目 录

测试之路在何方？

——一个阿里测试工程师三年的经历和总结

作者：王春生

摘要：

本篇文章讲述了我在阿里巴巴三年的自动化测试生涯，总结了工作中遇到了种种困难，以及自己的摸索和解决方案。希望能够给做测试的朋友们一点参考。

关键词：测试发展 职业规划 自动化测试

感谢好友 leeyupeng 的指点和建议，谢谢你！

2006 年 6 月，我加入雅虎中国，正式成为阿里巴巴的一名员工，从此开始了我在阿里三年测试生涯。三年里面，我从之前的一名开发工程师转变为测试工程师，从普通的测试工程师尝试转为自动化测试工程师，从单纯的测试工程师，努力向测试架构师转变。中间经历了各种各样的困难，疑惑，总结思考了很多的东西。今天总结成为文字，希望能给大家一点参考。

一 、我的工作经历

我在阿里巴巴三年的工作经历，大致可以分为三个阶段：第一个阶段是角色的转化，从开发工程师转变为一个测试工程师；第二个阶段是尝试提升，开始摸索自动化测试方面的工具，技术；第三个阶段则比较系统的思考自动化测试的整体解决方案，为团队提供工具、方法和培训。

1.1 从开发到测试的转变

在进入阿里巴巴之前，我主要从事的工作是开发。虽然之前有开发过 Bug 管理工具——BugFree，但真正自己去做测试工作，还是第一次。刚刚上班的感觉，很紧张。需要参加的会议很多，需要接触的团队和同事也很多，每个人的脚步都急匆匆的，说话的语调也很快，大部分的名词和术语都听不懂。怎么办？

这是我遇到的第一个问题：一个公司或者组织，不会停下来等你准备好了才上路。只能自己花时间，多多学习，向老同事请教学习，在最短的时间里面熟悉业务，掌握这些名词和术语。所谓做一行，爱一行。测试也是这样，只有对自己所测试的业务十分熟悉，才能更好的进行测试，保障产品的质量。

很快就有了测试任务，测试 ContentMatch(内容匹配，发展成为后来的阿里妈妈)。大名鼎鼎的车东时任 CM 的产品经理，他给了一些很复杂的公式来计算网页关键词和广告之间的匹配程度，等等。刚开始测试的时候，还尝试用笔计算。

后来发现不行，太复杂了，也容易出错。怎么办呢？

这是我遇到的另一个问题：所测的东西的复杂程度远远超出了手工测试的范畴，如何解决呢？后来我尝试写了一个小脚本程序，实现了他的计算公式。只要输入各种数值，就能计算出各种数据，然后和开发的程序的输出进行比对。我想这就是自动化测试的开始。这就引出了一个问题，什么是自动化测试？我想自动化测试不是说对一个东西实现了 100%的自动化，才叫自动化测试。凡是有助于省却重复劳动，提供测试效率的工具或者方法，都应属于自动化测试的范畴。

紧接着，就有性能测试任务来了。需要测试一个广告搜索引擎在不同数据规模、不同压力情况下面的性能数据。这次性能测试主要遇到了两个问题：一是测试数据如何产生。二是选择什么样的测试工具。

第一个问题比较容易解决，让开发人员帮忙从生产环境拷贝下来真实的广告数据和真实的流量数据，就可以保证了数据规模的真实，以及流量的真实。

第二个问题，也许有的朋友说，用 loadrunner 啊。我那个时候没有接触过 loadrunner，公司里面也不让用盗版软件。所以就找开源软件的解决方案。最后选择了 http_load，很容易就可以完成压力的测试。

就这样，不知不觉中开始了自己的测试之路。

1.2 自动化测试探索

很快我被调去负责搜索竞价的点击和过滤系统的测试。这两个系统是当时整个雅虎搜索竞价系统的核心子系统，每天涉及到的金额会有几十万到上百万，稍有疏忽，就会造成非常严重的后果。我第一次测试这个过滤系统的时候，是从另外一个同事那儿接手过来的。没有文档，只有简单的测试用例；没有数据，没有环境。第二天要上线。自己昏头昏脑的测了半天，终于测完了。突然间发现，自己测试的是老版本，不是最新的。只好加班，又重复测试了一遍。

从那个时候开始，我就开始琢磨，这样下去肯定不行，每次这样重复的测试，浪费时间和精力不说，也保障不了速度和质量。为什么这样说呢，是因为这些系统的测试都是没有界面的，只能自己构造数据，运行系统，然后自己检查数据。过程繁琐，极易出错。有的时候，跑一个测试用例，可能需要 1 个小时的时间。

用什么样的框架和工具来进行这种引擎系统的自动化测试呢？这是我遇到的新的问题。商业的软件，大都是用来做 GUI 自动化测试的。在我们当时的环境中，根本无用武之地。我还是把目光投向了开源软件。研究来研究去，我最终选择了 PHP 的自动化测试框架——PHPT。

大家知道 PHP 是世界上应用最为广泛的网站开发语言。它的代码质量是有目共睹的。全世界几千万的部署量，都没有什么问题，这靠的什么呢？有很多原因，其中有一个很重要的就是它们的自动化测试框架机制——PHPT。

PHPT 的运行机制其实很简单，就是你写一个测试脚本程序，里面有一定的代码逻辑，来完成特定的功能，然后有一个输出。PHPT 这个工具呢，会自动来运行这个测试脚本，获得它的输出，然后和实现设定的预期输出进行比对，相同则通过。不同则失败。就这样简单。我就拿来把它给改了改，就开始了过滤系统的自动化测试脚本的开发。

开源再次展现了它的威力！

选择了自动化测试框架，剩下的就是来写自动化测试脚本了。在这个过程中遇到的主要问题是测试数据如何生成以及如何测试这个过滤系统。测试数据我还是采用了手工准备的房方式。即一个测试用例，对应一个测试脚本，一套测试数据。然后模拟手工测试的过程，通过程序来实现和过滤系统的交互。比如第一步，搭建测试环境，第二步准备测试数据，第三步触发竞价引擎处理数据，第四步，分析过滤结果，进行比对。这个过程其实就是手工测试过程的再现！

1.3 如何在团队里面实施自动化测试？

07 年 5 月份开始，我所在的竞价部门的同事，开始陆续抽调到杭州，阿里妈妈的创业开始了。阿里巴巴的做事风格和雅虎的四平八稳就很不同，做事讲究速度，不拘一格，飞速地往前冲。

我被调到杭州的时候，马上面临的一个严峻问题就是测试人手严重不足。我所在的部门测试只有我一个，面对的则是两个研发 Team，大约有七八人左右。怎么办呢？

招聘！动用各种资源，开始疯狂的招聘，甚至和其他部门抢人。让人欣慰的是一年以后，杭州已经有了一个颇具规模和战斗力的测试团队。

人招进来之后，紧接着就产生了新的问题。如何让新来的同事尽快上手。答案两个字，培训。当时我把一些测试必须看的资料，掌握的工具，命令，方法，整理成系统的文档。这样新来的同事，你先让他看文档，就可以开始上手。然后再在实际的测试任务中成长。

团队逐渐组建起来了，越来越多的问题暴露了出来：没有测试用例，没有统一的测试工具、方法。大家都还停留在手工测试阶段，重复。无法保障测试的时间和质量。

首先来总结测试用例。每个人要把自己所负责的模块的测试用例点整理完毕。其他的文档都可以没有，但测试用例一定要保留下来。这样后面人员的变动就不怕了。

然后开始总结测试工具，方法，环境，用例等系列的问题。

测试框架，已经有了之前的 phpt 做基础，我重新开发，强化了它的自动化测试框架的功能，增加了和 bugfree 集成的 api，使之成为了每个测试工程师都在

使用的自动化测试框架。

自动化测试里面还有一个非常重要的问题就是测试数据的问题。我们曾经有一个日志文件，大约有 100 多个字段，一条日志大约占掉半屏的空间。我曾经看见同事手工修改一个字段的日志，大约需要花费十几分钟，乃至更长。实在是无法忍受，于是动手写了一个通用的测试数据生成工具。测试数据只需要配置格式，数量，即可生成想要的测试数据。

自动化测试里面还有一个问题就是依赖系统的问题。当时阿里妈妈的系统架构及其复杂，最多的分层可能要达到六七层。在做测试的时候，你如何控制这些依赖系统的行为，以保证你所测试的系统逻辑走向，是极具挑战性的。为此，我和另外一个开发的同事，一起设计开发了一个通用的 mock 工具。从而使得测试任何一层上面的系统，都不再麻烦。

就这样一个问题一个问题的解决，最终形成了一套整体的解决方案：

- 统一的测试用例要求。
- 统一的自动化测试框架。
- 通用的测试数据生成工具来解决测试数据的问题。
- 通用的接口 mock 工具，来解决测试环境依赖的问题。
- 通过虚拟化来解决测试环境不足和冲突的问题。
- 开发了各种小工具，来解决日常测试中遇到的问题。
- 通过系列文档来解决新人培训和学习的问题。

二 、如何做好测试？

三年的测试工作下来，我认为真正做一个好的测试工程师，需要具备一些品质和技能。

2.1 做好测试所需要的品质：

- 耐心

测试确实是一个非常磨练人意志的工作，没有耐心，就无法忍受重复的工作

- 细心

粗枝大叶，很难做好测试工作。

- 追求完美

不追求完美，就无法从更高的层面来保证产品的质量

- 乐观，坚强

作为交付用户前的最后一关，测试的压力也很大。需要乐观坚强的品质来自我调节。

2.2 做好测试所需要的技能：

从能力来讲，我认为可以从下面几个方面入手：

● 掌握一门编程语言

毋庸置疑，掌握了一门编程语言，对自己的测试工作是非常有帮助的。无论是写一个小工具，还是生成测试数据，懂得开发，都可以让自己的工作变得简单许多。编程语言可以选择脚本语言，比如 PHP, ruby, python 等等。

● 熟练运用各种工具

常用的各种工具，应该能够熟练应用。这包括各种办公软件的使用，常见的命令，各种浏览器及其插件的使用，等等。

● 学习和使用 linux 等开源软件

除了 windows 下面的这些软件和工具外，应当学习使用下各种开源的操作系统和软件。很多的自动化测试框架也都是开源软件。

● 了解各种常见网络协议

比如 HTTP 协议，FTP, SSH, mail, DNS 等等，也包括自己系统内部封装的 API。只有深入了解这些东西，才有可能找到更好的测试方法。

● 总结分享

勤于总结，学习，东西一定要将其整理成文字，写下来。自己的心得体会可以经常和团队的同事分享。哪怕是讲不明白，只要把自己讲明白就行了。:)我很多东西是在给别人讲的时候，讲着讲着，自己突然间明白了。

● 沟通能力

做测试需要和不同的人打交道，沟通能力是必须的。

三 、测试职业生涯应如何发展？

3.1 技术发展路线

如果你真想在测试技术这一个领域发展下去，那么有几个比较好的方向，性能测试和自动化测试。

性能测试是一个非常系统的工作，部署测试环境，制定测试策略，准备测试数据，构造压力，获得数据，分析，形成报告。如果能够找到瓶颈原因，那就更棒。不过性能测试不是说使用一个工具就可以了。真正做好性能测试，其实需要很系统的知识。

自动化测试是未来的一个发展方向。其实只要系统的结构好，更高的自动化测试脚本的覆盖率是完全可能的。自动化测试的目的不是自动化测试，而是更好的发现新的 bug。

其实技术路线走下去，会和开发汇合在架构师这一个层面上面。因为你走得越深，其实和开发考虑的越来越趋同，届时你可能就分不清楚测试和开发的区别。开发就是测试，测试就是开发。思考问题的角度会跳出单纯的测试或者开发的层面，更加全面。

3.2 管理发展路线

测试发展还有一个方向就是管理路线，做测试经理，这也是一个很好的方向。不过挑战也比较大。如何沟通，协调，调动大家的积极性，让每个人都能得到锻炼和发展，需要认真对待，真心对待。

3.3 跳出测试，走进项目管理

09 年 7 月份，我辞职离开北京，正式开始自己的创业之路。我选择的方向是做一款开源的项目管理软件。为什么选择这个方向呢，是因为我发现，测试管理固然重要，但更为重要的是项目管理。

我在阿里做测试这三年，深深的体会到了一点：测试工程师是在夹缝里面跳舞。很多需求本身就存在问题，开发人员的设计也存在问题，而我只能做到的就是保证测试这一关。但这很难。事情从开始就已经出现偏差了，到了测试这一关又能做什么呢？所以我在阿里三年，也在思考过程改进，流程再造方面的事情，也试图去做一些工具来支撑项目管理的流程。不过这些想法，在阿里都没有能够实现，所以我带着这些思考和总结，出来创业，就形成了今天的开源项目管理软件——禅道。

单纯的测试，只是覆盖到了整个项目管理的后期。虽然测试也有机会参与到项目管理的前期，但我想起的作用不大。所以我建议大家，可以去学习掌握下项目管理的知识和方法。测试对于一个组织很重要，但项目管理对于组织来讲，就更重要。

最近敏捷开发的思想在国内日渐流行，也许大家可以好好关注一下这些方面的动态，可以朝着这个方向发展。我想这也是测试的一个发展方向。

附录

通用测试数据生成方案

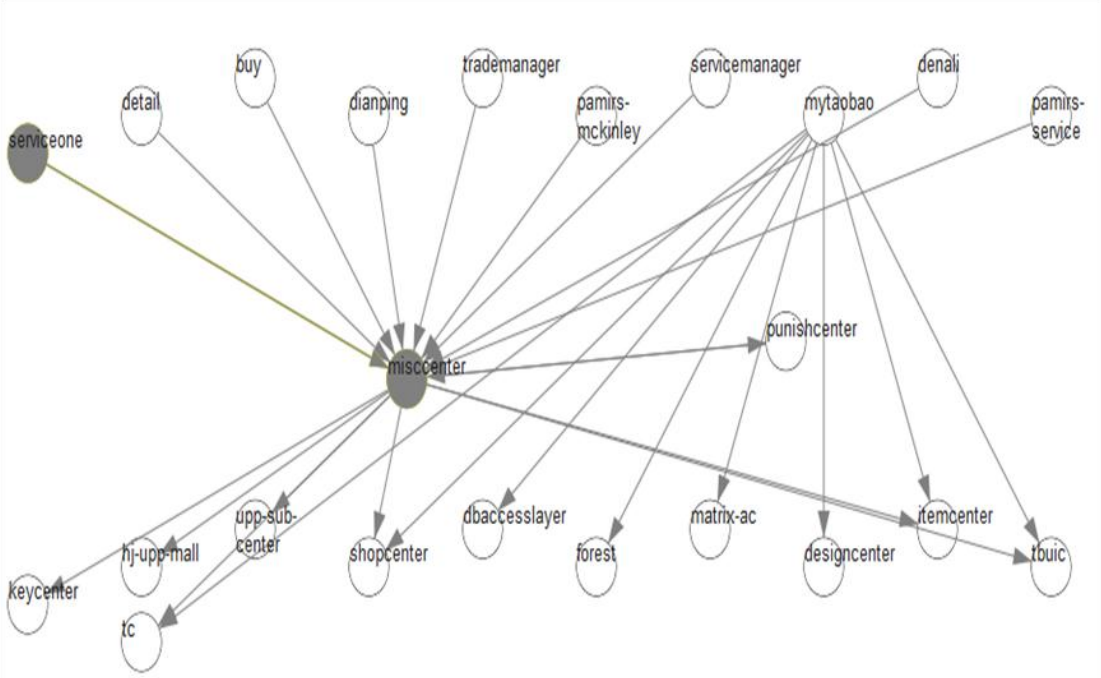
参考：http://www.51testing.com/html/13/n-224613.html

【淘测试专栏】接口测试的应用于实践

作者：叶渡

一 、为什么要做接口测试？

对于一个小型的 web 系统而言，通过界面的手工测试，可渗透到系统的各层，达到测试整个系统功能的目的。但随着产品功能的不断丰富，访问量不断增加，传统的 MVC 架构已经不能满足快速研发的需求，系统不断向着分布式、业务中心化和高可用性的方向发展而分为多个子系统，各个子系统之间通过接口契约相



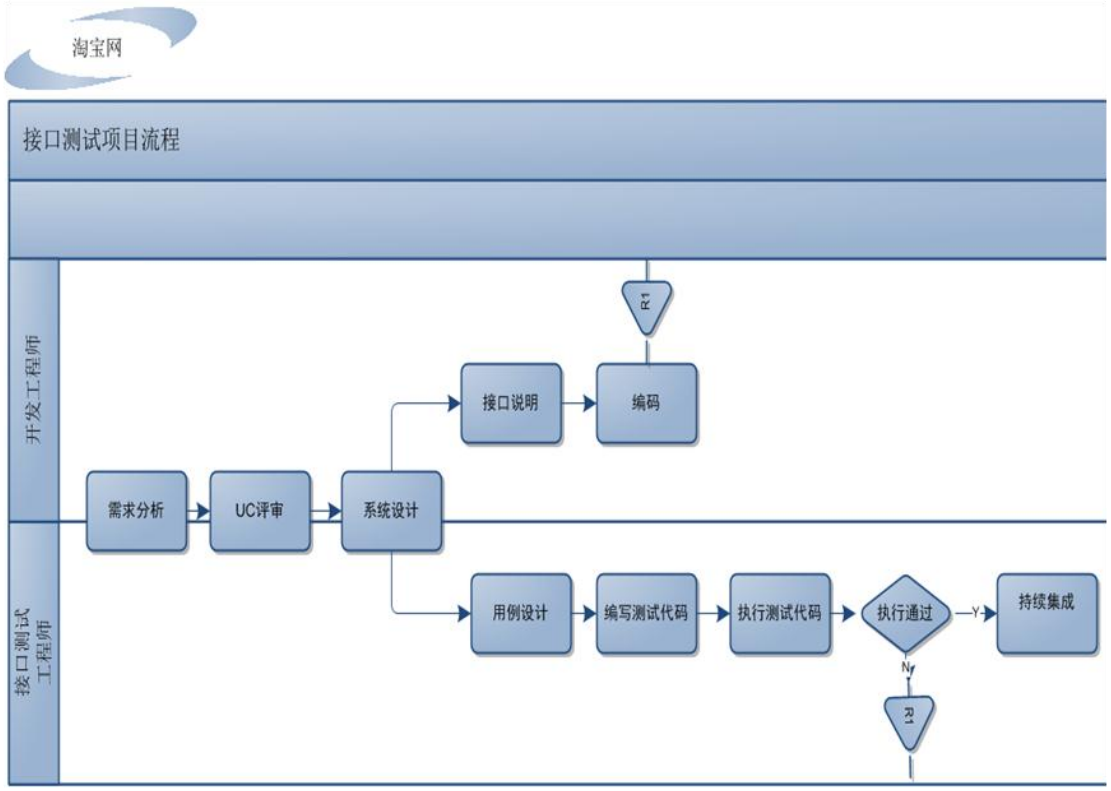
互提供服务，这些接口又会被上层应用调用，调用关系错综复杂，下图展示了淘宝某子系统跟其他子系统的交互关系。

