中文摘要

随着信息技术的飞速发展，软件产品已经深入到社会的各个领域，软件产品的质量成为人们共同关注的焦点，因此软件测试就成为保证软件质量的重要方法之一，它贯穿了软件工程的全过程。随着软件规模的扩大，软件复杂性的提高，软件测试技术的不断发展，越来越多的测试人员发现传统手工测试成本高、执行繁琐、效率低等特点已远远不能满足现实的需求。为了克服手工测试的这些缺点，自动化测试技术被广泛地引入进来，并逐渐成为软件测试的发展方向。自动化测试框架的出现表明软件自动化测试技术正在趋于成熟，早期使用录制回放和脚本工具的不足正在被克服，使得自动化测试更加经济、有效，更加有利于实施维护。

本文首先介绍课题背景，软件测试的历史与现状；然后阐述软件自动化测试的概念，介绍软件自动化测试生命周期，并通过与手工测试的比较，指出自动化测试的优点与局限；接着介绍现有各种自动化测试框架的概念，包括测试脚本模块化框架，测试库架构框架，数据驱动测试框架，关键字驱动测试框架，并利用这些自动化测试框架对被测应用程序进行测试，从而比较各种框架的优缺点；然后在对各种自动化测试框架比较的基础上提出一种软件自动化测试混合框架，该框架将现有各种自动化测试框架的优点有效地结合在一起，采用模块化结构组织测试列表，提高测试列表的复用性，运用数据驱动和关键字驱动技术降低测试脚本维护开销。文章着重阐述了软件自动化测试混合框架的结构以及设计方法，并结合自动化测试生命周期介绍运用此测试框架在决策信息管理系统上进行测试的详细步骤。最后，总结出软件自动化测试混合框架的优点，提出设计自动化测试框架的原则。

关键词：软件测试；软件自动化测试；软件自动化测试框架；WinRunner
分类号：TP311.56

ABSTRACT

With the rapid development of information technology, software is used in every field of society. More and more people focus on how to improve the quality of software, so software testing which impenetrate whole process of software engineering becomes one of the most important ways for quality assurance. With the enlargement of software scope, the improvement of software complexity and the development of software testing, more and more tester is realizing that the trait that high costs, fussy and inefficient execution of manual test is not satisfy with the requirement now. In order to overcome these respects, automated testing is being introduced into software testing and becoming the direction of software test. The appearance of software automation test framework indicates that the technology of software automated testing is going to be mature. Automated testing will become more and more economical and efficient, and be more advantageous to be implemented maintained.

This paper first introduces the background of task, history and actuality of software testing. Secondly, discuss the concept of automated testing and through the comparison with manual test to indicate the predominance and defect of automated testing. Thirdly, discuss conception of some software automation test frameworks, such as test script modularity framework, test library architecture, data-driven testing framework and keywords-driven testing framework. Then, use these frameworks for automated test and compare advantages and disadvantages among them. At last, this paper brings forward a hybrid software automation test framework which inherits the advantages of different software automation test frameworks. This framework adopts modular structure to organize test list, applies data-driven and keyword-driven technology to improve efficiency of test and reduce cost of script maintains.

This paper mainly discusses the structure and design way of hybrid software automation test framework and expatiate the process of using this framework for testing in decision information management system. At last paper summarizes the advantage of hybrid software automation test framework and puts forward the principle of software automation test framework design.

KEYWORDS: Software Testing; Software Automated Testing; Software Automation Test Framework; WinRunner

CLASSNO: TP311.56

致谢

两年半的研究生生活转眼间就要结束了，回顾这段难忘的时光让我充满了眷顾与感激。在这段时间里我学到了更多的知识，参与了更多的实践，认识了许多同学、老师和朋友。在这里我要对他们给予我的帮助、关心与鼓励表示衷心的感谢。

首先我要感谢我的导师须德教授。在研究生学习期间，须老师不仅在研究上给予指导，更在生活上无微不至的关心和照顾我们。须老师严谨的治学态度，为我们树立了榜样。从他身上我们不仅学到了知识，更重要的是学会了做人、治学的道理。论文撰写期间，须老师经常给我提出指导和修改意见，论文的顺利完成离不开他的悉心指导。

其次我要感谢北京交通大学计算机研究所多媒体数据库实验室老师和同学。感谢徐保明老师、李曦老师和王涛老师在项目中给予的指导和帮助，感谢宋泽海老师在科研环境以及科研设备中给予的大力支持，感谢实验室，以及研究生班上和我朝夕相处的所有同学在学习和生活中给予的关心和帮助。

此外我还要感谢我的父母和家人，感谢他们多年的养育之恩，感谢他们为我求学付出的种种艰辛，是他们的教诲伴我健康成长，是他们的关心让我有了战胜困难的勇气。

最后，衷心地感谢在百忙之中审阅论文的各位老师和专家，恳请各位老师多多批评指正，并提出宝贵的意见。

1 引言

1.1 课题背景

信息技术的飞速发展，使软件产品应用到社会的各个领域，上千万行的大型系统软件及百万行的应用软件已屡见不鲜，然而软件的质量却一直是令所有人头痛的问题，因为随着规模的扩大，对于质量的保证已经成为了一项异常艰苦的工作，而对软件进行测试就是保证软件质量最重要和最有效的方法。

软件危机曾经是软件界甚至整个计算机界的热门话题。为了解决这场危机，软件从业人员、专家和学者做出了大量的努力。现在人们已经逐步认识到所谓的软件危机实际上仅是一种状况，那就是软件中存在缺陷，正是这些缺陷导致了软件开发在成本、进度和质量上的失控。有缺陷是软件的属性，而且是无法改变的，因为软件是由人来完成的，所有由人来完成的工作都不会是完美无缺的。问题是在于应如何尽量去避免缺陷的产生和消除已经产生的缺陷，使程序中的缺陷密度达到尽可能低的程度。给软件带来缺陷的原因很多，具体地说，主要有以下几点：

（1）交流不够、交流上有误解或者根本不进行交流。在进行软件开发时，没有做充分的需求分析。即在不清楚该做什么或不该做什么的细节(应用的)的情况下，进行软件设计。

（2）软件复杂性。图形用户界面(GUI)，客户/服务器结构，分布式应用，数据通信，超大型关系型数据库以及庞大的系统规模，使得软件及系统的复杂性呈指数增长，没有现代软件开发经验的人很难理解它。

（3）程序设计错误。像所有的人一样，程序员也会出错。

（4）需求变化。需求变化的影响是多方面的，客户可能不了解需求变化带来的影响，也可能知道但又不得不那么做。需求变化的后果可能是造成系统的重新设计，设计人员的日程的重新安排，已经完成的工作可能要重做或者完全抛弃，对其他项目产生影响，硬件需求可能要因此改变，等等。如果有许多小的改变或者一次大的变化，项目各部分之间已知或未知的依赖性可能会相互影响而导致更多问题的出现，需求改变带来的复杂性可能导致错误，还可能影响工程参与者的积极性。

（5）时间压力。软件项目的日程表很难做到准确，很多时候需要预计和猜测。当最终期限迫近和关键时刻到来之际，错误也就跟着来了。

要解决上述问题，即保证软件的质量，提高产品的可靠性和有效性，软件在

开发的过程中必须进行测试，而且要尽早测试、经常测试、充分测试和全面测试。正如任何生产过程都离不开产品质量检验一样，测试工作也是软件开发过程中必不可少的环节。测试是迄今为止人们所能找到的保证软件质量的最好方法。测试的目的是要发现软件中的错误，降低软件运行时的风险。软件测试在开发成本中占有40%以上的比例，它是保证软件质量的重要手段[1]。

随着软件应用范围的扩大，软件复杂度的提高，以及软件设计技术的不断发展，软件开发规模越来越大，处理的问题愈来愈复杂。相对来说传统的软件测试技术和方法以及测试工具已无法满足大型的、复杂的软件测试需要。软件自动化测试已成为当前软件测试技术研究的重点和难点，有关软件自动化测试技术、理论的研究和软件自动化测试工具的研发越来越受到软件界的重视。

1.2 课题意义

随着软件规模的扩大，软件复杂度的提高，无论是软件开发商还是最终使用软件的用户对软件质量的要求越来越高。软件测试成为软件工程的重要环节，在软件生存周期中占有非常突出的位置，它直接关系到软件的质量、开发速度和成本。为了提高软件开发的效率和软件的质量，将自动化测试代替一部分手工测试是实现这一目标的行之有效的方法。然而要更好地实现软件自动化测试，就必须有一个理想的自动化测试框架，来保证自动化测试有效地进行。

本文在比较了几种常用的软件自动化测试框架的利弊之后，提出了一种混合各个软件自动化测试框架优点的新的测试框架，该框架既能够实现测试描述与测试实现的分离，又能够实现测试数据与测试用例的分离，此外还实现了测试脚本的模块化管理，从而提高了测试脚本可复用性和可维护性，进而提高软件自动化测试的效率，减少软件自动化测试的成本。

1.3 软件测试历史与现状

1.3.1 软件测试的历史

Edward Kit 在他的畅销书“Software Testing In The Real World : Improving The Process”中将整个软件开发历史分为三个阶段[2]：

第一个阶段是60年代及其以前，那时软件规模都很小、复杂程度低，软件开发的过程随意。开发人员的 Debug 过程被认为是唯一的测试活动。其实这并不是现代意义上的软件测试，当然这一阶段也还没有专门测试人员的出现。

第二个阶段是70年代，这个阶段开发的软件仍然不复杂，但人们已开始思考开发流程问题，并提出“软件工程 Software Engineering”的概念。把软件工程分为需求分析、设计、编码、测试和维护几个阶段的软件生存期的概念。同时，在软件开发的实践中，人们还认识到，在开发初期发现并排除软件错误所付出的代价，远比在完成编码以后经过测试发现错误并加以改正的代价小得多。但是这一阶段人们对软件测试的理解仅限于基本的功能验证和 Bug 搜寻，而且测试活动仅出现在整个软件开发流程的后期，虽然测试由专门的测试人员来承担，但测试人员都是行业和软件专业的入门新手。1972 年 6 月在美国北卡罗来纳大学召开了首届软件测试正式技术会议，成为软件测试技术发展中的一个重要里程碑。1975 年 W.C.Hetzel 将在北卡罗来纳大学召开的软件测试会议上发表的论文整理出版了《Program Test Methods》一书，书中纵览了测试方法以及各种自动测试工具，这是专题论述软件测试的第一本著作。70 年代中期 J.B.Goodenough 和 S.L.Gerhart 首先提出软件测试的理论，从而把软件测试这一实践性很强的学科提高到理论的高度，被认为是软件测试技术发展过程中具有开创性的工作。此后 W.E.Howden, Gerhart 进一步总结原有的测试理论并进一步加以完善，使软件测试成为有理论指导的实践性学科参考。

第三个阶段是 80 年代及其以后，软件和 IT 行业进入了大发展。软件趋向大型化。与之相应，人们为软件开发设计了各种复杂而精密的流程和管理方法（比如 CMM 和 MSF），并将“质量”的概念融入其中。软件测试已有了行业标准（IEEE/ANSI ），它再也不是一个一次性的，而且只是开发后期的活动，而是与整个开发流程融合成一体。软件测试已成为一个专业，需要运用专门的方法和手段，需要专门人才和专家来承担。软件公司为了保证产品质量，提高产品在市场上的竞争力，成立了独立的测试部门，承担软件测试的任务。90 年代，测试工具的出现，测试支持度、测试成熟度等新概念的提出进一步表明软件测试技术的不断发展。

1.3.2 国内外研究现状

软件测试作为软件工程学科的一个重要分支，随着软件的发展而发展。自从20世纪70年代以来，国内外许多学者和组织在软件测试方面进行了大量的研究工作，形成许多经典的软件测试技术和软件测试流程管理规范。

目前软件测试技术的研究主要向网络化、大型化和自动化方面发展，主要包括 Client/Server 系统、基于 Internet 的 Web 应用系统、嵌入式系统的软件测试技术的研究和测试工具的研制。

美国的 IEEE,ACM 等组织制定了一系列软件测试规范，国外的许多大学(如 Carnegie Mellon、华盛顿大学等)、研究机构(如 National Software Testing Lab 等)和公司(如 Software Research, Mercury Interactive, Rational Corporation 等)进行了大量软件测试的研究和应用工作。其中 Carnegie Mellon 大学侧重于回归测试和 Client/Server 测试技术的研究，George Mosan 侧重于基于规范的测试自动生成和面向对象测试技术。比较流行的软件测试工具集有 Software Research 的 TestWorks、Rational 的 Robot、Mercury Interactive 的 WinRunner/LoadRunner 等[3]。

国内的软件产品测试技术研究起步晚，在人力、物力上投入与国外相比差距较大[4]。

（1）上海市软件评测中心(SHSTC)[5]。评测环境:Solaris 平台、AIX 平台、OS 400 平台、HP- UX 平台、NT 和 Linux 平台、网络环境；评测业务包括软件评测和网络评测，软件评测包括：软件 产品登记测试、软件产品性能测试、单项功能确认测试、标准符合性测试、开发过程中的单元测试和集成测试、信息工程项目验收测试、信息工程监理；网络评测包括:网络布线认证测试、网络系统性能测试。

（2）中国软件评测中心(CSTC)[6]。它是国家级第三方计算机软件产品检测机构。对软件从功能上、兼容性、可扩充性、性能、安全稳定性、速度、易用性、用户文档、资源占用率等九个方面进行全面测试。

（3）I-TEST 测试管理系统[7]。I-TEST 测试管理系统专注于测试流程的管理，管理功能全面，对测试流程的设计科学、规范、合理。结合了开发人员在业界的经验和对国内软件开发现状的把握等基础上开发而成，非常贴近国内用户的需求；具有强大的测试用例、测试步骤的编辑和管理功能，BUG 的跟踪处理功能，所有输出结果自动生成 WORD 文档的功能，同时有强大的统计分析、决策支持能力，使用维护方便，具有良好的性价比，是目前国内市场不可多得的测试流程管理软件。

（4）北京大学、北京航空航天大学进行了一系列软件分析和测试工具的研究和开发，研制了一系列的程序理解工具和测试工具，比较具有代表性的工具有 SafePro C/C++，SafePro/javao 航空计算机研究所、南京大学在嵌入式系统测试方面进行较多工作，开发了一些静态分析工具和测试用例自动生成工具。西北工业大学在航空软件仿真测试以及 C/S 系统的测试方面进行许多研究工作并取得了一系列成果。

随着面向对象技术、软件重用技术以及 Internet 的广泛应用，软件测试技术面临着新的挑战。目前软件测试技术领域内的研究热点有[3]:

（1）针对新的软件开发技术开展的软件测试技术研究，包括面向对象技术、Internet 结构、Java 语言、自动生成软件等软件测试技术的研究；

（2）对某类软件的特点开展的实用软件测试技术和方法的研究，如实时软件、系统软件、嵌入式软件等不同类型软件的特点开展测试技术研究；

（3）测试自动化技术的研究，即以提高测试各阶段工作的自动化程度，减轻人工测试负担为目的开展的研究，如回归测试，覆盖测试等。

（4）测试工具与测试环境的研究，依据软件测试的技术和方法开发相应的测试工具与环境，如测试计划工具，测试设计工具、测试管理工具、静态分析工具、回归测试工具、性能及网络负载测试工具，为提高工具使用的有效性、工具间互操作性以及信息的共享性而开发的、由若干工具经过有机结合形成的测试环境等。

