摘要

随着个人计算机和互联网技术的不断进步,计算机应用系统越来越渗透到我们日常生活的每个方面，网上购物系统、网上银行系统和网上金融交易系统就是最典型的例子。但是随着计算机技术的更新换代；系统拥有的用户数量和数据量的激增；系统涉及的业务的增加等因素使得现有系统不得不面临重构。

大型金融系统重构项目中，最大的难点也是工作量最大的环节就是重构以后的系统的测试环节。一方面金融交易系统因为直接管理大量货币，所以要求有很高的正确性、稳定性、高效性等，故必须通过大量的测试来保证重构以后的系统的质量。另外一方面因为历史的原因，需要重构的系统往往存在需求文档缺失，具体需求难以获得等情况。如果从现有代码逆向分析出需求，无论在时间还是在资源上的代价都很大，还很容易造成需求的遗漏和错误。

本文结合作者参与的一个遗留的网上金融交易系统的重构项目中测试环节遇到的需求缺失，时间和预算不足，难以从现有代码获得需求等困难。提出，设计并实现一个以平行测试为主要测试方法的自动化测试系统。该系统通过使用大量测试数据来自动实现对重构以后系统的测试。有效地发现了系统重构中的 Bug，高效地完成了对重构以后系统的测试工作。提出了一种在需求缺失，或者获得需求代价过大的情况下用自动化测试来快速、低开销、有效地完成对重构项目的测试工作的新方法。

关键词： 自动化测试，金融软件系统，系统重构，平行测试

Abstract

Due to the growing number of the personal computers and the progress of Internet technology, the computer applications are increasingly penetrated into every part of our daily lives. Software systems such as online banking system and online financial transaction system are the most typical examples. The computer technology is upgrading everyday; and the number of users and data in the system are growing too; also, the increase requirement in the business makes the system to face the reconstruction.

The biggest difficulty in the reconstruction project of a large-scale financial used system is in the testing step. As the financial used system directly manages a large amounts of currency, so such systems demanded a high level of accuracy, downward compatibility, stability and efficiency and so on. How to ensure system quality? Of course, the answer is enough testing! However, because of some historical reasons the requirement documents are usually incomplete. Analysis of existing code to get the requirement needs plenty of time and human resource. But, this method is easily to make mistakes.

In this paper, a reconstruction project of a financial used system which is in use science 90s last centre was taken as the example. In this reconstruction project, there were many difficulties in test period such as the lack of requirement documents. And a test automation system was designed and developed using parallel testing to solve the difficulties of the testing in the reconstruction project. It gives a new solution of testing in the reconstruction project when there is neither not enough or clear requirement nor enough resource or time to analyze the requirement. The solution is using automated parallel test to cover the requirement, and it is a low cost and high active solution.

Keywords: Test Automation, Financial used system, Reconstruction Project, Parallel test

图目录

表目录

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权浙江大学可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月

致谢

在研究生两年半的学习生涯即将完成之际，我要首先由衷地感谢我的导师杨小虎教授。在我攻读硕士学位期间，杨老师渊博的知识、严谨的治学态度、一丝不苟的工作作风都给我留下了极深的印象，也给我树立了良好的榜样。不论是在平日的研究生学习和工作期间还是在论文的写作上，我自始至终都受到杨老师的悉心指导和鼓励，使我能始终保持一个饱满向上的态度来对待今后的工作和生活，让我受益终身。

其次，我要特别感谢和我一起从事实验室项目同事们：王特，曹炜，胡金俊，刘源根和戚力强，是你们陪伴我渡过了这意义非凡的两年半。在此期间，大家互相学习互相帮助，组成一个积极向上，团结而且和谐的团队。此外还要感谢项目组在美国 Boston 总部的同事们：Singh, Sangharsh; Bao, Xiaoyun; Pratap, Rana 和 Alancla, Rajesh，在两个团队的通力合作下，共同解决了很多项目中遇到的难题，出色完成了公司交给的任务。

另外，我还要特别感谢王帆和戚力强同学，感谢他们在我论文的撰写过程中，对我写作思路的梳理和灵感的启发，以及给予我技术上的帮助。

还要感谢计算机科学与技术学院 2007 级硕士班的其他同学在学习和生活上给予我的关心和帮助，并衷心地希望我们的班级里的每个成员都能施展抱负，出人头地。

最后衷心感谢我的父母多年来在学业上、生活上和精神上给予我的极大的关心、照顾、理解和支持，让我永远保持进步；以及女友蒋梅玲的监督和帮助，让我时刻保持前进，不曾松懈。

王雷

2010年1月 于求是园

第1章 绪论

1.1 引言

随着个人计算机的广泛普及和互联网技术的飞速发展，各种各样的计算机应用系统逐渐渗透到我们的生活中。超市购物系统、交通售票系统、财务管理系统等应用软件系统给生产力带来了质的飞跃；而之后出现的网上购物系统、网上银行系统、各种网上金融交易系统等则进一步给我们的日常生活带来了翻天覆地的变化。与此同时，随着时间的推移，计算机技术不断地更新换代；系统所拥有的用户数量不断增长；以及系统所涵盖的业务不断地扩展。直至软件系统原有的架构对新的需求渐渐地失去支持能力。所以到了这一阶段，系统便不得不面临重构以适应进一步的发展。重构是软件工程中一个极为重要的课题。重构能使开发人员修订一个现有软件系统的结构或者设计，从而使它变得更加容易增加（或者修改）功能[1]。

当对金融交易系统这种规模庞大；系统正确性，安全性要求高；逻辑结构又异常复杂的系统进行重构时，如何确保重构以后的系统的正确性和可靠性便成为一大难题。而软件测试是软件质量保证的关键所在，在软件生命周期中占有着非常重要的位置。不管开发人员采用怎样的开发技术、开发流程来保证软件质量，都不能代替软件测试的作用。因此，在金融系统的重构中，测试环节扮演着举足轻重的角色。

1.2 软件测试的分类

软件测试从测试的执行方式上一般可以分为两种：手工测试和自动化测试。

传统的手工测试是一个劳动密集型工作，但是其优点是可以充分利用测试人员的个人的能力。测试员可以根据被测试的代码设计一些专门的测试用例，也可以针对边界数据增加一些测试用例，有经验的测试人员还能发现很多自动化测试软件所不能预测的问题。但是手工测试的缺点十分明显，首先手工测试非常容易

出错；此外手工测试的效果很大程度上依赖于测试人员的经验；最后手工测试根本无法支持海量的测试案例。

软件自动化测试的优点是可以高效地完成一些重复性的测试；同时还能降低人为因素对测试过程的干扰；充分的测试还能有效降低随机性和盲目性；最后还能降低多个测试人员合作时的重复测试工作，减少遗漏等。软件自动化测试就是通过执行某种程序设计语言编制的自动测试程序，控制被测软件的执行，模拟手动测试的步骤，完成全自动或者半自动测试[2]。其目的在于提高软件系统测试效率，缩短测试周期，增强对被测试软件的测试覆盖率等，从而达到保证软件得到充分测试，保证软件质量的目的。

对比传统的手工测试方法，自动化测试有以下的优点[3]：

1. 最小化回归测试的成本。
2. 提高测试效率，并得到更好的测试效果。
3. 能完成一些手工测试不可能或者难以完成的测试任务。
4. 高度的一致性和可重复性的测试任务。
5. 最大化地提高人力资源的利用率。

因此，软件测试自动化工具和技术得到越来越广泛的运用。

1.3 自动化测试的历史和现状

从上世纪 90 年代起，针对软件自动化测试的研究就已经开始，相应的工具层出不穷。总体来说，按照产生的时间段及使用的技术特点，自动化测试可以分为以下几个阶段[4]：

第一代的自动化测试技术始于上世纪的90年代初，通过硬件的方式录制键盘的输入并播放，但是不具有检查点（checkpoint）的功能，而且测试脚本也很难以维护。

第二代的自动化测试技术兴起于上世纪的 90 年代末，这时已经由硬件录制/播放转变成使用软件录制/播放（capture/playback）的方式生成测试脚本（script），并且增加了检查点的功能，用来对软件做验证。测试的范围较上一

阶段增大了许多。但是存在最大的问题是灵活度不够高。

第三代的自动化测试技术诞生于 2001 年，我们称之为测试框架（test framework），测试框架的主要思想即是把测试脚本给抽象化，从而提高自动化测试的适用度和灵活性。具有代表性的工具有：Rational robot 和 Mercury loadrunner 等。

近年来随着软件开源模式的兴起，自动化测试技术进入第四代。其最大的特点是使用者可以根据自己的实际情况随意定制、选取、扩展所需的自动化测试模块，并可以和其它开源框架结合形成一个完整的自动化测试平台。

1.4 论文的课题背景和研究内容

本文针对大型网上金融交易系统的重构项目中遇到的诸如：此金融系统商业逻辑复杂，逻辑分支繁多；系统测试需要用大量的数据来覆盖边界条件；系统重构项目代码开发量大，工作周期紧张；还有系统的需求文档缺失严重，开发人员又缺乏金融知识背景等种种困难，提出了用自动化的平行测试来尽可能充分地覆盖系统边界条件；从而在尽可能少的人力资源投入下，尽可能高效地发现 Bug。另外用平行测试的原理实现了回归测试的自动化执行，进一步提高了开发人员在 Bug Fix 阶段的工作效率。 该自动化测试系统的实现方案的提出为一些业务逻辑复杂，需要测试工作量巨大、测试难度高、需求又不完整而且还极难使用通用自动化测试工具进行测试的系统重构项目提供一种低投入、高产出的自动化测试解决方案。

第二章将会对测试目标系统的相关背景，系统架构以及重构项目中测试环节面临的主要的挑战进行简要的介绍。

第三章将会对重构项目所需的测试以及面临的主要难点做分析，然后提出有效的解决方案。

第四章将会就该自动化测试系统主要功能点的实现方案和框架结构设计做详细阐述。

第五章将会就该自动化测试系统具体实现环节所采用的技术和实现方案做

详细阐述。

第六章将会就对该测试系统在项目的实际运用中产生的效果做客观的分析和估算。

第七章将重点介绍基于 BP 人工神经网络的算法实现的 Bug 定位模块的原理和具体实现方案。

第八章是本文的总结部分，指出自动化测试系统存在的缺陷，有待继续研究的方向等。

1.5 本章小结

本章简单介绍了自动化测试的发展历史和现状，并结合作者在一个大型网上金融交易系统重构项目中开发的自动化测试系统的实践。引出了用自动化的平行测试来解决系统重构项目中的测试问题。

第2章 系统重构项目概述

2.1 系统重构的背景

金融交易所包涵的内容极其丰富，包括：外汇交易、现金转账交易、黄金交易、股票，债券等票据的交易、期货交易、期权交易还有利率衍生品交易等各种类型的交易。在本文中作待测试系统的FTM（Financial Transaction Manager）系统主要的功能为：对客户或者交易经纪人发出的外汇交易、现金转账交易、股票，债券等交易请求做汇率，金额等的计算和转换；检查交易请求的完整性；评估交易请求的正确性和可行性；计算交易所需要支付的费用等。FTM 在整个系统在交易流程中的位置和作用如图 2.1 所示：

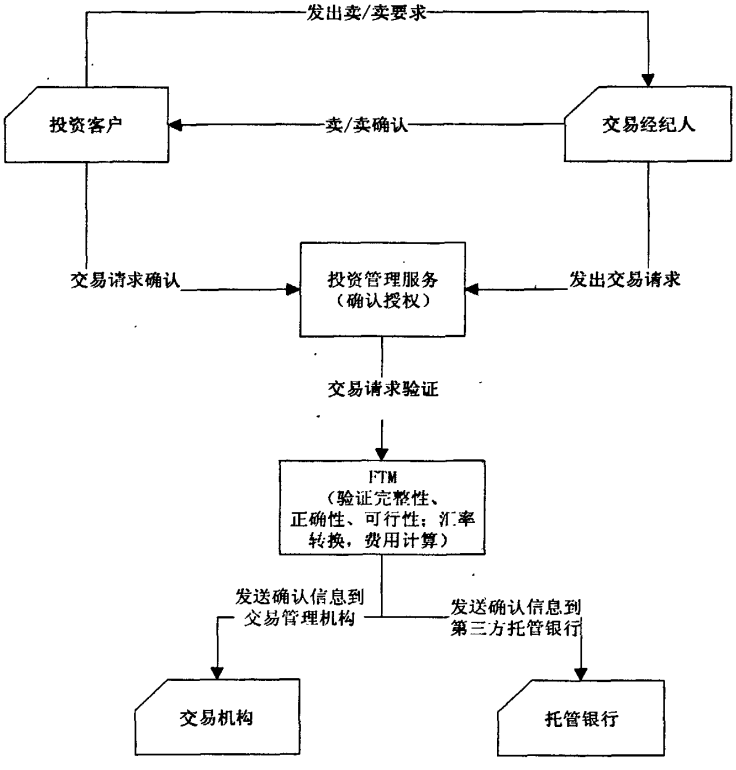

图 2.1 交易流程图

从系统的业务量上看，整个系统包含四大类业务；交易范围涵盖了80多个国家、地区，涉及50多种主要货币交易；日处理数据量平均在20万~25万条。因此，不难看出FTM是一个涉及的业务众多，包含的商业逻辑复杂，处理的数据量巨大的网上金融交易系统。