通过上图可以看出，随着系统的不断复杂，通过界面测试保证整个系统功能的做法不再有效或成本巨大，同时底层及业务中心对外提供服务的接口变得很关键，一旦这些接口出现 bug，将影响到大量的上层业务甚至使系统宕机。如何保证这些接口的功能正确性？一个有效的手段就是通过模拟上层调用场景对系统接口进行测试，保证接口对外提供的服务是正确的。接口测试的另外一个价值在于回归测试，一旦某个系统代码改动影响了接口服务契约，回归系统会马上发现报告给相关人员，避免了线上 bug 的发生。

既然接口测试如此重要，那么如何进行接口测试？下面从流程、框架选型和

持续集成方面做些阐述。

二 、接口测试的一般流程



接口测试需要在项目何时介入，具体要做哪些工作，在研发过程中接口测试工程师怎么跟开发工程师互相配合？下面的流程图展示了接口测试在项目中的重要活动。

准确的说接口测试工程师在项目启动初即参与进来，参与需求分析和系统设计，掌握需求并从测试的角度对系统的健壮性、可测性等方面提供建议。

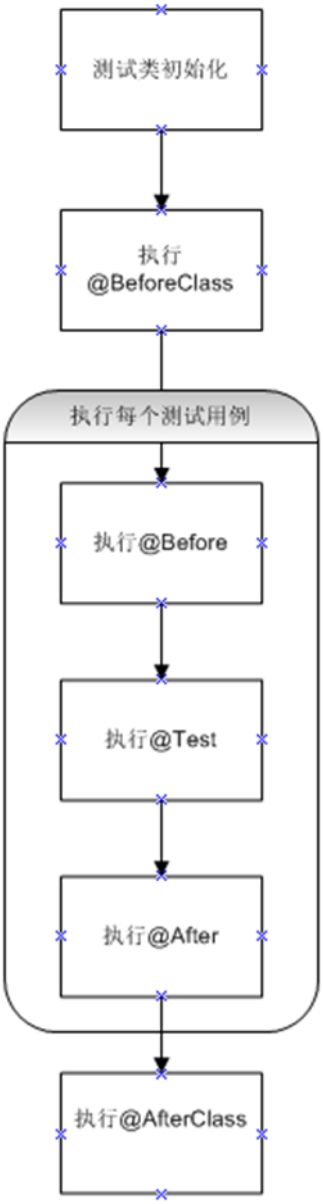
在开发工程师写接口说明和编码阶段，接口测试工程师设计测试用例，编写测试脚本，并对已经完成的接口功能进行测试，通过则纳入持续回归系统，否则将 bug 反馈给开发人员。

在上述各流程中，最有技术难度的就是编写测试脚本，使测试脚本发挥最大作用的就是持续回归系统，下面的内容着重介绍这两个方面。

三 、接口测试框架选型

Junit----最基本的测试框架

对于 Java 系统而言，选用 JUnit（http://www.junit.org/）做为基本测试框架是个不错的选择，也有些人偏爱 TestNG（http://testng.org/）， TestNG 在测试运行方面有些特性比 Junit 强，如分组运行、重新运行失败的用例等。但笔者认为从可扩展性方面考虑，Junit 仍然是第一选择。在 Junit4版本，通过注解的方式定



义了一系列流程，可以在测试类中方便使用满足不同阶段的需求。

下面的代码输出明确表明了其运行规则。

```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class Junit4Sample {
@BeforeClass
public static void beforeClass(){
    System.out.println("BeforeClass,to be run once before any of the test methods in the class");
}

@Before
public void before(){
   System.out.println("Before,to be run before the {@link org.junit.Test} method");
}

@Test
public void testA(){
    System.out.println("Test,can be run as a test case");
}

@Test
public void testB(){
    System.out.println("Test,can be run as a test case");
}

@After
public void after(){
 System.out.println("After，to be run after the {@link org.junit.Test} method");
}

@AfterClass
public static void afterClass(){
 System.out.println("AfterClass,to be run after all the tests in the class have been run");
}
}
```

上面的流程考虑到了用例执行切入点的方方面面，再加上其断言语句和报表功能，可以满足一般的单元测试需求，但应对复杂的接口测试尚显不足。

四 、Junit+DbUnit——扩展的数据管理

接口测试往往很难避开数据问题，要在测试前使数据库处于已知状态，在测试后清理相关数据给下个运行一个干净的数据环境。举个简单的例子，要测试某

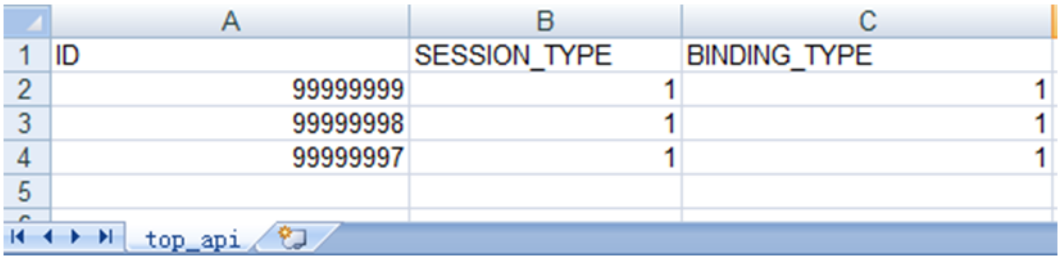
个接口的查询功能，需要在执行测试前向数据库插入一批已知数据，然后用特定的参数查询验证结果，在测试完成后需要清理之前插入的数据，恢复数据库到原有状态。Junit 提供了@Before 和@After 注解解决了流程上的问题，但直接用 Sql 操作解决此类数据问题就会使代码很难编写和维护，DbUnit 很好的解决了此类问题。使用 DbUnit 管理数据的一般思路如下：

1、将数据定义为数据集存放在 Excel 文件或 XML 文件里。

2、在测试执行前将数据集文件里的数据同步到数据库。

3、在测试执行后删除数据库里跟数据集文件匹配的数据。

DbUnit 支持 Excel 数据集和 XML 数据集，Excel 数据集的 sheet 名为表名，第一列为数据库字段名，其他列为数据，其格式示例如下。



	A	B	C
1	ID	SESSION_TYPE	BINDING_TYPE
2	99999999	1	1
3	99999998	1	1
4	99999997	1	1
5			

top_api

XML 数据集以 dataset 为根节点，子节点名为表名，子节点属性名称和值对应数据库字段名称和值，其格式示例如下

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
  <top_api ID="99999999" SESSION="1" BINDING_TYPE="1"/>
<top_api ID="99999998" SESSION="1" BINDING_TYPE="1"/>
<top_api ID="99999997" SESSION="1" BINDING_TYPE="1"/>
</dataset>
```

DbUnit 通过 DatabaseOperation 类定义了一系列的数据管理操作。

NONE：什么也不做

UPDATE ： 根据数据集内容更新表中数据

INSERT ： 将数据集内容插入表，可能会违反主键唯一约束

REFRESH ： 如果表中存在数据集中对应的数据则更新，不存在则插入

DELETE ： 从表中删除数据集中对应的内容

---------下面三条在公用数据库中慎用----------

DELETE_ALL ： 删除表中所有数据

TRUNCATE_TABLE：执行 truncate table 指令

CLEAN_INSERT ： 先执行 DELETE_ALL,再执行 INSERT

利用 DbUnit 提供的 Api 结合 Junit 的流程可以很方便的将数据集文件里的数据同步到数据库。

```
public class UserManagerTest {
static DatabaseConnection connection; //数据库连接
```

```
static XlsDataSet dataset; //数据集文件

    @BeforeClass
    public static void setup(){
        //定义数据库连接，包装了 java.sql.Connection，oracle 下需要指定 schemaName
connection = new DatabaseConnection(sqlConnection,schemaName);
//读取数据集文件
dataSet = new XlsDataSet(new DefaultResourceLoader().getResource("testData.xls").getInputStream());
}

@Before
public void initData(){
//执行插入操作
DatabaseOperation.INSERT. execute(connection, dataSet);
}
      @Test
      public void testFindByName() {
            User result = userManager.findByName("doe", "john");
            assertPropertyLenientEquals("userName", "jdoe", result);
}
@After
public void cleanData(){
//执行删除操作
DatabaseOperation.DELETE. execute(connection, dataSet);
}

@AfterClass
      public static void tearDown() throws SQLException{
           connection.close();
      }
}
```

利用 DbUnit 管理数据可以避免繁琐的 Sql 操作，数据相对比较容易管理，而且降低了编写和维护测试代码的成本，一些框架（如 Unitils）对此做了封装使其更加简单。

五 、Junit+DbUnit+Unitils----不错的解决方案

Unitils 提供通用的断言工具，支持数据库测试，支持使用 Mock 进行测试，并提供对 Spring 以及 Hibernate 的集成,使测试编码更加高效。以上文所说的数据管理为例，在测试类或方法加上@DataSet 注解，即可完成 DbUnit 数据集跟数据库的同步。

```
public class UserManagerTest extends UnitilsJUnit4 {
```

```
    @DataSet
    @Test
    public void testFindByName() {
        User result = userManager.findByName("doe", "john");
        assertPropertyLenientEquals("userName", "jdoe", result);
    }
}
```

上面的测试执行时会从测试类相同 classpath 路径寻找跟测试类同名的 xml 数据集文件同步到数据库。其数据管理策略和数据源连接配置在 unitils.properties 文件中。

```
DbUnitModule.DataSet.loadStrategy.default=org.unitils.dbunit.datasetloadstrategy.InsertLoadStrategy

database.driverClassName=oracle.jdbc.driver.OracleDriver
database.url=jdbc:oracle:thin:@yourmachine:1521:YOUR_DB
database.userName=username
database.password=secret
database.schemaNames=userSchema
```

Unitils 还为基于 Spring 的测试提供了便利的支持，可方便管理 Spring 上下文和依赖注入。

通过@SpringApplicationContext 管理 Spring 上下文，通过@SpringBean 实现依赖注入。

```
@SpringApplicationContext({"applicationContext-test.xml"})
public class UserManagerTest extends UnitilsJUnit4 {

    @SpringBean("userManager")
    protected UserManager userManager;

    @Test
    public void testFindUserByLastName() {
        List<User> users = userManager.findByLastName("Doe");
        assertPropertyLenientEquals("firstName", Arrays.asList("John", "Jane"), users);
    }
}
```

另外，Unitils 提供了大量的断言 Api 丰富了 Junit 的断言，对 EasyMock 做了封装降低使用复杂度，使用 Junit+DbUnit+Unitils 的框架模型，可以满足接口测试快速编码的需求，但也有不足之处，比如上面说的数据管理，只是解决了测试运行前跟数据库同步数据的功能，并不会在执行完成后自动清理数据，实际运用还需要做一些扩展。

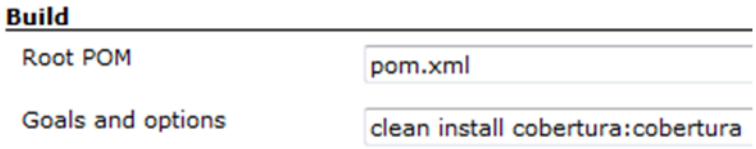
六 、持续集成和代码覆盖率

接口测试的另外一个重要价值在于其持续集成和自动化回归，当依赖很复杂时，不能再靠人工的方式估量某个系统的改动是否会影响到其他依赖系统，而高覆盖率的接口测试自动化回归可以为系统重构保驾护航。在 Java 领域有很多可以做持续集成的工具，推荐使用 Hudson（http://hudson-ci.org/）。 Hudson 安装、配置非常简单，界面友好，而且提供了良好的扩展特性，拥有丰富的插件。

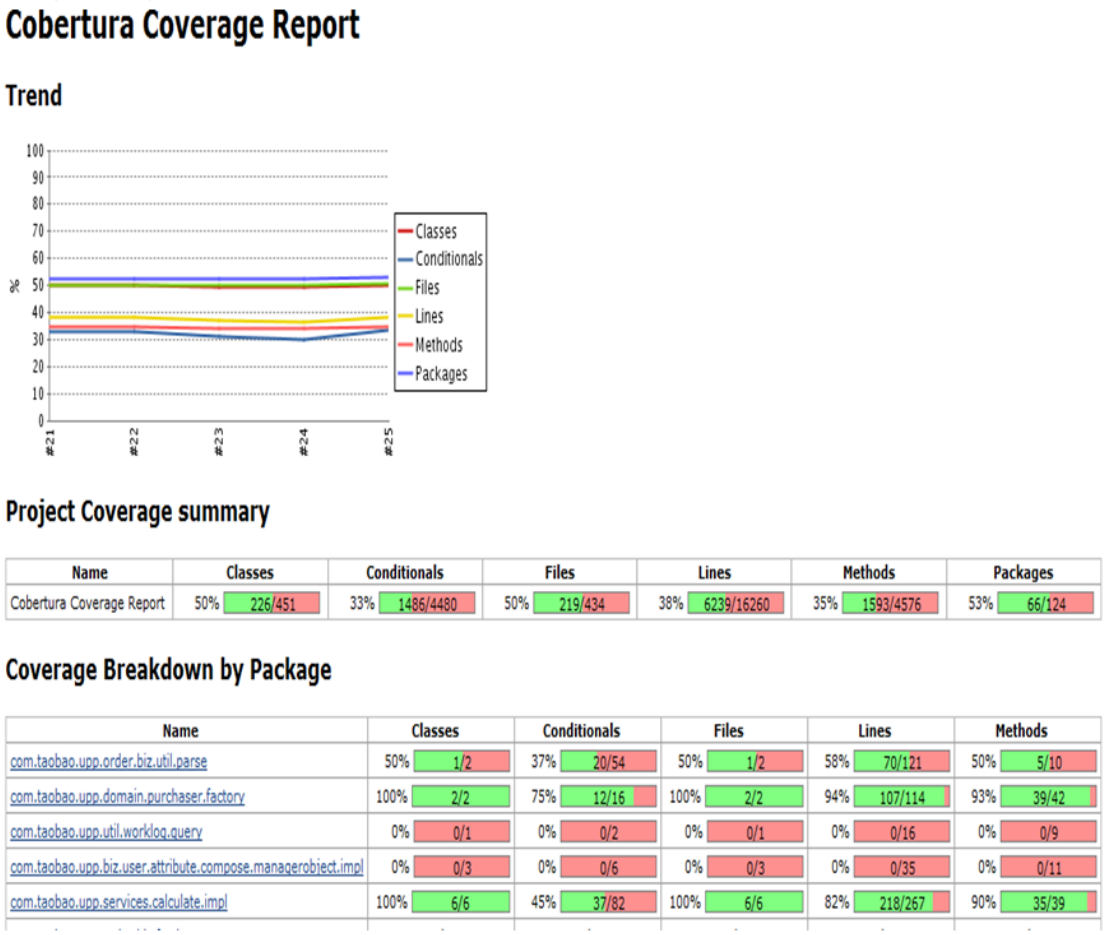
代码覆盖率是接口测试的指标之一，利用 Maven 插件和 Hudson 插件，可以在用例执行结束后出一份详尽的覆盖率报表，下面介绍利用 Cobertura（http://cobertura.sourceforge.net/） 来产生代码覆盖率报告。首先需要在 Maven 工程的 Pom 文件里配置 cobertura 的 Maven 插件。