1.4 论文组织安排

本文在分析、借鉴、吸收大量文献资料的科研成果的基础上，设计了一种软件自动化测试混合框架，并利用该框架进行了软件自动化测试的实践。论文组织结构如下：

第一章 引言 介绍课题研究的背景及意义，软件测试的历史以及国内外的研究现状，论文的组织安排。

第二章 软件自动化测试概述 介绍软件自动化测试的基础知识，包括软件自动化测试的概念，自动化测试生命周期，并通过与手工测试的比较，指出软件自动化测试的优势与局限。

第三章 软件自动化测试框架比较 通过具体测试实例介绍现有的四种自动化测试框架的概念，并比较了这四种测试框架的优缺点。

第四章 软件自动化测试混合框架的设计 提出一种结合现有自动化测试框架优点的软件自动化测试混合框架，并主要介绍该框架的组成，具体设计方法，以及该框架的执行流程。

第五章 软件自动化测试混合框架在DIMS中的应用 介绍了按照自动化测试生命周期的步骤，运用自行设计的软件自动化测试混合框架在决策信息管理系统（DIMS）进行自动化测试的详细过程。

第六章 结论 结合所做工作，总结软件自动化测试混合框架的优点以及需要改进的地方，提出设计自动化测试框架的应遵循的原则。

2 软件自动化测试概述

软件测试的工作量虽然很大(据统计，会用到40%的开发时间，一些可靠性要求高的软件测试时间甚至占到总开发时间的60%)，但是许多操作是重复性的、非智力创造性的工作。以往大多采用手工调试和测试的方法，或编制测试程序进行测试，即耗去大量时间又不规范，在将软件分发给用户使用时，常常发生问题，严重时导致系统瘫痪，这样不仅无法收回投资，而且造成巨大经济损失。采用自动化测试的方法可以大大提高软件测试的效率，因此应用自动化测试，企业在这方面的投资，会对整个开发工作的质量、成本、和周期带来非常明显的效果。

2.1 软件自动化测试的概念

软件自动化测试就是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，模拟手动测试步骤，完成全自动或半自动测试[8]。其目的在于缩短测试周期，增强对软件性能方面的测试能力等，从而达到保证软件质量并使软件能够提前上线。

全自动测试就是指在自动测试过程中，根本不需要人工干预，由程序自动完成测试的全过程[8]。

半自动测试就是指在自动测试过程中，需要人工手动输入测试用例或选择测试路径，再由自动测试程序按照人工指定的要求完成自动测试[8]。

软件自动化测试是一门技术。对于任何软件系统，测试者希望通过有限的测试用例发现软件中的大部分缺陷。自动化测试使得所取得的测试用例得以重复测试，并能保障测试的科学性、严密性、组织性。其次，自动化测试是一种机制，它不仅是指运用自动工具进行测试，而且包括如何管理测试自动化，如何确定自动化测试的方法以及如何组织测试等。

高效的自动化测试来源于好的测试软件，这些测试软件是由经验丰富的测试人员精心设计的，在此基础上再应用自动化测试技术就可以实现自动测试。自动化测试通常要比手工测试经济得多，其开销只是手工测试的一小部分。自动化测试的方法越好，长期使用获得的收益就越大。

2.2 自动化测试生命周期

软件自动化测试是一个复杂的过程，为了确保自动化测试的成功，就必须遵循软件开发流程。我们需要像开发软件项目一样把自动化测试作为一个项目来执行，自动化测试必须被视为一个完整的软件开发过程。因此自动化测试的执行应经过需求定义、测试计划、测试设计、测试开发等一系列的活动。为此，Dustin、Rashka 和 Paul 合作公布了自动化测试生命周期方法学（Automation Test Lifecycle Methodology，ATLM）——这是一种经过调整的结构化方法学，能确保自动化测试的成功实现[9]。它定义了以下六个阶段方法学：自动化测试的决定；自动化测试工具的选择；自动化测试的引入；自动化测试计划、设计和开发；自动化测试的执行和管理；自动化测试项目评审。

（1）自动化测试的决定。在这一阶段，测试团队应该找出能自动化的软件测试过程以及应该自动化的软件测试过程；知道自动化测试的预期结果和列出在正确执行自动化测试后的益处；同时，需要列出自动化测试工具的备选方案，这对于获得管理层的支持是非常有帮助的。

（2）自动化测试工具的选择。测试工具是用于促进测试过程的。测试工具能被用于实现一个过程并执行测试过程的各种规范。在很多情况下，测试工具自带的内建程序可以被理解为过程。然而，它们往往是不完整的，不能正确反映过程。一个最好的软件测试工具使你能够将它和你的测试需求达成一致。而且它们提供高度可自定义的工作流程和跟踪报告能力。因此，这一阶段用于指导测试人员对自动化测试工具进行评估，并确定使用的测试工具。

（3）自动化测试的引入。这个阶段概括了成功引入自动化测试到一个新的项目中所必须的步骤:①测试过程分析确保整个测试过程和测试策略适当，必要时可以加以改进，以便成功地引入自动化测试；②测试工具考查阶段测试工程师根据测试需求、可用的测试环境和人力资源、用户环境、平台以及被测的应用的产品特性，研究将自动化测试工具或实用程序引入测试工作是否对项目有好处。

（4）自动化测试计划、设计和开发。①测试计划：测试计划是测试过程中最重要的活动。它包括风险评估、鉴别和确定测试需求的优先级，估计测试资源的需求量，开发测试项目计划以及给测试小组成员分配测试职责。②测试设计：该阶段需要确定所要执行的测试数目、测试方式（如路径或功能），必须执行的测试条件，以及需要建立和遵循的测试设计标准；③测试开发：在此阶段，测试团队要在测试分析和设计的基础上，制定测试程序开发/执行进度表，创建具有可维护性、可重用性、简单性和健壮性的测试程序。

（5）测试执行和管理：测试团队必须根据测试程序执行进度执行测试脚本，

并改善这些脚本。在这个过程中还必须评审测试的结果，以避免错误的结果。系统的问题应通过系统问题报告记录在案，并帮助开发人员理解和重现这些问题。最后，测试团队需要进行回归测试来追踪和关闭这些问题。

（6）测试项目评审与评估：测试项目的评审与评估必须贯穿于整个自动化测试生命周期，以利于测试活动的不断改进，必须有相应的标准来衡量评审的结果。

通过运用 ATLM 中所概括的系统的方法，一个测试团队就能在测试资源受限的情况下利用这一途径组织和执行测试活动，并且达到使测试覆盖率最大的目的。很明显，强调自动化测试是软件工业的一个很大转变。这种转变不仅包括应用程序工具和测试自动化，它还贯穿于系统开发的整个生命周期。ATLM 和系统的开发生命周期是同步进行的。要使软件专业人员转到自动化测试上来，结构化的方法是必须采用的。因此，ATLM 是革命性的，它代表了一种结构化的方法，规定了确定测试方法和执行测试的流程，使得软件专业人员能进行可重复的软件测试。

2.3 软件自动化测试的优点

随着软件规模的扩大，软件复杂性的提高，传统的手工测试已远远不能满足现实需求，并逐渐被自动化测试所代替。其原因在于手工测试具有无法保证测试的科学性和严密性的缺点，这是因为[10]：

（1）测试人员要负责大量文档报表的制定和整理工作，会变得力不从心；

（2）受软件开发日期，开发成本，人员，资源等多方面因素的限制，难以进行全面的测试；

（3）如果改正软件中的错误需要很长时间，回归测试将变得异常困难；

（4）对测试过程中出现的大量缺陷缺乏科学有效的管理手段，责任含混不清，没有人能够向决策层提供精确的数据来衡量当前的工作进度和工作效率；

（5）反复测试带来的倦怠情绪及其它人为因素使得测试标准前后不一，测试花费的时间越长，测试的严格性也就越低。

而上面这些问题大都可由自动化测试工具来解决。自动化测试具有以下优点[11]：

（1）令软件新版本进行回归测试的开销最小。对于软件开发，每发布一个新版本，其中大部分功能和界面都和上一个版本相似或完全相同，这时要对新版本再次进行已有的测试，这部分工作多为重复工作，特别适合使用自动化测试来完成，从而令回归测试的开销达到最小。

（2）可以在更短的时间内完成更多的测试。基于计算机的高效计算能力，自动化测试的最根本的优点在于，与手工测试相比，能在更少的时间内完成更多的

测试工作，因此也就缩短了测试时间。

（3）可以完成一些手工测试不能或难以完成的测试。对于一些非功能性方面的测试，如：压力测试、并发测试、大数据量测试、崩溃性测试等，这些测试用手工测试是很难，甚至是不可能完成的．但自动化测试则能方便地执行这些测试，比如并发测试，使用自动化测试工具就可以模拟来自多方的并发操作了。

（4）测试具有一致性和可重复性。由于每次自动化测试运行的脚本是相同的，所以可以进行重复的测试，使得每次执行的测试具有一致性，手工测试则很难做到这点。有些测试可能在不同的硬件配置下执行，使用不同的操作系统或不同的数据库，此时要求多平台产品的跨平台质量的一致性，这在手工测试的情况下更不可能做到。好的自动测试机制还可以确保测试标准与开发标准的一致性。例如，此类工具可以测试每个应用程序的相同类型的功能以相同的方法实现。

（5）更好地利用资源。将繁琐的测试任务自动化，可以使测试人员解脱出来，将精力更多地投入到测试用例的设计和必要的手工测试当中。并且，理想的自动化测试能够按计划完全自动地运行，使得完全可以利用周末和晚上的时间执行自动化测试。

（6）测试的复用性高。好的自动化测试机制可以很好的提高测试脚本的复用性，当测试软件发生改变，只需要对测试脚本进行少量的修改就可以进行回归测试，大大降低了测试脚本的维护开销。

（7）可以更快地将软件推向市场。一旦一系列测试已经被自动化，则可以比手工测试更快地重复执行，因此缩短了软件开发的时间。

（8）增加软件信任度。软件通过强有力地自动化测试后，可以大大减少软件中存在的错误，提高软件质量，发布时对其的信任度也就相应提高了。

2.4 软件自动化测试的局限

虽然，软件自动化测试有着显著的优越性，自动化测试正在逐步替代手工测试。但是，自动化测试并不是万能的，自动化测试所完成的功能也有一定的局限性，主要表现在以下几个方面：

（1）某些情况下不宜使用自动化测试。很少运行的测试；对不稳定软件的测试；通过人的感官来判断的测试等。这些情况下使用自动化测试将得不偿失。

（2）自动化测试对测试质量的依赖性极大。测试工具只能判断实际结果与期望结果之间的区别。因此，在自动测试中必需保证期望输出结果的正确性。也就是对测试本身的质量有很高的要求。

（3）自动化测试工具本身并不具有灵活性。工具是软件，只能严格地按照指

令执行任务。而手工测试中，测试者可以根据具体的情况随时调整测试计划，处理意外事故，用想象力和创造力改进测试。

（4）自动化测试不能提高有效性。有效性是指发现软件缺陷的有效性，即能否发现缺陷。在这方面自动化测试并不会比手工测试更加有效。

（5）自动化测试比手工测试发现的缺陷更少。因为软件测试的首次进行是由手工方式完成，自动化测试应是对程序的“再测试”，也就是回归测试。长期测试的经验统计表明。再次测试发现的缺陷不会多于首次测试发现的缺陷。

（6）自动化测试可能会制约软件开发。软件的改变对自动化测试有较大的影响，因此与手工测试比较，自动化测试表现得更“脆弱”。由于设置自动测试比手工测试开销大，并且需要进行维护，这些都可能限制了软件系统的修改或改进。由于经济原因，对自动测试影响较大的软件修改可能受到限制从而使软件开发受到影响。

通过以上分析可以看出自动化测试并不能完全取代手工测试。认识到软件自动化测试的这些局限性有助于我们更加合理的对软件测试进行自动化，通过自动化测试与手工测试的完美结合，充分发挥各自的优势，以达到更好的测试效果。

3 软件自动化测试框架比较

自动化测试在过去的20年中已经有了很大的发展。最初的测试工具只提供了简单的捕捉/回放功能：记录并播放键盘按键，然后捕捉和比较屏幕。这些测试方法虽然最容易应用，但是几乎不可能维护。捕捉/回放工具最终被功能和灵活性更强的测试脚本工具代替。

但是，脚本工具也有自己的问题。他们实现起来需要很强的开发技术和经验，并且也不能确定它们是一定可以维护的。更糟糕的是高度个性化的脚本工具技术，加上没有什么文档记录，最后的结果经常是重写包含成千上万行代码的脚本库，成本开销巨大。

因此，一种新的自动化测试产品出现了。它可以减少实现和维护的成本，使测试人员可以把精力集中在应用程序的测试用例设计上，而不是开发复杂的测试脚本。这些工具提供预先写好的测试框架，可以极大的减少，甚至消除学习和使用脚本语言的需要。这个测试产品就是自动化测试框架。在下面的几节内容里将讨论几种常用的自动化测试框架，并比较它们之间的利弊。

3.1 测试脚本模块化框架

测试脚本模块化框架是通过创建小的独立的脚本来代表被测试应用程序的模块和函数，然后用一种分层的方式将这些小脚本组成更大的测试，从而实现一个特定的测试用例[12,13]。

为了提高自动化测试套件的可维护性，测试脚本模块化框架应用了抽象和封装的原则。它采用了一个很著名的编程策略，就是在一个部件前面构建一个抽象层以掩藏应用程序其它的部件。它把应用程序从部件的修改中隔离出来，并规定了在应用程序设计中的模块性。这种测试框架在以后所提到的各种框架中是最容易精通和掌握的。

为了更直观地说明这种框架的应用，下面以Windows计算器程序为被测应用程序，测试其在标准视图和科学视图下的基本功能：加、减、乘、除。采用测试脚本模块化框架，分别将加、减、乘、除四个子功能创建成独立的测试脚本作为测试脚本层次的最低层，以下分别是用WinRunner录制的标准视图下和科学视图下的加法功能脚本。

```
# Script for standard-view Addition
# 计算器（标准视图）的加法脚本
set_window ("计算器", 3);
button_press ("7");
button_press ("+");
button_press ("8");
button_press ("=");
#建立图像检查点，验证结果是否真确
obj_check_bitmap("Edit", "Img1", 1);
button_press ("C");
```

```
# Script for scientific-view Addition
# 计算器（科学视图）的加法脚本
set_window ("计算器", 4);
button_press ("7");
button_press ("+");
button_press ("9");
button_press ("=");
#建立图像检查点，验证结果是否真确
obj_check_bitmap("C", "Img1", 1);
button_press ("C_2");
```

然后在层次结构的上一层分别用两个脚本表示标准视图和科学视图中要测试的加、减、乘、除操作。就像下面两个脚本中表现的一样，每个脚本调用了低层创建的各功能模块的独立脚本。

```
#Script for Standard View
# Test Add Functionality
call "c:\\testscript\\calculator\\calc_standard_add"();
# Test Subtract Functionality
call "c:\\testscript\\calculator\\calc_standard_subtract"();
# Test Divide Functionality
call "c:\\testscript\\calculator\\calc_standard_divide"();
# Test Multiply Functionality
call "c:\\testscript\\calculator\\calc_standard_multiply"();
```

```
#Script for scientific-view
# Test Add Functionality
call "c:\\testscript\\calculator\\calc_scientific_add"();
# Test Subtract Functionality
call "c:\\testscript\\calculator\\calc_scientific_subtract"();
# Test Divide Functionality
call "c:\\testscript\\calculator\\calc_scientific_divide"();
# Test Multiply Functionality
```

```
call "c:\\testscript\\calculator\\calc_scientific_multiply"();
```

最后，在脚本层次结构中最顶层的脚本应该是用来测试应用程序不同视图的测试用例。该脚本分别调用中层创建的标准视图脚本和科学视图脚本。

```
# Top level script - represents test case
# Test the Standard View
call "c:\\testscript\\calculator\\calc_standard"();
# Test the Scientific View
call "c:\\testscript\\calculator\\calc_scientific"();
```