2.2 系统重构后的架构变化

现有的FTM系统从架构上分，自上而下一共有4层7个模块：

1. 处理输入信息的Trade Capture模块；
2. 处理具体商业逻辑的四个模块：CashProcess模块，SecurityProcess模块，IndirectFXProcess模块和FeeProcess模块；
3. 做业务金额结算的SIM模块；
4. 向下游模块发送信息和接收反馈信息的Outbound模块。

经过重构后的新FTM系统从架构上分，自上而下一共有以下5个模块：

1. 处理输入信息的Inbound模块；
2. 验证输入信息完整性的Trade Pretreatment模块；
3. 处理具体商业逻辑的Trade Processing模块；
4. 做交易结算的Settlement模块；
5. 往下游系统发送信息的Outbound模块。

重构前后的FTM系统框架对比如图2.2所示：

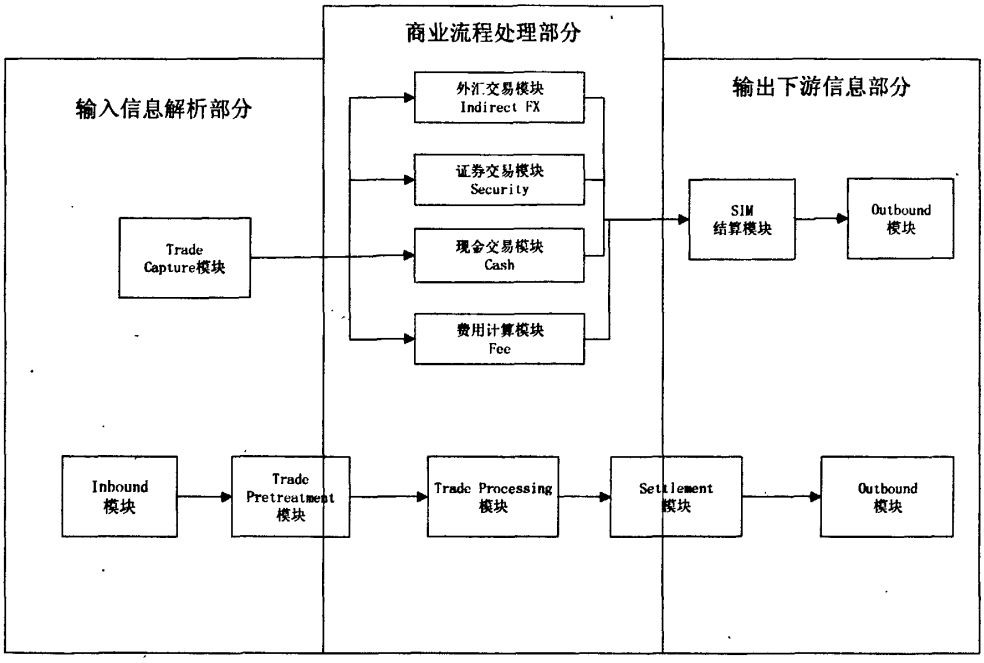

图 2.2 重构前后系统框架比较图

2.3 重构项目中测试工作面临的主要挑战

从上述不难看出，FTM 系统重构项目有以下 3 大特点：

1. FTM 是一个包含的商业逻辑复杂，交易流程分支众多的专业金融交易系统，商业需求复杂；
2. 重构前后的 FTM 系统在框架结构上有较大的改变；
3. FTM 系统处理的业务广（涉及 80 多个国家地区， 50 多种货币），数据量大。

以上的特点给重构项目的测试工作带来了以下的困难：

1. 商业流程和逻辑分支复杂，代码量庞大，使测试整个系统所需要的测试工作量非常巨大，而且测试用例的设计很难实现；
2. 系统架构的改变使功能模块级别的功能测试无法充分进行，所以针对整个系统做的集成测试工作量会大大增加；
3. FTM 系统处理的业务范围广泛，因此传统的手工测试方法很难达到一个满意的覆盖率；

4. 现有的 FTM 系统因为历史原因没有完整的需求文档，所以衡量重构以后的系统正确与否的标准只有以重构以后的系统运行结果是否于现有 FTM 系统运行结果一致为标准。

2.4 本章小结

本章简略介绍了该测试系统需要测试的 FTM 系统的特点和相关背景，以及重构项目的情况和特点的概述。再指出该重构项目的测试环节面临的主要困难和挑战。

第3章 测试难点和解决方案

3.1 测试方法选取

从测试的方法上来分，测试可以被分为白盒测试和黑盒测试两大类。

3.1.1 白盒测试

其中白盒测试也称结构测试或逻辑驱动测试，它是按照程序内部的结构来测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。白盒测试要求全面了解程序内部逻辑结构、对所有逻辑路径进行测试，白盒测试是穷举路径测试。因此，在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据[5]。

首先，对于重构项目来说，系统的内部逻辑结构和商业流程的具体实现过程并不是开发人员所关心的重点，需要关心的重点是系统的功能点的正确与否，即重构以后的系统对于同样的输入数据能否和原来的系统有相同的输出。因此没有选择白盒测试的需要。其次对于FTM这样一个庞大复杂的系统来说，整个系统包含的逻辑分支是数以千计的。因此，对整个系统进行集成测试的话，所需要的测试用例几乎是一个天文数字，因此对这样的系统使用白盒测试并不适合。所以黑盒测试是比较合适的测试方法。

3.1.2 黑盒测试

黑盒测试也称功能测试或者数据驱动测试，它是通过测试来检测功点是否都能正常使用。在测试中，把程序看作一个无法打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，通过程序接口进行测试。它只检查程序功能是否按照需求正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，即不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试[5]。

对黑盒测试来说必须有明确而且尽可能完整的需求才能较好地达到测试的目的和获得满意的测试覆率。而需求来源分为显示需求和隐式需求两大类。其中显式需求指的是：

(1)原始需求介绍说明书；

(2)产品规格书；

(3)软件需求文档；

(4)继承性文档；

(5)经验库；

(6)通用协议规范标准。

隐式需求指的是:用户的主观感受、市场主流观点、专业人士评价分析等等[5]。

3.2 获取测试需求的难点

3.2.1 需求文档缺失

一般情况下，软件测试的需求最主要来源于显式需求中的软件需求文档、继承性文档、通用协议规范标准和原始需求介绍说明书等。但是，一个实用的商业系统在运行过程中会不断地向系统中增加新的业务需求、删除被淘汰的业务、合并类似的业务和修改某些不适当的业务流程来满足不断变化的商业流程。举本文提及的FTM系统为例子。该系统从投入实际应用至今已经运行了十几年，期间经历的大、小改动不计其数，对于这样的系统，依靠软件需求文档来定义测试需求代价太大，而且很难保证需求的正确性。首先，收集、整理这么多的文档需要花费大量的时间和人力资源；其次，上述需求文档有的是系统级别的需求变更，有的是模块级别的需求变更，有的是功能点的需求变更；因此要把这些层次各异的需求文档统一转化为系统级别的测试案例工作量太过巨大，而且非常容易造成需求的重复或者遗漏；最后，实际上很少有系统能保持有如此完整的原始需求文档和变更需求文档。以本文涉及的FTM系统为例，只有2005年以后的需求文档是完整的，之前的文档存在部分缺失或完全缺失的情况。

3.2.2 逆向分析需求代价过大

在需求文档缺失的情况下，开发人员仍然可以通过分析现有代码转化为业务流程，然后再进一步还原为具体需求的逆向过程来分析、抽象、提取出具体需求。但是，这样的逆向分析过程最大的困难是费时、费力、难度大而且还容易出错。在FTM系统重构项目中，最大的困难表现为以下三点：

1. 要从程序代码中分析出商业流程和具体需求，要求分析者必须同时具备程序开发知识和金融交易相关的商业知识，而同时具备以上2种知识的人员非常稀缺；
2. 为了保证从代码中分析得出的需求的正确性，必须有相关的用户对产生的需求文档做出仔细检查和审核，发现问题再和分析人员沟通。这样的流程进一步加剧了时间和人力上的开销；
3. 系统运行期间多次打补丁，贴膏药式的代码修改增大了分析代码的难度，同时也增大了代码分析过程中出错的几率。

综上所述，从现有代码逆向分析得到商业需求的过程需要花费大量的时间，需要耗费大量的人力，而且还容易出错。所以这种方法并不适合时间和预算都不是很宽裕的项目中使用。

3.2.3 现有系统即是需求

在上文分析的两种方法都无法或者是很难获得测试需求时，我们还可以采用隐式需求作为黑盒测试的需求。简而言之就是：抛开任何需求和实现，对用户而言，如果对于同样的输入信息，重构以后的系统和重构前的系统有相同的输出值，那么这个功能点在重构中的实现就是正确的。以本文中的FTM系统重构为例子，则对重构后的系统做测试时，可以用现有的FTM系统的运行结果作为对重构以后的系统的集成测试的需求。

这种需求获得方法有以下两大优点：

1. 不再需要花费大量时间在需求获取和需求分析上；
2. 可靠性高，因为系统重构的前提要求是重构以后的系统要有和原有的系统有

相同的运行结果。

但是也有以2方面的缺点：

1. 由于没有显式的需求文档，对测试人员来说难以生成有效的测试用例，很容易造成80%的测试案例都在反复测试20%代码的情况；
2. 很难保证测试的覆盖率和边界情况的覆盖率。

因此，我们引入平行测试的方法来弥补上述2个缺点。

3.3 平行测试

平行测试（Parallel Testing）即通过输入相同的数据到新、旧两个系统中同时运行。然后通过比较两个系统的运行结果或者执行状态的方法来实现对新系统的测试的一种测试方法。在本文涉及的重构项目中，平行测试的工作原理如图3.1所示：

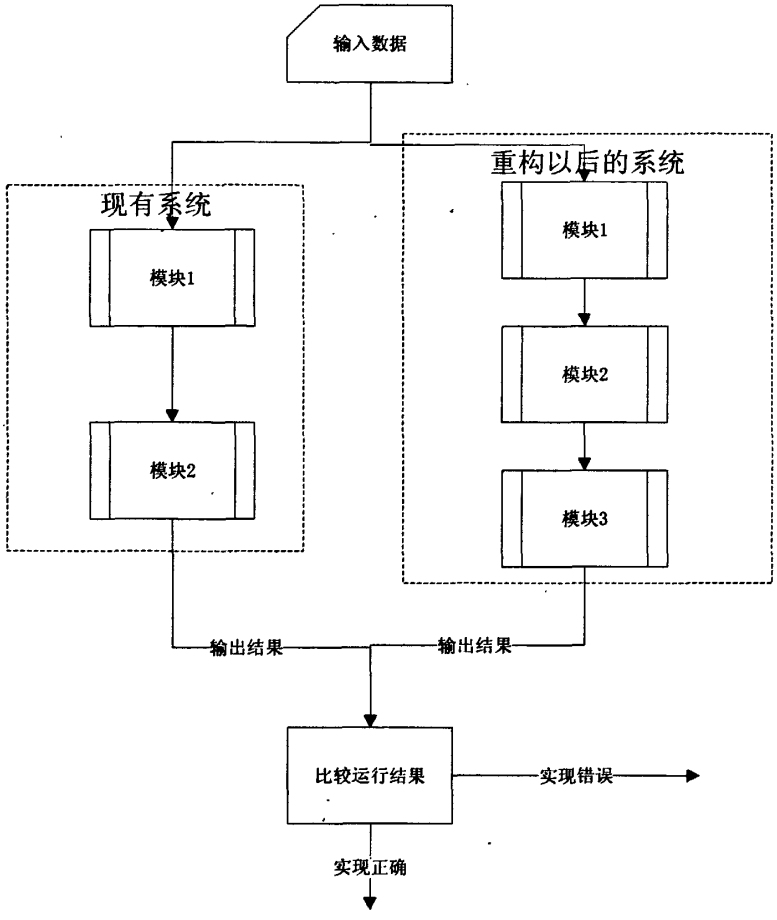

图 3.1 平行测试原理图

在本文讨论的系统重构项目中，平行测试在开发的各个阶段能起到不同的作用，按不同阶段分有以下四种：

1. 在集成测试之前，开发人员能用平行测试有针对性地对一些功能点设计测试用例实现模块级的功能点测试；
2. 在集成测试中，能用平行测试来判断功能点的实现是否正确；
3. 在开发后期的SIT测试(System Integration Testing)阶段和UAT测试(User Acceptance Testing)阶段用平行测试提高测试的覆盖率；
4. 在重构系统投入实际运行前，能通过海量数据的平行测试来尽可能地覆盖前面三个阶段无法覆盖到的边界条件，从而尽可能保证系统的正确性。

3.4 回归测试

除了上文阐述的在基准系统和待测试系统之间运行的平行测试外，重构项目中还需要一种特殊的平行测试——回归测试来加快系统开发的进度。所谓的回归测试（Regression Testing），即就是在软件生命周期中，只要软件发生了改变，就可能给该软件产生问题；所以，每当软件发生变化时，我们就必须重新测试现有的功能，以便确定修改是否达到了预期的目的，还要检查修改是否破坏原有的正常功能[6]。因此，回归测试实质上就是待测试系统在代码改动前和代码改动后分别针对同一基准系统之间进行的2次平行测试。