```
<project ...>
    ...
    <build>
        ...
        <plugins>
            ...
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>cobertura-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <formats>
                        <format>xml</format>
                    </formats>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>cobertura</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            ...
        </plugins>
        ...
    </build>
    ...
</project>
```

其次在 Hudson 工程设置里配置



这样，当运行完测试后就可以产出一份详细的代码覆盖率报表，包含类级别，语句级别等。



七 、总结和思考

在接口测试实施过程中一些问题不可避免，下面几条是比较常见而且重要的问题。

1、用例设计和用例代码的同步问题

在接口测试流程中定义了用例设计的活动，这个设计一方面指导测试工程师编码，另一方面用于跟项目中其他人员交流，通过评审的方式补充场景覆盖。将这个设计转换为测试代码的过程费时费力，那么如何将这些设计自动转换为测试代码，在设计改动时又如何保持测试代码的同步？

2、测试代码框架问题

接口测试需要掌握的技术很多，也有很多的框架可以选择，这就导致学习成

本很高，往往需要多种框架组合来提高编码效率，而且有些开源框架并不能满足一些复杂的需求，需要针对框架做扩展。这就带来另外一个问题，如何降低接口测试的技术门槛和学习成本

3、回归报表的集中展现

利用 Hudson 可以很好的完成回归工作，报表展现也不错，但这个仅仅限于单个项目，当项目很多时，如何能拿到一个集中的报表，通过各条子系统的测试代码通过情况了解整个系统的健康状况？

Web 网页安全测试的一些方法

作者：文青山

摘要：

本文主要对各类书籍以及网络上所介绍的相关 Web 安全测试的方法进行了一定的归纳，并结合实际工作经验进行了一些粗略的禅述，由于 Web 安全测试具有较高的难度性，在此也仅仅是依据一些常规测试点进行了粗略的策略上的描写，其内容的正确性和广泛性也没得到验证。另外本文档所针对的测试类型是 Web 网页端的，没有对服务器的安全进行涉猎，由于个人能力的问题肯定有许多地方还有其它测试方法或者方法出错的地方，希望能够得到指正。

关键词：web 网页安全测试

一 、Web 常有的安全漏洞

通过查询相关资料，Web 应用程序存在的常有漏洞在此做了简单的介绍，具体内容请参考下面的内容。

1.1 错误配置

App.config 中存在的一些错误配置、Sql 服务器、Web 服务器的错误配置为非法用户的攻击提供了一些机会。

1.2 隐藏字段

查看源代码中有些信息被注释掉，但这些信息在间接上为攻击者提供了一些信息。另外，许多 Web 应用允许恶意的用户修改 HTML 源文件中的一些字段，为他们提供了攻击的机会。

1.3 后门和调试漏洞

开发人员常常建立一些后门并依靠调试来排除应用程序的故障。一些常见的后门使用户不用口令就可以登录或者访问允许直接进行应用配置的特殊 URL。

1.4 跨站点脚本

一般来说，跨站点编写脚本是将代码插入由另一个源发送的网页之中的过程。

1.5 参数篡改

参数篡改包括操纵 URL 字符串，以检索用户以其他方式得不到的信息。

1.6 Cookie 安全

网站常常将一些包括用户 ID、口令、帐户等的 cookie 存储到用户系统上。

通过改变这些值，恶意的用户就可以访问不属于他们的帐户。

1.7 缓冲区溢出

缓冲区溢出是恶意的用户向服务器发送大量数据以使系统瘫痪的典型攻击手段，Web 应用缓冲区溢出攻击的典型例子也涉及到 HTML 文件。如果 HTML 文件上的一个字段中的数据足够的大，它就能创造一个缓冲器溢出条件。

1.8 直接访问浏览

直接访问浏览指直接访问应该需要验证的网页。

1.9 Sql 注入攻击

主要通过 web 与数据库的交互漏洞，来获取数据库的访问、操作、控制权限。

1.10 Web 认证攻击

主要是通过猜解、探索有用的组合，从而获取目标站点的部分或全部的控制权。

1.11 敏感数据的传输机制

敏感数据的传输需要加密，并且加密方式应具有一定的强度。

1.12 验证码生成方式及有效性检查

OCR 光学识别技术的发展，致使一些工具可以识别一些验证码。

1.13 日志文件

如日志文件泄露业务关键数据。

1.14 IE 内存泄漏

引起 IE 内存泄漏的主要情况为 js 对象实例跟 DOM 对象的相互引用、"内部函数引用(Closures)"以及 DOM 插入顺序泄漏，其中最常见的就是 JS 对象实例跟 DOM 对象的相互引用，对基于对象的 JS，一个通常用法是通过封装 JScript 对象来扩充 DOM 对象，在构建的过程中，通常在涉及 DOM 对象时，建立一个对 DOM 对象的引用，DOM 对象也建立一个指向 JS 对象实例的引用，这就形成了一个循环，虽然不管 JS 调用 DOM 还是从 DOM 反向找到实例都非常方便，但如果在对象销毁或 Document Unload 的时候不去解除他们之间的引用，就会引起内存泄漏。

1.15 目录遍历

通过目录遍历漏洞有可能得到一些文件的访问权限。

二 、实施测试的大体方略

通过上面的介绍，我们已知道 Web 网页端所存在的常见的安全漏洞，下面将针对上面的漏洞类型的检测方法做了一些粗略的介绍。

2.1 利用 Visual Studio.Net 2008 的 code Analysis 来检测

C#2008 中提供了一个 security Rules 规则，利用此规则可以检测出一些代码级的安全隐患，其中提供的检测规则有：

CA2012:Catch non-CLScomplaint exceptions in general hanlers（抓住一般 hanlers 非 CLScomplaint 例外）

CA2103:Review imperative security(检测命令性安全)

CA2104:Do not declare read only mutable reference types(不要声明只读可变强用类型)

CA2105:Array fields should not be read only(数据字段不应为只读)

CA2106:secure asserts(保护断言)

CA2107:review deny and permit only usage(检查 deny 权限和 permit only 权限的使用情况)

CA2108:review declarative security on value types(检查有关值类型的志明性安全)

CA2109:review visible event handlers(检查可见的事件处理程序)

CA2111:pointers should not be visible(指针应为不可见)

CA2112:secured types should not expose fields(受保护的类型不应公开字段)

CA2114:Method security should be a superset of type(方法安全性应是类型安全性的超集)

CA2115:call GC.KeepAlive when using native resources(使用本机资料时调用 GC.KeepAlive)

CA2116:Aptca methods should only call Aptca methods(Aptca 方法应只调用 Aptca 方法)

CA2117:Aptca types should only extend Aptca base types(Aptca 类型应只扩展 Aptca 基类型)

CA2118:review SuppressUnmanagedCodeSecurityAttribute usage(检查取消非托管代码安全性的使用情况)

CA2119:Seal methods that satisfy private interfaces(密封满意足私有接口的方法)

CA2120:Secure serialization constructors(保护序列化构造函数)

CA2121:static constructors should be private(静态构造函数应为私有)

CA2122:do not indirectly expose methods with link demands(不要使用链接请求间接公开方法)

CA2123:override link demands should be indentical to base(重写的链接请求应与基相同)

CA2124:wrap vulnerable finally clauses in outer try(在外部 try 块中包装易受攻击的 finally 子句)

CA2126:type link demands require inhertance demands(类型链接请求需要继承和请求)

CA2127:security transparent assemblies should not contain security critical code(安全透明的程序集不应该包含安全的关键代码)

CA2128:security transparent code should not assert(安全透明代码不应断言)

CA2129:security transparent code should not reference non-public security critical members(安全透明代码不应引用非公有制安全的关键成员)

使用此规则验证代码的方法：项目名->右键->Properties->Code Analysis*->Security Rules->只选择 Security Rules->具体选择要检查的规则->Run code Analysis，执行后结果将在 Error list warnings 中显示，如显示为：

Warning 3CA2122:Microsoft.Security : 'SendPackMain.ConvertToBytes(IntPtr, int)' calls into 'Marshal.Copy(IntPtr, byte[], int, int)' which has a LinkDemand. By making this call, 'Marshal.Copy(IntPtr, byte[], int, int)' is indirectly exposed to user code. Review the following call stack that might expose a way to circumvent security protection: ->'SendPackMain.ConvertToBytes(IntPtr, int)' ->'SendPackMain.ConvertToBytes(IntPtr, int)' C:\Documents and Settings\wqs592452\桌面\MC_PackSend\PackSend\SendPackMain.cs 316 PackSend

2.2 采用黑盒的方法进行测试

2.2.1 App.config 配置检查

ASP.NET 应用程序在生产环境中部署时，需要检查 Web.Config 文件是否存在以下 10 个不正确的配置，如果存在以下配置可能会导致安全漏洞：

1、Disabling custom errors

Vulnerable:

```
<configuration>
<system.web>
<custom mode="Off">
```

Secure:

```
<configuration>
<system.web>
<customErrors mode="RemoteOnly">
```

2、Leaving tracing enabled

Vulnerable:

```
<trace enabled="true" localOnly="false">
```

Secure:

```
<trace enabled="false" localOnly="true">
```

3、Enabling debugging

Vulnerable: Secure:

<compilation debug="true"> <compilation debug="false">

4、Making cookies accessible through client-side script

Vulnerable: Secure:

<httpCookies httpOnlyCookies="false"> <httpCookies httpOnlyCookies="true">

5、Enabling cookieless session state

Vulnerable: Secure:

<sessionState cookieless="UseUri"> <sessionState cookieless="UseCookies">

6、Enabling cookieless authentication

Vulnerable: Secure:

<authentication mode="Forms"> <authentication mode="Forms">

<forms cookieless="UseUri"> <forms cookieless="UseCookies">

7、Failing to require SSL for authentication cookies

Vulnerable: Secure:

<authentication mode="Forms"> <authentication mode="Forms">

<forms requireSSL="false"> <forms requireSSL="true">

8、Using sliding expiration

Vulnerable: Secure:

<authentication mode="Forms"> <authentication mode="Forms">

<forms slidingExpiration="true"> <forms slidingExpiration="false">

9、Using non-unique authentication cookies

Vulnerable: Secure:

<authentication mode="Forms"> <authentication mode="Forms">

<forms name=".ASPXAUTH"> <forms name="{abcd1234…}">

10、Using hard-coded credentials

Vulnerable: Secure:

<authentication mode="Forms"> <authentication mode="Forms">

<forms> <forms>

<credentials> …

…

</credentials> </forms>

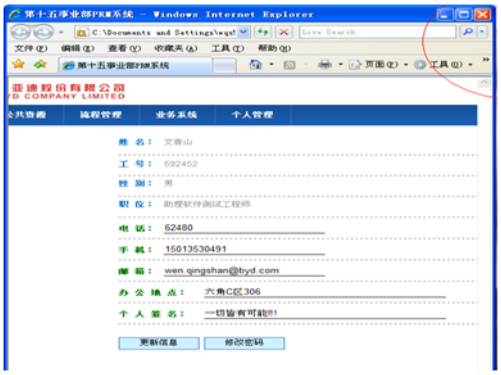
</forms>

以上内容请参考：《Top 10 security vulnerabilities in .NET configuration files》

2.3 隐藏字段

1、查看 HTML 源文件中注释掉的内容是否及时删除或泄露隐私信息

2、另存为本地表单修改后提交，即本地提交(注意 from 结构的表单)，如以



下示例为典型存在的安全缺陷

2.4 后门和调试漏洞

1、通过代码走审的方式去审核

2、对某些为测试目的而开的后门，在发布前进行测试是否取消（如验证码）

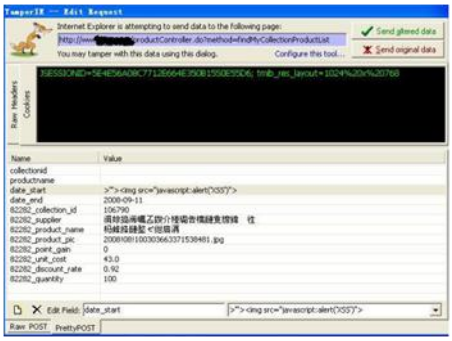
3、通过 Appscan 扫描

2.5 跨站点脚本

1、系统内各输入框各种构造 XSS 脚本测试，包括将其编译为 URL 编码和 HTML 编译后提交。

2、各个 URL 地址中的参数，构造 XSS 脚本测试，包括将其编译为 URL 编码和 HTML 编译后提交。

3、利用 TamperIE 截取所有输入框、下拉框、日期控件的 post/get 值，改变



成各种构造脚本测试，包括 URL 编译后、HTML 编译后发送。如：

4、有些类型的网站，通过 TamperIE 不能获得其 Post 所传输的值，我们可以利用.net 中 HttpWebRequest-HttpWebResponse 提供的基本方法，将 XSS 脚本测试的数据发送 Post 请求给 Web 应用程序，然后登录该网站，观察是否有一些恶劣的结果。（该方法请参考《.NET 软件测试自动化之道》中的介绍）

2.6 参数篡改

1、URL 参数传递时各种构造 XSS 脚本测试，包括将其编译为 URL 编码和 HTML 编译后提交。

2、各种 sql 注入脚本测试，包括将其编译为 URL 编码和 HTML 编译后提交。

3、改变为其它正常值，查看是否能获得其它不应该获得的值（良好的防护措施为，当 URL 参数值为其它时跳转至登录页）。

2.7 Cookie 安全

1、删除所有 cookie，将 cookie 完全屏蔽，正常运行系统（一般会导致系统无法运行，良好的系统，将会提示用户开启此浏览器设置）。

2、删除所有 cookie，将 cookie 设置为自动提示，正常运行系统，提示时随机选择启用和禁用（有可能会导致系统出现逻辑错误等，良好的系统，将会提示用户开启此浏览器设置）。

3、运行系统过程中，删除所有 cookie，继续运行系统（有可能会导致系统出现逻辑错误等）。

4、利用 Cookie Editor 工具修改系统 cookie（修改前需关闭 web，修改后重新提交访问）。

5、利用 Cookie Editor 工具查看关键数据是否加密（如密码）、过期时间是否设置合理以及是否正确设置了 Http only 和 Secure（勾选）等。

详细测试方法请参考陈能技《.NET 软件测试实战技术大全》。

2.8 缓冲区溢出

1、大数据量测试（如添加等）。

2、在相关输入框内提交超长数据（如长度大于 6553 个字符，一般说来输入框都要求设定 maxlength 属性）。

3、利用 TamperIE 截取所有输入框、下拉框、日期控件的 post/get 值，发送超长数据（如长度大于 6553 个字符）。

4、对上传文件处进行严格的容量测试和异常格式测试(注意 0KB 的文件上传)。

5、URL 参数传递值为特殊字符集或超长字符，包括将其编译为 URL 编码和 HTML 编译后的字符。

详细介绍请参考陆臻、顾健《测试与安全测试的关系》。

2.9 直接访问浏览

1、利用 Xenu 获取每个界面的链接，采用功能测试的方法，观察是否自动跳转至首页（登录页）。

（也可以利用 QTP 进行此项测试，将获得的链接存在入 EXCEL 中，使用

语句 systemutil.Run "iexplore.exe",sUrl,sUrl 参数化，当然应该还加入一些判断测试成功与否的逻辑。）

2.10 Sql 注入攻击

1、代码走查，数据库链接部分是否是通过参数化(或存储过程)实现的。

2、系统内各个输入框 sql 注入字符或组合。

3、URL 参数传递时各种 sql 注入字符或组合。

4、利用 TamperIE 截取所有输入框、下拉框、日期控件的 post/get 值，改变其值为各种 sql 注入字符或组合。

5、利用 VS2005 中 code Analysis 中的 CA2100:检查 Sql 查询中是否有安全漏洞来检查。

2.11 Web 认证攻击

1、主要在登录模块、新建账户/密码、修改密码、找回密码、权限分配等模块进行设计上和功能上的验证，防范暴力攻击等，具体测试点可参考下面的内容：

A：

登录模块安全性检查点	检查方法
1、应避免设计“记住我”等功能如设计有“记住我”等功能，则最好不记住密验证码	UI 评审时
2、监控“登录”的日志，并在用户登录之后，告知上次成功登录的时间和来源 IP，以及从那以后无效登录尝试的次数	UI 评审时
3、登录界面最好没有采用 Cookie 的方法保存相关信息，如采用 Cookie，则查看 Cookie 未记录密码且禁用 Cookie 后，有相应提示	代码检查
4、登录界面传值最好是采用 POST 方法	代码检查
5、登录界面的数据查询方式是通过参数查询用户和密码	代码检查
6、要有验证码功能，且验证码具有一定的难识别能力（如较多噪点等），最好从服务器端传送	功能验证
7、同一 IP,10/20 次登录失败后应采用相关帐户锁定策略（如当日不能再次尝试登录，要有相关提示信息）	功能验证
8、错误登录时，应给予较模糊的提示语“如用户名和密码错误”	功能验证
9、复制成功登录之后的每个页面或查询等 URL 地址，在另一个浏览器，检查是否能绕过登录界面	功能验证
10、在历史记录中访问存储的站点 URL 地址，检查是否能绕过登录界面	功能验证
输入一些错误值，如某个为空、密码或用户名为特殊字符、非中文字符、Sql 注入语句、JS 语句等，检查错误处理异常（参照登录模块的	功能验证

功能测试用例）	
11、正式发布后，应清空所有数据库中的用户名和密码	验收测试时
12、正式发布后，检查性能测试或自动化测试时设置的万能验证码是否取消	功能验证
13、网页保存到本地之后，表单提交	功能验证
B：	
新建帐户/密码模块安全性检查点	
1、用户名长度规则，长度>4；	功能验证
2、用户名字符规则，数字、字母（大小写）、下划线、点（最好如此）	功能验证
3、用户名不能同名	功能验证
4、密码应再次确认，即需要有密码、确认密码两项	UI 评审时
5、密码长度规则，长度>6	功能验证
6、安全密码规则，字母、数字、特殊字符的组合（参照功能测试点）	功能验证
7、输入一些错误值，如某个为空、密码为特殊字符、非中文字符、Sql 注入语句、JS 语句等，检查是否有异常（参照登录模块的功能测试用例）	功能验证
8、密码与用户名不能相同	功能验证
9、要有验证码功能，且验证码具有一定的难识别能力（如较多噪点等），最好从服务器端传送	功能验证
10、密码提示问题应提供多个，且具有一定的复杂性	功能验证
11、如通过 Email 发送注册信息，且需要通过某个链接，才能激活，则这个链接需要在一个时间内有效，且该链接中不能包含密码等重要信息(对安全性要求较高的网站，应采用这种方式）	功能验证
12、同一 IP，同一天连续注册超过 50（100）次，则给予相关的锁定机制（有提示信息）	功能验证
13、如注册时为默认密码，则登录之后需要在一定时间内给预修改	功能验证
14、注册成功后，应跳转到登录页面，而不是直接登录	UI 评审时/功能验证
15、数据插入方式是通过参数插入相关信息	代码检查
16、网页保存到本地之后，表单提交	功能验证
C：	
修改密码模块安全性检查点	
1、需要有“旧密码”、“新密码”、“确认新密码”	UI 验证
2、参照功能测试部分，进行各个错误数据测试	功能验证

3、监控“修改密码”日志，通过 Email 发送用户密码被修改的信息，但此信息不得泄露任何关于密码或关键数据的信息	功能验证
4、数据更新方式是通过参数更新的方式	功能验证
5、密码安全规则，长度，字符组合	功能验证
6、修改的新密码不得于旧密码相同	功能验证
7、网页保存到本地之后，表单提交	功能验证

D：

找回密码模块安全性检查点	
1、需要经过用户输入，且提示问题应跟注册时一样具有一定的复杂性	功能验证
2、如通过 Email 发送找回的密码，则应只能在较短时间内执行且该 URL 中不能暴露一些关键信息（较好的解决方案）	功能验证
不能有“暗示”功能	功能验证
3、要有验证码功能，且验证码具有一定的难识别能力（如较多噪点等），最好从服务器端传送	功能验证
4、如果安全性要求极高，则需通过外带方式，即打电话到呼叫中心，且呼叫中心需要有相应规则	功能验证
5、同一 IP，同一天连续找回密码超过 20（50）次，则给予相关的锁定机制（有提示信息）	功能验证
6、数据更新方式是通过参数更新的方式	代码检查
7、网页保存到本地之后，表单提交	功能验证

2.12 敏感数据的传输机制

利用 Wireshark 截取关键数据的提交，如银行卡号、密码等，观察是否加密（针对网络窃听）。

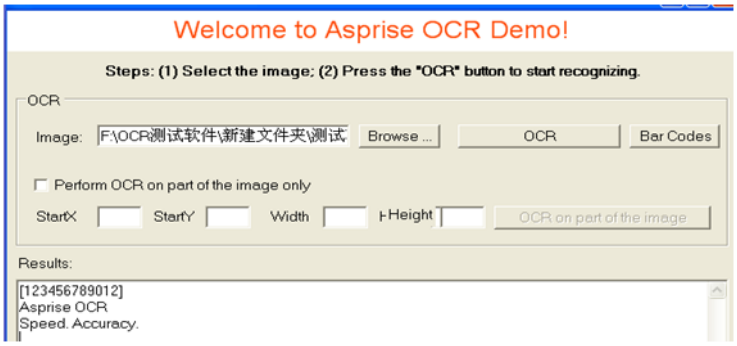
2、利用 WireShark 多次截取相同数据的提交，观察密文是否有关联（针对重放攻击）。

2.13 验证码生成方式及有效性检查

1、下载最新的 AspriseOCR.dll 库，开发 OCR 的应用软件或下载现成的软件，观察是否能够认识验证码。（OCR 只能识别一部分较弱的验证码。另外 lr9.5 以上版本的软件有 OCR 功能，能够识别一些验证码）。

如图片：

识别后的效果：



2、验证码生成的随机方式是否妥当，注意每次进入登录页时的验证码是否相同、以及从收藏夹中打开或历史记录进入登录页时的验证码是否相同。

2.14 日志文件

1、检查日志文件中是否暴露了敏感业务数据。

如下面的日志暴露了数据库的密码：

无法连接数据库：Data Source=10.9.146.182:Initial CataLog=T_Common User ID=sa;Password=123456; Aconnection was successfully established with the server,but then an error occurred during the login process.(provider:TCP Provider,error:0-远程主机强迫关闭一个现有的连接。)

2、日志文件过大时的处理方式，当日志文件(txt)超过 1G 时不仅有可能导致功能的实现，也有可能导致服务器岩机或在网页端暴露出一些异常。

2.15 IE 内存泄漏

1、利用 JSLeaksDetector 插件来检测

JSLeaksDetector 通过在访问每个 URL 结束时给出测试结果，如果没有发生内存泄漏，那么 URL 显示为绿色，如果发生了内存泄漏，显示为红色，可以记录下发生内存泄漏的详细信息，左侧部分显示发生内存泄漏的代码位置为粗体字，在中间的两格中显示详细信息与 CALL STACK，右侧显示完整的代码

2、利用 sIEve 插件来检测

SIEVE 通过在地址栏输入要访问的系统地址来进行操作测试，中间直接显示要访问的系统界面，下栏显示 COM 和 DOM 的使用情况，右侧显示实时数据：内存使用情况，内存泄漏等，如果发生内存泄漏可以通过右侧数据看出，然后点击 show leaks 的按钮可以看到发生内存泄漏的详细信息

2.16 目录遍历

1、在 URL 地址栏构造类似../../../../../../../../../../../boot.ini 的字符。../可以呈增加多个。

三、工具扫描的方法

1、开源软件(Wapiti)

Wapiti 是一个开源的安全测试工具，可用于 Web 应用程序漏洞扫描和安全检测，Wapiti 执行的是"黑盒"方式的扫描，可以直接对网页进行扫描。可以检测以下漏洞：文件处理错误

数据库注入（包括 PHP/JSP/ASP SQL 注入和 XPATH 注入）

XSS（跨站脚本）注入

LDAP 注入

命令执行检测（例如：eval(),System(),passtru()等）；

CRLF 注入

具体使用方法请参考陈能技《.NET 软件测试实战技术大全》。

2、商业软件（AppScan）

这是一款商业类的 Web 漏洞扫描程序。AppScan 在应用程序的整个开发周期都提供安全测试，从而测试简化了部件测试和开发早期的安全保证。它可以扫描许多常见的漏洞，如跨站脚本攻击、HTTP 响应拆分漏洞、参数篡改、隐式字段处理、后门/调试选项、缓冲区溢出等等。具体使用方法，请参考《AppScan Standard Edition 入门》。

比照法在行业产品测试中的应用初探

作者：朱志敏

摘要：

本文从比照法的定义出发，举例论证在行业产品中应用此方法的意义和价值，基于分类分层方式的测试应用构建出测试人员能力分层体系，对敏捷开发类的行业产品测试工作具有指导作用，也为测试管理工作提供了宝贵的借鉴经验。

关键词：比照法 行业产品 能力分层

一 、词条释义：

比照之名由来已久，在百度百科上搜索其释义，主要有以下两种解释：

◆ 按照已有的(事例、律条、标准等)。

如：明 唐顺之 《凤阳等处灾伤疏》:“比照 嘉靖 二年事例，截留十数万石，委官分赈济。”

◆ 比较，对照。

如：《宋史·选举志一》:“命州郡守倅结罪保明，比照字迹无伪，方许帘引注籍。”

近代以后，比照一词应用在各文人笔下的事例也是层出不穷，在此不一一而足。

比照法或比较法最初是语言学研究的一种方法，用它来揭示语言间的源流关系，此后更为广泛的应用在化学、生物、物理、生产等一系列的自然科学和社会活动领域。通过比照法的应用，将实际输出的结果与标准进行比较，能够更迅速、准确地判定输出结果是否符合预期，是否偏离正常轨迹。目前科学管理中最常用的 6 sigma 、TQM 中都有比照法的身影，其常用工具--过程能力指数 CPK、柱状分析图等工具等都是应用比照分析的例证。

作为质量保证的软件测试工作，比照法也可以为测试工作提供一种方法论的指导。

二 、行业属性分析：

1）行业是指从事国民经济中同性质的生产或其他经济社会的经营单位或者个体的组织结构体系，如纺织业，汽车业，冶炼业、装备制造业等。

2）按照生产过程中物质属性是否发生改变又可将行业划分为离散型行业和流程型行业，混合型行业是属于这两种行业的组合，如制药、化妆品、造酒等行

业。

3）因为行业属性及各家企业的生产组织方式的差异，也造成了软件在对业务匹配性上的千差万别，但是万变不离其宗，总有一条线索可以为我们的工作提供测试依据和结果的正确性指导。

三 、比照法应用示例：

对于行业性产品而言，其区别于通用产品的最大特点就是软件具有了行业特性，很多功能很难在通用产品中见到，这对于测试人员而言就是一种极大的挑战。测试设计中，预期或者输出结果是一个重要的组成部分，这就决定了测试设计人员在设计输出结果时必须严谨、科学，否则就会给应用者造成极大损失。

以下笔者将结合实际经验来分析一下比照法是如何应用在实际的测试工作中的：

3.1 与通用产品比照：

与通用产品比照相对来说更加容易，一般而言，一个系列的产品其业务逻辑、规则往往是固定的，是前人的经验积累和总结，比照标准产品的控制规则进行测试对完善行业产品功能，稳定行业产品质量有较强的参考意义。

通用产品也是各行业产品依赖的基础，拿业务环节举例，常用的规则包括：多计量组应用、权限控制（包括数据、金额权限）、审批流、工作流、单据编号规则、多模板、打印（套打、联打等），根据控制的规则，很容易判断当前的行业产品控制是否正确。此种方法适用于行业特性不突出，且与通用产品存在接口的功能点，比照控制的一致性，可以让测试迅速定位产品问题，开发快速处理问题，并做到与通用产品的平滑对接。

3.2 与其他行业产品比照：

*3.2.1 若通用产品中很难找到与当前测试产品的原型，可以在其他行业中寻找比照模型。*比如通用产品为 C/S 架构的，而当前测试的行业产品为 B/S 架构，且当前测试的行业产品需要与通用产品进行数据传输，此时可以借鉴本体系内其他行业类似架构产品的处理模式。结合 B/S 结构下产品通用测试技巧，重点测试：多浏览器模式下产品应用正确性、通过 B/S 推式生成到 C/S 产品中数据控制与回写正确性、同一标准产品单据上当前行业与其他行业接口的数据传递逻辑是否一致。

例如：A 行业为已有行业，B/S 结构生成到通用产品的销售订单，需要记录业务部门、客户、交期、业务员、单价和金额（含税）等关键信息，若当前测试的 B 行业也生成销售订单，但记录的信息中缺少了部门，或者单价和金额为无税信息，则此种处理就是错误的。

3.2.2 若某个行业 A 支持动态展现列功能，而 B 行业也需要实现此功能，则

测试时参照 A 行业的相关业务控制即可发现问题。

例如: A 行业按照存货动态展现其颜色属性：

存货名	颜色属性
C00	白色、黑色
C11	黑色、红色

光标焦点对准 C00 行，其右侧的颜色列展现为两个栏目：白色、黑色，同理，光标焦点对准 C11 行时，其对应右侧的颜色列展现两个栏目值：黑色、红色。比照此种控制逻辑，若 B 行业动态展现时，C00 出现了三个列：白色、黑色、红色，或者 C11 出现了三个列：空白列、黑色、红色（或者空白列+白色、红色，或者黑色、红色+空白列）均为错误。

*3.2.3 若本体系内没有可参照的产品，可以参考体系外的其他行业产品，*如伙伴、客户或标杆企业已有的其他同质产品等。此种情形相对较少，但是作为借鉴，在拓宽测试人员思路，增强行业产品易用性方面有很大的帮助。

