的问题。可是到了项目快结束的时候, 用户却并不买账。我们又仔细调查了用户需求: 原来用户想跑 Script 的目标节点是同时给多个组使用, 不同的组有不同的模板, 有的时候还要随时新建模板, 并且期待在不同模板间的频繁切换 (Why)。这样, 把脚本直接耦合在模板上就不合适了 (谁知道那个组用哪个模板呢)。所以, 用户实际的需求是要解决在不同模板切换的情况下依旧保持脚本可以顺利运行。因此, 目标节点应该和脚本直接耦合。搞清楚了 Why 才能弄明白用户是怎么玩的, 你的 How 才不会是无的放矢。

**小白** 怎么才能弄清楚用户动机呢?

**大鸟** 如果你能够直接问用户或者项目负责人, 那是最好的。如果没有这样条件, QA 可以做一些推理, 大胆假设, 小心求证。其实, 当你感觉这个地方动机不明、模糊的时候, 这些都是 QA 的机会, 别放过它。负责任的警探不会因为动机没搞清楚就匆匆结案, 同理, 负责任的 QA 也不会因为没搞清楚用户动机就匆忙测试。

### 2.4.3 如何保证软件需求的正确性: 正确的表达方式

**小白** 正确性的第二条是正确的表达方式。需要什么表达方式吗? 说出来不就完了, 写出来不就得了。

**大鸟** 表达是一种沟通方式的问题。我们做软件一般是通过文档沟通, 所有的文档都是书面语言。语言是一种非常无力的东西, 心里想的是什么, 一张口可能说的是另一个意思。

#### 场景: 解说员的故事

解说员韩某某:

“守门员将球回传给门将”;

“各位观众, 中秋节刚过, 我给大家拜个晚年”;

“可能有的观众刚刚打开电梯, 我们再把比分……”;

“球被守门员的后腿挡了一下!!!”;

“梅西又习惯性地舔舔自己的舌头”。

韩老师心里明明是这样想的，但表达出来后就是另一种意思了。这是语言描述的局限性。有些东西靠说是讲不清楚的，描述的过程中会造成信息扭曲。

可是文档的表述还不如口语呢，文档没有声音，没有图像，更没有语气。

**小白** 所以表达方式很重要。

**大鸟** 是的，所以我们要用正确的表达方式，具体包括如下几条。

- (1) 直接沟通。
- (2) 简洁描述。
- (3) 减少定性描述。
- (4) 词语的准确性。

### 2.4.3.1 直接沟通

**小白** 您的意思是尽量做一些直接沟通。

**大鸟** 我们说 PRD 是很重要的，这没错，但是它再重要也就是一个文档。文档是死的而人是活的，所以不要局限在这个文档上。文档只是我们过河的一条船，我们的目的是过河。如果可以的话，和写文档的人多做一些互动、交互不是更好吗？获取需求的最好办法就是直接和客户面对面沟通，尽量不打电话；如果可以打电话，尽量不要用 Skype；如果能用 Skype，尽量不要发邮件。越直接，效果越好。找到项目相关的人，最好找到项目的接口人进行面谈，或者直接去用户那里，看看使用环境。

### 2.4.3.2 简洁描述

**小白** 下一条是简洁描述。

**大鸟** 软件需求说明书，不是记叙文，而是应用文。应该概括段落大意，给出关键点。尽量用应用文的方式、短句的方式，不要长篇大论。

### 2.4.3.3 减少定性描述

**小白** 所谓定性描述是什么呢？

**大鸟** 定性的词语也就意味着不确定。比如“系统对报警提供了有效的支持”、“两个报表间存在有效依赖关系”这两句话，你看其中的两个词“有效”、“依赖”。什么叫有效呢？表现在什么方面呢？依赖是什么呢？是数据依赖？还是流程依赖？这些都是定性的词语，如果审核时候发现了这样词语，QA 就要问个为什么了，为什么不能用更加定量的方式来表述。