刚完成重构的系统中必然包含着大量的错误（Bug）和瑕疵（Defect），因此伴随而来的是大量的Bug Fix工作。而每次的Bug Fix都会或多或少带来代码的改变，这种情形下我们便需要大量的回归测试来保证Bug Fix的正确性。

在本文提及的系统重构项目中，我们需要以下三种不同规模的回归测试，它们的频率分别如表3.1所示：

表 3.1 回归测试执行表

规模	频率	作用
20~50 相关测试用例	每次Bug Fix以后进行	确定某一Bug Fix的正确性，以及完整性。
100~500通用的测试用例	每个工作日结束以后进行	确认该工作日Bug Fix的结果，以及确认是否引进新的Bug。
运行现有的所有的回归测试用例	每周（即每个Code Version）提交之后进行	更新Bug列表，制定下个周期的工作目标。

从上表可以发现，回归测试在整个重构过程中占的比重非常大，但是在整个开发过程中又是必不可缺的环节。

综上所述，在整个系统重构项目中占的工作量比重最大，同时也是主要测试方式的平行测试和回归测试有以下共同点：

1. 测试流程机械单一；
2. 测试重复性高；
3. 测试所需数据量大。

因此，这两种测试适合用自动化测试来完成。因此非常有必要设计一种专门针对平行测试和回归测试的自动化测试工具把开发人员从繁重而又重复的测试中解放出来，从而大大节省测试时间，进而大幅度缩短开发周期，加快开发速度，提高整个项目的资源利用率

3.5 本章小结

本章分析了系统重构项目中测试环节的需求和特点，提出了最适合系统重构项目使用的两种测试方法——平行测试和回归测试。然后分别分析了这两种测试的特点和作用，最后指出开发一种实现这两种测试方法的自动化测试工具的必要性。

第4章 自动化测试系统的框架和架构

4.1 自动化测试系统的工作流程

通过前文分析可知，该自动化测试系统主要解决的是系统重构项目中的平行测试和回归测试的自动化问题。

4.1.1 平行测试工作流程

从图 3.1 的平行测试的工作流程图中，可以发现平行测试的步骤分为以下三步：

1. 同步向 2 个系统（作为测试标准的 A 系统和作为测试目标的 B 系统）中投入相同的测试数据；
2. 分别从 A 和 B 两个系统中找出本次测试的运行记录，并根据测试要求提取相应的表示运行结果的字段；
3. 对运行结果做比较和分析，分析该测试案例对应的功能点的实现是正确还是错误。

上述 3 个步骤是开发人员执行平行测试时的流程。相应的，如果用自动化测试系统去模拟开发人员在上述三个环节中的工作，需要实现的功能分别是：

1. 把一组测试数据按相同的顺序，分别投入基准系统 A 和待测试系统 B 中。同时为了便于识别出该次测试的记录，在输入数据中设置能标识测试批次和测试顺序的字段；
2. 按照上一步输入的标识字段分别从 A 系统和 B 系统中选出前一步投入的测试用例的测试结果，并按照测试顺序值将测试结果一一配对；
3. 对每一对测试数据的结果字段进行分析，判断测试结果是否匹配，若不匹配则把不同之处高亮。
4. 汇总结果，输出报表。

具体工作流程图如图 4.1 所示：

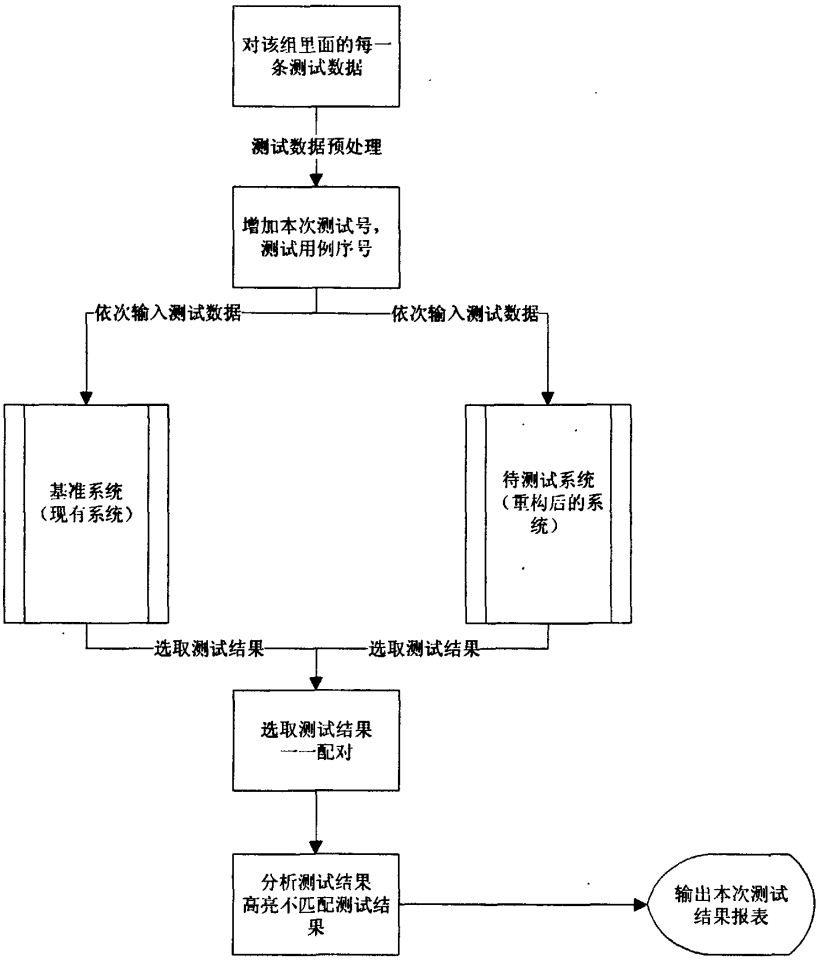

图 4.1 平行测试工作流程图

4.1.2 回归测试工作流程

一次回归测试的完整流程包括以下 5 步 [6] ：

1. 识别软件中被修改的部分；
2. 从原基线测试用例库 T 中，排除所有不再适用的测试用例，确定那些对新的软件版本依然有效的测试用例，来建立一个新的基线测试用例库 T0；
3. 依据一定的策略从 T0 中选择测试用例测试被修改的软件。
4. 如果必要，生成新的测试用例集 T1，用于测试 T0 无法充分测试的那部分软件代码。

5. 用测试用例集T1测试修改后的软件。

其中第2步和第3步测试用来验证本次修改修改是否破坏了现有的正常功能；第4步和第5步测试用来验证本次修改是否修正了发现的错误。

相应的，如果用自动化测试系统去模拟开发人员在上述三个环节中的工作，需要实现的功能点别是：

1. 在改动代码前设立一个基线，基线里包含了测试用例和测试结果；
2. 把改动代码之前做的测试按测试结果分2类，和基线相比测试结果为“匹配”的测试用例归为第一组；和基线相比测试结果为“不匹配”的测试用例归为第二组。
3. 修改代码并重新执行基线里的测试用例，再把获得的测试结果和基线的结果做比较。
4. 如果原本第一组的测试用例在这新的测试中结果为“不匹配”，则这些测试用例是由于本次Bug Fix造成的错误；如果原来在第二组的测试用例在这新的测试中结果为“匹配”则表示Bug Fix成功修正了错误。

具体工作流程图如图4.2所示：

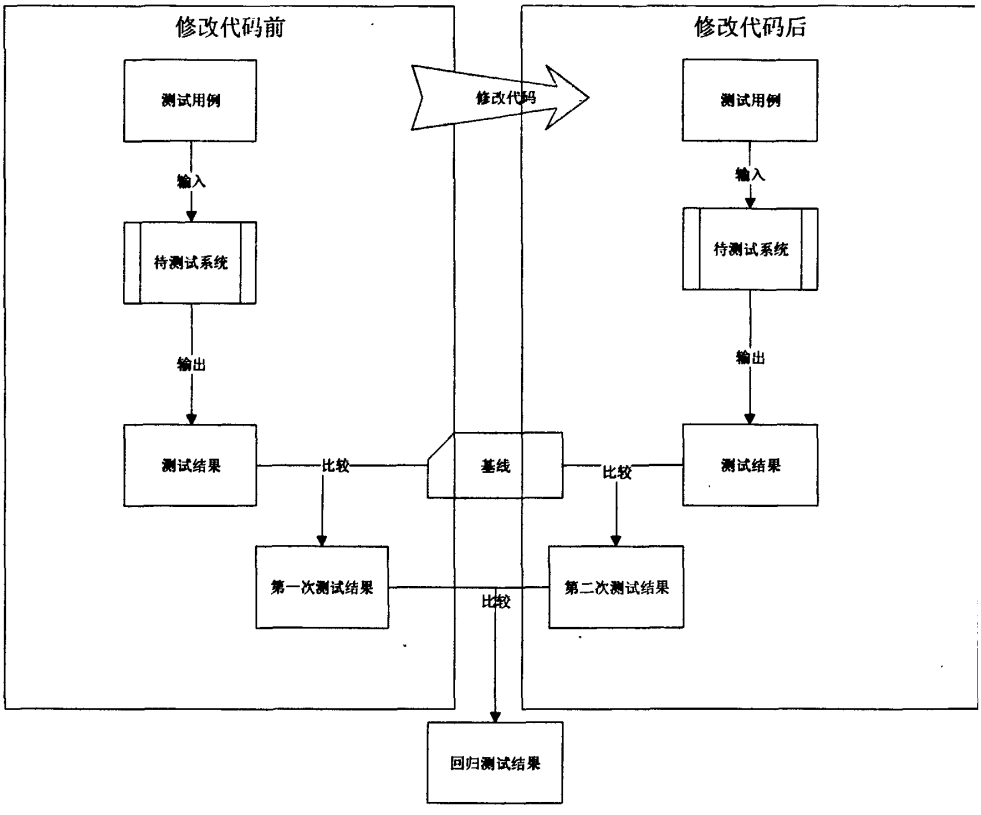

图 4.2 回归测试工作流程图

4.2 自动化测试系统的模块划分

按照功能划分，自动化测试系统可以分为以下 4 个模块：

1. 测试数据管理模块：用于新建、修改、删除测试数据；
2. 测试方案管理模块：用于选择测试数据、测试类型、测试环境以组成新的测试方案，也可以选择已有的测试方案再次执行，或者修改已有的测试方案；
3. 测试结果获取模块：根据测试方案中定义的参数从相应的环境中获取测试结果字段/消息，再按照测试案例将测试结果一一配对。
4. 测试结果比较模块：把获得的测试结果进行比较，若需要比较的只是简单的字段则用 String 比较即可；若需要比较的是复杂格式的输出消息，则先将待比较字段分割，再做 String 比较。比较后再把消息组合，还原。

自动化测试系统的模块结构图如图 4.3 所示：

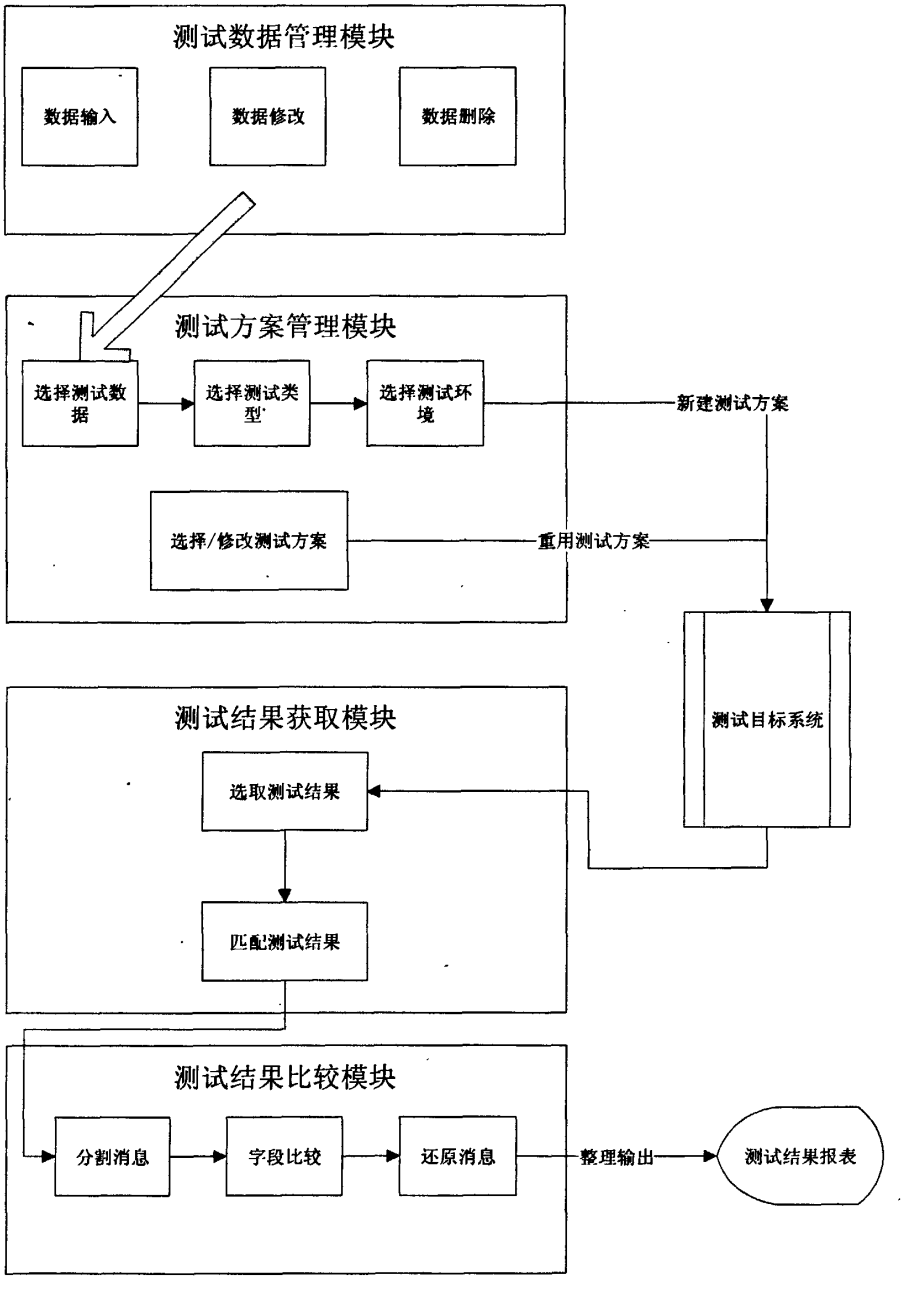

图 4.3 自动化测试系统模块图

4.3 本章小结

本章首先对手动执行平行测试和回归测试的流程做了分析，然后按照这上述测试流程设计了自动化测试系统执行平行测试和回归测试的步骤。最后再从功能上对自动化测试系统进行了模块划分，并且确定了每个模块的功能。