例：求标准偏差公式：微软中就给出了常用的计算公式：STDEV(number1,number2,...) Number1,number2,...? 为对应于总体样本的 1 到 255 个参数。也可以不使用这种用逗号分隔参数的形式，而用单个数组或对数组的引用。而 SQL 中以如下公式来表达：标准偏差σs＝SQRT (((Avg-value1)* (Avg-value1)+(Avg-value2)* (Avg-value2).... (Avg-valuen)* (Avg-valuen))/（n-1）) 说明：SQRT 代表表达式的平方根。首先将组织的数据通过微软的标准偏差公式进行计算，再将相应数据输入行业产品中，比照计算结果是否与微软 excel 中得到的结果一致，若有差异且差异较大则说明程序计算存在错误。同理，其他一些计算项目也可以应用此种方法来进行验证。

3.3 与客户典型场景比照：

很多时候，行业产品是没有同类产品可以参考的，往往是集体智慧的结晶---基于客户典型应用场景，由产品经理抽象出产品定义，需求人员根据业务场景再具象成产品原型，开发进行架构设计、代码实现。由于此类行业产品比较专业化，很多规则属于企业特有模式，加之需求规格不会面面俱到，测试在做用例设计和实际测试时很难设计出遍历所有逻辑分支的用例。比照客户场景的测试对于验证产品功能匹配性帮助作用十分明显。只要能够满足客户常用的业务模式就可以认为产品功能是满足需求的，因为企业常用的业务功能只占用软件功能的 20%，即通常所说的二八法则，测试工作中遵循此原则就能够保证 80%功能的匹配要求（good-enough 原则）。

例如：某 LED 行业里常用频率和光谱来决定料品的唯一属性值，而光谱又是由二极管颜色和波长构成的二维分布曲线，其常见光谱系如下表所示：

二极管颜色	波长 P （nm）

蓝色	460-465
绿色	550
红色	680-700
红外	910
硅光电	

客户下达订单时订单数量见下表：

料品名	二极管颜色	波长 P （nm）	数量（件数）
IN/GAN LED 二极管	蓝色	460-465	100
IN/GAN LED 二极管	绿色	550	200
IN/GAN LED 二极管	红色	680-700	100
IN/GAN LED 二极管	红外	910	200
IN/GAN LED 二极管	硅光电		100

该 LED 企业为客户发货时通常是以包为单位，每包里同一规格的料品要求满足如下配比关系：

二极管颜色	波长 P （nm）	配比关系
蓝色	460-465	9
绿色	550	5
红色	680-700	3
红外	910	7
硅光电		11

即每包的数量为 35 件，按照通常情况下的算法，客户总包数应为：订单数量 700/每包规格 35=20 包，但此处因为存在要求每包比例关系，实际上的打包数量应为：

料品名	二极管颜色	波长 P （nm）	数量（件数）	每包件数	每包最小规格数	总包数(尾数单独打包）
IN/GAN LED 二极管	蓝色	460-465	100	9	35	12
IN/GAN LED 二极管	绿色	550	200	5	35	40
IN/GAN LED 二极管	红色	680-700	100	3	35	34
IN/GAN LED 二极管	红外	910	200	7	35	29
IN/GAN LED 二极管	硅光电		100	11	35	10=9（整包数+1尾包数）

涉及到实际应用时，因需要考虑每包里的件数配比关系，就不可能按照总数量除以每包规格的方式进行打包发货；但若允许对尾数进行合并尾数打包，计算出的总包数才为 20 包。

因此在进行测试设计时需要综合考虑尾数是否合并打包这两种场景，这样设计出来的用例覆盖度更高。

例如：某化工企业规定了其采购结算合同如下：

a）Tfe≥ 62%以上时，64%>Tfe≥ 62%（按月加权平均），每上升 0.1%，以 60%为基数，单价奖 2 元/吨；Tfe≥ 60%（按月加权平均），每上升 0.1%，以 60%为基数，单价奖 1 元/吨；当 Tfe≤58%（不加权平均单独结算），每下降 0.1%，以 60%为基数，单价罚 2 元/吨；当 Tfe≤56%（不加权平均单独结算），每下降 0.1%，以 60%为基数，单价罚 3 元/吨；当 Tfe≤54%（不加权平均单独结算），每下降 0.1%，以 60%为基数，单价罚 4 元/吨；

b）S≤0.3%（允许加权平均），每上升 0.01%，以 0.3%为基数，单价罚 2 元/吨；当 S≥0.34%（不加权平均单独结算），每上升 0.01%，以 0.3%为基数，单价罚 3 元/吨；当 S≥0.38%（不加权平均单独结算），每上升 0.01%，以 0.3%为基数，单价罚 6 元/吨。

初看这个功能很适合用等价类划分的方法来进行用例设计，辅以边界值法应该可以满足基本的测试设计要求，但是这种设计却不能完全满足客户实际应用场景的要求。对于此类客户而言，最基本的原则是将企业的损失与付出降到最低，永远保证结算的价格是对企业最有利的。那么，两个指标 Tfe 和 S 是如何组织计算方式从而使企业达到结算价格最低的呢？此时我们应该尽量比照企业的实际结算单据规则，这样更加便于测试活动的展开，以下是等价类划分的值域范围，我们拿其中部分边界指标来做说明：

基准价格（60%Tfe and 0.3S）	Tfe（含量%）	S （含量%）	指标关系	计算公式
200	>=60	<=0.3	And	(60-Fe 加权)*10+(0.3-S 加权)*20
200	<= 54	>=0.38	Or	(60-Fe)*40+(S-0.3)*60
200	58<Fe<60	0.3<S<0.34	Or	100（计算时用基价来扣减此结果）

按照第二行计算出的结果可能为负值，而实际业务中不可能存在结算单价为负的情况，因此可以重点验证程序是否做了结算单价为负的校验。对于第三行，因合同中未对此类情况进行说明，若单纯按照等价类划分的话就会把这类情况漏掉，此时就不进行扣减计算，造成客户结算时出现损失。而实际业务中是经常会出现折价情况的，我们设置一个常量值，这样保证检验指标出现最后一种情况时也会进行扣价，将客户的损失降到最低。

3.4 与行业通用规则比照：

客户的业务千差万别，对行业产品的需求也自然是不一样的，此时，借鉴行业通用规则来辅助测试工作往往可以达到事半功倍的效果。虽然在第一种方法中我们说过可以比照通用产品的控制规则来进行测试，但很多情况下在 A 行业适用的控制规则并不适用于 B 行业，此时，比照本行业通用规则来进行业务数据组织、流程的验证就更加重要了。

例 1)：某厂需要从仓库领取钢材，通常情况下，钢材是以条状、块状进行存储的，且不会轻易分割，而钢材也一般是专材专用，不会存在多领或者业务部门互相占用材料的情况。领料规则就是不允许超可用量、应领量进行出库，但是钢材又是以重量为计量单位，实际领用时会出现超出应领量的情况：如应领 50 公斤，但一根钢材净重为 51 公斤，此时若按照通用产品的参数设置就无法完成出库。而针对此类行业通用规则，行业产品增加一个参数，保证在不允许超可用量，超应领的前提下进行出库，领料数量不大于 50*（1+允许出库上限）的范围，既满足了业务应用，又是对通用产品测试思路的一个有益补充，同时也是对通用产品需求的一个良好扩展，一举多得。

例 2)：?EAN-13 国际商品条码坚称 EAN-13 国标码，是当今世界上使用最广的商品条码，已成为电子数据交换（EDI）的基础。EAN-13 国标码由共计 13 位数字组成，前三位代表国家代码，中间四位代表厂商代码，后五位代表产品代码，最后一位是校验码，根据前 12 位数字计算得出。这种条码不同于企业内部的产品条码，其应用标准要求非常严格，使用国标码的企业一般都需要在工商局有备案。在测试时，可以比照国际通用标准进行校验，如国家代码不等于 4，或者输入不存在不存在的厂商代码及产品代码，也可以尝试输入非数字的字符等操作，如果这些情况下可以生成国标码，或者生成的国标码长度不等于 13 位数字，均属于控制的错误。对于行业通用规则的了解便于我们准确验证条码类及 RFID 等新型电子设备与软件数据传输的正确性。

四 、比照法应用的范围及要求：

1、适用范围：

比照法适用于敏捷开发类的产品，即开发团队规模在 30 人以下，产品交付周期较短，需求、产品设计规格说明书较为简略的情况。一般来说，敏捷开发类产品，因开发规模小，周期短，需求易变动，需求文档不够详尽，此时测试人员可参照的文档与规范就比较少，往往根据测试人员自身经验和能力来把握产品测试进度和产品质量。很多时候，行业类产品是某行业几个项目原型的交集，需求角色可能会缺失，测试工作基本上属于“摸石头过河”，在此类情况下，比照法的应用就显得非常重要。

2、应用要求：

首先，比照法应用时要求其比照的对象是标准、规范化的，即比照成熟的产品、规范化的流程及行业标杆，否则标准频繁异动对应用比照法会产生重大影响，造成测试结果的不准确。

其次，比照法应用时要求测试人员具有较丰富的行业知识和产品知识，能够明辨产品哪些地方是不合理之处，具有需求提炼的能力。针对上一节中比照法应

用实例的描述，我们可以将测试人员的能力对应划分为四层：基础产品层、行业产品层、项目层、行业层，测试人员不同能力的层面可以承担相应的测试任务。

具有基础产品层能力的一般为初级测试人员和外包人员，他们对通用产品的功能已经了解，可以完成简单行业产品的测试工作；具有行业产品层能力的测试员一般拥有一年以上测试经验，初步了解了该行业产品的总体特征，可以完成行业产品中稍复杂功能的验证；项目层能力的测试人员拥有较为丰富的项目交付经验，参与过某个行业项目的测试工作，对客户实际应用场景有比较深入的了解；行业层能力的测试人员拥有跨多个行业产品的测试经验，丰富的经验使其在测试中可以根据客户的功能描述提炼出通用需求，并基于业务流程层面进行行业产品的验证（基本可以模拟进行客户验证或者β测试）。

分层分类能力体系可以帮助比照法应用时明确实施主体和对象，帮助测试人员准确定位，厘清职业发展的上升通道，激励测试人员通过不断地学习来完善自身，从而最终实现个人职业发展目标。

3、应用原则：

- ◆ 沟通是万能钥匙，可以打开任何未知世界的大门
- ◆ 角色互换，换位思考会获得意外惊喜
- ◆ 善于总结经验并形成分享机制
- ◆ 照猫画虎有时候很有用，尤其是其他人已经总结出相关经验时

五 、综述：

比照法在实际应用中还有很多方方面面，因篇幅和时间限制不能一一展开。笔者建议根据分层次的能力体系，全方位应用此方法，可以使产品最大程度满足客户的实际应用场景，避免为客户输出华而不实的行业产品，也可以避免经常面对客户“不好用，不专业”抱怨时的尴尬。

当然，对于行业产品的测试方法还有很多，在此只列举出一种，期待达到抛砖引玉的效果，与更多人来进行学习和分享。

软件架构师 vs 资深测试工程师

作者：董杰

也许你是一个头衔为资深测试工程师，高级测试工程师，测试工程师的测试从业人员，但在本文中什么头衔并不重要，重要的是你是否希望在自动化测试以外的领域不断提升自己的测试功力，有一天能与软件架构师一起畅谈产品架构设计，模块设计，甚至能辅导年轻的软件架构师如何设计一个有高可靠性的高质量架构。本文起这个标题不只是“吸引眼球”，同时也是告诉大家测试是从需求阶段就已开始，资深测试工程师从需求阶段开始就已与软件架构师一起合作，一起PK。

软件架构师与我们测试有啥关联吗？我的观点是：软件架构师与资深测试工程师大部分的 IT 知识是相通的，从其中一个微观角度看：当软件的架构被设计出来后，程序员需要用编程语言来实现设计逻辑，而测试员需要用测试语言（testcase 等）来不断验证实现的质量和驱动软件进行设计改进，当然我们 tester 也会用到一些自动化测试的编程语言。而在软件架构被设计出来前，测试人员与软件架构师是可以一起合作分工的，测试人员一样可以在需求澄清中与开发人员一起工作，在架构设计阶段一起讨论最终选择的技术框架，讨论如何让架构可支撑起多种需求，特别是如何设计才能满足可靠性和性能等质量强相关的需求。如果测试人员经验比软件架构师经验更丰富，你甚至可以影响和指导软件架构师对架构的某些设计做出很大的重构，给软件架构师提供某些设计方案也是可以的。

可为什么国内很少有测试人员参与产品的架构设计？有人会说：“测试人员能力和经验少呗”。我的观点与你一样：“说的对”。的确是这个原因，导致这个现状的主要原因有 4 个：

1、工作经验少了：产品架构设计人员往往有 3－5 年以上的工程化项目经验，而测试人员呢——可能才毕业，或只工作了一年，或是做测试管理很久了，或是主要精力放在了自动化和工具实现上，对商业产品的架构设计关注投入时间不够，对产品设计没有和软件架构师一样吃透。结果在架构设计阶段，明显测试人员的工程化经验比开发架构人员少，那么你又怎么能与开发一起完全平起平坐，甚至自己都觉得在需求和架构设计阶段，自己可发挥的作用太少了，还是想早点回去搞自动化脚本，心里来着实在些。但反过来如果一个项目组在需求和架构阶段，是这样一个组织结构：软件架构师只做软件开发 5 年，且第一次独立承担商业化的整产品设计；资深测试工程师是一个一直在技术一线工作了 8 年，且经历了多个不同架构的产品工程化的全过程（见多识广）。那么资深测试工程师不但能与这个年轻的软件架构师一起在需求和架构阶段工作，保障产品的可靠性设计

及架构的质量属性，更能辅导软件架构师做正确的事，牵引他把架构设计从质量角度考虑的更全面些，更长久些，减少他架构推翻重来的风险。

举一个我的案例：某项目就是这样一个背景，我是项目中工程化经验最多的人，软件架构师是第一次领导设计整产品的架构。在需求阶段，我一眼就发现他的可靠性需求，可测试性需求，性能需求，兼容性需求，扩展性需求，描述过简单过粗，甚至可以说几乎不合格，那你能想象在架构设计时他会把可靠性、性能，扩展性，可测试性这些设计好吗？最起码：可靠性是必须要保障的吧，没有可靠性的性能，UCD 一切等于零！后来，在设计阶段我带着他们一起做 FMEA，针对该类产品应用场景中常见的严重问题，我提出要在架构中补充对线程资源进行管理维护的模块，对进程状态进行管理修复的模块，对死锁进行自动修复的模块，这样能大大提升架构对各类故障的容错能力，这样他就不用穷举法对所有异常进行处理（而这些是这位年轻的软件架构师从未想到和考虑到的）。后来在大框架定下后，我又为高风险的大模块进行测试建模，在测试建模过程中，我把模块设计方案中的所有状态迁移图都吃透了的，并且每个测试建模图比原来的设计建模图都更完善，直接帮助年轻的模块设计师补充改进了设计状态迁移图，在编码前提升了设计质量。当然我们做测试的人，第一目标是质量，因此我们不要完全把开发的活都抢了，所以在需求和设计阶段，将你的精力放在如何提升需求和设计质量上，不断聚焦不断练习，你在如何设计高质量的软件架构上就能做到你的核心竞争力（从质量入手倒推架构的设计考虑是否足够）。当然要具备这样的能力需要一定的机遇和条件。下面 3 点就是这样的机遇和条件：

2、测试从业人员的计算机基础知识不够：计算机的基础知识包含：操作系统知识（进程、线程、锁、内存管理、信号量、IO 存储）；数据库知识（SQL 编程、索引、事务、联表）；数据结构（二叉树、hash、B＋树、双向链表）；编译原理（语法分析、语义分析）；面向对象编程（继承、接口）；TCP/IP。为什么我要列出这些知识点？因为我自己在过去的项目经验中涉及到了以上所列举的全部知识点，有幸于本科阶段的基础，所以在与开发定位问题，在阅读软件设计方案时并不费力。也许有朋友会说：编译原理的知识不做编译器就不会用到，有什么用啊？很遗憾，我也没做过编译器的项目，但我做过一个开发数据库的项目（不是应用数据库），当我们要解析 SQL 语句时，需要用的就是先做语法分析来验证 SQL 语法无错误，再做语义分析来选择走不同的处理流程，而做这个项目时我已 8 年多时间没看编译原理的书了，幸好当年编译原理的课学得扎实，关键算法也编程实现过，运用实现过语法分析和语义分析的代码，因此事隔 8 年多也能立刻在项目中应用开展工作。所以，根据我这些年不同项目的经验总结来看，有着扎实的计算机基础知识将有利于测试人员与开发设计人员一起平等交流业务。否

则，有一天你会发现你“只有产品应用知识，脚本开发能力，而这些知识毕业生2年就能掌握了”。

遗憾的是很多测试朋友，包括计算机科班出生的测试人员也都止步于毕业前计算机基础知识的认知，在从事测试后就不再继续深入学习计算机基础知识，认为那是开发人员的事。可恰好相反，我认为：“测试人员反而比开发人员更需要补充全面的计算机基础知识（含理论知识）”。因为我们做测试工作，特别是做测试分析设计工作就是一个把产品架构疱丁解牛的过程，你没有足够的基础知识框架，用什么工具来分解“这头牛身上的不同器官呢？”开发人员把一部分精力放在了编程实现的工具及语言熟悉上，这时测试人员正好可以利用这段时间来不断补充计算机基础知识。只有有了众多的计算机基础知识你才能知道如何来评估和验证被测产品的质量，建立起系统地产品质量评估体系，以及在需求和架构阶段（不需要精通编程语法）才能与软件架构师用计算机基础知识PK。对于不同行业的测试人员（科班和非科班），我可以给大家一些学习建议，针对不同的被测产品不断补充自己的计算机基础知识。如果被测产品是JAVA开发出来的，至少再好好把面向对象编程、TCP/IP知识、操作系统知识、数据库知识温习温习；如果被测产品是C++语言开发的，除了学习JAVA同学的课程外，数据结构也是要再温习的并了解C语言开发常犯的错误。 无论是什么项目的测试，操作系统知识都是一切的核心，因为它几乎融入了计算机的主要基础知识，任何软件架构的设计都会围绕和使用着操作系统的知识及其中的设计思想来开展的。如果一个来应聘测试的毕业生，他能对操作系统的基础知识有着全面深刻的理解，难道学一门脚本语言，掌握2－3种测试工具会对他是个难题吗？我相信未来他在日常测试中对bug的根因认识深度，以及对开发设计文档理解的效率上都会是不错的，能有机会成长为系统级的测试工程师。

3、缺少一个好的平台，缺少大型复杂项目经历：这个是机遇问题，但却是一个非常重要的客观因素。 如果你有机会参与2个代码几百万级，开发人员超过百人的项目，并且有机会学习产品的架构设计文档，那你是幸运的，因为你有了量变到质变的原始积累了，但有运气不代表你就一定会获得质的提升，还需要自己努力把实现业务流程的设计全搞明白，最佳检验你把设计流程搞明白的方法就是——灰盒测试分析与设计，你基于模块的设计流程开发出了原来纯从用户场景分析没有开发出的用例，那说明你真理解了该模块的软件设计了（通常在这个活动过程中，你会发现原来的软件设计居然还有这么多的问题，我还可以发现软件设计的问题啊）。

4、缺少独立开发一个软件项目的经历：我们在日常生活中常提倡“换位思考”，因为能让我们很容易体会到对方的痛苦和需求。做测试除了代表客户来测

试外，如果能理解开发人员的思维习惯，了解软件设计常见的问题类型，那么你就能在软件架构设计阶段更自信的参与工作。假设：你在读书期间，工作后时而自己独立设计一个小软件系统，你就能通过编程调试的经历，很快学习掌握该项目工作所需的脚本语言，C 语言，面向对象的高级语言。通过自己设计小软件系统的软件架构来运用计算机基础知识，运用过的知识才是你自己的，你才有知识与软件架构师一起沟通交流。

最后以我为例，我的 IT 学习时间大部分都在学习软件工程、产品开发实现的相关技术等，余下的时间才是了解业界的测试知识，并没有大部分时间只放在测试领域的知识上。总结一句话：“跳出测试来做测试，你会忽然发现 IT 专业技能的功力提升有了很大的一片空间，测试的舞台不只是在后期的 bug 查找和自动化实现。”大家对文中感兴趣的内容，有疑惑的看法，或者不同的观点，欢迎与我联系(testarchitectjie@gmail.com），一起探讨。

操作系统安全测评技术研究

作者：蒋发群 李艳波

一 、操作系统安全测评概况

随着计算机与网络技术的普及应用，信息安全已经成为关系到国家安全的关键因素。在计算机系统安全中，操作系统安全是整个计算机信息系统安全的基石。如果不经过安全测评，操作系统的安全性就得不到保障。随着我国操作系统研发的不断发展，研究操作系统安全测评技术已成为迫切的需求。

操作系统安全测评涉及到安全操作系统、安全等级评估、评估标准等多方面内容。目前国内在操作系统安全测评领域的研究还处在逐步发展的阶段，在操作系统安全等级评估方面已经取得了一定的成果，制定出了一系列等级评估相关标准。

随着操作系统在计算机系统安全中的重要作用越来越引起人们的重视，如何测评操作系统安全性成为一个重要的课题。本文首先介绍了操作系统安全测评的基础和准则，给出了操作系统安全测评的方法，最后对现有的操作系统安全测评系统进行了比较分析，为国内操作系统安全测评和国内自主开发安全操作系统提供技术支持。

二 、操作系统安全测评的基础和标准

为了对操作系统的安全性进行统一的评价，为操作系统产品厂商提供权威的系统安全性标准，需要有相应的安全测评标准来支持。目前，国际上信息安全评估标准的制定已经取得了长足的发展。

美国国防部于 1983 年推出了历史上第一个计算机系统安全评测准则 TCSEC(Trusted Computer System Evaluation Criteria)，又称桔皮书，从而带动了国际上计算机系统安全评测的研究。为了方便安全信息系统的统一评价，德国、英国、加拿大、西欧等纷纷制定了各自的计算机系统安全评价标准，其中较为著名的有 ITSEC（Information Technology Security Evaluation Criteria）、CC（Common Criteria for IT Security Evaluation）。我国在借鉴、吸收 TCSEC 和 CC 等基础上制定了相应的国家标准 GB/T18336-2001 和 GB/T 20008-2005 等标准。