从这个简单的例子中我们可以了解到这种框架是如何产生高度的模块化，并且增加测试套件的全面的可维护性。如果以后计算器上的某一个操作键发生了变化，我们只需要改变低层使用到这个操作键的脚本，而对上层调用该操作键进行测试的脚本无需作任何改动。

3.2 测试库构架框架

测试库构架框架和测试脚本模块化框架非常相似，有着同样的优势。但是它把被测应用程序分成过程和函数，而不是脚本。这种框架要求创建库文件来代表被测应用程序模块、零件或函数，然后这些库文件被测试用例脚本直接调用[12,13]。下面仍以上述计算器测试用例为例说明本框架的使用。采用 SQABasic 的库文件，创建包含计算器加、减、乘、除操作的函数，并在头文件中声明该库文件。以下是头文件（header file.sbh）和库文件（library file.sbl）。

```
'Header file
Declare Sub StandardViewFunction BasicLib "Functions Library" (OperandOne As Integer, _
OperandTwo As Integer, _ Operation As String)

'Library Source File
Sub StandardViewFunction(OperandOne As Integer, _ OperandTwo As Integer, _ Operation As
                         String)
  'Click on first operand
  Select Case OperandOne
      Case 0   PushButton Click, "ObjectIndex=8"
      Case 1   PushButton Click, "ObjectIndex=7"
      Case 2   PushButton Click, "ObjectIndex=11"
      Case 3   PushButton Click, "ObjectIndex=15"
      Case 4   PushButton Click, "ObjectIndex=6"
```

```
        Case 5   PushButton Click, "ObjectIndex=10"
        Case 6   PushButton Click, "ObjectIndex=14"
        Case 7   PushButton Click, "ObjectIndex=5"
        Case 8   PushButton Click, "ObjectIndex=9"
        Case 9   PushButton Click, "ObjectIndex=13"
    End Select
    'Click on operator
    Select Case Operation
        Case "+"   PushButton Click, "ObjectIndex=20"
        Case "-"   PushButton Click, "ObjectIndex=19"
        Case "*"   PushButton Click, "ObjectIndex=18"
        Case "/"   PushButton Click, "ObjectIndex=17"
    End Select
    'Click on second operand
    Select Case OperandTwo
        Case 0   PushButton Click, "ObjectIndex=8"
        Case 1   PushButton Click, "ObjectIndex=7"
        Case 2   PushButton Click, "ObjectIndex=11"
        Case 3   PushButton Click, "ObjectIndex=15"
        Case 4   PushButton Click, "ObjectIndex=6"
        Case 5   PushButton Click, "ObjectIndex=10"
        Case 6   PushButton Click, "ObjectIndex=14"
        Case 7   PushButton Click, "ObjectIndex=5"
        Case 8   PushButton Click, "ObjectIndex=9"
        Case 9   PushButton Click, "ObjectIndex=13"
    End Select
    PushButton Click, "ObjectIndex=21"
End Sub
```

利用上述头文件和库文件可以产生如下的测试用例脚本：

```
Test Library Acrhitecture Framework
'Test Case Script
'$Include "Functions Library.sbh"
Sub Main
    'Test the Standard View
    Window SetContext, "Caption=Calculator", ""
    'Test Add Functionality
    StandardViewFunction 3,4, "+"
    Result = LabelVP(CompareProperties, "Text=7.", "VP=Add")
    'Test Subtract Functionality
    StandardViewFunction 3,2, "-"
    Result = LabelVP(CompareProperties, "Text=1.", "VP=Sub")
     'Test Divide Functionality
    StandardViewFunction 4,2, "/"
```

```
    Result = LabelVP(CompareProperties, "Text=2.", "VP=Div")
    'Test Multiply Functionality
    StandardViewFunction 5,7, "*"
    Result = LabelVP(CompareProperties, "Text=35.", "VP=Mult")
End Sub
```

从这个例子中，我们能够看到这种框架也产生了高度的模块化，同样增加了测试套件的全面可维护性。就像在测试脚本模块化框架里一样，如果计算器中的某个操作键发生变化，我们所要做的只是修改库文件里对应操作键的信息，这样同时也更新了所有调用该操作键的脚本。

上述两种测试框架虽然使用模块化的方法，提高了测试套件的维护性，但是由于测试数据都是写在脚本中的，一个测试脚本只能执行一个测试用例，当软件规模很大时，往往需要大量的脚本才能实现自动化测试，增加了测试脚本的维护开销。因此，人们采用数据驱动脚本技术，实现了数据驱动测试框架，采用这种测试框架大大减少了测试脚本的数量。

3.3 数据驱动测试框架

将数据驱动脚本技术运用到自动化测试框架中就形成了数据驱动测试框架。这种框架从某个数据文件(例如 ODBC 源文件、Excel 文件、.CSV 文件、ADO 对象文件等)中读取输入、输出的测试数据，然后通过变量传入事先录制好的或手工编写的测试脚本中。其中，这些变量被用作传递(输入/输出)用来验证应用程序的测试数据。在这个过程当中，数据文件的读取、测试状态和所有测试信息都被编写进测试脚本里；测试数据只包含在数据文件中，而不是脚本里，测试脚本只是一个“驱动”，或者说是一个传送数据的机制[14]。

数据驱动的自动化测试利用相同的测试过程测试不同的输入、输出组合。它将测试输入和预期输出组织为表，表中的一行对应一个测试。然后创建一个从表中逐行读入的自动化测试过程，执行每个输入步骤，并检验预期结果。这些表可以记录在如 Excel 文件(电子表格)，. CSV 文件(易输出的以隔断文本文件格式存放的数据文件)等方便录入测试数据的数据文件中。当把数据驱动测试过程放在一起后，就可以反复使用该过程来执行新测试。这种手段对于有很多不同数据选项的应用来说最有效。

下面以 WinRunner 自带的 Flight Reservation 范例程序作为被测应用程序，利用数据驱动测试框架测试不同的订单中飞机票价总额计算是否正确。在测试脚本中将订单号设为变量，采用 Excel 文件作为数据文件，在其中定义不同的订单号来

实现数据驱动的自动化测试。测试脚本如下：

```
# 设置数据文件名称
table = "datadriver.xls";
# 获取数据文件记录，判断数据文件是否为空
rc = ddt_open(table, DDT_MODE_READ);
if (rc!= E_OK && rc != E_FILE_OPEN)
    pause("Cannot open table.");
ddt_get_row_count(table,table_RowCount);
# 依次读取数据文件中每一行的记录，取代脚本中对应的变量执行测试
for(table_Row = 1; table_Row <= table_RowCount; table_Row ++)
{
   ddt_set_row(table,table_Row);
    # Flight Reservation
    set_window ("Flight Reservation", 2);
    menu_select_item ("File;Open Order...");

    # Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    # 用数据文件中的订单号作为文本框的输入
    edit_set ("Edit_1", ddt_val(table,"Order_Num"));
    button_press ("OK");

    # Flight Reservation
    set_window ("Flight Reservation", 2);
    menu_select_item ("File;Fax Order...");

    # Fax Order No. 3
    set_window ("Fax Order No. 3", 3);
    edit_get_text("# Tickets:",tickets);
    edit_get_text("Ticket Price:",price);
    edit_get_text("Total:",total);
    if(tickets*price == total)
        tl_step("total",0,"Correct. "tickets" tickets at $"price"cost $"total".");
    else
        tl_step("total",1,"Error."tickets" tickets at $"price"does not equal $"total".");
        button_press ("Cancel");
}
ddt_close(table);
```

通过上面的例子我们可以看出，利用数据驱动测试框架有以下优点：（1）在应用程序开发的同时就可以同步建立测试脚本，而且当应用程序功能变动时，只需要修改业务功能部分的脚本即可；（2）利用模型化的设计，避免重复的脚本，

减少建立或维护脚本的成本；（3）测试输入数据，验证数据和预期的测试结果与脚本分开，存放在另外的数据文件里，利于测试人员修改和维护；（4）透过判断功能回传值是“True”或“False”，可作错误处理，增加了测试脚本的健壮性；（5）支持非程序设计测试员。自动化测试开发人员创建数据驱动的测试过程，测试人员创建测试数据；（6）采用测试过程来收集测试结果，并在输入数据的语境中表示测试结果，这样可以简化手工结果分析。

然而数据驱动测试框架也存在以下缺点：（1）测试人员对自动化测试工具里的脚本语言必须非常精通；（2）每个脚本都会对应多个数据文件，这些数据文件需要根据脚本的功能类别存放在各自的目录中，增加了使用的复杂性；（3）测试人员除了需要根据具体测试数据维护相应的测试计划，还要将这些数据写入不同需要的数据文件中；（4）在编辑数据文件时，必须注意测试脚本所需要的传输格式，否则会在处理脚本时产生错误。

如果有专门的技术人员对其进行维护的话，依赖于数据驱动脚本的自动化测试框架实现起来是最容易，也是最快的。但是，最难的也是维护工作，而且还需要保持这种数据驱动的模式，这样，经过长时间的维持也会导致失败。

3.4 关键字驱动测试框架

到目前为止，比较理想的自动化测试框架是“关键字”驱动的自动化测试框架，有时候也称为表驱动自动化测试框架，它是对数据驱动自动化测试的有效改进和补充。这个框架需要开发数据表和关键字。这些数据表和关键字独立于执行它们的测试自动化工具，并可以用来“驱动”待测应用程序和数据的测试脚本代码，使自动化测试框架独立于应用程序。关键字驱动测试看上去与手工测试用例很类似。在一个关键字驱动测试中，把待测应用程序的功能和每个测试用例的执行步骤一起写到一个表中[13]。

下面仍以测试计算器加、减、乘、除操作的用例为例，说明这种测试框架是如何运作的。

表 3.1 表示了测试计算器加法功能所需的电子数据表的形式，该表可以保存成一个“tab 分隔”文件。在实际运行测试过程中，这个文件被控制器脚本读取并处理。当遇到列 1 里的关键字时，控制器脚本就会调用与该关键字相关联的实用脚本。然后控制器脚本用列 2 和列 3 里的数据创建一个列表，它包含要进行的具体动作、要输入或验证的数据等，并把该列表作为输入参数传递给实用脚本。实用脚本调用用户定义的函数来执行具体的动作或使用测试工具自带的函数来完成。这个动作会继续执行直到到达“列表结束”状态或者出现致命的错误。然后实用

表 3.1 含有关键字的计算器测试数据表
Table 3.1 Test Data Table Contained Keywords

列 1 关键字	列 2 字段/界面名称	列 3 输入/验证数据	列 4 注释	列 5 通过/失败
Start-Test（启动测试）	计算器	主菜单	验证起始点	
Select（选择）	查看	标准型	选择视图类型	
Action（动作）	Button-press	7	按下第一个操作数	
Action（动作）	Button-press	+	按下操作类型	
Action（动作）	Button-press	8	按下第二个操作数	
Action（动作）	Button-press	=	按下等于键	
Verify-Result	结果文本框	15.	验证结果是否正确	
Action（动作）	Button-press	C	按下 clear 键，结果归零	

脚本将控制权返回给控制器脚本，控制器脚本会继续处理 tab 分隔文件直到到达“文件结束”状态或收到来自使用脚本的“致命错误”返回码。为了处理大量的脚本，控制器脚本被驱动脚本调用。驱动脚本包含所有要运行的测试用例的名字，并为每个测试用例调用控制器脚本，向它传递每个测试用例的名字（保存的 tab 分隔文件）[14]。

这种体系结构非常灵活，因为驱动脚本不局限于调用控制器脚本来处理关键字驱动测试用例。它也能调用业务功能类型脚本或者记录脚本。这种体系结构支持所有的自动化测试方法。

关键字驱动自动化测试框架是一种截然不同的思想，它把传统测试脚本中变化的与不变的东西进行了分离，这种分离使得分工更明确，并且避免了它们相互之间的影响。使用关键字驱动自动化的思想，测试人员可以轻易地实现对测试的修改和定制，以及对测试脚本的维护，提高了测试用例的重用性[15]。

关键字驱动自动化测试框架的开发和实现与传统的测试框架相比可能是困难的，而且是最耗时的，因为，软件测试人员正在努力完全地将测试和自动化工具以及应用程序本身的变化隔离开来。为了实现这个目标，最重要是要增强自动化工具所提供的组件的功能，例如，错误纠正、避免数据同步。但是这样的投资是一次性的，一旦开发结束投入使用，关键字驱动自动化测试框架所带来的效益是巨大的，是自动化测试框架中最容易维护和使用的，不仅维护简单，而且可以反复应用于各种应用，并能长期发挥它的作用。

4 软件自动化测试混合框架的设计

通过上一章对现有几种测试框架的比较，发现最为成功的自动化测试框架是将上述各种测试框架中优秀的设计思想组合起来，建立一个混合的自动化测试框架。这种测试框架既要采用测试脚本模块化的思想，将测试脚本进行分层次、嵌套管理；又要运用测试库构架框架中的库文件实现被测应用程序的功能、过程或方法，在测试脚本文件中直接调用这些库文件；此外，还可以采用数据驱动脚本技术，实现测试数据与测试脚本的分离；同时运用关键字驱动测试技术将测试描述与测试实现分离开来。因此，想达到比较理想的自动化测试效果，最好的办法是以模块化、数据驱动脚本作为输入，通过关键字驱动测试框架调用应用程序库文件进行处理得到测试结果，完成自动化测试过程。

本章针对前面讨论的软件自动化测试技术，结合各种自动化测试框架的设计思想，运用 WinRunner 提供的测试工具箱实现软件自动化测试混合框架。下面具体讨论这种测试框架的结构以及使用该测试框架进行自动化测试的流程。

4.1 SAFS 与 WinRunner 工具箱

软件测试是软件开发生命周期中不可或缺的一项活动，同时它也是一项既复杂又费时，开销又大的活动。因此提高测试效率，减少测试开销成为亟待测试人员解决的问题。

为此 Carl Nagle 为 SAS 研究所开发了一个基于关键字驱动的软件自动化测试框架支持工具 SAFS（Software Automation Framework Support）[16,17]。该工具提供的组件对软件功能测试自动化有利，它不仅适用于采用关键字驱动引擎进行测试的工程，同时也适用于其他自动化测试项目。这个工具最主要的目标是实现一个可在测试工具和测试平台间移植的通用的关键字驱动引擎。设计者在设计的过程中本着使被测应用程序或者测试工具的改变对自动化测试影响最小的思想，实现了测试与自动化工具的分离，可以将测试轻松地从一个测试工具移植到另一个测试工具，甚至在同一个测试中使用多种测试工具。

WinRunner 是目前最常用的软件自动化测试工具之一，用于检验应用程序功能是否正确。通过自动捕获、检测和重放用户的交互操作，WinRunner 能够发现系统缺陷，并且确保那些跨越多个应用程序和数据库的业务流程在初次发布就能避免故障的出现，保持其长期稳定地运行。

Keith Zambeilch 根据“测试计划驱动”自动化测试方法开发了一个用于 WinRunner 的测试工具箱，该工具箱基本上由用户自定义函数和能使测试者更好地控制 WinRunner 测试工具的实用脚本组成。不管特殊测试需求规定了什么测试方法或方式，这些函数和实用工具都可以使用户容易使用这个工具箱，而不用受到测试工具固有的限制[14]。在测试工具箱中有一个 WRSAFS 工具它是基于 SAFS 设计的自动化测试引擎，本文将利用此工具实现软件自动化测试混合框架。