**小白** 怎样更加定量呢？

**大鸟** 用一个指标性或者经验值之类的东西来描述。比如写可靠性时，不写“高

可靠性”，而是写“7×24 小时不间断的服务”；写易用性时，不写“达到高易用性”，而是写“没有接触过本软件的初级用户能够不在帮忙的情况下 30 分钟理解所有功能点”。

#### 2.4.3.4 词语的准确性

##### 场景漫画：开封菜的故事

甲：“对对对，我想起来了，这个地方有个挺好吃的，叫什么开封菜还不错。你到那里等我哦。”

乙：“开封菜？”

甲：“就是一白头发老头做的广告，叫什么 KFC (KaiFengCai)，再见！”



**小白** 词语的准确性是指什么？

**大鸟** 有时候需求文档中同一个词有多种含义，对于含混的表达，文档的作者必须基于这样一个理论：即文档的读者必须有一定的认知水平和认知标准。如果认知标准不同，必须加上清晰的注释。比如 IB 这个词是个缩写，在软件领域可以理解为 IN BOUND CALL 也可以认为是 Infini Band，这种缩写一定要在需求文档中定义清楚。

#### 2.4.4 一致性

**小白** 需求的一致性是指什么呢？

**大鸟** 一致性是指与其他软件需求或高层（系统、业务）需求不相矛盾。在开发前必须解决所有需求间的不一致部分。只有进行一番调查研究，才能知道某一项需求是否确实一致。

**案例** 系统管理员可以具备访问一切数据内容的权限。PCM 用户的数据库仅允许自己访问。

如果系统管理员可以访问一切数据库的内容,那也就可以访问 PCM 用户的数据库,可是 PCM 数据库仅允许自己访问。这就是不一致。

## 2.5 范围约束

**小白** 范围约束是指什么呢?

**大鸟** 范围约束限制了开发人员设计和构建系统时的选择范围。范围约束一般有如下三种。

- (1) 用户的期盼超出了实现的能力。
- (2) 非技术因素决定的技术选型。
- (3) 预期的使用环境限制。

### 2.5.1 用户的期盼超出了实现的能力

#### 场景：玩具飞机的故事

一个稚气未消的三岁小孩,将一个玩具飞机举过头顶,看着爸爸说:“爸爸,你帮我把这个玩具飞机送到太空上去吧,我要把月球上的嫦娥姐姐接下来玩”。爸爸:“-\_-|||”!