表 1 国内外计算机安全测评标准的概况

标准名称	颁布的国家和组织	颁布年份
美国 TCSEC	美国国防部	1983
美国 TCSEC 修订版	美国国防部	1985

德国标准	前西德	1988
英国标准	英国	1989
加拿大标准 VI	加拿大	1989
欧洲 ITSEC	西欧四国（英、法、荷、德）	1990
联邦标准草案（ FC ）	美国	1992
加拿大标准 V3	加拿大	1993
CC V1	美、荷、法、德、英、加	1996
中国军标 GJB2646-96	中国国防科学技术委员会	1996
CC V2	美、荷、法、德、英、加	1997
国际 CC(ISO/IEC 15408)	国际标准化组织	1999
中国 GB17859-1999	中国国家质量技术监督局	1999
中国 GB/T18336-2001	中国国家质量技术监督局	2001
中国 GA/T 388-2002	中华人民共和国公安部	2002
中国 GB/T 20008-2005	中华人民共和国国家质量监督检验检疫总局、中国国家标准化管理委员会	2005
中国 GB/T 20272-2006	中华人民共和国国家质量监督检验检疫总局、中国国家标准化管理委员会	2006

基于相关安全需求，TCSEC 在用户登录、授权管理、访问控制、审计跟踪、隐蔽通道分析、可信通路建立、安全检测、生命周期保障、文档撰写等方面均提出了规范性要求，并根据所采用的安全策略及系统所具备的安全功能设定四类（A~D）及七个安全级别，从低到高依次为 D、C1、C2、B1、B2、B3、A1，各级别描述由满足安全策略、审计和保证的主要控制目标及文档要求共四部分组成。

我国的 GB17859-1999《计算机信息系统安全保护等级划分准则》把计算机信息系统的安全保护能力划分为五级，从低到高依次为用户自主保护级、系统审计保护级、安全标记保护级、结构化保护级、访问验证保护级，相关要求分别对应 TCSEC 的 C1 级、C2 级、B1 级、B2 级和 B3 级，并稍作调整。

信息技术安全评价通用准则 CC 基于欧洲 ITSEC、美国 TCSEC、加拿大 CTCPEC 及 ISO SC27 WG3 安全评价标准而形成，是目前最全面的信息技术安全评估标准。其提出了保护轮廓的概念，将评估内容划分为安全功能要求和安全保证要求等两个方面，并均按照类、族、组件的层次结构分别展开描述。其提供和定义了七个逐步增强的评估保证等级 EAL1~7，依次为功能测试级、结构测试级、系统测试检查级、系统设计测试复查级、半形式化设计测试级、半形式化验证设计测试级和形式化验证设计测试级。各评估保证等级结构上由评估保证等级名

称、目标、适用性说明和一组保证组件以及相应保证组件间的所有依赖关系构成，注意在每个保证级中至多包含从属于各保证族的一个组件。评估保证等级的增强通过同族保证组件的替换或它族保证组件的增加来实现。

三 、操作系统安全测评方法

操作系统的安全测评是实现安全操作系统的一个极为重要的环节。为了实现操作系统安全测评自动化，需要对操作系统安全测评方法进行研究。目前，评测操作系统安全性的方法主要有三种：形式化验证、非形式化确认和模拟入侵检测。这些方法可以独立使用，也可以将它们综合起来评估操作系统的安全性。

（1）形式化验证

形式化验证是分析安全操作系统最精确的方法，通常用基于形式化需求描述和设计方法构建起来的操作系统的安全测评过程。在形式化验证中，安全操作系统被简化为一个要证明的“定理”。定量断言该安全操作系统是正确的，即它提供了所有应该提供的安全特性，而不提供任何其它功能。但是，证明整个安全操作系统正确性的工作量是巨大而复杂的。对于稍具规模的安全操作系统而言，试图运用形式化方法加以描述或验证往往不大可能。为此，操作系统一般仅在某些安全模型的分析和设计上采用形式化描述的方式，因而只能针对有关安全模型展开局部的形式化验证，而并非整个安全操作系统。

（2）非形式化确认

确认是比验证更为普遍的术语，它包括验证，但也包括一些不太严格，但能让人们相信程序正确性的方法。完成一个安全操作系统的确认有以下几种不同的方法。

1）安全需求检查，通过源代码或系统运行时所表现的安全功能，交叉检查操作系统的每个安全需求。其目标是，认证系统所做的每件事是否都在安全功能需求表中列出，这一过程有助于说明系统是否完成了预期的安全任务。

2）设计及代码的安全检查

3）模块及系统安全测试

（3）模拟入侵检测

模拟入侵检测可作为操作系统安全测评的辅助手段。其基本出发点是在掌握操作系统安全漏洞的基础上，试图发现并利用系统中的安全缺陷。例如，利用安全漏洞扫描软件等尝试发现可能存在的系统配置等安全漏洞，并进一步通过植入黑客窥视文件或特洛伊木马窃取口令账号、完成关键系统文件非授权修改，甚至摧毁正在测试中的安全操作系统。

四 、操作系统安全测评系统分析

构建自动化的操作系统安全测评系统，为操作系统安全测评工作提供支撑，从而实现操作系统的安全测评工作的系统化、规范化和自动化。这对于改善和提高操作系统的安全测评速度及质量具有重要的现实意义和实用价值。本节内容将对代表性的操作系统安全测评系统进行分析，主要包括安全操作系统等级评测系统 SLES、基于数据驱动方法的操作系统安全测评系统等。

（1）操作系统安全等级测评系统

为了对不同的操作系统进行不同安全等级的自动测评，魏丕会等在国内率先给出了安全操作系统等级评测系统 SLES 的设计，如图 3.1 所示。

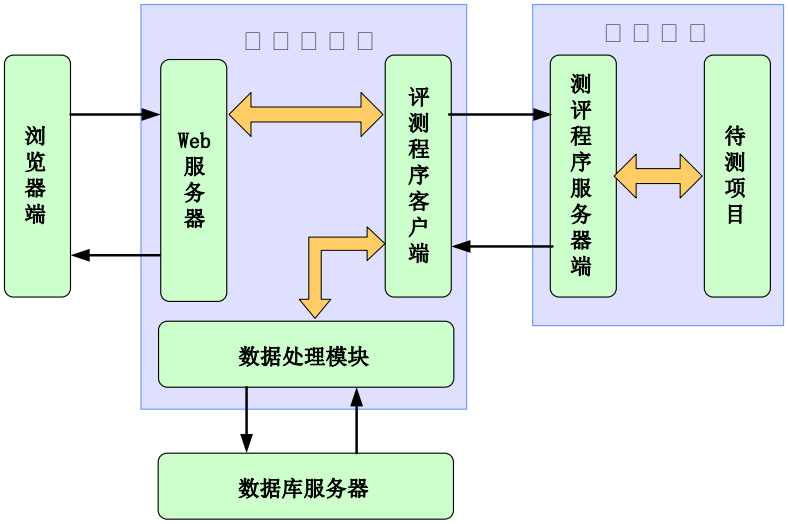


图 3.1 SLES 系统体系结构

SLES 的核心是测试服务器，它是根据规则具体执行评测的模块。SLES 系统中任何一步评测都是由规则来指导进行的。SLES 通过引入规则的概念，解决了对不同操作系统测评的跨平台问题。但是没有解决安全测评标准中的安全功能组件如何对应为具体的测试用例、测试命令的问题。

（2）基于数据驱动方法的操作系统安全测评系统

为了实现对 Unix/Linux 安全测评的自动化，陆幼骊等提出了基于数据驱动方法的操作系统安全测评系统，如图 3.2 所示。

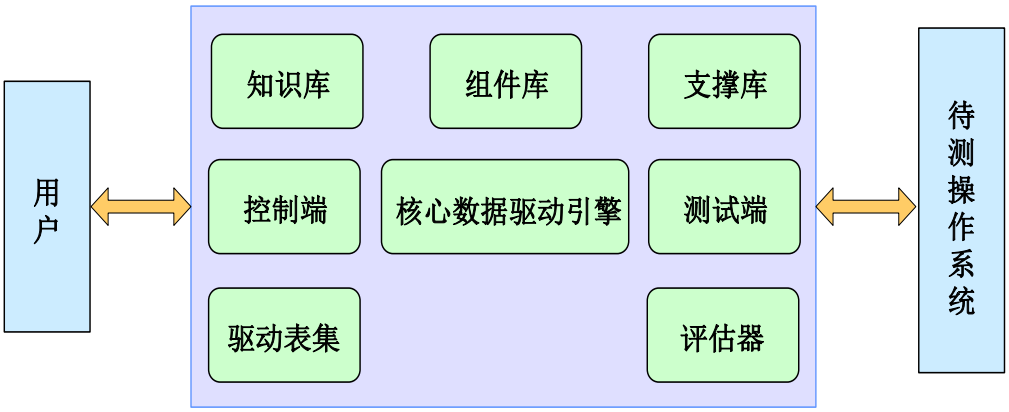
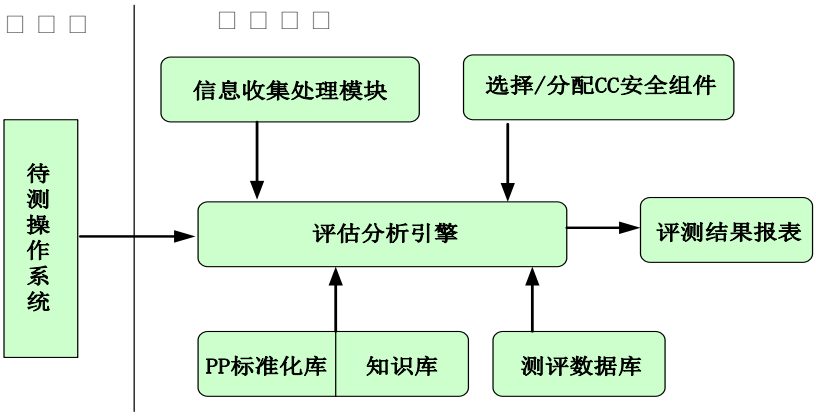


图 3.2 基于数据驱动方法的操作系统安全测评系统结构

该系统以自动化测评框架为基础，通过数据抽象将测试数据与测试脚本本身分离开，并将其包含在测试数据文件中，测试脚本的功能及执行由外部测试数据所引导。该系统利用了软件测试自动化中的模块化测试脚本方法，将编写好的测试脚本和测试例程存储在组件库和支持库中，充分利用这些脚本的重用性。同时利用软件测试自动化中的数据驱动的方法，实现测试需求到测试数据再到测试动作的转化，弥补了 SLES 系统不能将安全功能需求和测试用例对应的问题。但是该系统只适用于一种操作系统，测试不同的操作系统时需要重新建立测试脚本、测试例程、驱动表。

（3）基于 CC 标准的安全操作系统测评系统

金怡等利用 CC 标准实现了对操作系统的测评，提出了安全操作系统的测评系统 SOSES，如图 3.3 所示。SOSES 对安全操作系统安全功能测评的基础上，



根据不同的安全等级所对应的安全功能类不同的原则，对操作系统进行评估并给出它的安全等级。

图 3.3 SOSES 系统结构

此外，陈晨等通过分析上述典型的操作系统安全测评系统设计方案，并在吸取了其优点和融合了软件测试自动化方法的基础上，给出了一种改进的操作系统安全测评自动化框架 OSSEF。

鉴于操作系统的规模复杂性特征，其安全测评的工作量非常庞大。为此，操作系统安全测评过程的自动化已成为当前操作系统测评研究领域的热点方向之一。毫无疑问，如果能够实现或部分实现安全测评过程的自动化，则对于改善和提高操作系统的安全测评速度及质量具有重要的现实意义和实用价值。

随着计算机系统安全变得越来越重要，加强对操作系统安全测评技术的研究，设计自动化的操作系统安全测评工具或系统是一项很迫切的任务。这将成为我国信息系统建设的重要环节，不但能够确保整个信息系统根基的安全，而且能

够为我国安全操作系统的设计、开发提供有力的技术支持。

如何使用自动“Bug 邻域”分析来识别未修复 Bug

作者：淘宝宛蝴

尽管现在已经有了很多自动识别 Bug 的静态分析技术，这些技术可以帮助识别空指针引用类型的 Bug。但是极少有技术可以做到分析这类型 Bug 是否应该修复，及该如何修复。修复这些 Bug 有时不彻底，因为有其他相关的 Bug 没有得到修复。在这篇论文中，我们定义了“Bug 的完全修复”，它是指要修复由一个程序传递到另一个程序的无效值，包含空指针引用。在 Java 程序中，这种分析建立在“Bug 邻域（Bug neighborhood）”的定义上，“bug 邻域”包含一定范围内的一组无效值，文章展示了一种自动分析技术。假设程序开发中有两个版本 P 和 P'，我们将采用的动态分析，在第二个版本 P'中，识别出应当被修复的 Bug，并判断出这些 Bug 是否彻底被修复了。最后以空指针引用为例，源代码采用开源的工程，应用这项技术进行分析。分析结果表明，在这个工程中，许多的错误并没有完全被修复，因此程序在以后的执行中可能出现失败。

一 、总体介绍

Java 程序经常包含许多 Bug，例如对空值的引用，数组下标不正确，导致 Java 虚拟机抛出运行时异常。这些 Bug 包含一组无效值，由一个程序传递到另一程序，从而导致运行时异常。尽管许多自动化监测技术可以静态地发现这些 Bug，但是极少有技术可以做到分析这类型 Bug 是否应该修复，及该如何修复。一种修复该错误的尝试可能只修复了表面的错误，而未能彻底的完成修复，从而可能导致另外一次运行中出错。

以空指针引用异常为例来说明，这个异常是由于将空指针分配（null-pointerassignment， NPA）导致一个空指针引用（null-pointer dereference，NPR）。NPA 一般是由于一个空值的声明造成的，常包括下面三种情况：X=null；return null； foo(null)；。而 NPR 是在一个声明中，引用的变量为空。在图 1 中，左边灰色的方框描述在程序 P 中出现的空指针异常，NPA1 导致了 NPR1。NPR2 代表由 NPA1 所导致的，可能在 p'中出现的错误，NPR2 是否出现，取决于 NPA1 如何被修复。而 NPR3 代表 NPA1 所导致的其他空指针引用，它在 P 和 P'的另外一次运行中出现。

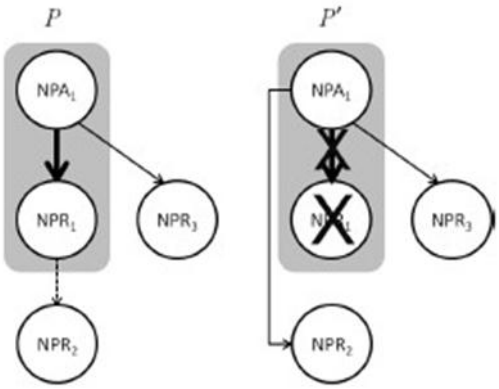


Figure 1. Example to illustrate an incomplete bug fix.

假设一个开发人员尝试去修复这些 NPR1 中出现的空指针引用错误，他采用的方法是在第二个版本 P'的开发中，在 NPR1 上加一个空值检验。图 1 右边的部分表现出相应的结果，这种改动消除了 NPA1 到 NPR1 的空值流，并且能保证在 P'的任何一次运行中，NPR1 都不会出现。但是这种方法可能会导致其他异常。因为 NPA1 可能会导致 NPR2，NPA1 中的空值也有可能会传递到 NPR3 ，开发尝试修复一对（NPA，NPR）时，如果 NPA 或者 NPR 有可能在 P'的另外一次运行中导致异常，那么这个 Bug 修复就不彻底。

现有的研究没有考虑到 Bug 修复的彻底性，并且本文的研究发现，很多实例中 Bug 都没有得到完全的修复。

首先谈一下“bug 完全修复”的概念，这种概念基于“bug 邻域”的定义，“bug 邻域”是指一组相关的无效值，如果想完全修复 bug，那么“bug 邻域”中的无效值要全部修复，文章中描述了修复 bug 时给“bug 邻域”带来的各种改变。本文介绍的技术可以适用于以下类型的 bug，它们将一组无效值从一个程序带到另一个程序，从而导致运行时异常，常包括空指针异常，数组下标异常，类抛出异常，这篇文章主要说明在空指针方面的应用。

第二，文章介绍一种动态的分析，它能识别 P 中的一组（NPA，NPR）在 P'中是否得到的修复，并能判断这种修复是否彻底。对于 P 中的一组（NPA，NPR），这种分析技术可以识别在 P'中对应的(NPA'， NPR？？')，接下来，它可以判断在 P'中，是否有其他的与 NPA'相关的 NPR 或者与 NPR'相关的 NPA，会导致 P'运行时抛出空指针异常。为了达成这个目标，这种技术采用 NPR'后向分析及 NPA'前向分析，判断某个 bug 的尝试修复是否与（NPA，NPR）相关，如果这种尝试修复已经进行了，那就判断这种修复是否彻底，这一步分析利用了过程间空指针分析工具“XYLEM”，这是这篇文章之前研究的一个成果。

在文章中我们使用开源代码和商用软件，演示个技术的实际运行效果，我们的研究表明“bug 邻域”包含的范围是非常大的，在很多实例中 bug 都没有得到

完全修复，这些成果可以提醒开发在尝试修复 bug 的过程中要注意完全修复 bug，以免在程序其他运行中导致异常。

这种技术的最大优势在于它可以自动发现未完成的 bug 修复，高亮标志出应当引起注意的代码，这项技术可以应用于交互式调错工具（debugging tool），对于一个 bug，当开发在修复时，可以给出建议如何才能彻底修复，因此，使用这种工具可以让 bug 修复变得更为有效。

这篇文章的主要贡献包括：

1、提供一种方法来帮助开发定位 bug 是否得到了完全修复，提供相关信息帮助其完全修复 bug。

2、技术上实现了空引用 bug 的定位

3、实验证明，“bug 领域”的范围是非常大的，(NPA， NPR)组也存在多种变化，所以很多 bug 修复都是不彻底的。

二 、bug 修复的彻底性

第一部分主要介绍 Java 程序中 NPR 的修复的相关概念。

图 2 中演示了(NPA1， NPR1)的“bug 邻域”，主要包括四种类型的 NPA，NPR：MaybeNPAs， Maybe NPRs， Forward NPRs， Backward NPAs.图中可以看到，Maybe NPAs 和 Maybe NPRs 分别与 NPR1 和 NPA1 相关，用实线来表明他们会在程序的其他运行中发生，Forward NPRs，Backward NPAs 也分别与 NPR1 和 NPA1 相关，用虚线来表明他们可能会在程序的其他运行中发生，这种可能性取决于(NPA1， NPR1)如何被修复。

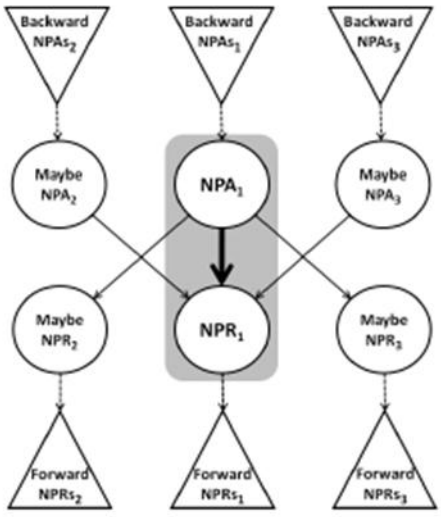


Figure 2. Bug neighborhood for (NPA_1,NPR_1) pair.

给定 （NPA1，NPR1） 组，“bug 邻域”分析将自动识别出邻域中其他的

NPA，NPR。

（1）Maybe NPRs：Maybe NPRs 是指在另外一次程序运行中，由于 （NPA1，NPR1） 修复不彻底，由 NPA1 引起的空指针引用，我们的技术采用 NPA1 前向分析来发现这些 Maybe NPRs ，在图 2 中主要包括两种 NPR 与 NPA1 相关，Maybe NPR22 和 MaybeNPR3 ，从而将（NPA？？，Maybe NPR22_ ）and （NPA？？，Maybe NPR3） 加到 （NPA1，NPR1） 的“bug 邻域”中。

（2）Maybe NPAs：Maybe NPAs 与 Maybe NPRs 类似，在另外一次程序运行中，由于 （NPA1， NPR1） 修复不彻底，空指针引用可能会导致 NPR1，因此采用 NPR1 后向分析来发现 Maybe NPAs。在图 2 中，有两种 Maybe NPAs 与 NPR1 相关，Maybe NPA22 和 MaybeNPA3， 从而将 （Maybe NPA22，NPR1） and （Maybe NPA3，NPR1） 加到 （NPA1， NPR1） 的“bug 邻域”中。

（3）Forward NPRs：由于程序改动，NPR1 不再是空指针引用，Forward NPRs 是指由 NPA1 引起的另一个空指针引用，这些 Forward NPRs 被 NPR1 隐藏，当 NPR1 被改动，在程序 P'中 NPA1 就不会导致 NPR1，取而代之，NPA1 导致 Forward NPRs 的产生，因此，我们计算 Forward NPRs 直到 NPA1 不会导致任何 NPRs，这是建立在 NPR1 和所有的 Forward NPRs 都得到了修复的基础上。在图二中有一棵树，根是 NPR1，叶子是 Forward NPRs，对于 n 个 Forward NPRs，（NPA1，Forward NPR1）…..（NPA1，Forward NPRn）都将加到（NPA1， NPR1）的“bug 邻域”中。

为了能完全修复 bug，“bug 邻域”分析要求对 Maybe NPRs 导致到 Forward NPRs 也进行同样的计算，否则，就算（NPA1， NPR1）修复了，由于 Maybe NPRs 导致到 Forward NPRs 仍然可能发生，也会导致其他的空指针异常，

（4）Backward NPAs：Backward NPAs 与 Forward NPRs 类似，当程序改动后，NPA1 不会再导致 NPR1，但是这些 Backward NPAs 可能就会导致 NPR1，这些 Backward NPAs 被 NPA1 隐藏，另外 Backward NPAs 还有可能隐藏其他的 Backward NPAs，因此只有当所有的 Backward NPAs 和 NPA1 修复了，才不会有任何的 NPA 导致 NPR1， 图二中 NPA1 上方的树根是 NPA1，对于 n 个 Backward NPAs，（Backward NPA1，NPR1）…..（Backward NPAn， NPR1）都将加到（NPA1，NPR1）的“bug 邻域”中。

与 Forward NPRs 计算类似，Maybe NPAs 导致的 Backward NPAs 也都要计算在内。

（5）定义：下面给出“bug 邻域”以及“完全修复”的定义说明

“bug 邻域”：（NPA1， NPR1）的“bug 邻域”是指多组（NPA，NPR）集合，这组集合由 MaybeNPAs， Maybe NPRs， Forward NPRs， Backward NPAs

共同导致，“bug 邻域”的大小就是（NPA，NPR）的组数。

“完全修复”：P 中（NPA1， NPR1）完全修复意味着 P'中“bug 邻域”的（NPA，NPR）集合为空，需要注意的是保证 bug 修复的正确性，因此有可能一个“完全修复”是错误的。

第二部分介绍尝试修复对“bug 邻域”带来的影响。具体情况参见图三。

（1）移除 NPA 及 NPR：这种修复可以删除（NPA，NPR）组，那么在 P'中（NPA，NPR）的“bug 邻域”为空，这种修复是彻底的。

（2）移除 NPA 和 NPR 之间的流：（NPA，NPR）的修复有可能只移除 NPA 和 NPR 之间的流，这种情况下，我们仍要考虑 MaybeNPAs， Maybe NPRs，并将在 p'中寻找它们，此外，因为 NPA，NPR 都没有消除，所以 Forward NPRs，Backward NPAs 也是需要考虑的。

（3）只移除 NPA：在 P'中删除 NPA，这样 p'中 NPA 无法导致 NPR，另外举一个例子，比如 NPA 中新建了一个对象，那么这个空值就无法从这里流向 P'中的 NPR，这种情况下，仍然可能存在 MaybeNPAs，它们会导致 NPR，但是在这种情况下，Maybe NPRs，以及 Forward NPRs 是不需要再考虑的。因此“bug 邻域”的范围缩小了，只要考虑 NPR 关联的“bug 邻域”。

（4）只移除 NPR：这种情况产生于在 P'中删除 NPR，另外一种情况就是，在 P'中 NPR 之前加一个条件语句，那么 NPA 就无法流入 NPR，这种情况下仍然要考虑 Maybe NPRs，因为 NPA 有可能会产生这些 Maybe NPRs，这种情况下 Backward NPAs，MaybeNPAs 是不需要考虑的。因此“bug 邻域”的范围也缩小了，只要考虑 NPA 关联的“bug 邻域”。

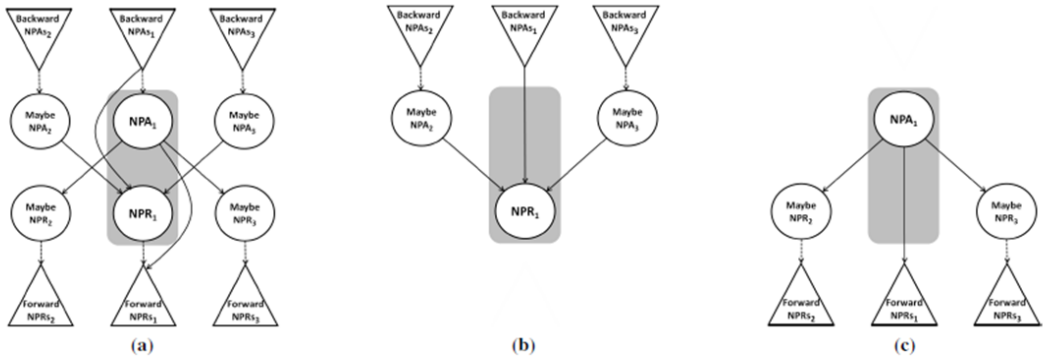


Figure 3. Potential bug neighborhoods after attempted fixes: (a) removing the flow between the NPA and the NPR, (b) removing only the NPA, and (c) removing only the NPR.

三 、自动分析

这一部分我们主要讲解自动分析技术帮助识别 P 中的空指针 bug 是否在 P'中尝试修复，并且判断出这些尝试的修复是否彻底。分析时要借助空引用分析工具 XYLEM。

foo(int i, j) { foo'(int i, int j) {