第5章 自动化测试系统的实现

5.1 测试数据管理模块的实现

测试数据管理模块主要用于对测试数据的维护、管理和查询，该模块在整个系统中位于底层位置。该模块提供的主要接口为测试数据的查找、新建、修改、删除等操作。同时为了方便管理，此条测试数据在基准环境下运行的结果也就是“基线”的值，也由该模块统一管理。

每一条测试数据构成的记录应该包含以下内容：

1. 测试数据的ID：这一项为数据库表的主键，是用来标识测试数据的列；
2. 测试数据的类型：因为系统支持XML格式，TEXT格式和SWIFT（Society for Worldwide Interbank Financial Telecommu-nication 即全世界银行间金融电信学会定义的消息格式）格式的消息，所以该列用于标识该测试数据的格式；
3. 测试数据的业务：本文介绍的系统支持外汇交易、现金转账交易、股票，债券等多种业务，这一列用于标识此条测试数据针对的业务种类；
4. 描述：用来简短描述本条测试数据的特点；
5. 消息体：存储测试数据的消息内容；

除了测试数据表外，为了方便回归测试的进行系统还负责维护一张Test_Data_Base_Line表，每一条测试数据基线表的记录应该包含以下内容：

1. 测试数据ID：用该列和测试数据表中的记录相关联，外键；
2. 基线基准环境：建立基线的基准环境，该列和测试数据ID一起成为本表的主键；
3. 基线结果：本条测试数据在基准系统中运行得到的结果，基线结果包含了在所有测试模板中都可能被用来做比较的字段的结果。

5.1.1 存储层实现

Test_Data数据库表的表结构定义具体如表5.1所示：

表 5.1 Test_Data 表表定义

列名	类型	长度	备注
Test_Data_ID	VARCHAR2	30	测试数据的 ID,主键;
MSG_Type	VARCHAR2	10	测试数据的类型;
TRAN_Type	VARCHAR2	10	测试数据的业务;
Test_Data_DESC	VARCHAR2	50	测试数据简单描述;
Test_Data_MSG	VARCHAR2	2000	测试数据的消息内容;
Added_User	VARCHAR2	15	添加该测试数据的用户;
Create_Date	TIMESTAMP		本条测试数据创建时间;
Update_Date	TIMESTAMP		本条测试数据更新时间。

Test_Data_Base_Line 数据库表的表结构定义具体如表 5.2 所示:

表 5.2 Test_Data_Base_Line 表表定义

列名	类型	长度	备注
Test_Data_ID	VARCHAR2	30	测试数据的 ID,外键;
Base_Line_ENV	VARCHAR2	10	基线建立的基准环境;
Test_Data_RSLT	VARCHAR2	2000	测试数据在基准系统中运行得到的结果;
Create_Date	TIMESTAMP		该基线的创建时间;
Update_Date	TIMESTAMP		该基线的更新时间。

5.1.2 逻辑层实现

在代码实现上,测试数据由对象 TestDateBO 来实现; TestDateBO 的对象用于实例化一条测试数据,因此其包含的属性为 Test_Data 表和 Test_Data_Base_Line 表的列的并集。

5.1.3 UI 实现

测试数据管理模块的界面比较简单，无非是实现测试数据的查找和测试数据的管理这 2 大类的功能：

1. 测试数据的查找：该界面的作用是，用户通过输入一定的查询条件来查找需要的测试数据。测试数据查询 UI 的实现如图 5.1 所示：

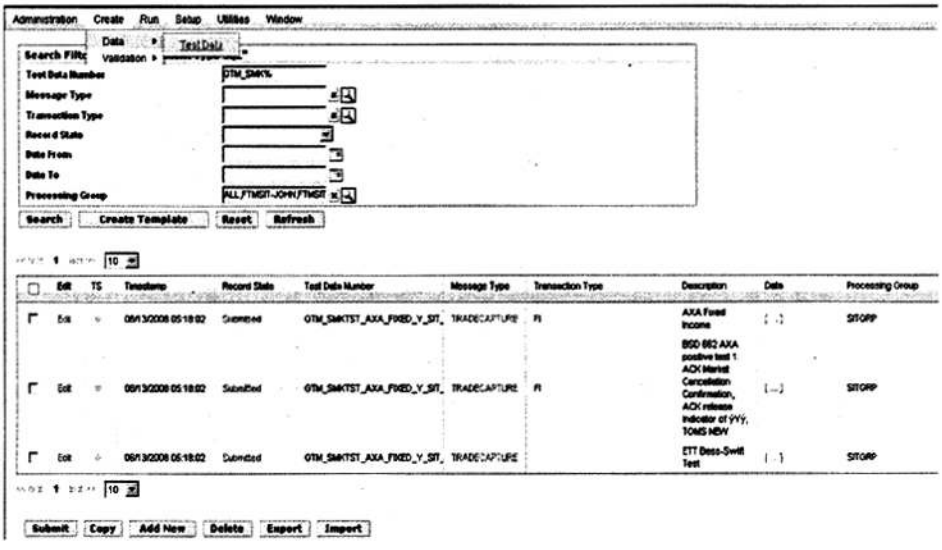

图 5.1 测试数据查询界面

2. 测试数据的管理：该界面允许用户新建、修改、删除测试数据。

测试数据管理 UI 的实现如图 5.2 所示：

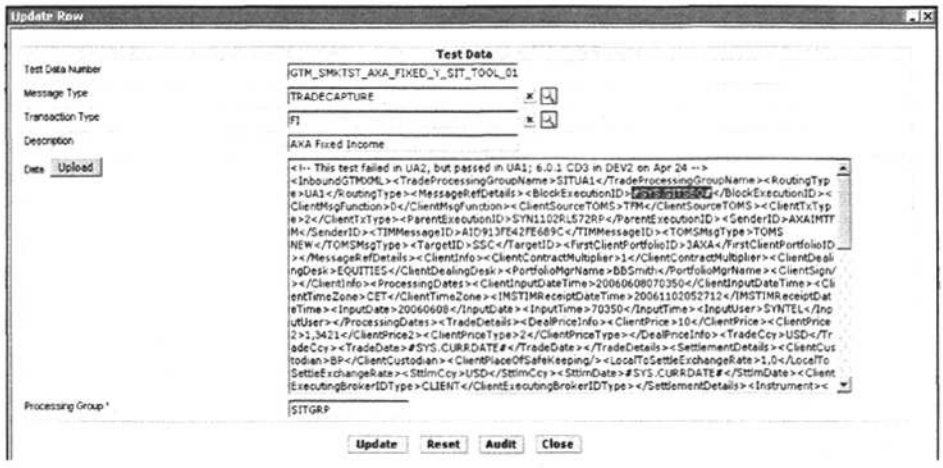

图 5.2 测试数据管理界面

5.2 测试方案管理模块的实现

测试方案是系统中最基本的功能单位，它代表的意思是一份测试计划。因此，一个测试方案必须包含以下信息：

1. 测试方案 ID：本次测试方案的名称，其值唯一，是该数据库表的主键；
2. 测试数据：本次测试方案中被使用到的测试数据；
3. 测试类型：是并行测试（Parallel Test）还是（Regression Test）；
4. 测试环境：选择需要测试的环境（通常开发环境有若干个，分别对应不同代码版本）；
5. 基准环境：作为测试结果比较的基准环境；
6. 比较模板：比较模板中定义了运行结果里面哪些字段/列将用来做比较；
7. 状态：测试方案的运行状态——Added，Fail 和 Success。

5.2.1 存储层实现

在具体实现上，为了节省存储空间，用“查询测试数据用的条件”代替测试数据存在 Test_Batch 数据库表中。Test_Batch 数据库表的表结构定义具体如表 5.3 所示：

表 5.3 Test_Batch 表表定义

列名	类型	长度	备注
Test_Batch_ID	VARCHAR2	30	测试方案的名称，主键；
Test_Data_CRE	VARCHAR2	50	选取测试数据的条件；
Test_Type	CHAR	1	测试类型，P：并行测试；R：回归测试；
Target_ENV	VARCHAR2	10	测试的目标环境；
Criter_ENV	VARCHAR2	10	测试的基准环境；
Template_NUM	VARCHAR2	10	测试结果比较使用的模板号；
Test_Status	VARCHAR2	5	测试方案运行状态；
Update_Date	TIMESTAMP		本测试方案最后更新时间；
Update_User	VARCHAR2	10	最后更新本测试方案的用户；
Test_Data_NUM	INTEGER	10	测试数据数量；
User_Group	VARCHAR2	10	本测试方案的用户组；
Base_Line_Tag	CHAR	1	标识本测试方案是否已经建立基线（Base Line）；
Base_Line_Value	VARCHAR2	2000	用来保存在本次测试方案的基线中。

5.2.2 逻辑层实现

在代码实现上，测试方案由对象 Test_BatchBO 来实现；Test_BatchBO 的一个对象表示一次测试的测试方案。系统解析一个 Test_BatchBO 对象的各个属性值然后执行此次测试。

在测试方案的执行环节上，回归测试比平行测试多一个建立“基线”的过程，如果新建的是回归测试，则必须先执行一次平行测试建立起“基线”才能进一步执行回归测试。而“基线”的建立也分两个步骤，第一步是从基准环境中取出运行的结果存入基线表(Test_Data_Base_Line)中；第二步是更新测试方案里面的