**小白** 什么叫用户的期盼超出了实现的能力?

**大鸟** 这句话的本质就是实现的局限性。

**小白** 所谓的“属下无能,听凭教主发落”。

**大鸟** 用户提出了大量的需求,有些是技术上根本无法实现的,比如说一百年前如果有用户提出移动电话的需求,那么当时是实现不了的。

**小白** 后来技术突破了,就解决了。

**大鸟** 对,有些是费用预算内无法实现的。就像孩子不知道航天飞机上月球要多少钱一样,用户也不知道自己提出的需求要多大的成本。

**小白** 用户想掏买馒头的钱吃鲍鱼,显然不可能。

**大鸟** 还有一种情况是,我能实现,但是在这个时间期限内根本做不完。做不完怎么办?加人或者砍 Scope,做不完的就砍掉了,也就是放在范围以外了。或者是这个版本我不支持,下个版本再支持。

### 2.5.2 非技术因素决定的技术选型

**小白** 非技术因素决定的技术选型是指什么?

**大鸟** 对于软件开发而言,有些技术选型并非由技术团队决定,而会受到企业/组织实际情况的影响,例如 HPC 软件产品原来采用的是 Oracle 数据库系统,但

是现在由于一些实际的原因（法律或者其他非技术因素），采用了 PostgreSQL 数据库。

### 2.5.3 预期的使用环境

**小白** 预期的使用环境是指什么？

**大鸟** 用户的使用环境（使用场合、软硬件环境等）也会对软件的开发产生很大的影响，如果忽略了这方面的因素会给项目带来一些不必要的麻烦。一般使用环境包括如下几个方面。

（1）硬件平台范围：产品支持哪些机器？内存大于多少？硬盘多少。

比如：

- 机器类型——ETT/Dell 系列/苹果系列等。
- 内存——不小于 4G。
- 硬盘——至少 500G。

注意：每类机器有不同的型号，如 ETT3750，DellR510。

（2）OS（操作系统）范围，支持哪些操作系统？

比如：

- DOS。
- Windows。
- Linux（Rhel/Sles/Ubuntu）。
- UNIX（Solaris/AIX/HP-UX）。

注意：OS 还有小版本，如 Red Hat 5.8，Windows 7 等。

**案例** 原来假设用户的环境是 Sles11.2，没想到用户实际环境只有 Sles10，从而导致很多基于 Sles11.2 开发的客户端程序不支持。

（3）浏览器支持范围，支持哪种浏览器？这里要注意一点，浏览器支持往往与软件或硬件相关。

浏览器包括：Firefox，IE，Chrome，Safari，Opera 等。

注意以下几点：

- 浏览器往往支持多个操作系统，但其针对不同的操作系统有不同的版本。  
比如 Firefox，要弄清楚是哪个操作系统上的 Firefox。
- 有些浏览器的版本变化很快，要搞清是支持哪个版本。
- 有些浏览器是支持移动设备的，比如 Android 上的 UC 浏览器。

（4）预期的使用依赖。

有些软件预期的使用前提是有 .NET 平台支持，或者有其他第三方软件，忽略

这样的情况，会造成实际使用时的尴尬。如果实在没办法解决，那么至少在对软件需求规格的说明中列举出影响需求陈述的假设因素。

#### 2.5.4 审核 Scope

**小白** 对于 QA 而言，在审核 Scope 时要做什么呢？

**大鸟** 要注意以下三点。

(1) 要搞清楚范围和超出范围的界定。如果确定不做的，就要将其写入范围之外 (out-of-scope)，不要不了了之。

(2) 要把超出范围而不去做的原因搞清楚。在此期间，QA 可能要做一些调查，而不要完全听从 PM 和 DEV 的“忽悠”。搞清楚是 Can't (不能) 还是 Won't (不愿)？

(3) 搞清楚完成此模块开发的假定事项或者依赖事项是否已表述清楚。

**小白** PM 和 DEV 还会“忽悠”QA？

**大鸟** 多了去了，人都有惰性。明明可以把事情做好，但不愿做的事情也蛮多的。从 QA 角度，你要搞清楚哪个是不能做的，哪个是不愿做的。

## 第3章 用户故事

**为**了彻底解决需求含混性的问题，我们需要一个画面式的场景来全真模拟用户的行为：我们的产品要为哪些人服务？他们的背景如何？他们有哪些亟待满足的需求？他们需要软件产品的什么功能？他们怎么去玩？想达到一个什么样的目的，实现什么样的价值？这些都是我们要为用户想到的。用户故事就是这样一个东西，通过有效、及时的沟通，帮助用户澄清和优化需求。

### 3.1 什么用户故事

**小白** 大鸟，前两章我们讲了测试的本质是验证软件需求，还讲了如何去审核软件需求，保障软件需求的完整性和正确性。到目前为止，我们的需求含混性解决了吗？

**大鸟** 我只能告诉你：这样还不够！还不足以“设身处地”地替用户去考虑。我们需要一个画面式的场景来全真模拟用户的行为：我们的产品要为哪些人服务？他们的背景如何？他们有哪些亟待满足的需求？他们需要软件产品的什么功能？他们怎么去玩？想达到一个什么样的目的？想实现什么样的价值？这些都是我们要为用户想到的。

**小白** 那我们具体该如何“设身处地”地替用户去考虑呢？

**大鸟** 通过用户故事。

**小白** 用户故事是什么？

**大鸟** 你可以理解为：用户故事是一个备忘录似的交互模型，是从用户角度来描述用户渴望得到的功能。它强调的是通过有效、及时的沟通，帮助用户澄清和优化需求，用户故事有一个通用的模板，如下：

As a role (作为一个角色), I want to do something or a piece of functionality (通过某项功能, 执行一些操作), so that achieve some business value or statement of intent. (以便能够实现特定的目标/价值)