4.2 软件自动化测试混合框架的组成

本节将着重讨论软件自动化测试混合框架的组成。根据现有各种自动化测试框架的优点，基于 WRSAFS 设计的软件自动化测试混合框架由模块化测试列表，核心数据驱动引擎，数据文件、关键字词库、组件函数库、支持库和应用映射文件组成，具体结构如图 4.1 所示：

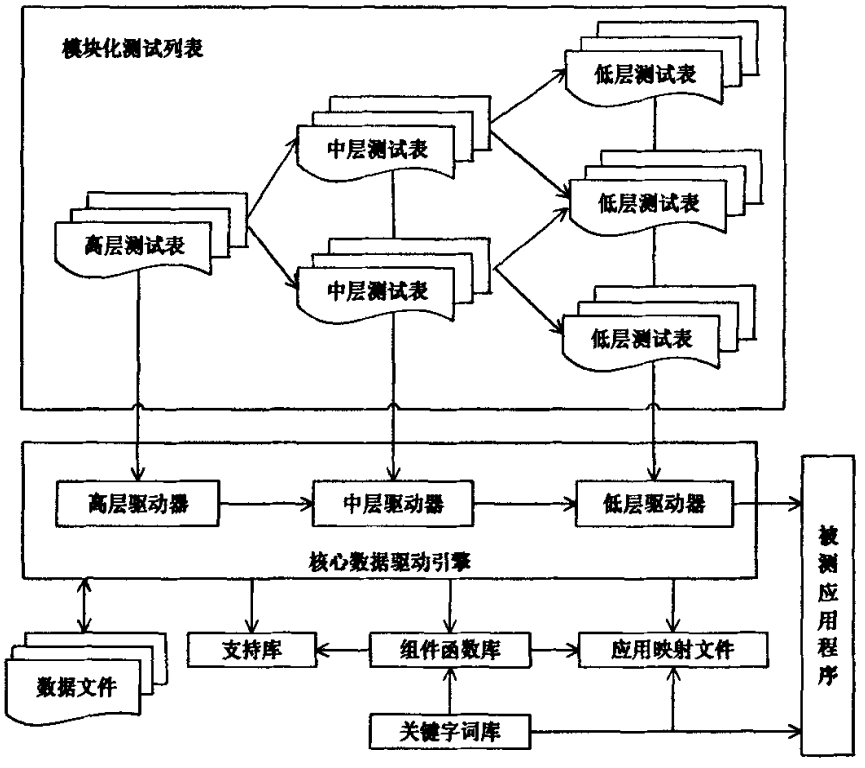

图 4.1 软件自动化测试混合框架结构图

Figure 4.1 Structure of Hybrid Automated Test Framework

模块化测试列表由高层测试表、中层测试表和低层测试表三部分组成，用于分层描述测试用例。核心数据驱动引擎由高层驱动器、中层驱动器和低层驱动器组成，分别与模块化测试列表中的三层测试表对应，用于驱动整个测试用例的执行。组件函数库是实现用户操作界面对象的指令函数的集合，低层驱动器通过调用组件函数库中的具体函数来执行测试用例中指定地对界面对象的具体操作。应用映射文件是模块化测试列表中的逻辑对象与被测应用程序中的物理对象之间联系的桥梁，自动化测试工具通过应用映射文件中对逻辑对象的物理描述来识别实际操作的界面对象。数据文件用于存储测试所需的数据，实现测试数据与测试用例的分离。支持库为自动化测试框架提供基础支持，用于提高自动化测试框架的处理能力。下面具体阐述软件自动化测试混合框架各组成部分的详细设计。

4.2.1 应用映射文件

为了实现界面元素名与测试内部对象名的分离，在进行测试设计之前，测试人员首先对被测应用中的每一个对象定义一套命名规范，并利用映射文件把这些名字和自动化测试工具识别的对象名联系起来，使自动化测试工具能准确地定位和操纵对象。测试脚本使用逻辑对象名代表物理对象，测试时测试工具通过应用映射文件查找该逻辑对象对应的物理描述，通过该物理描述定位实际操作的物理对象。例如在表 4.1 所示的按钮对象映射表中，SaveButton 只是逻辑对象名，物理描述包括逻辑对象的物理类型信息，以及使自动化测试工具能够识别该逻辑对象所需的属性。

表 4.1 按钮对象映射表
Table 4.1 Mappfing Table of Button Object

逻辑名	物理描述
SaveButton	Type=PushButton；Name=Yes；VisualText=Yes

在上例中逻辑对象 SaveBotton 对应的物理类型为“按钮”，VisualText 属性表示在应用程序界面上该按钮上的字符为“Yes”，Name 属性表示在应用程序代码里使用“Yes”代表该按钮，通过这两个属性即可惟一标识该按钮。

这样在维护测试脚本时，我们只需要进行单点维护。当应用程序中的物理对象发生变化时，只需修改物理描述部分对应的信息即可，而不用修改所有使用该物理对象的脚本。例如当 SaveButton 所代表的按钮上的字符改为“是”时，只需修改 VisualText 属性值为“是”，对测试脚本不需要做任何修改，自动化测试工具

仍然可以识别该逻辑对象。

（1）应用映射文件的结构

应用映射文件中的所有元素都是按照标准配置文件的格式组织的，文件的后缀名为.map。应用映射文件由部件名称，部件里包含的属性名和对应的属性值组成。其中部件名称有两类：Application-Constants 和 Windows。ApplicationConstants 是应用映射文件的保留部件名，其下的属性元素键值对用于指定被测应用程序的路径，以及测试所需的数据。Windows 部件名用于指定应用程序窗口的名称，紧跟其后的属性元素键值对定义了该窗口里所有组件的逻辑名和对应的物理描述。

下面以一个应用程序映射文件的一部分为例，说明应用程序映射文件的格式。

```
[ApplicationConstants]
FlightApp="D:\ProgramFiles\MercuryInteractive\WinRunner\samples\vb\app\
fltvb41a.exe"
FilghtPath="D:\ProgramFiles\MercuryInteractive\WinRunner\samples\vb\app"
UserName="jdoe"
Password="mercury"
[Login]
Login={class: "window", label: "Login", MSW_class: "ThunderRT6FormDC"}
Help={class: "push_button", vb_name: "cmdHelp"}
Cancel={class: "push_button", vb_name: "cmdCancel"}
OK={class: "push_button", vb_name: "cmdOK"}
AgentsName={class: "edit", vb_name: "txtAgentsName"}
Password={class: "edit", vb_name: "txtPassword"}
```

上例中 ApplicationConstants 部件后的 FlightApp 和 FilghtPath 属性用于定义应用程序的执行路径，便于自动化测试框架启动应用程序进行测试。用户可根据具体的应用程序定义这类属性。属性元素 UserName 和 Password 对应的值即指定了用于登录应用程序的用户名和密码。[Login]为 Windows 部件名，表示应用程序有一个名为 Login 的登录窗口。其后的属性元素定义了登录窗口以及该窗口里的所有组件的逻辑对象名和物理描述。根据应用程序中窗口的数目，在一个应用映射文件中可以有多个 Windows 部件。

（2）设计应用映射文件

应用映射文件是软件自动化测试混合框架中实现测试组件与被测应用程序隔离的重要部件。WinRunner 可以通过 GUI Map Editor[18]学习应用程序中的 GUI 对象，从而建立扩展名为.gui 的应用映射文件。WinRunner 会透过 GUI 对象的属性（Physical properties），如 class、label、width、height、handle 和 enabled 等，来识别 GUI 对象。在学习的过程中 WinRunner 只会记录最少但可组合成唯一的属性来识别 GUI 对象。学习完毕，WinRunner 会给每个识别到的 GUI 对象分配一个逻辑

名称。这些逻辑名称用在模块化测试列表中，执行测试时 WinRunner 利用这些逻辑名到应用映射文件找该 GUI 对象的物理描述，从而识别该 GUI 对象。

图 4.2 为应用 WinRunner 的 GUI Map Editor 学习得到的窗口组件的信息。保存之后会得到后缀名为.gui 的文件。

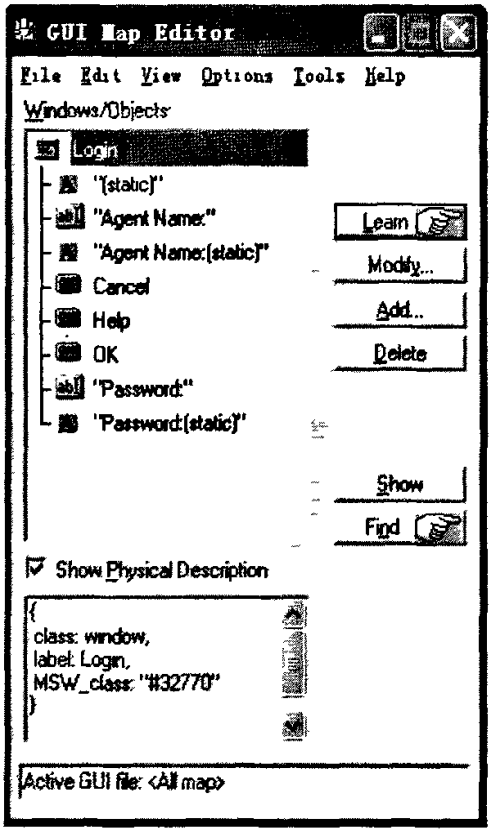

图 4.2 GUI Map Editor

Figure 4.2 GUI Map Editor

下面列出的即是上例中利用 GUI Map Editor 生成的.gui 映射文件的格式：

```
Login:
{ class: window,
  label: Login,
  MSW_class: ThunderRT6FormDC
}
Login."Agent Name:";
{ class: edit,
  vb_name: txtAgentsName
}
```

我们可以看出利用 GUI Map Editor 生成的.gui 文件与上一小节介绍的后缀名为.map 的应用映射文件的格式并不一致，不能直接被自动化测试框架运用，但可以通过它建立符合自动化测试框架所需格式的应用映射文件。只需要将组件的逻辑名和对应的大括号括起的物理描述的表示形式改成由等号连接的键值对形式，并把对应窗口下的组件的键值对放到 Windows 组件名下即可。此外还需要添加 ApplicationConstants 部件名及其下所包含的属性元素，指定被测应用程序执行的路径。

应用映射文件是软件自动化测试混合框架识别逻辑对象名的工具，要使自动化测试工具 WinRunner 也能够识别逻辑对象名还需要在执行测试的第一步将.gui 文件装载入 WinRunner 的“GUI Map”编辑器。WinRunner 驱动命令函数中的 SetApplicationMap 函数将.gui 文件的文件名和路径名作为输入参数来完成这一步骤，它把当前 GUI Map 编辑器中的 GUI Map 卸载，然后装入应用程序特定的 GUI Map。

4.2.2 模块化测试列表

模块化测试列表是软件自动化测试混合框架中的重要组成部分，它是测试用例的描述，用于引导测试的执行。在设计过程中，采用模块化脚本测试框架中的模块化思想，将测试列表分为低层测试表、中层测试表和高层测试表。

低层测试表指定测试的每一步指令的细节，这些指令都是直接作用在应用程序界面对象上的，是无法再细分的指令。低层测试表定义了实现相应动作的具体步骤，所以低层测试表又叫做步骤表。低层测试表中使用了应用映射表中定义的逻辑对象名，和由组件函数定义的低层关键字词库。

中层测试表并不包含低层面向应用程序对象的指令，而是把低层测试表组装起来执行更多有用的任务。因此中层测试表也可以叫做组装测试表。同一个低层测试表可以用于多个中层测试表，所以我们应该开发尽可能少的低层测试表，然后把它们按照不同的目的组装起来，实现最大的重用性。

高层测试表把中层测试表组装起来，形成一个测试循环，每个循环是一个完整的测试用例，可以定制不同类型和数量的测试。因此高层测试表也可以叫做循环测试表。

（1）测试表格命名规范

测试表格是自动化测试过程中必需设计的一系列文件，它包括完成测试所需的指令。也就是说我们需要把测试用例转化成软件自动化测试混合框架所能识别

的测试表格。在软件自动化测试混合框架中，测试表格按照三个层级进行组织，即高层测试列表，中层测试列表和低层测试列表。这些测试列表组合起来构成一个测试用例，某些测试用例里的低层测试列表可以重用在其它测试用例中。采用这种方式组织测试表的最大优点就是提高了测试表格的重用性。

测试表的命名规则应该遵从于文件的命名规则，为了便于核心数据引擎识别，每种测试表要有对应的扩展名。高层测试表的扩展名为.CDD，这个表不仅包含调用中层数据驱动引擎进行处理的中层测试表格的名称，而且还包含执行其它高层驱动命令的函数。当高层测试表格中包含一个记录，例如“Perform Login Test”，这实际上是高层驱动引擎调用中层驱动引擎处理文件“Perform Login Test. STD”的过程（因为中层测试表格文件的扩展名为.STD）。

中层测试表格的默认扩展名为.STD。这个表不仅包括调用低层驱动引擎进行处理的低层测试表格的名称，而且还包含执行其它中层驱动命令的函数。当中层测试表包含一个记录，例如“Verify Login Name”，这实际上是中层驱动引擎调用低层驱动引擎处理文件“Verify Login Name. SDD”的过程（因为低层测试表文件的扩展名为.SDD）。

低层测试表格的默认扩展名为.SDD。这个表不仅调用组件函数和工具脚本执行具体的测试，而且还包含执行其它低层驱动引擎命令的函数。

当使用微软的 Excel 或其它数据表格工具时，最好遵循上边的命名约定，每个文件工作簿的名字都包含文件扩展名字。当引擎处理表格时，把每个工作簿的内容导出到和当前工作簿同名的文件中。

（2）测试表的结构

了解了测试表的命名规范，下面介绍测试表的结构。与一般的表格相同，测试表格也是由分隔符隔开的一列一列组成， 每一列叫做一个字段。为了不与数据中的字符混淆，需要将电子数据表格保存成一个“tab分隔”文件，即用tab分隔符来分隔每一个字段。每行包含的字段数不定，包括必选字段、可选字段和未使用字段。图4.3是用Excel创建的一个测试列表。

测试表中的一行称为一个测试记录。测试记录可以分为处理记录和忽略记录。若测试表的一行为空或是注释，则被视为忽略记录，核心驱动引擎不执行任何操作。若测试表的一行包含测试记录类型或执行某个动作的指令和参数，则被视为处理记录。

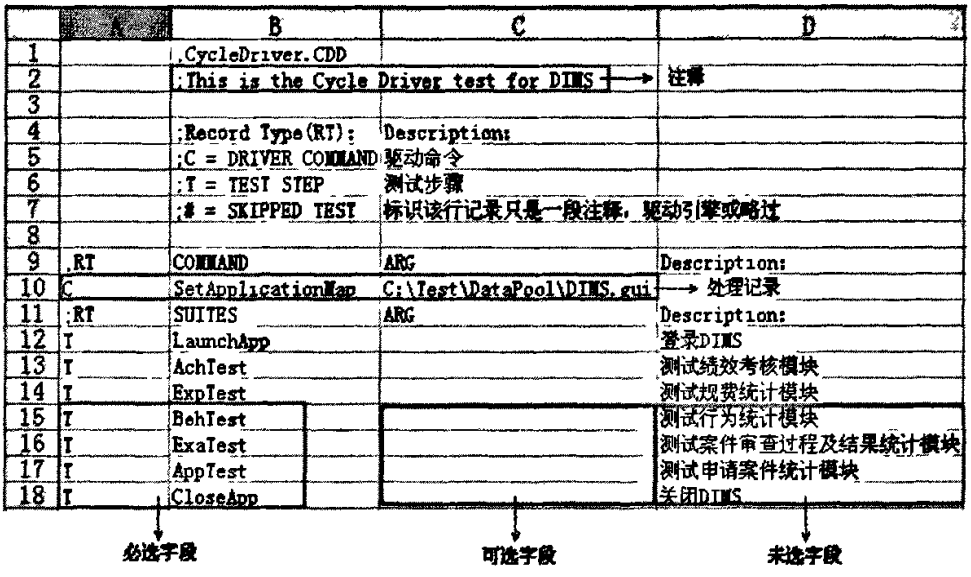

图 4.3 测试列表

Figure 4.3 Test List Table

必选字段用于存放处理记录的类型，以及该条记录需要执行的指令或测试表格的名称。可选字段用于存放处理记录所需的参数变量，可以在可选字段直接为参数变量赋值。如果希望一条测试记录可以执行多次，并且每次使用不同的数据，就可以通过数据驱动引擎的一段程序从数据文件中读取所需的测试数据为测试记录中的参数变量赋值。 在测试表格中若需要对某行记录进行注释，可将注释写在必选字段和可选字段后面的未使用字段，数据驱动引擎并不会处理这些字段，这些字段只用于增加测试表格的可读性。