基线测试结果（Base_Line_Value）列，把测试案例按测试结果结果为“匹配”和“不匹配”分为两组。

5.2.3 UI 实现

与测试数据管理模块类似，测试方案管理模块的界面也主要分为测试案例查询界面和测试方案管理界面2类：

1. 测试方案的查询界面，在该界面中用户输入查询条件，系统将符合查询条件的测试方案罗列在页面下半部分的列表中。具体实现如图5.3所示：

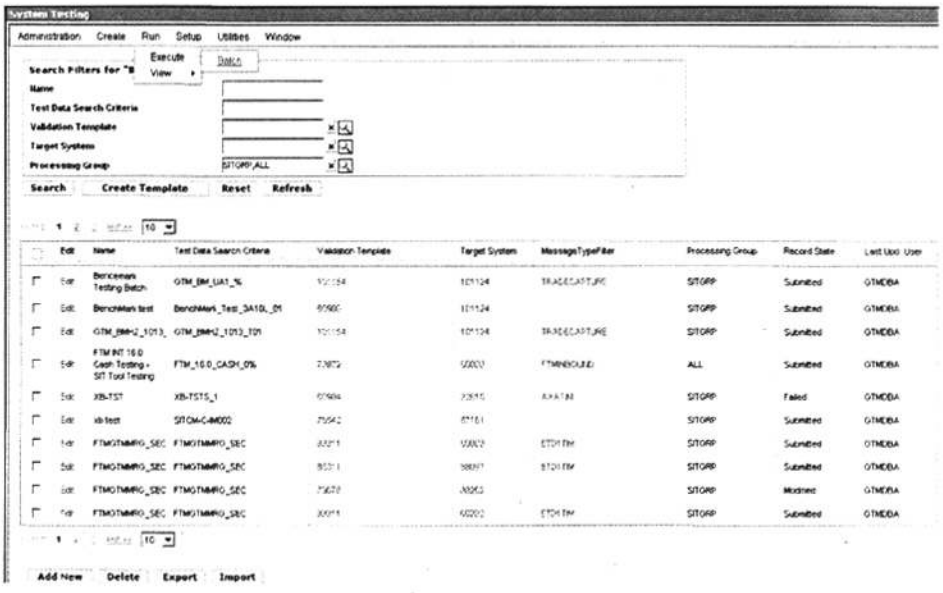

图 5.3 测试方案查询界面

2. 测试方案的管理界面，在该界面中，用户能新建、修改或者删除选中的测试方案。

测试方案管理UI的实现如图5.4所示：

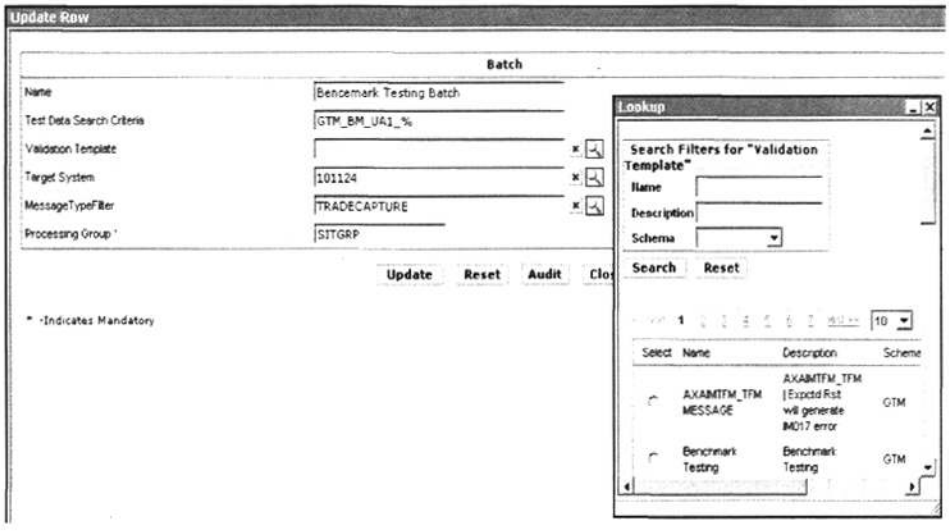

图 5.4 测试方案管理界面

5.3 测试结果提取模块的实现

测试结果的选取过程的代码运行不会自动运行，而是需要用户驱动，例如用户在自动化测试工具的结果页面选中某一个显示为提交成功（Success）的测试方案，然后点击“Show Report”按钮；或者是通过后台的脚本定时触发查看测试结果请求，生成测试结果报表。此时上述代码段才会被调用运行。

测试结果选取的过程可以分为以下三个步骤：

1. 第一步，解析被选中方案包含的测试数据、测试环境、基准环境、测试类型和比较测试结果的模板；
2. 第二步，把第一步解析获得的数值作为参数，动态生成查询结果的 SQL 语句，分别从测试环境和基准环境中获取测试结果字段或者是输出消息。若是回归测试，则选取基线表中的相应结果作为基准环境的运行结果；
3. 最后一步，按照测试方案中测试案例的顺序号把获得的测试结果排序，再把测试环境中的测试结果和基准环境中的测试结果/基线测试结果一一配对，存入测试结果表中给测试结果比较模块读取。

具体工作流程如图 5.5 所示：

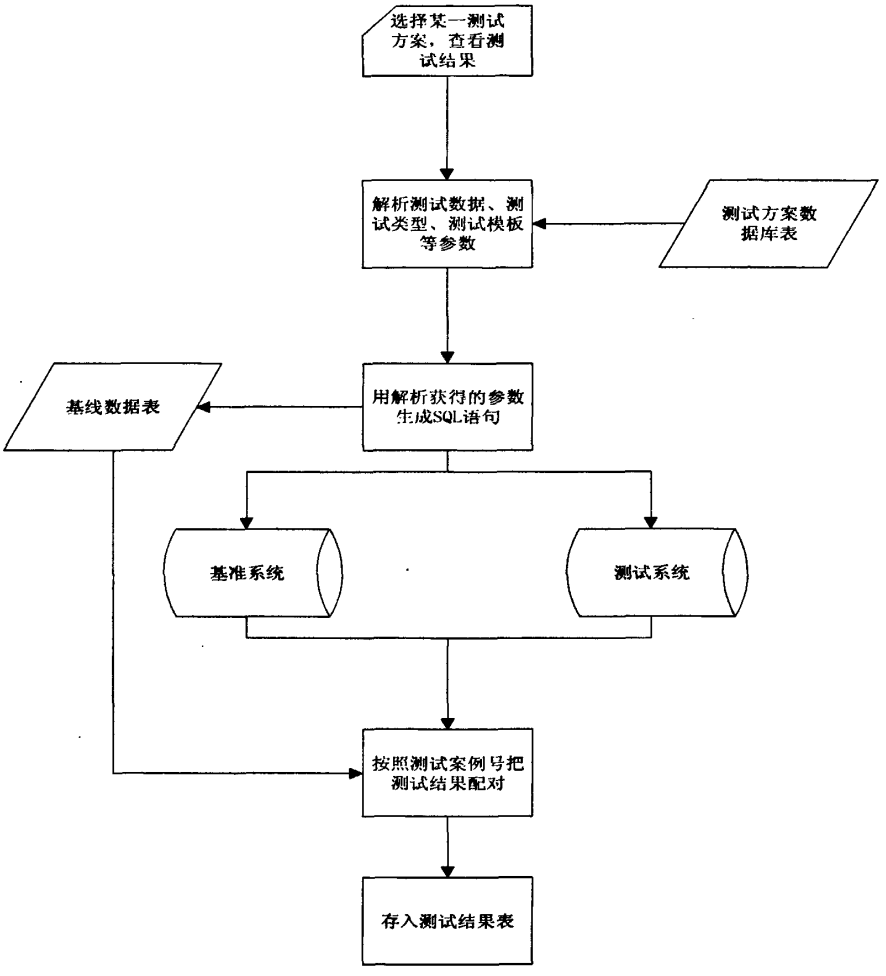

图 5.5 测试结果提取流程图

5.3.1 存储层实现

测试结果是由测试系统分别从测试环境和基准环境的数据库中（或者是基线表）查询得到。一般测试环境和基准环境的数据库数据量都比较大，容易导致查询速度缓慢；如果测试方案包含很多测试案例，那么测试系统查询测试结果的时间开销将会长得无法接受。所以，如果每次查看结果都要从2个环境中选取测试结果将会大大降低系统的性能。所以测试结果应该在本模块有数据库表有备份以提高系统性能。查询结果时，系统先在本模块的测试结果表中查找，只有找不到数据或者是用户手动点击“Update”按钮时，才重新去外部系统查找测试结果。

测试结果 Test_Result 表应该包含以下信息：

1. 测试方案 ID：标识本条测试结果属于哪个测试方案；
2. 测试数据 ID：标识本条测试结果属于哪条测试数据的运行结果；
3. 输出消息结果：测试结果最主要的是比较系统的 OutBound 消息，因为 FTM 系统有多种消息格式的 OutBound 消息所以每种格式的消息存为一列；
4. 其他字段结果：除了比较系统的 OutBound 消息外，还有一些表示系统运行结果的状态列也需要比较。

Test_Result 数据库表的表结构定义具体如表 5.4 所示：

表 5.4 Test_Result 表表定义

列名	类型	长度	备注
Test_Batch_ID	VARCHAR2	30	标识本条测试结果属于哪个测试方案；
Test_Data_ID	VARCHAR2	30	标识本条测试结果属于哪条测试数据的运行结果；
Outbound_MSG1_Data1/ Outbound_MSG1_Data2	VARCHAR2	2000	用来存放 Outbound 消息，目标系统的存 1 列，基准系统存 2 列；
Outbound_MSG2_Data1/ Outbound_MSG2_Data2	VARCHAR2	2000	有些测试案例有多条 Outbound 消息，其他同上；
Outbound_MSG3_Data1/ Outbound_MSG3_Data2	VARCHAR2	2000	同上；
Outbound_MSG4_Data1/ Outbound_MSG4_Data2	VARCHAR2	2000	同上；
Other_Field1/ Other_Field2	VARCHAR2	2000	其他需要比较的字段，用“字段名|字段值~字段名|字段值”格式组织。

5.3.2 逻辑层实现

逻辑部分主要需要实现以下 2 大块功能是：测试模板解析和查询测试结果的 SQL 语句生成。测试方案对象解析和查询 SQL 语句生成的具体工作流程图如图 5.6 所示：

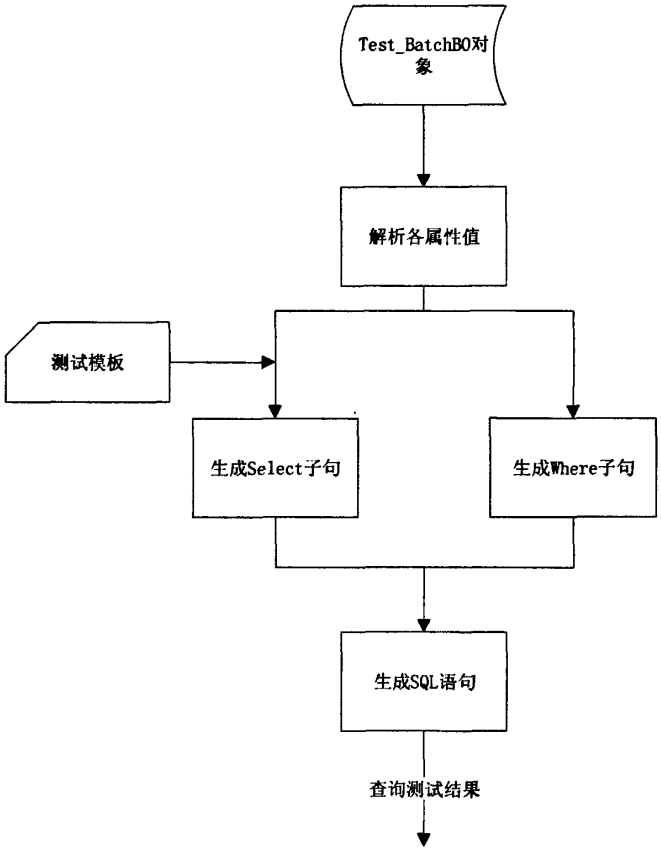

图 5.6 测试结果查询流程图

5.4 测试结果比较模块的实现

测试结果的比较模块的主要功能是把测试结果获取模块配对、整理好的数据进行比较得出测试结果。对于有输出消息的测试案例，如果测试环境的输出消息和基准环境的输出消息相比多了或者少了一部分，或者某个字段的数值不对。这种情况下，如果只是简单地告诉用户测试结果为“不匹配”，那显然是不合理的。

系统在输出测试结果时应该把有差异的部分用显目的颜色高亮以避免开发人员浪费时间在肉眼对比上。

因此，在流程上本模块主要分为以下4个步骤：

1. 从Test_Result表中选取需要比较的数据，先做简单比较判断整个测试案例的测试结果是否为“匹配”；
2. 若用户点击“View Detail”按钮查看详细情况，则把这条测试数据不匹配的字段或者OutBound消息分割成业务逻辑上的“最小字段”保存在VO对象中；
3. 把分割过以后的对象、字段按照具体不同的类型做比较；
4. 把比较以后得到的对象、字段按照比较结果格式化显示，生成最终报表。