在这个模板中，有三个不同的关键点，分别为：



- 用户角色 (Who)。
- 某项功能 (功能即用户能亲自执行的操作)。
- 实现了某个目标, 获得了某种价值 (Goal/Value)。

下面是一个例子:

例 作为一个皇马的球迷, 通过点击皇马官网的最新新闻栏, 便能够实时了解最新的皇马动态。

- “皇马球迷” (用户角色)。
- “点击皇马官网的最新新闻栏” (功能/用户操作)。
- “了解皇马的动态” (客户目标/价值)。

## 3.2 用户故事特点

### 3.2.1 体现用户价值

**小白** 用户故事主要是为了通过某个功能来体现用户价值。

**大鸟** 对产品而言, 永远应该把那些最能体现用户价值观的功能置于最高优先级。我们先看下面两个反例。

例1 如果把“保存”按钮统一放在页面上端而非下面, 那么对屏幕上侧的控件做些修改时, 就无需滚屏即可保存了。

例2 所有自定义字段, 统一改为 1000 长度。

第一个故事勉强可以写为:

作为一个用户, 可以方便地点击上端的“保存”按钮, 以便在某些控件修改的时候无需滚屏即可保存。

但是这个故事仅属于易用性优化级别而不是功能级别, 颗粒度显得过小。而作为一个用户, 这个需求显然和“增删改查”等功能需求, 在价值上相差甚远。

第二个故事, 找不到“用户”的位置, 因为它是我们自己要做的改进, 客户完全可以不感知。

所以, 上面这两个用户故事都不能很好地体现用户价值。

### 3.2.2 不要出现技术术语

**小白** 用户故事由谁来写? 是开发人员吗?

**大鸟** 不对, 开发人员容易站在自己的角度去思考和划分故事, 这样就背离了用

户故事的初衷。一般用户故事是由用户来写的，或者由用户口述，我们的需求人员（有可能就是 PM）来整理的。下面，让我们看看“卖香烟”这个例子。

#### 用户故事：卖香烟

1. 用户投入一些钱。
2. 售货机显示用户已经投了多少钱。
3. 如果投入的钱足够买某种香烟，这种香烟对应的按钮的灯就会亮。
4. 用户按了某个亮了的按钮。
5. 售货机卖出一包香烟给他。
6. 售货机找零钱给他。

注意到，一个用户故事里面的事件可以这样描述：

1. 用户做 AA。
2. 系统做 BB。
3. 用户做 CC。
4. 系统做 DD。
5. ....

用户故事只是描述系统的外在行为，以客户能够明白的方式，描述了一个系统的外在行为，它完全忽略了系统的内部动作。如果是下面这样的用户故事：

1. 用户投入一些钱。
2. 售货机将塞进来的钱存在钱箱里，然后发送一条命令给屏幕，屏幕显示目前已经投入的金额。
3. 售货机查询数据库里面所有香烟的价格，判定钱足够买哪些香烟，对于钱足够买的那些香烟，对应的按钮的灯就会亮起来。
4. 用户按下一个亮起来的按钮。
5. 售货机卖出一包香烟给用户，然后将数据库里面该香烟的存货数量减 1。
6. 售货机找零钱给用户。