在软件自动化测试混合框架中，模块化测试列表处理记录的第一个字段都是表示记录类型的字段。数据驱动引擎首先处理这个字段，通过该字段识别控制流。软件自动化测试混合框架中主要有以下三种记录类型，如表4.2所示：

表 4.2 记录类型

Table 4.2 Record Type

记录类型	描述
C	驱动命令
T	测试步骤
#	标识该行记录只是一段注释

如果记录类型是“C”，数据驱动引擎就会根据该记录第二个字段的函数名调用相关的命令执行具体的操作。例如下面的测试记录表示将决策支持系统的

DIMS.gui 文件加载到 WinRunner 的 GUI Map editor 中。

C	SetApplicationMap	C:\test\datapool\DIMS.gui

如果记录类型为“T”，则该记录第二个字段就要指明需要调用的测试表的名字。例如，若“T”出现在高层测试表的某条记录，则高层驱动器就会调用中层驱动器处理该记录第二个字段对应的中层测试表。

如果记录类型为“#”则表示该行仅是注释，数据驱动引擎就会跳过该行不做任何操作。

A．高层测试表结构

高层测试表（也叫做循环测试表）是测试列表中最抽象的一层，往往一个测试框架仅需要一个高层测试表定义所有将要执行的测试循环（即测试用例）。高层测试表的每一条测试记录即代表一个测试循环。高层测试表结构如表 4.3 所示：

表 4.3 高层测试表结构

Table 4.3 High-Level Test Table Structure

列序号	列名	描述
1	RT	标识测试记录类型
2	SUITES/COMMAND	中层测试表名/执行命令
>2	ARG	传递给中层测试表或执行指令的参数

如前面所讲高层测试表的第一字段用于表示该记录的类型，根据此字段高层驱动器执行相应的控制流程。如果记录类型是“T”，控制流就会转移到中层驱动器。高层测试表调用一系列中层测试表完成一个测试循环。如果测试记录类型是“C”，数据驱动引擎就会转去调用已经定义好的驱动命令执行相应的操作。表 4.4 是一个“验证用户登录许可”的高层测试表。

表 4.4 “验证用户登录许可”的高层测试表

Table 4.4 High-Level Test Table of Verify Authentication Function

高层测试表：VerifyAuthenticationFunction.CDD			
RT	SUITES/COMMAND	ARG	DESCRIPTION
C	SetApplicationMap	C:\test\datapool\DIMS.gui	加载应用映射文件
T	VerifyInvalidLogin		验证错误的用户登录
T	VerifyBlandLogin		验证空白的用户登录
T	VerifyValidLogin		验证合法的用户登录

B. 中层测试表结构

中层测试表与高层测试表一样也属于数据模块化测试列表的抽象层，只用来描述测试事例，一个测试框架中可以有多个中层测试表被不同的测试循环调用。与高层测试表不同，当测试记录类型为“T”时，测试表格第二个字段列出低层测试表的名字，控制流转向低层驱动器处理相关的低层测试表。中层测试表结构如表 4.5 所示：

表 4.5 中层测试表的结构

Table 4.5 Intermediate-Level Test Table Structure

列序号	列名	描述
1	RT	标识测试记录类型
2	STEPS/COMMAND	低层测试表名/执行命令
>2	ARG	传递给低层测试表或执行命令的参数

前面提到的“验证用户登录许可”的高层测试表中使用了三个中层测试表分别为：VerifyInvalidLogin.STD、VerifyBlandLogin.STD和VerifyValidLogin.STD。根据中层测试表的结构，这三个中层测试表的形式如表4.6至表4.8所示。

表4.6 “验证非法登录”的中层测试表

Table 4.6 Intermediate-Level Test Table of Verify Invalid Login

中层测试表：VerifyInvalidLogin.STD			
RT	SUITES/COMMAND	ARG1	ARG2
T	LaunchSite		
T	Login	UserName= “BadUser”	Password= “dims”
T	VerifyLogin		
T	Login	UserName= “dims”	Password= “BadPassword”
T	VerifyLogin		
T	Login	UserName= “BadUser”	Password= “BadPassword”
T	VerifyLogin		
T	ExitLogin		

表4.7 “验证登录用户为空”的中层测试表
Table 4.7 Intermediate-Level Test Table of Verify Bland Login

中层测试表：VerifyBlandLogin.STD			
RT	SUITES/COMMAND	ARG1	ARG2
T	LaunchSite		
T	Login	UserName=“”	Password=“”
T	VerifyLogin		
T	ExitLogin		

表4.8 “验证合法登录”的中层测试表
Table 4.8 Intermediate-Level Test Table of Verify Valid Login

中层测试表：VerifyValidLogin.STD			
RT	SUITES/COMMAND	ARG1	ARG2
T	LaunchSite		
T	Login	UserName= “dims”	Password= “dims”
T	VerifyLogin		
T	ExitLogin		

（3） 低层测试表结构

直到低层测试表我们才对具体的应用程序组件执行相应的操作，因此低层测试表会详细描述实现测试用例的每一步操作，那么低层测试表的记录类型字段就均为“T”。低层驱动器根据低层测试表描述的测试动作，调用相应的组件函数或子程序执行具体的操作。低层测试表也有多个，并被多个中层测试表调用。因此在设计的过程中同样要考虑低层测试表的重用性。低层测试表的结构如表 4.9 所示：

表 4.9 低层测试表的结构
Table 4.9 Low-Level Test Table Structure

列序号	列名	描述
1	RT	标识测试记录类型
2	Window	执行动作的窗口名
3	Component	被操作的组件名
4	Action	执行操作所需的命令
>4	ARG	组件函数所需的参数

由于低层测试表涉及到对应用程序组件的具体操作，在设计的过程中要参考应用映射文件与组件函数库，若发现组件函数库中缺少必须的关键字、行为或命令时，脚本开发人员就要对相应的组件函数模块、驱动器命令和支持库进行扩展。同样以“验证用户登录许可”的测试用例为例，该测试用例用到的低层测试表如表 4.10 至表 4.13 所示。

表4.10 “打开网页”操作低层测试表
Table 4.10 Low-Level Test Table of Launch Site

低层测试表：LaunchSite.SDD				
RT	Window	Component	Action	ARG
T	LaunchBrowser	Address	InputText	dimsweb/login.jsp
T	LaunchBrowser	Address	Enter	

表4.11 “登录”操作低层测试表
Table 4.11 Low-Level Test Table of Login

低层测试表：Login.SDD				
RT	Window	Component	Action	ARG
T	LoginPage	UserIDField	InputText	“dims”
T	LoginPage	PasswordField	InputText	“dims”
T	LoginPage	SubmitButton	Click	

表4.12 “验证登录”操作低层测试表
Table 4.12 Low-Level Test Table of Verify Login

低层测试表：VerifyLogin.SDD				
RT	Window	Component	Action	ARG
T	LoginPage	UserIDTextBox	VerifyValue	“dims”
T	LoginPage	PasswordTextBox	VerifyValue	“dims”

表4.13 “退出登录”操作低层测试表
Table 4.13 Low-Level Test Table of Exit Login

低层测试表：ExitLogin.SDD				
RT	Window	Component	Action	ARG
T	LoginPage	ExitButton	Click	

4.2.3 组件函数库和关键字词库

组件函数是实现用户对界面上的组件进行操作的指令函数，一个类型的组件对应一个组件函数模块。例如对于一个文本框组件，测试人员可能会对它执行多种操作：输入文本、验证文本框的值、验证文本框的某些属性等，实现这些操作行为的函数就被放在文本框的组件函数模块中。一般的测试工具都提供了这样的函数，而我们可以在其中加入额外的代码来检测错误、纠正错误和帮助同步，这类代码是实现无人职守的自动化测试所必需的。

组件函数相当于在应用和自动化工具之间提供了一个隔离层，如果没有这个隔离层，自动化测试工具或被测应用程序的改变或提高就会影响已有的脚本，但是有了组件函数，我们可以增加一段修补代码来适应这些变化，转移对测试的破坏。

WinRunner用自己的脚本语言（TSL）[19] 编写了大量应用程序常用组件的组件函数，在设计软件自动化测试混合框架的组件函数库时，只需要将这些组件函数按照组件的类型进行封装，构成该组件对应的组件函数模块，然后再将所有组件函数模块组合起来即构成组件函数库。

封装的过程实际是将操作某一类型组件的所有组件函数作为子函数写在一个TSL脚本中，然后在脚本最后添加一个main函数，该函数利用switch-case语句，根据组件函数关键字调用对应的组件函数执行相应的操作。例如对于文本框组件，WinRunner提供的组件函数有SetTextValue，SetTextCharacters，GetTextValue，则文本框组件函数模块的main函数如下所示：

```
function EditMain()
{
    auto mode;
    mode = 0;
    #do stuff in here
    switch(toupper(StepDriverState["testcommand"]))
    {
    case SET_TEXT_VALUE_COMMAND:
        mode = 0;
        SetTextValue(mode);
        break;
    case SET_UNVERIFIED_TEXT_VALUE:
        mode = 1;
        SetTextValue(mode);
        break;
    case SET_TEXT_CHARACTERS:
```

```
            mode = 0;
            SetTextCharacters(mode);
            break;
        case SET_UNVERIFIED_TEXT_CHARACTERS:
            mode = 1;
            SetTextCharacters(mode);
            break;
        case GET_TEXT_VALUE:
            GetTextValue();
            break;
        default:
            GenericObjectsMain();
        }
    }
```

其中case语句后面的条件即为组件函数的关键字。从上面的代码可以看出，有些组件函数带有参数，相同的组件函数，当传递的参数不同时会对组件进行不同的操作，因此在为组件函数定义关键字时，还应考虑参数的情况，同一个组件函数可能对应多个关键字。

组件函数关键字和它们所需的参数构成软件自动化测试混合框架底层的词库，有了底层词库和应用映射文件，就可以在此基础上建立模块化测试列表中的低层测试表。低层测试表中的Action字段即为组件函数关键字，通过该字段自动化测试工具就能知道调用组件的哪个指令执行具体的操作。

4.2.4 核心数据驱动引擎

与模块化测试列表相对应的分别是低层驱动器、中层驱动器和高层驱动器，它们构成的核心驱动引擎是软件自动化测试混合框架的另一重要组成部分。高层驱动器处理高层测试表的每条记录，如果遇到记录类型为T的处理记录，就激发中层驱动器处理中层测试表的每条记录，在处理中层测试表的记录时，如果遇到记录类型为T的处理记录，就激发低层驱动器处理低层测试表中的每条记录，并根据记录中的关键字调用关键字词库中低层指令所对应的组件函数来完成相应操作。

数据驱动引擎是执行自动化测试所需的控制脚本，控制测试流的转换与执行，它由一系列软件模块组成，这些软件模块由用 Mercury Test Scripting Language（TSL）编写的控制脚本文件组成。WinRunner 的 WRSAFS 工具中已经包含了我们实现数据驱动引擎所需要的软件模块。表 4.14 列出了数据驱动引擎所包含的软件模块。

表 4.14 数据驱动引擎的组成
Table 4.14 Compoment of Data Driver Engine

数据驱动引擎名称	模块名称
高层驱动器	CycleDriver、CycleDriverSTACK
中层驱动器	SuiteDriver、SuiteDriverSTACK
低层驱动器	StepDriver、StepDriverSTACK
其他引擎部件	WIN32.SBH、CustomDriverCommands、CustomLogUtilities、ustomRecordTypes、DDDriverCommands、SAFSUtilities、DDDriverCounterCommands、DDDriverFileCommands、DDDriverFlowCommands、DDDriverLogCommands、DDDriverStringCommands、DDEngine.SBH、StartCycleTest DDGUIUtilities、DDUtilities、DDVariableStore、STAFUtilities

WRSAFS中的这些软件模块只是对WinRunner提供的TSL函数进行了封装，增加了一些额外的逻辑。这些封装后的函数在模块化测试列表中被调用来执行期望的操作。

4.2.5 数据文件

为了实现同一个测试用例采用不同的测试数据进行测试，我们在软件自动化测试混合框架中引入数据文件机制。数据文件继承了数据驱动测试框架的思想，将测试所需的数据存放在指定的文件中，模块化测试列表中仅用参数变量名的形式表示。在测试执行过程中核心数据驱动引擎会根据变量名到数据文件中查找该变量代表的实际数据进行替换。采用这种方法可以使同样的模块化测试列表实现不同的测试用例，减少了测试列表的数量，降低了维护的开销。为了使核心数据驱动引擎能够正确识别变量名代表的实际数据，数据文件的格式如表4.15所示：

表4.15 数据文件结构
Table 4.15 Structure of Data File

列编号	列名	描述
1	VarName	标识变量名
2	DataNum	标识测试数据个数
>2	DataN	指定变量代表的实际数值(N 从 1 开始，表示测试数据的个数)

当测试表中的某条记录里有参数变量，并且该变量未被赋值，就表明需要到数据文件中读取该变量的实际值。在核心数据驱动引擎的控制脚本里有一个测试用例循环次数标识，核心数据驱动引擎会根据参数变量名到数据文件中找到该参数变量所在的记录，然后将DataNum字段的值赋给该标识，用于控制测试用例的循环执行。若DataNum的值大于1，则每执行一次测试循环，核心数据驱动引擎的控制脚本就会从某一列的Data字段获取对应参数变量的实际数值进行替换，直到Data1至DataN的数据均被替换之后循环终止。

与模块化测试列表一样，数据文件也要保存为“tab分隔”文件的形式才能被核心数据驱动引擎识别。为了方便起见，可以先用Excel建立好测试数据表格，然后再转化成“tab分隔”文件。

4.3 软件自动化测试混合框架控制流

软件自动化测试混合框架执行测试首先由初始脚本开始，这个脚本把高层测试表传递给高层驱动器，高层驱动器在处理高层测试表的过程中，若遇到中层测试表就将控制权转向中层驱动器，中层驱动器在处理中层测试表的过程中，若遇到低层测试表就将控制权转向低层驱动器。当低层驱动器处理低层测试表的时候，它试着使应用与测试保持同步。当低层驱动器遇到关于某一个成员的低层关键字指令时，它判断这个成员的类型并调用相应的组件函数模块来处理这个指令操作。所有这些元素都要依靠应用映射文件中的信息，它是自动化测试框架和被测应用之间的桥梁。待低层测试表的所有记录执行完毕，就返回调用该低层测试表的中层测试表，此时控制权又转回到中层驱动器，中层驱动器继续执行中层测试表的下一条记录。以此类推，待中层测试表的所有记录执行完毕，就返回调用该中层测试表的高层测试表，并将控制权转到高层驱动器，直到所有高层测试表的记录执行完毕，测试循环结束。软件自动化测试混合框架的控制流如图 4.4 所示。