具体的工作流程图如图5.7所示：

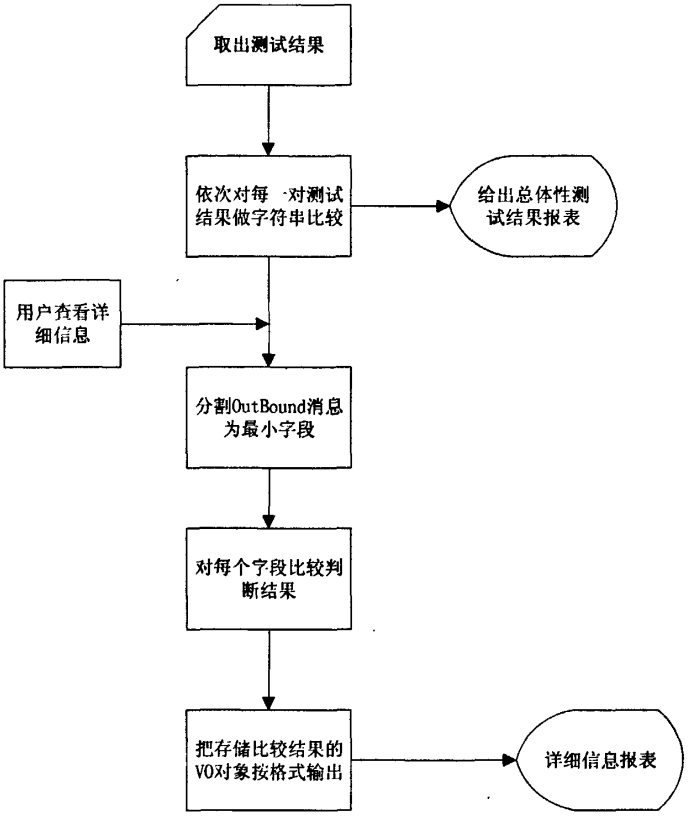

图 5.7 测试结果比较流程图

5.4.1 逻辑层实现

本模块包含的 Java 类主要分为 2 大类，第一类就是用来存储比较结果的 VO 类，这些类之间的类关系图如图 5.8 所示：

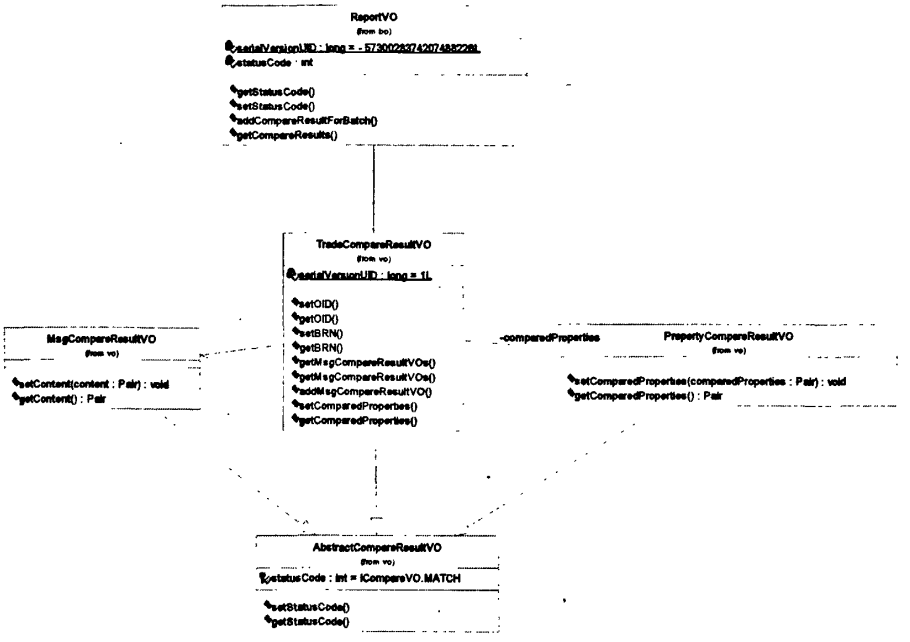

图 5.8 VO 类的类关系图

第二类则是把 VO 对象的值格式化输出到 HTML 中生成详细信息报表的 VOFormatter 类，这些类之间的类关系图如图 5.9 所示：

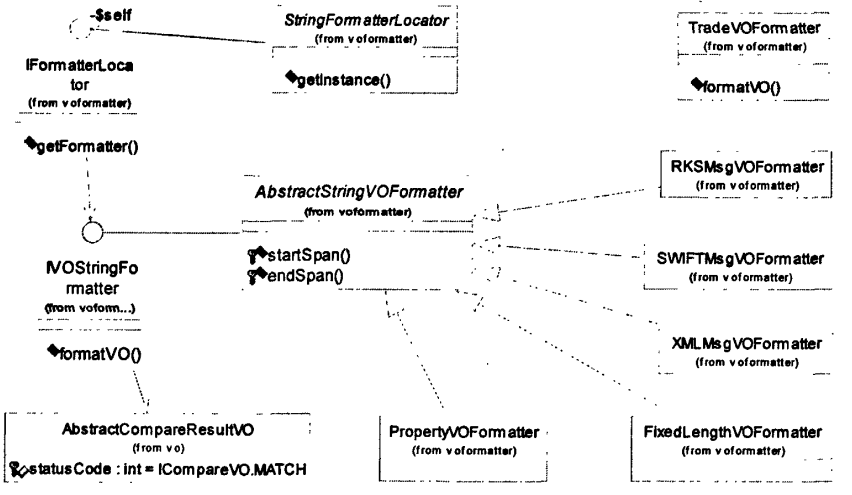

图 5.9 VOFormatter 类的类关系图

5.4.2 UI 实现

本模块的界面用来向用户显示测试结果的报表，一共有 2 个界面。第一个界面是用列表显示一个 Batch 的全部测试结果，在这个界面中，也提供本次测试的结果统计信息；第二个界面是显示一个测试案例的测试结果，不同部分高亮显示。

显示 Batch 测试结果的界面（Test Result Listing UI）如图 5.10 所示：

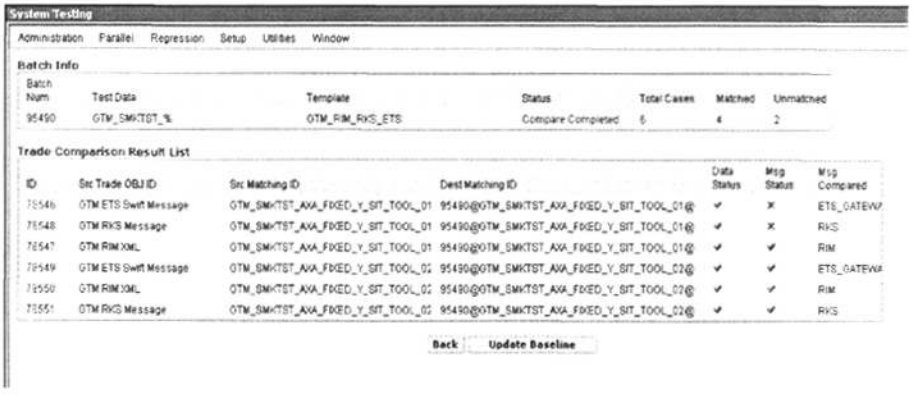

图 5.10 测试结果列表界面

显示详细测试结果的界面（Test Result Detail UI）如图 5.11 所示：

图 5.11 测试结果详细信息界面

5.5 本章小结

本章分模块详细阐述了各个模块需要实现的功能点和业务流程以及如何在UI界面层、代码逻辑层和后台数据库层中分别实现上述功能的细节。

第6章 应用结果分析

在本文中描述的网上金融交易系统重构项目，于09年的8月基本完成Coding，开始进入Bug Fix阶段。重构以后的新系统被部署到真实应用环境的镜像环境中，开始用真实的交易数据对重构以后的系统进行全面测试。自动化测试数据在每天系统数据库负载最轻的下午 3 点（美国东部时间晚上 2 点）针对过去 24 小时内运行的数据自动做一次平行测试。同时一个 SQA 小组的 3 个测试人员从系统所包含的商业逻辑作为需求，对重构以后的系统同步开展 UAT(User Acceptance Test)测试。

从 8 月初开始的平行测试中，重构以后的系统和现有的 FTM 系统运行结果的“不匹配”比率高达 95%以上，到 1 月份不到 1/1000 的“不匹配”比率。不得不说 Bug Fix 工作进行地是成功的！自动化测试系统在这期间做出了巨大的贡献。其贡献表现在两个方面：

1. 通过每天对 Production 环境上的平行测试发现了大量的 Bug，节约了开发人员通过集成测试寻找 Bug 的时间。
2. 在 Bug Fix 阶段，自动化测试系统能用来执行回归测试，以提高开发人员的 Bug Fix 效率，加快项目进度。

本文拟用量化的方法对自动化测试系统在重构项目中的贡献进行估算。

6.1 平行测试自动化效果分析

整个重构项目组从去年8月到今年一月一共Fix的Bug数量大概为1700左右，其中大约有 540 个的 Bug 是由 SQA 小组报告给开发组的。换而言之，被 Fix 的这些 Bug 中有接近 1200 个是独立被自动化测试工具发现的。

抛开被发现的 Bug 的重要性和对系统的影响大小，单从发现 Bug 的数量上看，自动化测试系统的工作效率可以简单的视为是 SQA 组的：(1700-540)/540 = 2.15 倍。相当于 6 个 SQA 测试人员的工作效果。实际上，开发组每天只有组长花大概半天的时间分析测试报告，从中发现和选取一部分待解决的问题分配给开发人员

研究然后再修正。

考虑到 SQA 的测试人员和开发组的开发人员在测试经验和业务逻辑知识上的差距，可以毫不夸张地说，自动化测试系统使开发团队集成测试的效率提高了 15 倍以上。

6.2 回归测试自动化效果分析

自动化测试系统的平行测试功能在发现 Bug 上发挥了巨大作用，其回归测试的功能在 Bug Fix 环节也同样发挥了非常重要的作用。

按照每天每个开发人员 Fix 3 个 Bug 的平均数目来做估算：一般情况下，很少有 Bug Fix 能一次成功的，假设每个 Bug 平均需要 2.5 次的回归测试来完成；每次回归测试用的测试案例在 20~50 个之间，假设平均需要的测试案例为 30 个。如果使用纯手工的方式执行回归测试的话，每个测试数据在使用前要需要花 5S 来修改测试数据做准备工作，在运行以后又需要花至少 25S 分别从基准系统和被测试系统选取输出信息和相关表示运行状态的列，再用肉眼（或者其他工具）比较 2 边的输出消息是否一致。这样一次回归测试需要花费至少 30*0.5=15 分钟；按上述假设，则每个开发人员每天花在回归测试上的时间至少为 2 小时。使用自动化测试系统则可以把执行一次回归测试的时间控制在 3 分钟左右。所以使用自动化测试系统的回归测试功能能使开发人员在 Bug Fix 的过程中提高 25%的工作效率。

除了 Bug Fix 中做的回归测试外，开发组每天还要执行一次测试案例数量在 100~500 条的回归测试；每周还要执行一次测试案例增量累加的回归测试。最后一种回归测试如果纯用手工完成，其花费的时间几乎是难以接受的。

6.3 本章小结

本章从平行测试和回归测试 2 个角度来估算自动化测试系统在发现 Bug 和提高 Bug Fix 效率 2 个方面的作用。并用量化的方法证明了自动化测试系统给重构项目带来的效率提升。

第7章 基于BP神经网络的Bug定位功能

7.1 Bug定位问题

采用本文介绍的自动化测试系统能有效的进行平行测试和回归测试从而把开发人员从机械而繁重的测试工作中解放出来，进而能大幅度提高开发效率。但是在使用自动化测试系统大大加快测试的速度以后另外一个开发过程中的瓶颈出现了——Bug定位的问题。

根据本次系统重构项目的经验数据，开发人员平均花在Bug寻找、分析上的时间占了Bug Fix的总时间的70%左右，而用于修改代码的时间平均只有20%左右。因此不难看出在自动化测试工具解决了测试这个瓶颈以后，Bug定位和分析成为另外一个阻碍项目进度的瓶颈。很不幸的是在Bug的寻找、分析这个难点上，目前没有成熟的软件或者工具可以辅助开发人员分析；而且对于Bug处理的研究更多的是基于如何从设计上剔除Bug[7]。开发人员只能通过分析代码、分析数据、打印日志消息或者Debug等几种传统方法来寻找出Bug然后分析并且修正它们。使用这样的纯人工方法定位Bug产生位置在小规模的系统中并无太大缺陷，但是如果这种方法应用在一个庞大的系统中就往往力不从心了。

7.1.1 人工Bug定位法

在本文涉及的重构项目中，开发人员采用的都是人工寻找、分析 Bug。在这个过程中遇到的最大的四点困难是：

1. 金融应用类系统对系统的强壮性要求很高，系统不会轻易终止对一条数据的处理。因此，如果不能定位Bug产生的模块，那么开发人员要分析的代码将将是整个系统！
2. 系统经过重构以后，即使最稳定的存储层也是变动巨大的。而仅仅靠开发人员用肉眼寻找、比较这些数据几乎不可能分析出Bug产生的原因，因此程序员通常只能通过分析代码或者到处打印Log寻找Bug；

3. 由于重构项目中的金融系统包含了丰富的商业逻辑，因此很多逻辑过程复杂而且难以理解。复杂的商业逻辑无疑进一步加大了开发人员寻找和分析 Bug 的难度；
4. 采用本文设计的自动化测试系统进行平行测试，一般只会给出重构以后的系统和现有系统之间最终运行结果的比较。因此，分析 Bug 时候只能采用回溯的方法。

所以开发人员只能靠从下游模块到上游模块的“接力”的方式人工分析、寻找 Bug。但是这样的 Bug 分析模式有很大的缺陷：

1. 下游的模块的开发人员将承担更加繁重的任务，因为每次的 Bug 分析工作都要由他们开始；
2. 在这样的模式下，大量的任务会被积压在下游模块；
3. 人力资源浪费严重，因为很多情况下，下游模块的开发人员在做重复的而且完全可以避免的工作。

显而易见，程序员单纯靠人工分析代码达到 Bug 定位方法不但效率低、工作分配不合理还浪费了大量人力资源。

7.1.2 使用软件定位 Bug 方法

既然人工 Bug 定位的方法既费时又低效，而且还会导致工作量分配不均还有人力资源浪费，那么是否可以考虑使用一些软件辅助开发人员定位和分析 Bug 呢？目前出现了一些分析和寻找 Bug 的软件，例如针对 Java 语言编写的程序的 FindBugs 等。但是此类软件的实质上只是对程序静态扫描、分析。这类软件对只有在动态运行中才能体现 出来的 Bug 束手无策。而其他的诸如 BugFree 类的软件实现的是 Bug 的管理而非分析。

因此，目前并没有一种现成的工具软件能有效地帮助开发人员在本文举例的系统开发过程中有效地分析和定位 Bug。

7.2 BP人工神经网络Bug定位方法

7.2.1 人工神经网络概述

人工神经网络（Artificial Neural Networks, ANN），是一种模仿动物神经网络行为的特征，实行分布式并行信息处理的算法的数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间互相连接的关系，从而达到处理信息的目的。人工神经网络具有学习和自适应的能力，可以通过预先准备好的一批相互对应好的输入－输出数据对，分析掌握两者之间的潜在的规律，最终根据这些规律，用新的输入数据来推算输出的结果，这种学习-分析的过程被称为“训练”[8]。