不管是口头描述，还是书面形式，这样的内容是描述用户故事时一个很常见的错误。特别是千万不要提及任何有关数据库、记录、字段之类的、对客户一点意义都没有的东西。

### 3.2.3 可测试性

**小白** 可测试性是指什么？

**大鸟** 通常，不可测试的故事发生在一些非功能性的需求上。所以，可测试性意

味着用户故事尽量采用功能性来做描述。

### 3.3 用户故事分解、细化、合并

**小白** 用户故事是对用户、对需求的描写，可能非常庞杂，甚至含混。

**大鸟** 的确如此，用户故事是对需求的细化和切分。既然是细化，就得有一个度，需求的颗粒度需要多少才能称之为用户故事？这就牵扯出和用户故事一起出现的另外一个关键的单词叫 Epic（史诗级故事），通俗来说就是大型的故事。Epic 有一些自身的特点：它是由许许多多的、较大的、不确定的需求组成，另外 Epic 本身不能直接通过其完成迭代（逐层分解）和开发，而是首先需要划分成较小的真正的用户故事。除了这两点，Epic 因为包含了太多的模糊性需求，所以常常混杂了很多不同的特性，而一个特性就是一组可以归为一类的需求。Epic 分为两类：

- （1）复合故事（compound story）：由多个小的故事组成。
- （2）复杂故事（complex story）：本身就很大且不容易分解的故事。

对于太小的故事，不值得去写故事和评估，则可以将其合并到需要半天或几天完成的故事中，比如缺陷报表和用户界面变更。

**小白** 那该如何细化呢？

**大鸟** 细化可以分成如下几种。

- （1）按用户角色细化。
- （2）根据数据边界来细化。
- （3）根据用户的操作来细化。

#### 3.3.1 按用户角色细化

**小白** 你是说按用户角色来分，而没有说按用户来分。那用户和角色的区别是什么？

**大鸟** 这里有三个概念，即用户、角色和权限。

在应用系统中，权限的表现是什么？

（1）对功能模块的操作，对上传文件的删改，对菜单的访问，甚至对页面上某个按钮、某个图片的可见性控制，都属于权限的范畴。

（2）此外，有些权限设计，会把功能操作作为一类，而把文件、页面元素、机器列表等作为另一类，这样构成“资源”的模型。

角色是什么？可以理解为一定数量的权限的集合、权限的载体。例如一个论坛系统，“超级管理员”、“版主”、“普通用户”都是角色。版主可管理版内的帖子，

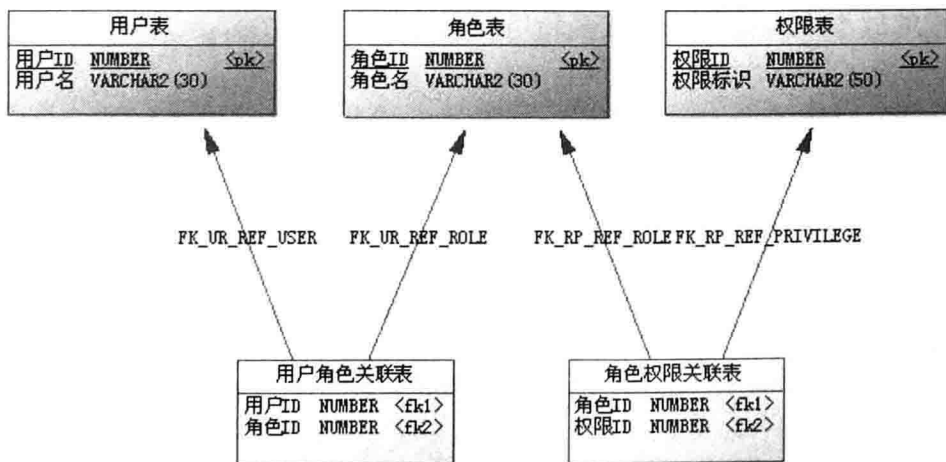
可管理版内的用户，可以下载某些资源等，这些都是权限。要给某个用户授予这些权限，不需要直接将权限授予用户，可将“版主”这个角色赋予该用户。

用户通过角色与权限进行关联。简单地说，一个用户拥有若干角色，每一个角色拥有若干权限。这样，就构造成“用户-角色-权限”的授权模型。在这种模型中，用户与角色之间，角色与权限之间，一般是多对多的关系。而在做数据表建模时，可把功能操作和资源统一管理，也就是将它们都直接与权限表进行关联，这样可能更具便捷性和易扩展性（如下图所示）。