WinRunner 通过运行“开始循环测试（StartCycleTest）”脚本开始执行测试。这个脚本是进入 WRSAFS 引擎的入口点，它设置好引擎所需要的所有变量和环境的值，为自动化测试框架的运行做好准备。由于在执行的过程中有许多信息在脚本之间交互，所以在 StartCycleTest 脚本中定义了许多全局变量。其中 TDPATH 和 PROJECTDIR 用于指定软件自动化测试混合框架中核心数据驱动引擎和模块化测试列表存放的路径。在 WRSAFS 引擎提供的 WRSAFS.ini 初始化文件中定义了这两个全局变量。在执行自动化测试之前我们需要根据实际情况为这两个变量赋值，以便 StartCycleTest 脚本进行初始化。CycleDriverState、SuiteDriverState、StepDriverState 三个变量分别是高层驱动器、中层驱动器和低层驱动器用于存储关

键字对的数组序列，里面包含许多关键字和与之相关联的值。在执行测试的时候WRSASF 引擎会根据需要为这些关键字赋值或提取关键字的值。

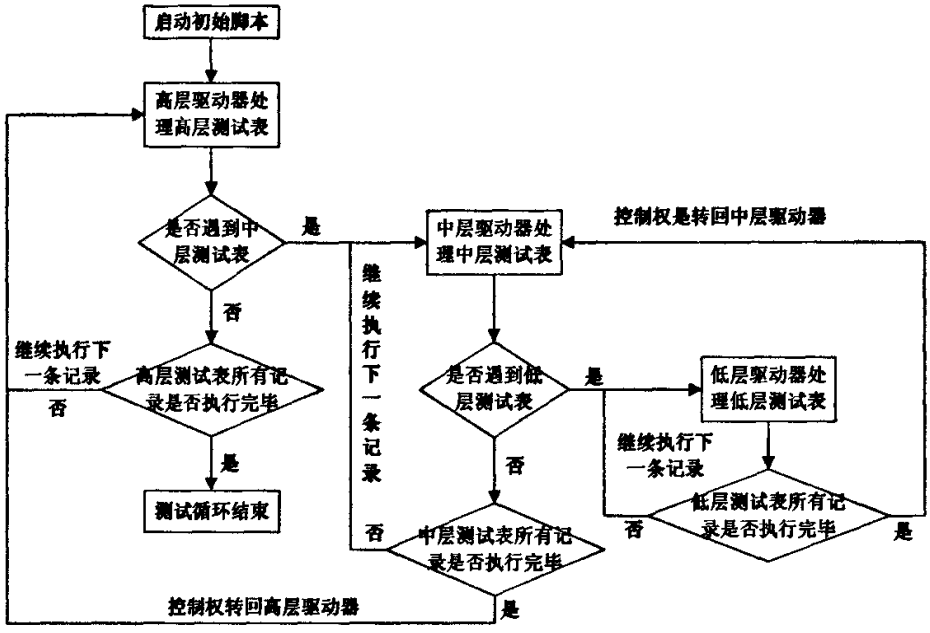

图 4.4 软件自动化测试混合框架控制流

Figure 4.4 Control Flow of Hybrid Automated Test Framework

当载入测试框架所需的所有库文件时，StartCycleDriver 会调用数据驱动引擎（DDEngine）脚本。所有的库文件都被保存为编译过的模块。若执行过程中检测到库文件中存在语法错误，则执行将被中断。当所有库文件加载完毕，控制流重新回到 StartCycleTest 脚本，然后调用 CycleDriver（循环驱动）脚本中的 CDCycleDriver 函数执行。CDCycleDriver 函数使用表 4. 16 所示的五个参数。

表 4.16 CDCycleDriver 函数使用的参数

Table 4.16 Paramater of CDCycleDriver Function

参数编号	参数名称	描述
1	Separator	存放模块测试列表各字段之间的分隔符，一般为“/t”
2	CycleDriverState	存放高层测试列表详细资料的全局数组
3	SuiteDriverState	存放中层测试列表详细资料的全局数组
4	StepDriverState	存放低层测试列表详细资料的全局数组
5	CycleDrivenMode	存放循环驱动脚本额外输出信息

在 CDCycleDriver 函数中，高层测试表被打开，并读取测试表中的一条一条记

录。首先记录的第一项即记录类型被提取出来，根据这个记录类型，CDCycleDriver 函数选择不同的执行路径。

如图 4.5 所示，如果高层测试表中第一个字段里的记录类型是“C”，则 DDDriverCommand 脚本被调用。在该函数内部有一个定义了所有驱动命令的 switch-case 语句。如果找到记录第二字段指定的命令，则该命令将被执行，相应的状态码被设置，然后测试返回到 CycleDriver 脚本，从高层测试表中读取下一条记录。执行过程中测试记录中的变量由实际的测试数据替代。DDVariabeStore 库中定义两个函数 DDSubstituteVariables 和 DDVExtractVariables 完成这一工作。

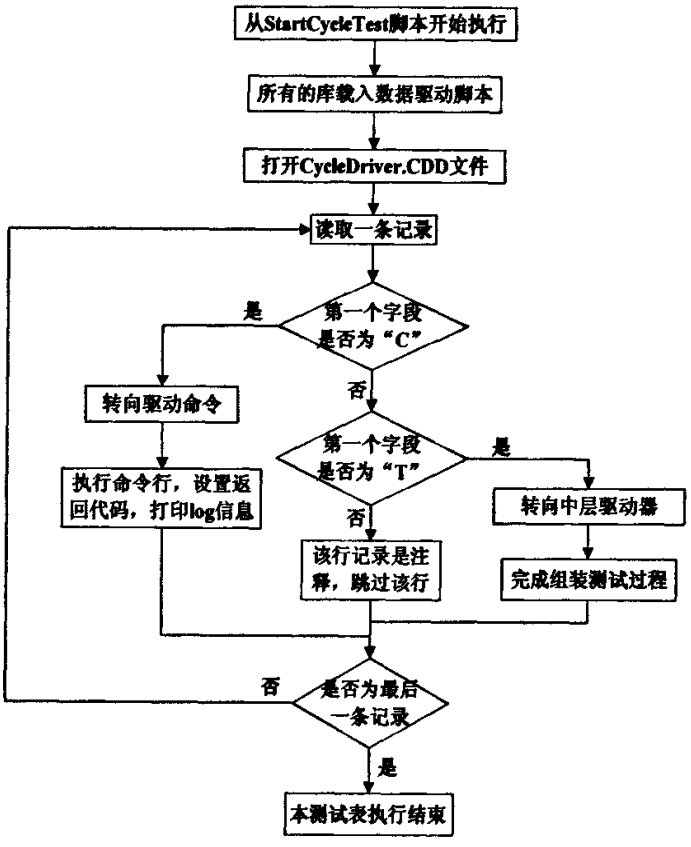

图 4.5 高层驱动器中的控制流

Figure 4.5 Control Flow of High-level Driver

如果第一个字段里的记录类型为“#”或者字段里的值的第一位为“;”，则表示该行记录是注释或说明性的内容，CycleDriver 脚本不作任何处理，跳过该行继续读取下一条记录。

如果第一个字段里的记录类型是“T”，则该记录的第二字段被提取出来，它的值是中层测试表的名字。此时高层驱动器将控制权转向中层驱动器，SuiteDriver

脚本中的 STSuiteDriver 函数被调用，用来执行中层测试表文件。该函数从中层测试表中读取记录，并提取出记录的第一个字段，根据记录类型的值选择不同的执行路径，处理过程与高层驱动器类似，如图 4.6 所示：

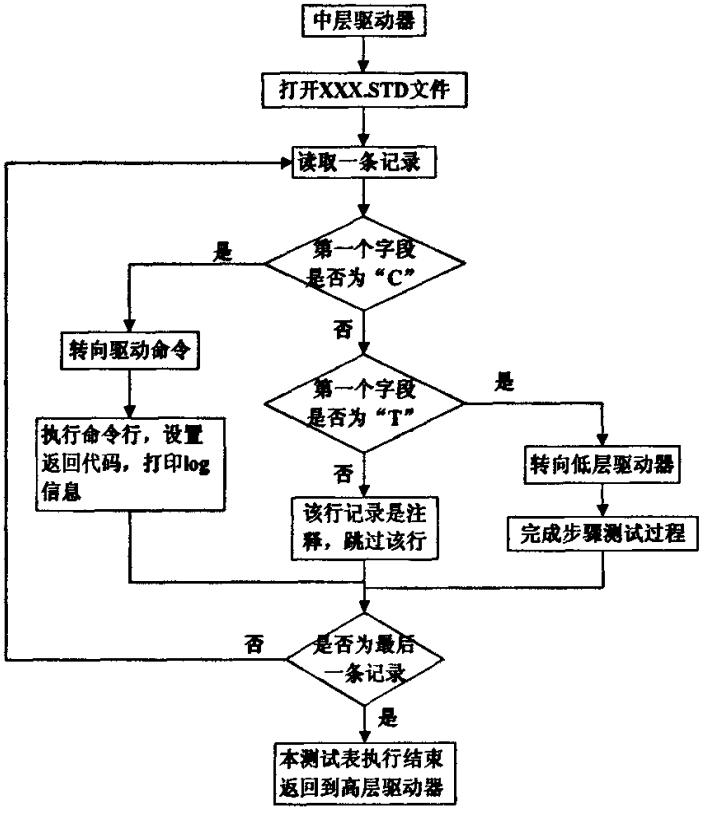

图 4.6 中层数据驱动引擎中的控制流

Figure 4.6 Control Flow of Intermediate-level Driver

如果中层测试表中第一字段的记录类型是“C”，DDDriverCommand 脚本被执行，在该函数内部定义了一个包含所有驱动命令的驱动函数。如果找到所需的驱动命令，中层驱动器将执行这个命令，并且返回相应的状态码。

如果第一个字段里的记录类型为“#”或者字段里的值的第一位为“;”，则表示该行记录是注释或说明性的内容，SuiteDriver 脚本不作任何处理，跳过该行继续读取下一条记录。

如果第一个字段的记录类型是“T”，则该记录的第二字段被提取出来，这个字段指定了低层测试表的名字。中层驱动器将控制权转向低层驱动器，即调用“StepDriver”脚本定义的 SDStepDriver 函数来执行低层测试表。该函数从低层测试表中读取记录，和处理高层测试表和中层测试表一样提取出记录的第二个字段，

选择不同的执行路径。图 4.7 表示低层驱动器中的控制流。

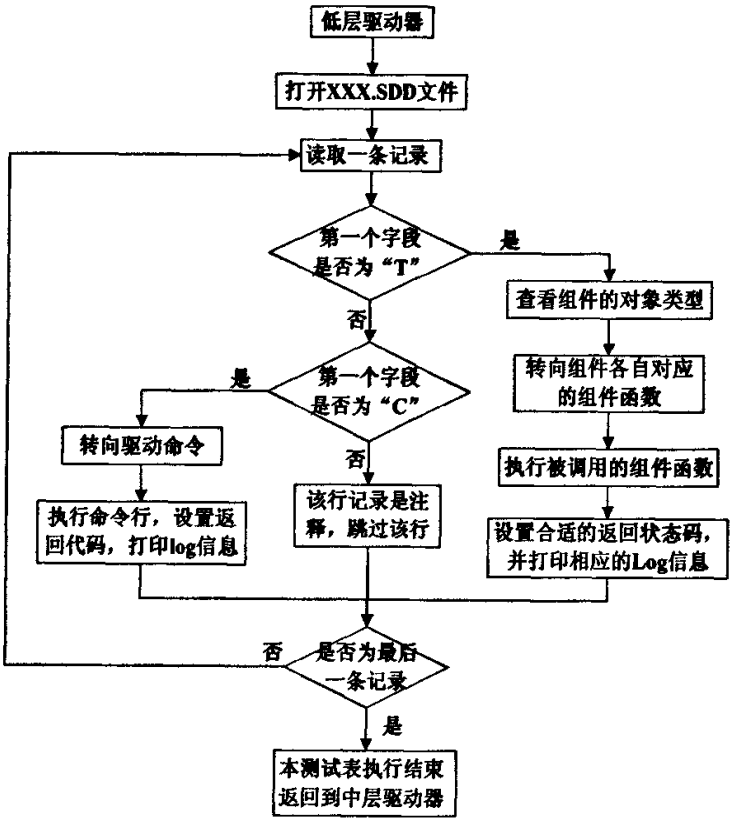

图 4.7 低层数据驱动引擎中的控制流
Figure 4.7 Control Flow of Low-level Driver

如果低层测试表第一字段的记录类型是“C”，则低层驱动器就调用 DDDriverCommand 脚本。在该函数的内部有一个定义了所有驱动命令的 switch-case 语句，如果找到所需的命令，就执行该命令并设置相应的状态码。命令执行完毕，测试继续返回处理该低层测试表的 StepDriver 脚本，读取下一条记录，直到所有的记录都被执行完。然后，测试返回到中层测试表，处理下一条记录，直到中层测试表中的所有的记录处理完。之后，再返回到高层测试表处理下一条记录，一直到高层测试表中所有的记录都处理完，则测试执行完毕。

如果低层测试表第一字段的记录类型是“T”，则该记录中的窗口名、组件名和需要执行的动作将被提取出来，StepDriver 脚本调用 GetObjectType 函数获取组件的类型信息。然后，在 StepDriver 脚本中有一个 switch-case 语句，该语句定义了组件函数库中所有组件函数模块的 main 函数名称。根据组件的类型，相应的主函数被调用。例如，如果组件类型是 PushButton，那么在 PushButtonFunctions 脚

本中定义的 PushButtonMain()函数就会被调用。在这个 PushButton 类的主函数中，利用包含 PushButton 类所有组件函数的 switch-case 语句，将低层测试表中 Action 字段的值和这些分支比较， 如果发现匹配的，该函数就会被调用，从而控制测试工具对被测应用程序的组件执行相应的操作。

当中层测试表的一条记录执行完毕，测试就返回到处理该中层测试表的 SuiteDriver 脚本，继续读取下一条记录，直到中层测试表中的所有记录都被处理过之后，测试返回到调用该 SuitDriver 脚本的 CycleDriver 脚本，再读取高层测试表的下一条记录，直到高层测试表中的所有记录都执行完毕。

在组件函数执行之后，根据执行的结果将状态码设置成通过或失败，然后程序又将控制权交给低层数据驱动（StepDriver）脚本，读取低层测试表的下一条记录，直到低层测试表中所有的记录都处理过。之后，控制流回到中层测试表，处理它的下一条记录。重复此过程直到处理完中层测试表中的所有的记录，最后是处理高层测试表中的记录，待高层测试表中的记录执行完后，测试执行完毕。

每个函数执行完之后，相应的结果都可以用 LogUtilities 库中的 LogMessage 函数来记录。最后这个执行周期的结果也记录下来。我们也可通过 WinRunner 的测试结果窗口看到测试结果。

5 软件自动化测试混合框架在DIMS中的应用

软件自动化测试是一个复杂的过程，为了确保自动化测试的成功，我们需要把软件自动化测试作为一个项目来执行。自动化测试必须被视为一个完整的软件开发过程，遵循软件开发的流程，只有这样软件自动化测试才能发挥其最大的效用。本章将介绍如何使用软件自动化测试混合框架在DIMS项目中进行有效的自动化测试。

5.1 自动化测试决定

要有效地进行自动化测试首先要对被测应用程序有清楚的了解，知道哪些测试可以自动化执行，适于进行自动化。只有脑中有了清楚的认识，才能制定出合适的测试计划，设计出好的测试用例。自动化测试决定就是测试人员经过分析被测应用程序，找出需要引入自动化测试的部分，制定测试目标，评审出可用的测试工具类型，选择测试工具的过程。