人工神经网络具有以下四个基本特征：

1. 非线性 非线性关系是自然界最普遍的特性。大脑的智慧就是一种非线性的现象。人工神经元处于激活或者抑制二种不同的状态，这种行为在数学上表现为一种非线性的关系。具有阈值的神经元所构成的网络具有更好的性能，可以提高容错性和存储容量；
2. 非局限性 一个神经网络通常是由多个神经元广泛连接而成的。一个系统的整体行为不仅仅取决于单个神经元的行为，更可能主要是由单元之间的相互作用、相互连接所决定的。人工神经网络通过单元之间的大量连接来模拟大脑的非局限性。其中，联想记忆是非局限性的典型例子；
3. 非常定性 人工神经网络具有自适应、自组织、自学习的能力。神经网络不但处理的信息可以有各种的变化，而且在处理信息的时候，非线性系统本身也在不断地变化。经常采用迭代的过程来描写动力系统的演化过程；
4. 非凸性 一个系统的演化方向，在一定条件下将取决于某个特定的状态函数。例如能量函数，它的极值相应于系统比较稳定的状态。非凸性是指这种函数有多个极值，故系统具有多个较稳定的平衡态，这将导致系统演化的多样性。

人工神经网络中，神经元处理单元可表示不同的对象，例如特征、字母、概念，或者一些有意义的抽象模式。人工神经网络中处理单元的类型可以分为三类：输入单元、输出单元和隐单元。输入单元接受外部世界传入的信号与数据；输出

单元输出系统处理的结果；隐单元是处在输入和输出单元之间的一层单元，从系统外部无法观察其状态。神经元之间的连接权值反映了单元之间的连接强度；信息的表示和处理则体现在网络处理单元的连接关系中。人工神经网络是并行的分布式系统，采用了与传统的人工智能和信息处理技术完全不同的机理，因此能克服传统的基于逻辑符号的人工智能算法在处理直觉、非结构化的信息方面的缺陷，具有自适应、自组织和实时学习的特点。

7.2.2 BP人工神经网络概述

在最早的人工神经网络算法当中，信息的传递只是单向的，即：输入单元→隐单元→输出单元。这样处于隐单元层各单元之间的连接权值很难根据外界反应相应地正确改变。一直到1986年由Rumelhart和McCelland为首的科学家小组提出了BP（Back Propagation）网络模型，BP网络能学习和存贮大量的输入→输出模式的映射关系，而无需事前揭示和描述这种映射关系的数学方程。它的学习规则是使用最快速下降法，再通过反向传播来不断调整网络的权值和阈值，使网络的误差平方和达到最小[9]。

BP人工神经网络的算法的流程如下[9]：

1. 创建一个具有i个输入层单元；h个隐藏层单元；o个输出层单元的网络。
2. 初始化所有的网络中的边的权值为一个比较小的初始值。
3. 在遇到终止条件前做以下的训练：
   1) 对于每个输入值Input，把输入数据沿网络向前传播，并且计算出每个输出单元的结果值；
   2) 把得到的输出结果和期待的结果进行比较；计算它的误差项，用误差项来修正输出单元的权值；
   3) 用输出单元的误差项逐层求隐藏层单元的误差项；并且用误差项修正隐藏单元的权值。
4. 结束训练。

7.2.3 BP 人工神经网络 Bug 定位方法

首先，本文提及的重构项目中的网上金融交易系统对数据完整性有很高的要求。因为即使系统运行过程中遭遇严重错误导致崩溃，系统也要具备正确地恢复到事故前状态的能力。因此该系统在设计之初就要求在各模块的数据库表中存储足够的数据以恢复任何一条运行记录。所以，每个模块的数据库表中有足够的数据来确定任何一条交易记录在该模块的运行状态。因此，通过对数据库表数据的分析来判断某条交易数据在该模块执行后的状态是否是正确是完全可行的。

其次，重构以后的必须保证现有系统的数据的完全支持。因此无论重构以后的系统和现有系统在系统架构、实现技术、实现方法上有多大的差异，在存储层上的差异还是相对小的。在本文描述的重构项目中，现有系统中有 95%以上的数据库的列元素能在新系统的数据库表中找到对应的列。

综上两点，通过对照分析重构以后的系统和现有系统中任何模块的数据库表元素的值，来预测该模块是否正确运行是完全可行的。因为每个模块的数据库表的列元素值和整个模块最终运行结果之间存在的联系实际上就是我们通过程序实现的商业逻辑。

BP 人工神经网络擅长的是处理那些规律隐藏在一大堆数据中的映射逼近问题，特别是需要通过学习自适应可调的问题[10]。因此，本文使用 BP 人工神经网络的算法，针对该网上金融易系统重构项目中寻找，定位 Bug 困难的问题，在自动化测试系统上设计了一个 Bug 分析子系统。该子系统的作用是对自动化测试系统给出结果为“不匹配”的测试案例做进一步分析，尝试定位造成这一错误的 Bug 所处的模块。该系统的输入值是测试结果中结果显示为“不匹配”的一组记录的 ID 号。

7.3 基于 BP 人工神经网络 Bug 定位系统

Bug 定位子系统位于整个自动化测试系统的末端，作用为当测试结果显示某一条测试案例在重构以后的系统和旧系统中运行结果不一致的时候，开发人员通过启用此子系统来辅助开发人员定位这条测试案例产生 Bug 的具体模块。

7.3.1 Bug定位系统框架设计

Bug定位子系统的工作流程为：

1. 输入测试结果为“不匹配”的测试案例的一对ID；
2. 根据输入的ID号分别从基系统和待测试系统中取得所需要的数据；
3. 把取得的数据分割、整理、配对再做比较。把数据通过预处理转换为BP人工神经网络能识别的“0”，“1”串输入；
4. 把预处理后的结果输入BP人工神经网络分析模块分析；
5. 根据编码表，把输出的“0”，“1”串转化为具体的模块信息。

具体的工作流程图如图7.1所示：

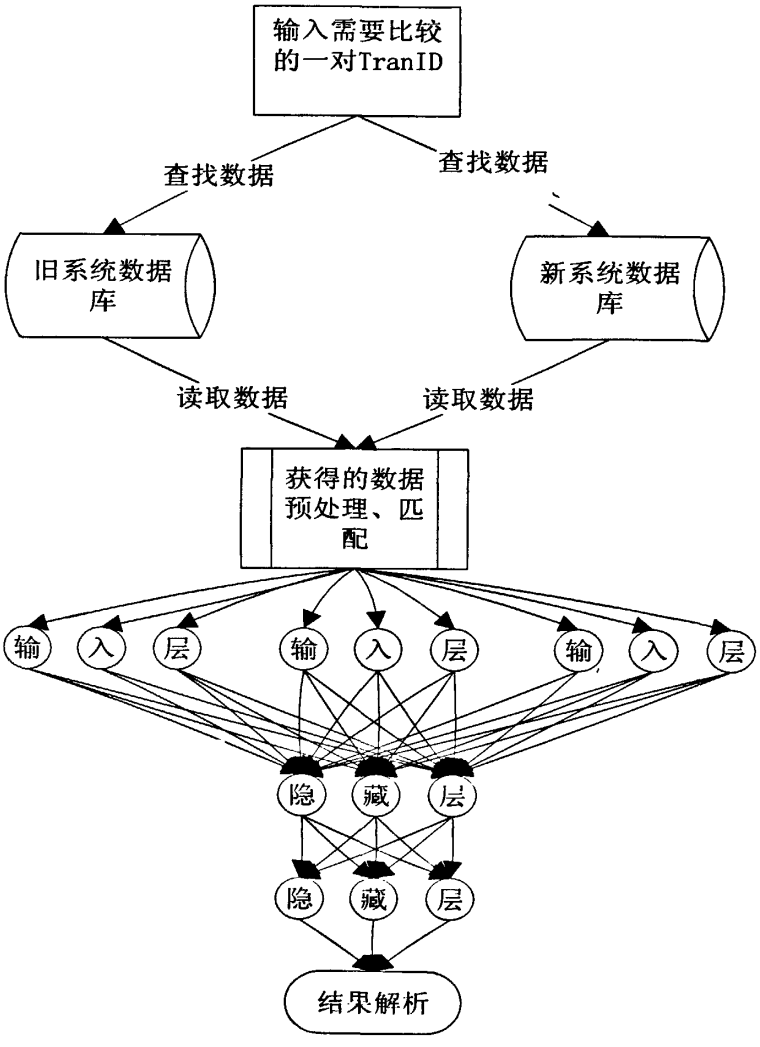

图 7.1 Bug 定位系统工作流程图

7.3.2 数据预处理模块

因为 Bug 定位子系统的分析数据来源——网上金融交易系统 FTM，该系统的数据库表里包含的数据类型有字符型、字符串型、布尔类型还有整数型和浮点小数。必须对这些取出来的数据做预处理才能得到适合 BP 神经网络处理的输入数据。此外，更大的难点在于，新、旧两个系统中数据库表的列并非完全对应。因此，根据数据来源对应状态的不同，需要做以下 3 类预处理操作：

1. 数据来源的数据表列在新系统和旧系统的数据库表中有直接的对应关系。这类数据的预处理只需要把取自2个系统的值进行简单的比较，如果比较结果为“匹配”则赋值 “1”，反之赋值“0”；
2. 数据来源在新系统和旧系统的数据库表中为1:n或者是n:1的对应关系（例如旧系统中把交易额表示为“货币类型+交易数目”存为一列，而新系统中把货币类型和数额分为两列存储）。这类的处理方法是把包含信息多的数据列拆开，再一一对应，然后按1中的方法处理；
3. 数据来源在新系统和旧系统的数据库表中为m:n的关系。对这类数据源，处理的方法是先把相关数据值拼接，再按照最小粒度分割，再把分割以后的单元一一对应按照步骤1处理。

经过以上处理后得到的就是适合BP人工神经网络处理的“0”和“1”串了。但是在实际应用中，0和1都是使用sigmoid单元的人工神经网络所无法达到的权值[11]，因此在实际实现过程中用0.1代替0；用0.9代替1。

预处理模块的工作流程图如图7.2所示：

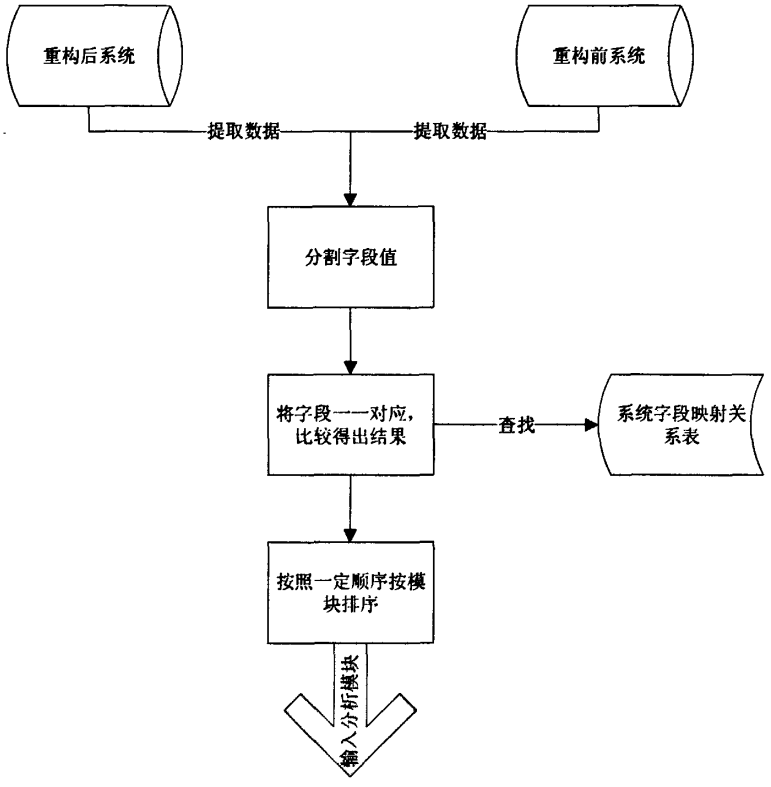

图 7.2 预处理模块工作流程图

7.3.3 BP 神经网络分析模块

已有理论研究证明：只要有一个隐含层的三层 BP 神经网络，就可以逼近任何映射函数以完成给定的映射任务[12]。在三层 BP 网络的基础上再增加一层隐藏层，虽然能进一步提高逼近的精度，但是会导致网络复杂度激增和训练时间的大量增加。所以本文中选择三层的 BP 神经网络。

输入层单元的数目由需要处理的数据元素数量决定。一般一个输入单元对应接受一个输入元素的输入信息。所以在本文中，输入层单元的数量等于经过预处理以后得到的单元数。

隐藏层节点的作用是从样本中提取并存储其内在规律。因此，隐藏层单元数目对网络性能至关重要，如果隐藏层单元太少，则无法产生足够的连接权组合数来满足样本对学习的要求；如果隐藏层单元过多，则学习以后网络泛化能力变差

[10]。隐藏层节点数 m 的确定方法，三个经验公式[13]具体定义见公式 7.1、7.2、7.3：

1. $k < \sum_{i=1}^{n} C\binom{m}{i}_i$　　公式(7.1)