张三、李四、王五都是用户，但是很有可能他们的权限都一样。他们都是一种角色，所以按照用户来划分，会造成用户故事没有区分度。

**小白** 明白了，张三、李四、王五属于一种角色。

**大鸟** 他们属于一个“组”，当用户的数量非常大时，要给系统每个用户逐一授权（授角色），是件非常烦琐的事情。这时，就需要给用户分组，每个用户组内有多个用户。除了可给用户授权外，还可以给用户组授权。这样一来，用户拥有的所有权限，就是用户个人拥有的权限与该用户所在的用户组拥有的权限之和。



当故事只为单一用户角色编写时，故事的可读性是最强的。比如这句话：“作为一个用户，我可以访问资产负债表数据。”可以做如下分解：

**例** 作为一个用户，我可以访问资产负债表数据。

——作为系统管理员（实际上是系统管理员组），我可以访问资产负债表数据；

——作为普通用户（实际上是普通用户组），我可以访问资产负债表数据；

——作为根（root）用户，我可以访问资产负债表数据。

### 3.3.2 根据用户的操作细化

**小白** 什么叫根据用户的操作来细化？

**大鸟** 一般我们用 CRUD（Create/Read/Update/Delete，即增删读改）来细化。可以做如下分解：

**案例** 作为一个用户，我可以访问资产负债表数据。

——作为一个用户，我可以**增加**资产负债表数据；

——作为一个用户，我可以**删除**资产负债表数据；

——作为一个用户，我可以**修改**资产负债表数据；

——作为一个用户，我可以**阅读**资产负债表数据。

**小白** 但是真实情况可能不那么简单。除了增、删、读、改等操作外应该还有激活、冻结等。如果要是安装测试的话，应该还有安装、卸载之类的。

**大鸟** 你说得没错，具体的内容我会在测试的分类里详细介绍。

### 3.3.3 按用户角色合并

**小白** 什么叫做按用户角色合并？

**大鸟** 对于太小的故事，不值得去写故事和评估，则可以将其合并到需要半天或几天完成的故事中，比如缺陷报表和用户界面变更。

## 第4章 审核FS

**前** 三章，我们讲了测试的本质是验证软件需求，还讲了如何去审核软件需求。无论是审核 PRD 还是写用户故事，都是为了解决用户“要什么，为什么”的问题，本质上是解决目标需求的含混性。

事实上，对于软件开发来说，含混性存在两个方面，一个是对目标需求的含混性，另一方面是实现方式上的含混性。实现方式的含混性主要是解决“给用户什么，怎么给，什么样的呈现方式”的问题。FS (Functional Specification, 功能规格书) 是更进一步回答“各个模块具体怎么做”的问题。

### 4.1 实现的含混性

**小白** 为什么会出现实现的含混性呢？

**大鸟** 原因有如下两个。

(1) 一个是科学的方法大多需要量化的度量技术。含混性的量化公式是：

$$\text{含混性} = \frac{\text{实现需求的最大花费}}{\text{实现需求的最小花费}}$$

如果需求过于抽象，那么这个公式给出的含混性就可能非常大，而这种随意性很大的估值是非常忌讳的，很可能会造成亏本。

(2) 另一个原因，是我们拔高了我们的底层实现。很多时候，歧义来自于我们自身的能力，因为我们有足够的能力做到多样性，而这些多样性都体现出各自的特点，所以我们才会有了如此众多不同的选择。

通过严格的分析后发现，决定含混性的，始终还是应该由实现相应需求的不同方案的数目来确定的。含混性毕竟还是对这种实现的不确定性的度量。按照成本来度量，可能会存在两方面的问题。其一，许多不同的方案花费相差不大，但在具体实现的时候可能会造成一些执行上的偏差，而这种偏差在最开始度量的时候体现不明显；其二，虽然成本一般都对应着相应的解决方案，但是不排除少数解决方案之间存在大的成本跨度。这些情况大多主要影响的是实现，不过确实也需要提前注意。

---

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

**备用QQ:2404062482**