```
[1] x = null; // NPA [1]x = null; // NPA
[2] if ( i == 0 ) [2] if ( i == 0 )
[3] x = new C(); [3] x = new C();
[4] y = x; [4] y = x;
[5] if ( j > 10 ) {[5] if ( j > 0 ) {
[6]if ( x != null )
[7] x.m1(); // NPR[7] x.m1();
[8] x.m2(); // F-NPR [8] x.m2(); // F-NPR
} else { }else {
[9] if ( i == 0 ) [9] if ( i == 0 )
[10] y.m(); [10] y.m();
[11] x.m3(); // M-NPR [11] x.m3(); // M-NPR
[12] x.m4(); // M-F-NPR [12] x.m4(); // M-F-NPR
}}
} }
```

Figure 4. Example to illustrate XYLEM and bug-neighborhhood analyses.

在上图程序中，foo()是原始版本，foo'是修改后版本，在 foo'第六行加上了一个空值检验，从声明语句 S 开始，在声明中引用了变量 v，分析工具 XYLEM 可以识别 v 的空值流向 S 的路径，在分析过程中，它传递了过程间控制流图中的 v 状态的改变信息，开始的时候断言 v 在 s 中是空值，在整个路径回溯中更新 v 的状态,如果这种更新不连续，那么 v 没有空值流向 S，因此，回溯过程就会中断。

以上述程序为例，在 foo()中的第七行引用变量 X，分析时假定{X=null}，回溯时到达第五行的条件语句，加上（j>10）这个条件，第四行的语句并没有更新状态，从第四行回溯有两条路径，沿着其中一条路径到第三行发现分配了一个新的对象给 X，这个跟现在的假定{X=null}是相反的，因此，这条路径结束了，另一条路径则转到第二行，加上（i！=0）这个条件，接着转到第一行，发现 x 被分配了空值，这时就可识别出第七行的语句是 NPR。

algorithmBugNeighborhoodAnalysis

input（s_a,s_r）：(NPA, NPR) pair in the original program P

outputFix status: {“unfixed”,“complete fix”, “incomplete fix”}

declareBugNeighborhood’:pairs of reaching NPAs for s_r‘(mapped from s_r)

reachable NPRs for s_a‘(mapped from s_a) in modified program P’

begin

1. s_a‘=matching null-assignment statement in P’;

2. s_r‘=matching dereference statement in p’;
3. BugNeighborhood’= //initialization

//check whether s_r‘has reaching NPAs in P’

4. If s_r‘ null then //s_r‘exists in P’
5. Sa’=NPAsidentified by the XYLEM analysis starting at s_r‘
6. Foreacha’ Sa’ do add(a’,s_r‘)toBugNeighborhood’

//check whether s_a‘has reachable NPRs in P’

7. If (s_a‘ null) then //s_a‘exist in P’
8. S’$_{deref}$= derefences (excludings_r‘) reachable froms_a‘ in P’
9. Foreachd’ S’$_{deref}$ do
10. Sa’= NPAs identified by th XYLEM analysis starting at d’;
11. If (s_a‘ Sa’) then Add(s_a‘, d’) to BugNeighborhood’
12. Foreach (NPA, NPR) BugNeighbirhood’ do

Find forward NPRs,Backward NPAs

//set FixStatusfor(s_a,s_r)

13. if(s_a’,s_r’) $\in$ BugNeighbirhood’ then FixStatus=unfixed
14. else if BugNeighborhood’=$\emptyset$ then FixStatus=complete fix
15. else FixStatus=incomplete fix
16. return BugNeighborhood’,FixStatus

End

图五展示了“bug 邻域”的算法，算法输入一组 P 中的（NPA，NPR），即（sa，sr），算法首先判断有没有尝试修复（sa，sr），如果没有， bug 状态为“unfixed”，如果有，则判断在 p'中是否彻底修复，状态分别为“完成”或者“未完成”，算法的 1-2 行将 sa，sr 分别映射为 P'中空分配和空引用的声明，映射的计算与我们的分析成正交关系，它可以采用多种方式计算，有不同的计算成本和准确度。将 sa，sr 分别映射为 sa'，sr'后，算法剩下部分就是看是否有尝试修复以及存在尝试修复的情况下，修复是否彻底。

第三行初始化 bug 邻域，第四行到第六行判断 P'中是否有可到达的 NPAs（reaching NPAs）与 sr'相关，如果存在 sr'并且 sr'不为空，这个算法使用 XYLEM 分析技术，从 sr'开始，来识别 Sa'（可到达的 NPAs 的集合），对于 Sa'中的所有声明 a'，（a'，Sa'）都会被加入到“bug 邻域”中。

以 foo（）中（1，7）这个（NPA，NPR）组为例应用这一算法，在 foo'（）中，在将 NPR 对应到 foo'（）中的第七行，算法使用 XYLEM 分析，因为加了空值校验，找不到可到达的 NPA，因此 Sa'这时为空.

7-11 行判断是否有 sa'引起的 NPR，算法分为两步，第一步，算法进行前后可到达分析从 sa'开始，识别是否有 sa'引起的空值引用（第 8 行），第二步（9-10 行），对于每个引用 d'，识别出来后使用 XYLEM 来判断，对于每一个引用，sa'将被判断是否是 NPA，如果被判断为 NPA，将（sa'，d'）加入“bug 邻域”中，这两步的好处是利用 XYLEM 过滤掉不正确的可到达引用（这些引用是第一步识别出来的），第 12 行使用类似的前向，后向分析来识别 Forward NPRs，Backward NPRs，将他们加入“bug 邻域”，13-15 行判断修复状态，如果（sa'，sr'）存在于“bug 邻域”中，那么就会有空值与（sa'，sr'）相关，（sa'，sr'）也就没有修复，bug 的修复状态为“unfixed”，否则，有两种情况，如果“bug 邻域”为空，修复状态为“完全修复（complete fix）”， 如果“bug 邻域”不为空，修复状态为“不完全修复（incomplete fix）”，最后，算法返回果“bug 邻域”和修复状态。

将算法应用于 foo（）中的（1，7）组，从 foo'（）第一行（Sa'）开始，算法使用前向分析找出引用 7，8，10，11 和 12.对于每一个引用，算法使用 XYLEM，对于第 8，11，12 行的引用，XYLEM 判断出第一行的声明是一个可到达的 NPA，第 7，10 行，则没有发现可到达 NPA，因此，算法的第 13 行得到“bug 邻域”为{(1，8)，(1，11)，(1，12)}：第 8 行是一个 Forward NPR，它被识别出来并且尝试修复，11 行是一个 Maybe NPR，12 行是一个与 11 行相关的 Forward NPR，因此，这种修复是用来修复（1，7）这个组的，因为这一组现在不存在于“bug 邻域”中了，但是这种并不彻底，因为整个“bug 邻域”不为空。

四 、实验

为了评价这种方法，我们进行两次实验来研究“bug 邻域”和 bug 修复是否彻底。

A、数据准备

我们实验包括三个开源项目(Ant-1.6.0， Lucene-2.2.0， and Tomcat-4.1.27)，和三个商用项目(这里称为 App-A， App-B， and App-C)，表格 1 里列出了类，方法，字节码说明和(NPA，NPR)组的个数，我们分析时采用了 XYLEM 工具，实验主要包括两个部分：映射部分和“bug 邻域”分析部分，现在假设程序 P 中空指针 bug 为（NPA0，NPR0）。

Table I
SUBJECTS USED IN THE EMPIRICAL STUDIES.

Subject	Classes	Methods	Bytecode Instructions	(NPA, NPR) Pairs
Ant-1.6.0	1858	17204	443254	167
Lucene-2.2.0	381	2815	72691	86
Tomcat-4.1.27	260	4077	101075	97
App-A	278	3933	98225	63
App-B	169	1876	46286	119
App-C	2488	13746	340896	107

Table II
EIGHT BUG-NEIGHBORHOOD CATEGORIES FOR AN (NPA, NPR) PAIR.

Category	Maybe NPA Present	Maybe NPR Present	Forward NPR Present
1	no	no	no
2	no	no	yes
3	no	yes	no
4	no	yes	yes
5	yes	no	no
6	yes	no	yes
7	yes	yes	no
8	yes	yes	yes

其中映射部分将每句语句按照顺序进行映射，（NPA0，NPR0）的映射结果有可能在 P'中的另一个（NPA，NPR）组，也有可能没有任何对应的 NPA 或者 NPR。

“bug 邻域”分析时用了 XYLEM 来识别 P'中与（NPA0，NPR0）对应的 bug，过程中用了上文提到的 BugNeighborhoodAnalysis 算法，然后使用前向后向分析来发现 P'中的 Maybe NPAs，Maybe NPRs，ForwardNPRs，目前，还没有计算 Backward NPAs。

使用 P'中的 NPAs 和 NPRs 以及（NPA0，NPR0）在 P'中的映射，实验中创建出新的空指针 bug，联同映射得到的（NPA，NPR）组一起加入“bug 邻域”中，实验中将判断（NPA0，NPR0）的状态“not fixed”，“尝试修复但是不彻底”，“彻底修复”。

B、邻域分类及大小

（1）目标和方法：实验的目标是发现“bug 邻域”的大小和连续性，因此，我们针对每个表格 1 的项目进行试验。

为了俩接“bug 邻域”的连续性概念，我们将其分为表格 2 中展示的 8 类，每种分类基于 Maybe NPA， Maybe NPR， 或 Forward NPR 是否存在于“bug 邻域”中（我们目前的实验没有考虑 Backward NPAs）。

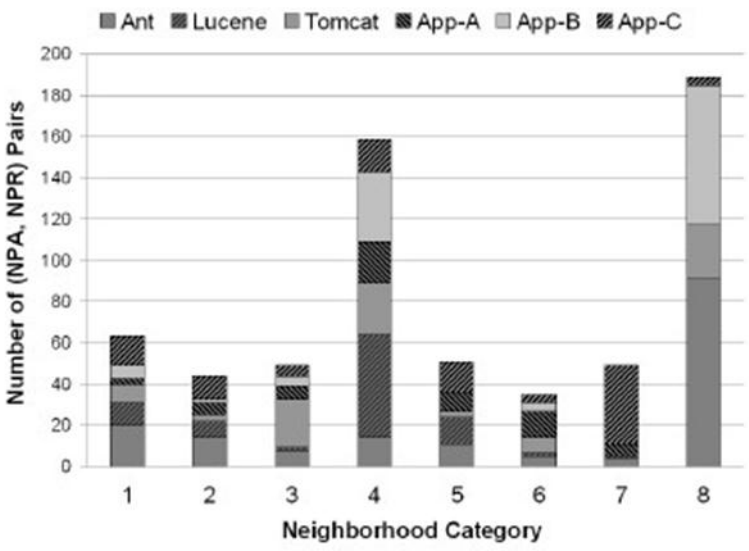


Figure 6. Number of occurrences of bug-neighborhood categories, aggregated over the subjects. Each bar segment shows the number of (NPA, NPR) pairs in a bug-neighborhood category that occur in a subject.

例如，第一类代表了最简单的“bug 邻域”，里面没有 Maybe NPA， Maybe NPR， 或 Forward NPR，因此，在这种情况下，图三中每一种尝试修复 bug 都是彻底的修复，第 8 类代表最复杂的“bug 邻域”。

为了计算“bug 邻域”的大小，我们定义三种大小：小（5 个一下（NPA，NPR）组），中（（NPA，NPR）组大于五个，小于十五个），大（（NPA，NPR）组高于 15 个）。

（2）结果与分析：图六展示了每一类“bug 邻域”出现的次数，横坐标是 8 种类别，纵坐标是每一种类别在不同工程中对应的数目，工程在每一个柱状图中出现的顺序都相同。第一个是 Ant，最后一个是 APP-C，综合所有的工程，第一类“bug 邻域”出现 63 次，其中 Ant 中出现 20 次，Lucene 中出现 11 次，Tomcat 中出现 9 次，App-A， App-B， Ant-C 分别出现 3，6，14 次，第八类“bug 邻域”共出现 189 次。

这项数据表明第八类“bug 邻域”出现次数最多，第四类也是较复杂的一类，出现次数也很多，另一项发现表明几种“bug 邻域”类别中，包含 Maybe NPRs 但是不包含 Forward NPRs 的经常出现，所以对于这种“bug 邻域”，开发可以多加关注 NPR，Maybe NPR，而不用考虑 Forward NPRs。

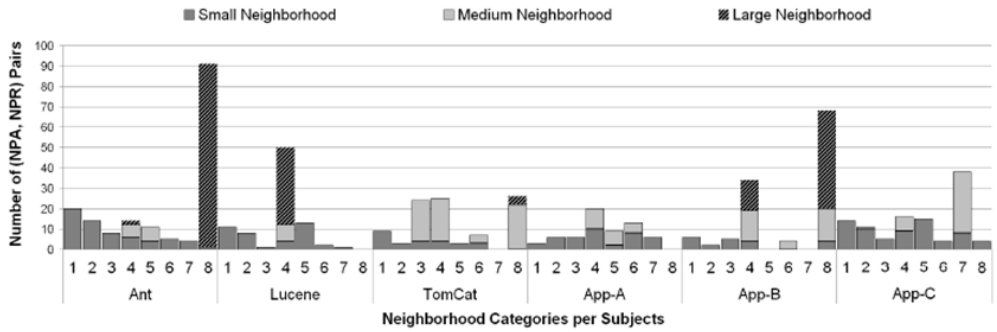


Figure 7. Number of occurrences of bug-neighborhood categories per subject. The bar segments show the number of (NPA, NPR) pairs in a bug-neighborhood category that have small, medium, and large neighborhoods.

图七中展示了数据的另一种视图，横坐标表示各个工程，纵坐标表示（NPA，NPR）组的数目，由图中可以看到，Ant 和 App-B 包含第 8 类“bug 邻域”的绝大部分，Ant 和 App-C 包含所有的类别，其他的工程除了 App-B 至少含有 7 类，App-c“bug 邻域”的分布与其他几个不同，它第七类“bug 邻域”最多，所有的工程中，第七类是含有最少（NPA，NPR）组的类别之一。

这项数据也显示了“bug 邻域”大小与类别之间的关系，所有大的“bug 邻域”都包含复杂的“bug 邻域”类别（4 和 8 类），中等“bug 邻域”一般包含 Maybe NPAs 或 Maybe NPRs（3 和 8 类），小的“bug 邻域”一般各种类别的都包括。

总的来说，复杂的“bug 邻域”经常出现，并且一般是大的“bug 邻域”，因此，文章中提出的“bug 邻域”的概念在实践中会很有作用，可以帮助开发提供 bug 的信息，帮助他们彻底修复 bug。

C．彻底修复 bug

（1）目标与方法：研究的目的是发现不彻底修复的出现频率。文章中考虑项目连续发布的情况，识别出两个连续发布的版本之间 bug 修复情况。

映射部分用来识别 P 和 P'之间（NPA，NPR）关系，同时带来了一些误报（报告为尝试修复但是并没有真正尝试），我们只想报道精确的结果，所有手工的改了四个项目的分析结果-Ant，Lucene，Tomcat，App-A，移除了误报。