5.1.1 分析被测应用程序

决策信息管理系统（DIMS）是台湾省智慧财产局企业级应用中的一个子项目，系统主要用于搜集各种案件的统计数据及非统计数据，可供经济部智慧财产局不同阶层人员作为决策之用。它包括：

A. 前台：提供统计选项，显示统计图表部分。根据使用者选择的统计内容，自动生成各式统计图表，包括折线图、长条图、圆饼图、雷达图。按照统计内容的不同，系统分为如下几个模块：绩效考核，规费统计，行为统计，案件审查过程及结果统计，以及申请案件统计。各模块进行统计的操作流程均相同如图5.1所示，用户请求某项统计服务，系统显示该统计所需选项供用户选择，用户选择统计选项并提交给系统，系统根据统计选项产生相应的统计结果图表返回给用户。

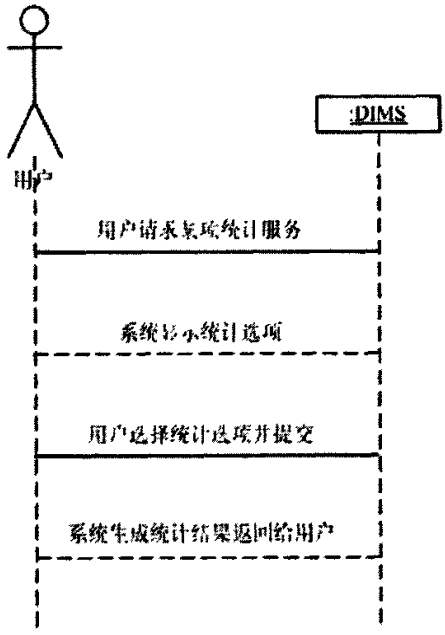

图 5.1 用户、系统交互图

Figure 5.1 User-System Interaction Diagrame

B. 后台：整合局内各项业务项目、资源及使用者记录，运用 OLAP（在线实时分析 OnLine Analytical Processing）技术，帮助使用者有效率地、方便地进行各项资源多角度的分析，以供决策之用。

C. 用户登录 DIMS 系统的时候，系统从目录服务器获取该用户权限，提供其权限范围内的操作界面。因此，前台还包括一个权限管理模块，用于对用户进行分组管理，按照智慧财产局工作人员的级别将用户分为不同的群组，每个群组具有不同的操作权限。权限管理模块包括：添加群组、更改用户群组、删除用户群组、用户权限验证几个功能。整个 DIMS 系统的架构如图 5.2 所示。

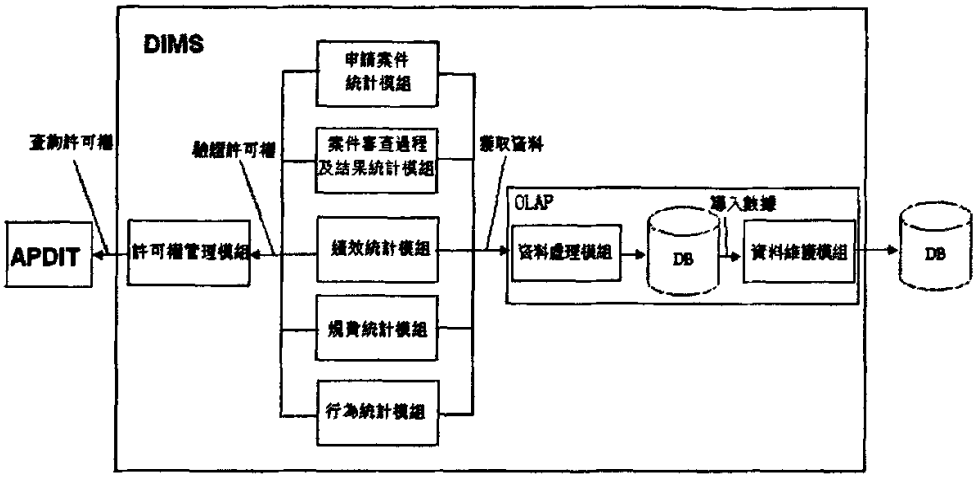

图 5.2 DIMS 架构图

Figure 5.2 DIMS Structure Diagrame

5.1.2 确定自动化测试部分

白盒测试与黑盒测试是很广泛使用的两类测试方法。

白盒测试（White-box Testing）又称结构测试、逻辑测试或基于规格说明的测试。采用这一测试方法，测试者可以看到被测的源程序，它可用以分析程序内部构造，并且根据其内部构造设计测试用例。这时测试者可以完全不顾程序的功能[20,21]。

黑盒测试（Black-box Testing）又称功能测试、数据驱动测试或基于规格说明的测试。用这种方法进行测试时，被测程序被当作打不开的黑盒，因而无法了解其内部构造。在完全不考虑程序内部结构和内部特性的情况下，测试者只知道该程序输入输出之间的关系，或是程序的功能。它必须依靠能够反映这一关系和程序功能的需求规格说明书考虑确定测试用例，和推断测试结果正确性。即所依据的只能是程序的外部特性。因此，黑盒测试是从用户观点出发的测试[20,21]。

通过黑盒测试和白盒测试的定义，我们发现黑盒测试更适合于自动化。因为基于规格说明的功能测试存在大量的重复工作，而且不涉及程序内部结构，我们只需要设计好测试输入和预期输出，自动化测试工具会根据输入自动执行被测应用程序，并将实际输出与预期输出进行比较，报告测试结果。

对于 DIMS 系统，系统功能需求如下：

A. 各种专利、商标申请案件统计

（1）各类申请案件量之周、月、季、年统计；（2）依专利类别、产业别统计；

（3）依商标类别、商品类别统计；（4）申请人为法人或自然人统计；（5）按申请国别统计；（6）专利、商标代理案件统计

B. 提供各类专利、商标等各种人民申请案件之审查过程及结果统计

（1）各类案件办理情形统计；（2）各类案件办理时效统计；（3）各类案件审查结果统计；（4）各类案件审理状态统计

C. 规费统计

（1）各类案件收费、退费之按日、周、月、季、年统计，需能区分使用在线规费及其它规费机制之金额统计；（2）使用者为个人、公司或代理人之统计

D. 行为统计

（1）提供各类案件于各阶段承办累计时间统计

E. 绩效考核之综合评量

（1）提供各承办人员收件、待办、持有、办结等作业阶段之统计；（2）提供各主管收件、待办、持有、办结等作业阶段之统计；（3）提供各阶段待办案件量、未结案件量、逾时案件量之统计；（4）依各承办个人、单位别之统计

由上面的系统需求我们发现，每个统计模块提供给用户的统计选项很多，用户可以根据自己的需要进行组合，因此在对系统进行功能测试时，会存在大量测试用例，将这部分测试实施自动化，必然可以提高测试效率。由于每个统计模块的操作流程大致相同，采用软件自动化测试混合框架可以更好的提高模块化测试列表的复用性，减少测试维护开销。

5.1.3 制定测试目标

测试工作要达到什么目的呢？通常情况下，测试要证实软件达到了特定的标准，并且满足最终用户或客户的需求。测试的主要目标是找出应用中的缺陷，从而防止、检测、消除缺陷，生成一个稳定的系统[20]。

测试的主要目标是提高被测应用在各种条件下都能正常运行并达到所定义需求要求的概率，从而尽可能发现错误，使最终用户感到满意。自动化测试最终可以减少测试工作量，缩短测试进度、产生可靠的系统、增强测试过程的可重复性。

在确定自动化测试部分，我们选择将功能测试部分进行自动化，功能测试就是验证软件是否实现了需求说明书的要求。结合 DIMS 系统需求说明制定以下测试目标：

（1） 给定有效的统计输入，系统产生正确的统计输出。

（2） 给定无效的统计输入，系统正常执行并能产生错误提示。

（3） 无论给定有效输入还是无效输入，系统没有挂起或崩溃。

（4） 系统的行为与指定的一致。

（5） 系统是否达到软件需求规格说明书的功能要求。

5.1.4 选择测试工具

针对自动化测试决定阶段确定的系统需要自动化测试部分，我们将利用软件自动化测试混合框架进行系统功能测试，为此需要选择合适的功能测试工具。功能测试工具有很多，例如 WinRunner，Rational robot，QARun，SilkTest，e-Tester，WebFT，PureTest[22-24]等，面对市面上林林总总的测试工具，我们需要根据测试的需要选择合适的测试工具。选择功能测试工具可以考虑以下几个方面[25]：

（1）是否易于使用，容易维护，容易安装。如果一个自动化测试工具安装困难并且难于维护，那么这个测试工具是不会被使用的。

（2）是否支持多平台。随着计算机技术的发展，越来越多的应用程序被设计成能够在不同的平台上运行，这就要求在测试的过程中，要在不同的平台上对应用程序进行验证，因此也就需要测试工具能够立即移植到各个平台上并在其上运行，这样才能保证测试的顺利进行。

（3）测试工具使用的脚本语言是否灵活、健壮，复杂，是否允许采用模块化脚本开发方法。自动化测试说简单一点实际是测试工具自动执行测试脚本的过程，灵活、健壮的脚本语言就可以开发出灵活、健壮的脚本；允许采用模块化脚本开发方法的脚本语言可以提高测试脚本的复用性，生成的脚本易于维护。若脚本语言过于复杂，测试人员学习、使用起来就比较困难，这样的测试工具也就不会被使用。

（4）是否可以建立可重用函数库。自动化测试工具运行的核心就是控制脚本调用函数库中的相应函数对组件进行操作，从而达到自动化测试的目的。测试工具具有可重用函数库就可以降低开发测试脚本的开销，提高测试脚本的可重用性和可维护性。

（5）是否具有报告生成能力。测试报告是自动化测试的重要组成部分，一个完整的自动化测试过程应该有测试报告详细地记录本次测试的执行情况，并能以图形或表格的形式记录测试结果，供测试人员进行分析得出相应的测试结论。

通过对 DIMS 系统的分析，鉴于系统各模块操作流程相同，采用软件自动化测试混合框架进行测试能够充分提高测试用例的重用性。软件自动化测试混合框架是基于关键字驱动的软件自动化测试框架支持工具（SAFS）进行设计的，为此我们选择 WinRunner 作为测试工具。

WinRunner具有以下特点：（1）易于安装和使用；（2）它提供的脚本语言简单、灵活、健壮，能够开发出功能强大的测试脚本；（3）自动生成测试报告的功能为测试人员分析测试结果提供了极大的便利；（4）可重用和扩充的组件函数库既能提高测试脚本的重用性，又能扩展测试工具的功能。

5.2 自动化测试计划

任何一项活动要想有条不紊、顺利地进行，都离不开一个周密的计划。软件自动化测试过程也不例外，测试计划是高效测试的基础。测试计划包含丰富的信息，其中包括测试任务，测试设计流程，测试环境准备。

5.2.1 测试任务

在测试计划中首先要确定测试需求，即明确测试任务，为制定测试用例提供基础。在DIMS项目中，根据系统功能需求说明，制定如表5.1所示的测试任务表。

表5.1 测试任务表
Table 5.1 Test Task Table

功能模块	测试任务
登录	1. 验证当用户名、密码正确时，用户能否正常登录，并正确显示该用户所在群组权限范围内模块导航链接 2. 验证当用户名、密码错误时，系统能否给出拒绝登录的提示
绩效考核	1. 验证用户按不同的统计类别提交正确统计选项时，系统是否能够给出正确的统计结果 2. 验证用户提交错误统计选项时（统计日期格式错误），系统能否给出出错提示
规费统计	1. 验证用户按不同的统计类别提交正确统计选项时，系统是否能够给出正确的统计结果 2. 验证用户提交错误统计选项时（统计日期格式错误），系统能否给出出错提示
行为统计	1. 验证用户按不同的统计类别提交正确统计选项时，系统是否能够给出正确的统计结果 2. 验证用户提交错误统计选项时（统计日期格式错误），系统能否给出出错提示
案件审查过程及结果统计	1. 验证用户按不同的统计类别提交正确统计选项时，系统是否能够给出正确的统计结果 2. 验证用户提交错误统计选项时（统计日期格式错误），系统能否给出出错提示
申请案件统计	1. 验证用户按不同的统计类别提交正确统计选项时，系统是否能够给出正确的统计结果 2. 验证用户提交错误统计选项时（统计日期格式错误），系统能否给出出错提示

5.2.2 测试设计流程

在运用软件自动化测试混合框架进行测试设计之前，需要制定合理的自动化测试设计流程，来指导测试的设计与开发，才能确保自动化测试顺利执行。根据软件自动化测试混合框架的结构制定如图 5.3 所示的测试设计流程。

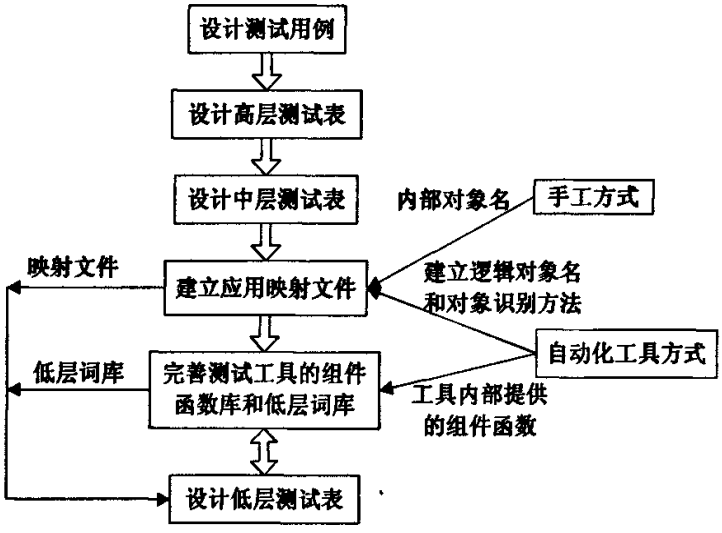

图 5.3 自动化测试设计流程
Figure 5.3 Automated Testing Design Flow

首先由测试设计人员根据系统功能需求说明和测试任务表设计测试用例。测试用例是执行自动化测试的蓝图，它需要测试设计人员根据自己的测试经验，结合被测应用程序的特点，并充分考虑测试列表的复用性设计出来。测试用例的设计应遵循以下几条原则：

（1）测试用例要能全面覆盖系统功能需求说明中列出的系统所有功能。

（2）测试用例不仅要能测试系统正常执行情况，还要设计错误输入数据，验证系统是否出现错误提示信息。

（3）要采用等价划分的方法设计测试用例，利用等价类别或等价区间把过多（无限）的测试用例减小到具有相同测试目标或者暴露相同软件缺陷的一组测试用例。

待测试用例确定下来之后，我们就能够全面、方便、快捷地将所要执行的测试任务转化成模块化测试列表。首先是设计高层测试表，每个高层测试表定义了一个测试的执行周期，其中包括多个待实现的子任务(即高层关键字)。接着设计中层测试表，每个中层测试表对应着高层测试表的一个子任务，在中层测试表中

根据实际测试经验对高层循环测试表中的每一个子任务进行详细扩展。

一般高层和中间层的测试设计都十分抽象，所以不需要引用具体的界面元素。但是当我们向最终的低层测试设计靠近时，就必须要引用具体的界面元素。因此在设计低层测试表之前，需要设计应用映射文件。我们可以利用自动化测试工具学习的方式获得每一个界面元素的逻辑名称、类型和识别方法，将逻辑对象名和识别方法对应起来就形成了映射文件。

自动化测试工具已经为我们提供了常用组件的组件函数，而在建立映射文件的过程中，我们可能会发现新的没有被组件函数处理的组件类型。因此，需要为每个新组件类型实现一个新的组件函数模块，或决定已有的组件函数能否最佳处理这些新类型，从而进一步完善组件函数库和低层词库。若新增了组件函数，就需要在低层词库中添加该组件函数对应的关键字及所需的参数。