其中K表示样本数，n表示输入层单元数。当$i > m$时，$C\binom{m}{i}=0$。

2. $m = \sqrt{n+o} + a$　　公式(7.2)

O为输出层单元数，a是一个[1,10]之间的常数。

3. $m=\log_2 n$　　公式(7.3)

本文设计的 Bug 分析系统输入层单元数量大约为 340，为了使该网络有足够的隐藏层节点来产生连接权组合数来满足样本对学习的要求，但又不会因为隐藏层节点数目过多而影响网络的泛化能力。在本例中选取经验公式 7.2 来确定隐藏层节点数目，本文取隐藏层单元数目为 60。

输出层单元的数目则由需要表示的输出信息种类和数目所决定。由本文第 2 章可知，重构以后的 FTM 系统，从上游到下游可以依次分为如下五个相互独立的功能模块：

1. 处理输入信息的 Inbound 模块；
2. 验证交易信息完整性的 Trade Capture 模块；
3. 处理具体商业逻辑的 Trade Processing 模块；
4. 做交易结算的 Settlement 模块；
5. 往下游系统发送消息的 Outbound 模块。

Bug 分析子系统最主要的目的是把产生 Bug 的代码定位到某个模块（如果有超过一个模块有 Bug，则报告最先产生 Bug 的那个模块）。因此，用 3 个输出层单元就足以表示所有 7 种可能发生的情况了，具体的编码-状态对应关系如表 7.1 所示：

表 7.1 输出层单元信息真值表

编码值	表示的状态
111	执行正确，无Bug；
110	Inbound产生Bug
101	Trade Capture产生Bug
100	Trade Processing产生Bug
011	Settlement产生Bug
010	Outbound产生Bug
001	N/A
000	输入信息缺失

最后，把每个输入层单元、隐藏层单元之间建立连接；每个隐藏层单元、输出层单元之间也建立连接来完成该BP神经网络的构建。

7.3.4 算法和关键参数的选取

本文选取随机梯度下降的包含双层Sigmoid单元的BP网络反向传播算法，算法简单描述如下[11]：

1. 建立并初始化网络，为了加快训练速度，把网络的权值初始化为(-2.4/N, 2.4/N)，其中N为该层单元数量[14]。
2. 在遇到终止条件前做一下训练：

   对于训练样例<X, T>，把输入按照Sigmoid函数映射关系沿网络前向传递：

   1) 把实例X输入网络，并计算网络中每个输出单元的输出值On；

   使误差沿网络反向传播：

   2) 对于网络每个输出层单元 k，计算它的误差项δ_k，计算方法如公式 7.4 定义：

$$n_k = o_k(1-o_k)(t_k-o_k) \qquad \text{公式(7.4)}$$

   3) 对于网络的每个隐藏层单元h，计算它的误差项δ_h，计算方法如公式7.5

定义：

$$\delta_h = o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k \qquad \text{公式(7.5)}$$

4) 更新每个网络权值w_{ij}，计算方法如公式7.6定义：

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \text{公式(7.6)}$$

为了防止网络在训练时收敛于局部最小值，并且加快训练时的收敛速度，按公式7.7加入冲量项[11]：

$$\Delta w_{ij}(n) = i\ p_i x_{ij} + m\Delta w_{ij}(n-1) \qquad \text{公式(7.7)}$$

上式中，η_l为学习步长，学习步长是该算法中非常重要的一个参数。学习步长太小会导致训练时收敛速度太慢；学习步长过大则会导致学习时候预测值发生振荡。因此本文采用变步长算法[15]，步长计算方法如公式7.8所定义：

$$\begin{cases} i = i \times 1.05 & E(t) < E(t+1) \\ i = i \times 0.7 & E(t) > 1.04 \times E(t+1) \\ i = i \end{cases} \qquad \text{公式(7.8)}$$

来使网络在训练中能尽量比较平滑而迅速地收敛。

7.4 Bug定位子系统预测准确率

有理论研究表明:神经网络中连接总数和训练该网络所需要的样本数有如公式7.9确定的近似关系:

$$N = \frac{W}{\varepsilon} \qquad \text{公式(7.9)}$$

其中ε为接近10的常数[13]。因此，本文设计的网络大约需要1800个训练样本才能达到稳定。

因为Bug定位子系统的数据预处理部分因为项目原因没有完成，故本文中用人工预处理数据的方法获得实验数据对BP神经网络分析模块进行训练和预测准确率测试。实验选取了重构项目从09年7月到9月的3个月中，Inbound模块修正76个bug；Trade Capture模块修正47个bug；Trade Processing模块修正

317个bug；Settlement模块修正114个bug；Outbound模块修正211个bug，总计765个bug。

本文针对每个bug取3组数据，得到2300个样本。再根据训练样本：测试样本为7：3的比值从各个模块的样本中按这一比例选取1600个训练样本和700个测试样本。使用这1600个训练样本完成对该网络的训练。此后再从训练样本集中选取700个训练样本，用这些训练样本和700个测试样本分别对该网络进行测试。测试结果如表7.2所示：

表7.2 bug分析系统测试结果

Bug所在模块	训练样本准确率	测试样本准确率
Inbound	90.6%	84.8%
Trade Pretreatment	94.6%	89.9%
Trade Processing	87.5%	78.8%
Settlement	88.3%	81.5%
Outbound	91.2%	85.9%

从上述实验结果可以看出，该bug分析系统对训练样本的识别率均在90%左右，对测试样本的识别率也基本在80%以上。若将各个模块发现bug数量定义为权值，则该系统整体的bug定位准确率为：对训练样本为90.2%，对测试样本为83.2%。

因此，完全可以认为本文提出的Bug分析系统对该网上外汇交易系统重构项目中的Bug有很高的识别率，从而在bug修正过程中能帮助程序员节省大量寻找bug的时间。

7.5 本章小结

本章首先提出了开发人员在用自动化测试系统获得测试结果以后，在定位Bug所在模块所面临的难题。然后提出了用BP人工神经网络的算法设计一个Bug定位分析子系统来辅助开发人员定位Bug产生的模块。并设计和实现了该基于BP神经网络的Bug定位子系统，最后用实验模拟来证明该系统确实能够帮助开发人

员定位 Bug 产生的模块。

第8章 总结与展望

8.1 总结

软件系统重构问题既是软件工程领域的一个重要的研究课题同时也是目前大量现有软件系统正在面临或是将要面临的问题。系统重构项目中从最开始的需求获取，到新系统的架构设计，再到代码编写阶段都充满了挑战。但是最大的挑战还是在测试阶段，重构项目工程量通常非常大，大到几乎不亚于重新开发一个相同规模的系统；而且还要考虑对原有系统数据的兼容性。因此，如何在有限的时间和人月代价下，有限的时间内尽可能充分地完成对重构后的系统的测试，以最大限度保证重构后系统的正确性以及同重构以前系统的一致性是一个非常有挑战的课题。

本文中遇到的系统重构项目还有其特别的困难：一、该系统需求文档有比较严重的缺失现象，而且没有足够的时间和预算让开发人员从现有的代码里分析、还原出完整的商业需求；二、该系统包含的业务多，范围广，使对金融业务流程和逻辑几乎没有了解的开发人员很难在集成测试阶段覆盖这些边界情况。本文通过项目实践，提出了一种通过自动化的平行测试来有效地解决：在开发人员“无明确需求；无充足时间；无金融知识背景”的“三无”情况下完成对一个商业逻辑复杂，业务覆盖面广的金融交易系统的测试难题。通过使用本文提出的方案，开发人员在较小的人月投入、较短的时间周期内完成了对重构以后的金融交易系统有效、充分的集成测试，达到了最大限度保证重构前后的系统功能上的一致性和正确性的目的。

本文的具体工作如下：

1. 分析了系统重构项目中测试环节面临的困难，并提出了并行化测试的解决方案；
2. 具体分析了平行测试和回归测试的工作流程和特点，并且设计、实现了一个能够有效地将这两种测试自动化运行的测试系统，并且证明了使用

该自动化测试系统确实能大大减少开发人员花费在测试上的时间，从而大幅加快开发进度；

3. 设计了运用BP人工神经网络的技术，来对测试中“不匹配”的测试案例进行进一步分析，定位出Bug产生模块的Bug分析定位子系统来进一步节省开发人员的时间，加快项目进度。

文中的创新之处在于：

1. 本文提出的自动化的平行测试的方法很好地解决了开发人员“无明确需求；无充足时间；无金融知识背景”的“三无”情况下完成对一个商业逻辑复杂，业务覆盖面广的金融交易系统的测试问题。把本来无法实现或者是很难完成的测试任务顺利完成了。
2. 本文提出的将BP人工神经网络的技术应用到软件Bug分析中来解决Bug分析、定位的难题具有创新性和想象力。

8.2 未来工作的展望

随着软件产业的不断发展，软件系统的重构这个课题也将越来越得到人们的重视。因此，如何在各种应用软件系统的重构项目中快速有效地完成重构以后系统的一致性和正确性的测试也将越来越受到业界人士的重视。本文阐述的自动化的平行测试的方法在“需求不明确，预算不充分”的困难条件下完成系统重构后的集成测试，符合目前很多系统重构项目的情况，具有较高的实用参考价值。但是本文提出的解决方案还有如下的缺陷，有待下一步研究和改进：

1. 文中设计的自动化平行测试系统有明显的测试盲点：当某一测试用例在重构之前的系统中运行的结果为“运行正确”和“运行错误”之外的其它状态时，该测试系统无法有效识别测试案例在重构前后的两个系统中运行结果是否为“匹配”。
2. 该自动化测试系统无法识别同类的错误，使测试人员无法从测试报告中准确了解测试的真实结果。例如，假设某一Bug造成100个测试案例中的20个出现错误，测试系统无法将这20个错误归为一类。

3. 由于重构前后的系统字段映射表内容的大幅缺失和数据预处理模块存在一些未解决的技术难点，Bug 定位系统仍然处在原型阶段。需要在具体实践应用中发现和解决问题。

参考文献

[1] Ginancarlo Succi, Michele Marchesi 著，张辉 译. Extreme Programming Examined [M]. 北京：人民邮电出版社， 2002.6

[2] DanielJ.Mosley 邓波等译. 软件测试自动化[M]. 北京:机械工业出版社, 2003.10

[3] Paul C.Jorgensen 著, 韩柯, 杜旭涛译. 软件测试[M]. 北京:机械工业出版社, 2003

[4] Fewster M,Granam D 著. 舒智勇,译. 软件测试自动化技术与实例详解[M]. 北京:电子工业出版社, 2000.

[5] 邓松良，刘海岩，陆丽娜 编著. 软件工程 [M]. 西安： 西安电子科技大学出版社, 2004.2

[6] PATTON R. 软件测试[M]. 北京:机械工业出版社, 2002

[7] Frederick P.Brooks.Jr. The Mythical Man-Month［M］. 汪 颖，译. 北京：清华大学出版社, 2007 : 142-144.

[8] 王伟 著. 人工神经网络原理[M]. 北京航空航天大学出版社.1995

[9] SimonHaykin 著・叶世伟,史忠植 译.神经网络原理[M].北京:机械工业出版社, 2004.

[10] 张立明. 人工神经网络的模型及其应用［M］. 上海: 复旦大学出版社, 1993： 32-47.

[11] (美) Tom M.Mitchell 著， Machine Learning［M］. 曾华军，张银奎，译. 北京：机械工业出版社, 2003 : 60-92.

[12] 靳 蕃 著. 神经计算智能基础 原理.方法［M］. 成都:西南交通大学出版社, 2000 : 130-137.

[13] 张立明 著：多层神经网络泛化特性分析，1997 年中国神经计算科学大会论文集（一）[C]. 北京：人民邮电出版社，1997: 36-39.

[14] 罗四维 著. 大规模人工神经网络基础[M]. 北京:清华大学出版社, 2004: 38-70.

[15] 胡伍生 编著. 神经网络理论及其工程应用[M]. 北京: 测绘出版社, 2006: 63-72.

[16] 虞和济 著. 基于神经网络的智能诊断[M]. 北京: 冶金工业出版社, 2000: 10-15.

[17] Hopfield J. Artificial Neural Networks [J]. IEEE Circuits and Devices Magazine, 1998, 4(5): 3-10.

[18] Tomas Velasco, Mark R. Rowe. Back propagation artificial neural network for the analysis of quality control charts[J]. Computers & Industrial Engineering, 1993, 25(4): 397-400.

[19] Robert V. Binder. Testing Objects: Myth and Reality [J]. Object Magazine, 1995, 5 (2): 73~75.

[20] S.Ntafos. On random and partition testing[J]. Software Engineering Notes, 1998, 23 (2):42-48

[21] Hideo Fujiwara, Tomoo Inoue. Optimal Granularity and Scheme of Parallel Test Generation in a Distributed System[J]. IEEE Transactions on Parallel and Distributed Systems 1995, 6(7).

[22] Christoph Stoermer, PAnthony Rowe. Model-centric software architecture reconstruction[J]. Software-Practice & Experience. 2006, 36(4).

[23] Stephane Ducasse, Damien Pollet. Software Architecture Reconstruction: A Process-Oriented Taxonomy[J]. IEEE Transactions on Software Engineering. 2009, 35(4).

[24] Bradley S. Green. Software test automation[J]. ACM SIGSOFT Software Engineering Notes. 2000, 25(3).