（2）结果与分析

表格三中描述了不彻底的修复数目以及尝试修复的数目，在 Ant 中，26 个尝试修复中 4 个修复不彻底，其中这四个的类别是 2，4，4，5，图中显示了有 5 个不彻底的修复属于第四类，其他两个属于第二类和第五类，不彻底的修复中包含的（NPA，NPR）有小的“bug 邻域”，中等“bug 邻域”，和大的“bug 邻域”，最大的“bug 邻域”的大小是 36。

实验表明，尝试修复有可能不彻底，那么，与尝试修复过程相关的空指针 bug 有可能子啊程序的另外一次运行中出现，另外，这些未完的修复的“bug 邻域”有可能非常大，因此很难人工发现，例如，两个不彻底的修复的“bug 邻域”大小超过 30，如果没有自动化的分析，判断这些与（NPR，NPA）相关的组是否存在，以及定位这些组都将消耗大量的时间。

D、实验的一些缺陷

在实验中，一些操作也许会影响我们获得结果的准确性，我们使用 XYLEM 分析来实验，因此我们必须相信它所产出的结果，但是在映射部分，发现一些误报，我们采取手工的修复。

实验结果的普遍适用性还没有完全证实，但是该项成果已经应用于一些各种各样的实践中，因此我们可以得出结论，空指针 bug 的不彻底修复确实存在，并且它们的“bug 邻域”非常大。

另外就是实验只是针对空指针型的 bug，其他类似 bug 的分析将在后续工作中进行。

五 、相关工作

在之前的工作中，我们展示了识别以及修复造成运行时异常的 bug，当声明中出现空指针异常时，则采用 XYLEM 来识别 NPA，maybe NPAs ，maybe NPRs。这篇文章中讲述了 bug 如何被修复，定义了“bug 邻域”的概念和彻底修复的概念，采用两个程序说明了这一点。

前人的研究主要集中于错误信息库中的 bug，这些 bug 很常见，而我们的研究还可以应用于一种类型的 bug（包含一组无效值的 bug）。

Table III
NUMBER OF INCOMPLETE BUG FIX ATTEMPTS AND THEIR NEIGHBORHOOD CATEGORIES AND SIZES.

Subject	Number of Incomplete/ Attempted Fixes	Neighborhood Categories of Incomplete Fixes	Neighborhood Sizes of Incomplete Fixes
Ant	4 / 26	2, 4, 4, 5	Small, 3 x Medium
Lucene	3 / 17	4, 4, 4	Small, 2 x Large
Tomcat	0 / 9	—	—
App-A	0 / 7	—	—

Ayewah 的研究主要手工的影响功能使用的 bug 分类，他们也检查了程序修改时是否定位及修复 bug，但是没有说明如何修复 bug 才是彻底。

Spacco， Hovemeyer， and Pugh 说明了 bug 的演变，展示了将 bug 从一个版本映射到另一个版本的方法，报告了缺陷的周期和趋势和密度，也没有说明如何修复 bug 才是彻底。

Williams，Hollingsworth 建立了一种检查器来发现一类 bug，这类 bug 通常产生于使用一个函数的返回值之前没有去检测值是否为空，他们也建立了警告机制，在与已经修复的 bug 相关的函数前加一个检验，他们的工作在“彻底修复”方面与我们类似，但是没有对 bug 修复彻底作出分类。

现有的工作主要包括：基于已修复 bug 的类型给出修复 bug 的建议，相关的软件产品，基于信息挖掘给出 bug 的信息，我们的工作与这些有类似地方，但是不同的是，方法是基于代码分析而不是软件库中的挖掘信息。

六 、结论与未来工作

文章中讨论了自动识别未彻底修复 bug 的方法，引入了“bug 邻域”的概念，将 bug 进行分类，利用“bug 邻域”的概念来判断 bug 是否完全修复（看邻域是否为空）。

我们的方法可以协助调错，可以通过分析程序的不同版本发现未彻底修复的 bug，当发现这类 bug 时，开发可以利用“bug 邻域”中的现有信息来完全修复它们。

为了验证结果进行相应的实验，结果表明复杂的“bug 邻域”经常出现，尝试修复往往不彻底。

今后工作的方向包括：1、解决映射部分不精确的问题，将尝试其他映射技术，如 JDiff，能有更好的效果。2、继续研究未彻底修复 bug 与“bug 邻域”大小有无关系？开源项目汇总的未彻底修复 bug 与商用项目是否相同？是不是在开源项目中未彻底修复 bug 更多？3、对其他类型会导致无效值流的 bug 进行研究，我们将研究是否存在定义准确的错误类型以及修复结果类型，这些类型之间有什么关系。我们的目标就是帮助开发可以完全修复 bug，对于某些错误类型，我们也许可以使用自动方法修复。对于其他类型，则可以给开发提出建议帮助他们修复。

最后，我们将研究（NPA，NPR）的优先级，利用直觉可视化技术来展示“bug 邻域”，这些技术将减少人工调错的时间和成本。

重新定义需求分析

作者：成韩丽

1988 年，在苹果电脑，我度过了最糟糕的一周。苹果的开发系统小组开发了一个叫 ASMCVT 的工具，目的是为了要转换汇编源格式文件到另外一种格式化的更合适的新的汇编程序，该工具依然在开发之中。我被告知，由于相应的汇编测试团队的领导者，我将会在几天内收到这个工具。但是没有人逐字逐句的检查底层，直到开发者已经在下层的测试部分之上进行工作时，没有任何有关工具的细节。当我询问那个开发者，并且向他说明为了规划一套像样的规范，我们需要一个规格时，他告诉我，该工具非常简单并且写一个规格是完全没有问题的。

第二天他将一页纸放在了我的面前。那是一个漂亮的小文档，用 30 号的醒目黑体 Helvetica “ASMCVT”横跨页面的顶部，还有一些其他的标记。我记得，还有一些水平的标尺，开发者的全名、日期、在他们旁边还有一类带着“N/A”的标题，和大量的空格。如果隔着一段距离或许会被误认为是一个真正的技术文档——带着莫名的称赞说谁设计的这个文档模板。但是它仅仅包含一个有技术含量句子：“这个工具从旧的汇编格式文件转换成新的格式”。

那个开发者好像很诧异，当我践踏他的文档，并朝他皱起了眉头时。

一 、讨论风险

他提供了一个微弱的防御措施，“几乎有 1500 种不同的文件需要从旧的文件转换成新的文件。当然，你别指望我将它们全部关闭。”是的，先生，我会那样做，如果你能够编译他们，你就能够写出来。我下一步的计划是我的老板 Chris Brown，也是开发系统质量管理者。

“1500 种，Chris，这就是他所说的。但是看看这个，”我将规格书举过头顶然后仍下来，他像羽毛一样随着空气飘动。“他称这个为说明书，我拒绝测试这个产品直到他告诉我这是什么。”

Chris 似乎对于我示威的言行毫无反应：“Jim，”他说，“我们需要谈谈风险。”

使我惊奇的是，Chris 没有看到测试一个没有规格的工具问题。他像这样解释道：“ASMCVT 是一个将文本文件作为输入然后生产出一个新的文本文件作为输出的工具，这只是一个一次性的过程，它不会用任何方式改变原来的文件。源文件和新的文件都是人类能看懂的，在新文件中的错误将会很容易被发现，通

过新的汇编程序来运行它，并且将会打印错误报告或制造一个新的对象文件，该对象文件并不是和旧的文件及旧的汇编程序制造的对象文件毫无差别的。如果我们不提供这个工具，我们的客户就必须手动转换它，尽管一个小的机动工具将会潜在的减轻他们很多的工作量。”

“这并不是一个高风险的情况，”他继续说道，“James，测试无非是风险管理。即使是一个简单的测试过程，基于我们假设没有什么工具，可能对于一个不太好的工具的风险管理已经足够。此外，你已经了解了足够多的有关新的汇编语法和旧的汇编语法之间的不同。如果任何的规格书问题提升了功能性或工具的风险，和开发者坐下来细细的询问他们。但是别指望开放信息垃圾。”

“仅仅是通过 ASMCVT 运行大量的源文件，”他总结道，“仔细检查结果，对于所有异常信息书写报告。如果现在在这里有一个非常重要的问题，你也许会发现它。”

二 、纪律或者惩罚？

我感到有些苦恼，因为我的经理将违背软件工程的一个既定原则：软件应该依照一个清晰的规格说明书来执行。虽然我没有找出他的过错，因为适应于当前局势，但我很担忧，对于在这种情况下的做法，可能会提供一种不遵从良好的软件开发过程的一个借口。当时我所没有意识到的是我对于“良好的过程”——人对于过程的关系——是如此幼稚。我所想到的过程仅仅定义为一连串有顺序的任务，而 Chris 把它看成一个团队达成目标的一种途径。

软件过程的想法应该被严格的定义，任何人都可以重复，并且遵循谨慎的想法，以及深受高级软件管理者和 SQA 热衷者喜爱的。他们希望是可控制的以及可预见的，并且他们认为可以通过刺激那种惩罚来得到它。这个合乎情理的看法的问题是，在实践中，严格和惩罚会被误认为是经常性教条和轻率的行为。这个不太好的惩罚过程的问题是用固定过程代替了它——不管在当时情况下他们的价值是什么，而执着追求于顺序的任务和目标。

如何判别这个固定并不是一个错误的事情，仅仅只是我们依照它自己的理由来执行它。与其我们为我们所理解的过程提供所需要的服务，倒不如了解我们需要解决的真正的问题是什么。在 ASMCVT 案例中，我变成固定在拿到一个进行数据转换的工具规格书的目标上，即使作为一个测试经理为了完成我的真正任务并不一定真正需要那个文档。我也把自己固定在了这样的目标中，就是那个开发者用他自己的时间遵循了官方过程，尽管可能还有其他的一些事情更为重要。

对他来说，ASMCVT 开发者就是任务的固定者。他的确应该写规格书。他遵照了我们所定义的规格，就是减少使用官方的规格模板。但是他所提供的完全忽略了我真正需要的是什么。我们没有进行有效的谈判，以解决我们之间的冲突。

这是一个情况下定义的过程，就是我们的操作没有为我们服务。

遗憾的是，许多我们所选择的以规范和组织我们的项目的过程都是建立在根本错误的想法之上，那就是团队是如何成功的创造出软件。我们希望控制软件项目的愿望诱使我们接受一些无用的操作和工程。软件开发过程，实际上是被人类的认识所控制的，并不是机械的，强制劳动，或是物理定律支配。软件是一种相互协作的创造物。软件开发是一种探索的和自我调整的对话，包括口吃的、卷舌的和支吾的，以及下意识的失言。这一对话输出了一个产品。当软件过程被强制成了一个制作模型，已经严重削弱了对话，或者已经将它驱赶到了地下。

如何寻求对话使得软件开发目标被重新定义？作为一个例子，让我们先看看需求分析和文档。

三 、要求重新审视

在普遍接受的软件开发实践中，需求分析应该是在设计以前就有的，并且设计应该是在编码和测试以前就有的。一些文件被认为应该是从需求分析中产生出来的。在 IEEE 830-1993 推荐的软件需求规格书包括一系列的质量属性列表：正确、完整、清楚、一致、重要性分级、可验证、可修改、并具有可追溯性。该规格被认为是应该指导后续的软件活动。

所谓的普遍接受的实践，尽管如此，并没有被普遍实践。正如我所遇到的他们一样，需求规格是一个模棱两可的问题状态和设计决定：他们表现的太过模糊或者太过精确，为了解释说明通常需要大量的背景知识，并且有些是在设计阶段就不用的或者忘记一半的。这些听起来熟悉吗？

对于一个拥有被大众接受的实践框架的不太好的需求规格，至少有一种简单的解决方法：做得更多。而不是压缩成了几周的访谈、会议、和熬夜写作，应该花费几个月的时间。一直坚持直到没有任何模棱两可和设计的风险。利用测试者、模拟器、完整模型，以及讨论组。探索全方位技术解决方法和评估收购与建造战略。让开发者隐姓埋名到客户现场，并且让他们像人类学家一样和潜在使用者生活在一起。

关于优秀的需求探索和定义的技巧，有大量的文献资料。然而，了解到能做什么并不是最主要的问题。主要的问题是，几乎没有人认为所有的这一切是必须的。大多数的人要么认为一个快速并且劣质的需求分析已经足够好了，要么根本没有意识到他们的过程是快速的和劣质的。另一个问题是，只有少数人能够熟练的执行所有需要的分析，并且一遍就能完成。

四 、以寻求对话为目标

实际上，需要制作出一个好的产品真正需要的是一系列想法，开发者能够很

好的理解产品、实际执行，以及（如果实施了）就会制作出一个足够理想的产品。我们应该利用任何一种满足需要的并且在恰当时机出现的过程。它甚至有可能满足需求，而没有编写任何需求文档或者任何的需求分析阶段。无论文档和阶段的价值是什么，他们都是达到目的一种手段，并且对于手段和目的都保持清晰的认识是对抗固定的最强大的防御。

引用需求专家 Brian Lawrence 的一句话，“许多人在现有的需求规格中关注需求。需求是设计选择的驱动程序——是我们为什么决定建造什么的原因。这些原因存在于我们思想中，因此，整套的需求都是在所有产品利益相关者的共享空间里面。”一个需求文档仅仅是那些共享空间中的一个信息模型，但是它帮助我们管理误解的、不切实际的和不好的需求的风险。

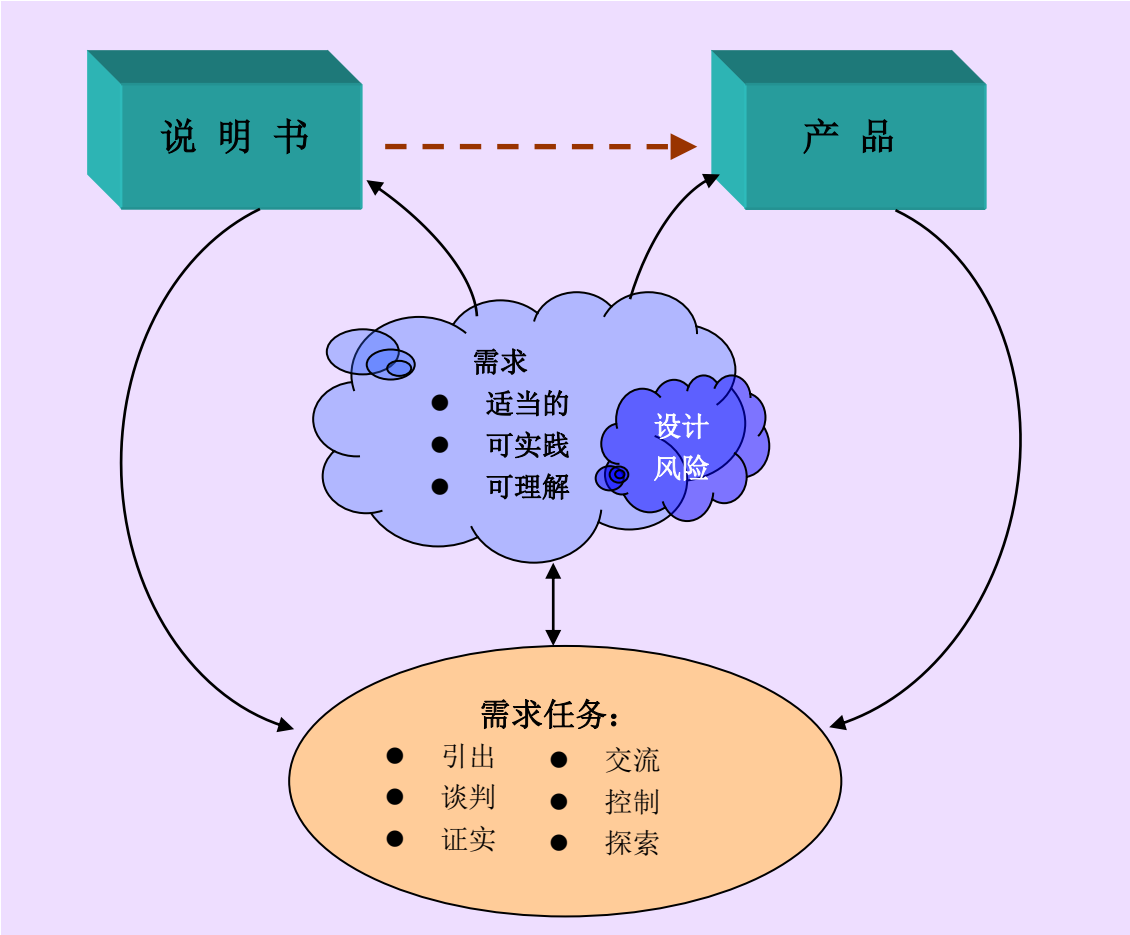


图 1 需求演化。云形状的“需求”描绘出被理解的需求。立方体形状的“说明书”描绘出被质疑的需求。云形状的“设计风险”描绘出由于不太好的需求可能发生的问题。有同时进行的两个对话：定义和体现。定义对话，在左边，是一个定义了我们想要得到什么的循环。体现对话，在右边，描绘了什么可以合理实现的循环。

图 1 是描述需求演化的一个对话，而不是作为一个整体阶段。任何的需求阶

段，从严密的纯净的开发过程到纯理论的堆砌，都可以称为这个模型，该模型描述了同时进行的两个对话：定义和体现。定义中的对话涉及对于要做什么的理解，而体现中的对话涉及对于如何才能做的合理的理解。全部的对话是以目标为导向的活动。按照优先顺序排序的目标是：

1、制作一个好的产品。

2、确保足够好的理解了需求，并有足够的实践了和满足期望，使得威胁到首要目标的设计风险足够小。

3、如果有必要，创建一个需求文档，支持前两个目标。

立即使用下面的模型进行实践：

● 需求规格是一个减轻需求管理的工具。即使是未完成的或者模棱两可的，也能够提供有用的线索，对于更深的和共享的理解需求。

● 由于该需求文档是不完整的，所以它不能成为后续工作唯一的基础。应该通过其他渠道获取信息，例如口头交流、专业知识和原型。

● 需求规格并不是需求讨论和谈判的替代品。

● 开发的需求阶段并不是对所有的已经发生的进行定义，相反的是发生了足够的事情，需要增加交流的和没有关注到要将简便的工具或重复的大量工作引入的风险。

● 继续判定需求就像技术上的限制一样都是偶然碰到的和需要探索的。需求的优先顺序可能会因为面对的困难和意识到他们对于一个产品的价值时而发生改变。

当我们固定在详细需求的过程中，我们试图顺服那些可能让我们履行过程的管理者，并且当他们不支持我们时我们会抱怨。当我们固定在一个快速的过程中，我们试图通过执行最小的需求来获得满意的过程。但是，还有第三种方法。我们可以拒绝固定在过程中，并且用解决问题来替代它。我们可以以寻求对话为目标重新定义需求过程，它们的目的是管理建立在错误产品上的风险。

在 ASMCVT 案例中，对话非常的简单。Chris 提出了合乎情理的需求，然后进行了对话，为了完成我们的测试任务，我们不需要对于这个特定的工具额外详细设计规格，历史证明他是对的，正如我所说的。我们轻易的解决了这个问题。但是他的论证已经远远超过了节省一个测试项目的成果。他使我开始了长达 10 年的冒险之旅，调查风险分析和管理的方法，使得它更加合乎情理。