最后，设计低层测试表。低层测试表将中层测试表中的关键字细化为具体的、对界面元素的直接操作行为。在这个过程中，我们会用到在应用映射文件中定义的逻辑对象名和低层词库定义的关键字对低层测试表进行填空。如果测试设计人员发现组件函数中缺少必须的关键字、行为或命令时，脚本开发人员就要对相应的组件函数模块、驱动器命令和支持库进行扩展，或找到其他一些解决方法来提供这些功能。

5.2.3 测试环境准备

软件自动化测试混合框架中的核心驱动引擎 WRSAFS 是基于 SAFS 设计的，因此必须安装 WRSAFS 和 SAFS 这两个工具包，并在测试机器上进行相应配置，才能保证测试框架的正常运行。这两个工具包在 sourceforge.net 的网站上都可下载到最新版本。在 SAFS 工具包中有一个名为 SetupSAFS.vbs 的脚本程序，在运行里输入 cmd 命令进入命令窗口，输入 cscript setupSAFS.vbs 命令，执行脚本程序，安装完毕后，脚本程序会自动在环境变量里添加如表 5.2 所示的变量。这些环境变量指定了自动化测试框架运行所需的 jar 包的路径

表 5.2 SAFS 环境变量设置
Table 5.2 Environment Variable of SAFS

Var	Value
CLASSPATH	;C:\STAF\bin\USTAF.jar;C:\SAFS\lib\jakarta-regexp-1.3.jar; C:\SAFS\lib\safscust.jar;C:\SAFS\lib\safsrational.jar; C:\SAFS\lib\safs.jar; C:\SAFS\lib\safsjrex.jar; C:\SAFS\lib\JRex.jar
SAFSDIR	C:\SAFS
STAFDIR	C:\STAF

此外要使 WRSAFS 正常运行，安装完 WRSAFS 工具包之后，还要在注册表 HKEY_LOCAL_MACHINE/Software/WRSAFS 路径下创建表 5.3 所示的键值。

表 5.3 WRSAFS 注册表键值设置
Table 5.3 Register Value of WRASFS

Key	Value	备注
EnginePath	C:\WRSAFS	WRSAFS 引擎的安装路径
CyclePath	Cycle	高层循环驱动引擎的路径名
SuitePath	Suite	中层循化驱动引擎的路径名
StepPath	Step	低层循化驱动引擎的路径名

5.3 自动化测试设计

在自动化测试决定和自动化测试计划阶段，我们确定了被测应用程序需要执行自动化测试的部分，明确了测试任务，并制定了测试设计流程，下面就要进行详细的测试设计。

5.3.1 设计测试用例

根据 DIMS 系统的特点，分别对各个模块的功能进行测试，测试每个模块用户在提交各种统计选项的组合之后，系统能否做出正确的响应。为了将测试用例与模块化测试列表相对应，测试用例按照层次进行组织与编写，表 5.4 是测试用例的最高层级叫做测试用例集。编号说明：测试用例集编号格式为 TC290-用例集序号，测试用例编号格式为 TC290-所属用例集序号-用例序号，其中 TC 为 Test Case

的缩写，290 是 DIMS 的系统代号。

表 5.4 DIMS 系统功能测试用例集
Table 5.4 Functional Test Case Suite of DIMS

测试用例集	TC290_DIMS 系统功能测试
测试用例集内容	使用者进入 DIMS 系统，选择各个模块进行统计操作
测试用例集	
测试用例集编号	测试用例集名称
TC290-01	用户登录
TC290-02	绩效考核
TC290-03	规费统计
TC290-04	行为统计
TC290-05	案件审查过程及结果统计
TC290-06	申请案件统计
TC290-07	用户退出 DIMS

上述测试用例集只是对将要进行的测试的抽象概括，指出要对系统哪些模块进行测试，并没有涉及到具体的测试细节，可将其与模块化测试列表中的高层测试列表相对应。下一步是对测试用例集的细化，制定实施每个测试用例集要进行哪些测试。表 5.5 和表 5.6 分别是“用户登录”用例集与“行为统计”用例集的细化。

表 5.5 用户登录用例集
Table 5.5 Test Case Suite of User Login

测试用例集名称	TC290-01_用户登录	
测试用例集内容	使用者登录 DIMS 系统	
测试用例		
测试用例编号	对应屏幕/选单	测试用例内容
TC290-01-1	SC290-UC03-01-01_用户登录页面	使用者输入用户名、密码进行登录
TC290-01-2	SC290-UC03-01-02_验证用户权限—功能模块索引画面	使用者通过权限验证，登录 DIMS 系统
TC290-01-3	SC290-UC03-01-03_验证用户权限—错误提示信息画面	使用者未通过权限验证拒绝登录

表 5.6 行为统计用例集
Table 5.6 Test Case Suite of Behavior Statistic

测试用例集名称	TC290-04_行为统计	
测试用例集内容	使用者进行行为统计	
程序版本	3.0	
测试环境	Window XP，P4，CPU 3.0GHz，内存 1GB	
测试时间	2006-06-21	
测试用例		
测试用例编号	对应屏幕/选单	测试用例内容
TC290-04-1	SC290-UC02-03-01_行为统计首页	使用者选择行为统计对象之功能
TC290-04-2	SC290-UC02-03-02_行为统计详细选项画面	使用者选择行为统计案件类型、输出样式、统计时段之功能
TC290-04-3	SC290-UC02-03-03_行为统计——信息确认画面	使用者对所选信息确认之功能
TC290-04-4	SC290-UC02-03-04_行为统计——统计结果输出画面	使用者更换图表类型，查看统计结果的图表之功能

通过上述两个表对“用户登录”和“行为统计”用例集的细化我们发现，这两个表仍然没有指明如何具体进行测试，还属于逻辑设计的部分，这正好与模块化测试列表中的中层测试表相对应。下面就要对每个测试用例集中的各个测试用例进行进一步的细化，制定测试执行的详细步骤，即设计与低层测试表相对应的测试用例的详细执行步骤。表 5.7 至表 5.9 是用户登录用例集中测试用例的细化。

表 5.7 测试用例 TC290-01-1
Table 5.7 Test Case TC290-01-1

测试用例集	TC290-01_用户登录	
测试用例	TC290-01-1_使用者未输入用户名或密码进行登录	
步骤	测试程序/输入	预期输出
1.	在浏览器地址栏输入登录页面地址	进入 SC290-UC03-01-01_用户登录页面
2.	于用户名输入字段不输入用户名	用户名输入字段文本框为空
3.	于密码输入字段输入用户密码“dims”	密码输入字段文本框内以不可见形式显示用户输入的密码，****
4.	按“登录”按钮	移至 SC290-UC03-01-01_用户登录页面，并在页面上显示“用户名或密码不能为空，请重新输入用户名和密码” 的错误信息

表 5.8 测试用例 TC290-01-2
Table 5.8 Test Case TC290-01-2

测试用例集	TC290-01 用户登录	
测试用例	TC290-01-2_使用者通过权限验证登录 DIMS 系统	
步骤	测试程序/输入	预期输出
1.	在浏览器地址栏输入登录页面地址	进入 SC290-UC03-01-01_用户登录页面
2.	于用户名输入字段输入用户名“dims”	用户名输入字段文本框内显示用户输入字符串，dims
3.	于密码输入字段输入用户密码“dims”	密码输入字段文本框内以不可见形式显示用户输入的密码****
4.	按“登录”按钮	用户通过权限验证，移至 SC290-UC03-01-02_验证用户权限——功能模块索引画面

表 5.9 测试用例 TC290-01-3
Table 5.9 Test Case TC290-01-3

测试用例集	TC290-01_用户登录	
测试用例	TC290-01-3_使用者未通过权限验证，显示出错信息	
步骤	测试程序/输入	预期输出
1.	在浏览器地址栏输入登录页面地址	进入 SC290-UC03-01-01_用户登录页面
2.	于用户名输入字段输入用户名“abc”	用户名输入字段文本框内显示用户输入字符串，abc
3.	于密码输入字段输入用户密码“abc”	密码输入字段文本框内以不可见形式显示用户输入的密码，***
4.	按“登录”按钮	用户未通过权限验证，移至 SC290-UC03-01-03_验证用户权限——错误提示信息画面
5.	按“返回”按钮	移至 SC290-UC03-01-01_用户登录页面

上述三个表详细描述了对应用程序各个组件的具体操作，制定了执行每个测试用例所需要的步骤，因此该层测试用例可与模块化测试列表中的低层测试表相对应。

5.3.2 设计高层和中层测试列表

通过对测试用例的分层设计，为接下来进行模块化测试列表的设计提供了基础，我们可以根据测试用例的组织层次及其内容来设计模块化测试列表。首先采

用 Excel 进行测试表格编辑，表格的形式按照第四章介绍的模块化测试列表的格式进行设计。首先设计高层测试表，其结构如图 5.4 所示。表所有处理记录的类型均为 T，表示将要调用中层测试表进行处理，SUITES 字段为需要处理的中层测试表的名字，这些测试表正好与系统功能测试用例集中所列的测试用例集相对应，所有这些测试用例集构成一个测试循环。

	A	B	C	D
1		;CycleDriver.CDD		
2		;This is the Cycle Driver test for DIMS		
3				
4		;Record Type(RT):	Description:	
5		;C = DRIVER COMMAND	驱动命令	
6		;T = TEST STEP	测试步骤	
7		;# = SKIPPED TEST	标识该行记录只是一段注释，驱动引擎或略过	
8				
9	;RT	COMMAND	ARG	Description:
10	C	SetApplicationMap	C:\Test\DataPool\DIMS.gui	
11	;RT	SUITES	ARG	Description:
12	T	LaunchApp		登录DIMS
13	T	AchTest		测试绩效考核模块
14	T	ExpTest		测试规费统计模块
15	T	BehTest		测试行为统计模块
16	T	ExaTest		测试案件审查过程及结果统计模块
17	T	AppTest		测试申请案件统计模块
18	T	CloseApp		关闭DIMS

图 5.4 高层测试表

Figure 5.4 High-Level Test Table

以“用户登录”测试用例集为例，说明其对应的中层测试表的设计。如图 5.5 所示：

	A	B	C	D
1		;LaunchApp.STD		
2		;This test will be process by SuiteDriver in the WRAFS		
3		;Record Types (RT):	Description:	
4		;C = DRIVER COMMAND	驱动命令	
5		;T = TEST STEP	测试步骤	
6		;# = SKIPPED TEST	标识该行记录只是一段注释，驱动引擎或略	
7	;RT	COMMAND	ARG	Description
8	C	SetApplicationMap	C:\Test\DataPool\DIMS.gui	Set the GUI Map File
9				
10	;RT	Steps	ARG1	ARG2
11	T	LaunchDIMS		
12	T	Login	*userName	*Password

图 5.5 中层测试表 LaunchApp.STD

Figure 5.5 Intemediate-Level Test Table LaunchApp.STD

第一条处理记录的类型为 C，表示中层数据驱动引擎调用 SetApplicationMap 命令为测试工具的 GUI Map Editor 加载映射文件。后两条处理记录的类型均为 T，表示将要调用低层测试表进行处理，Steps 字段为所要处理的低层测试表的名称。其中第三条记录里 ARG1 和 ARG2 字段^userName 和^Password 为参数变量，表示处理 Login.SDD 低层测试表时需要传递参数，当中层数据驱动引擎处理到这两个参数变量时就会用存放在数据文件中的实际测试数据来代替它们，从而实现用同一条记录执行不同的测试用例。图 5.4 中其它测试用例集对应的中层测试表的形式与图 5.5 所示的相同，按照中层测试用例表的内容（如图 5.5），在每个测试表内，记录类型为 T 的处理记录的 Steps 字段定义低层测试表的名字，该低层测试表的名字与测试用例集所要执行的所有测试用例对应。

5.3.3 设计应用映射文件

由于高层测试表和中层测试表只是逻辑设计，不涉及对具体应用程序组件的操作，所以可以直接根据测试用例集以及各用例集细化的测试用例进行设计，但在设计低层测试表时，要设计每个测试用例的详细执行步骤，必然会涉及到对某个窗口，某个组件的具体操作，就必须知道窗口和组件的逻辑名称，以及调用组件函数对该组件进行操作的关键字，因此在设计低层测试表之前，需要使用测试工具建立应用映射文件。若发现应用程序中的某个组件没有对应的组件函数对其操作，还需要完善组件函数库和低层词库。

按照第四章讲述的建立应用映射文件的方法，采用 WinRunner 的 GUI Map Editor 学习 DIMS 系统各模块每个页面内的所有组件。学习完毕，GUI Map Editor 会为每个组件分配一个逻辑映射名，标识各个组件的类型，以及生成可以唯一识别该组件的最少的物理描述，并将所有这些信息存放在.gui 格式的文件中。然后再把.gui 格式的文件转化成自动化测试框架能够识别的后缀名为.map 的应用映射文件。在应用映射文件中我们需要添加 ApplicationConstants 部件，在该部件中指定进入 DIMS 系统的 URL 路径。应用映射文件的形式见附录 A：DIMS 应用映射文件。

5.3.4 设计低层测试表

DIMS 系统的所有组件涉及到的操作都有 WinRunner 提供的组件函数与之对应，所以不需要完善或增加自动化测试框架的组件函数库以及低层词库，就可以直接进行低层测试表的设计了。我们只需要按照第四章讲述的低层测试表的结构，

将低层的各个测试用例所描述的测试步骤转化成低层测试表的形式即可。

仍然以“用户登录”测试用例集为例，以 LaunchApp.STD 表中 Steps 字段的值为低层测试表的名字，按照低层测试用例细化的测试步骤，生成低层测试表如图 5.6 和图 5.7 所示：

	A	B	C	D	E
1		;LaunchDIMS.SDD			
2		;Record Type(RT):	Description:		
3		;C = DRIVER COMMAND	驱动命令		
4		;T = TEST STEP	测试步骤		
5		;# = SKIPPED TEST	标识该行记录只是一段注释，驱动引擎或略		
6	;RT	COMMAD	ARG1	Description:	
7	C	LaunchApplication	"DIMSSite	进入DIMS登录首页	

图 5.6 低层测试表 LaunchDIMS.SDD

Figure 5.6 Low-Level Test Table LaunchDIMS.SDD

	A	B	C	D	E	F
1		;Login.SDD				
2		;Record Type(RT):	Description:			
3		;C = DRIVER COMMAND	驱动命令			
4		;T = TEST STEP	测试步骤			
5		;# = SKIPPED TEST	标识该行记录只是一段注释，驱动引擎或略			
6	;RT	WINDOW	COMP	ACTION	ARG1	Description
7	T	Login	UserName	SetTextValue	"userName	输入用户名
8	T	Login	Password	SetUnverifiedTextValue	"Password	输入密码
9	T	Login	Login	Click		

图 5.7 低层测试表 Login.SDD

Figure 5.7 low-level test table Login.SDD

由图 5.6 和图 5.7 可见，直到低层测试表才涉及对应用程序界面组件的操作，Action 字段为关键字，通过该字段调用相应组件函数执行具体的操作。ARG 字段的参数变量名与中层测试表 ARG 字段的参数变量名对应，中层数据驱动引擎将控制权转到低层数据驱动引擎时会将其获取的相应参数变量的实际数值也传递给低层数据驱动引擎，以便测试工具按照指定的数据进行测试。模块化测试列表的组成见附录 B。

5.3.5 设计数据文件

DIMS 系统的每项统计都要求按照日、周、月、季、年的统计方式进行，因此在设计测试用例时必须将这五种情况都考虑进去。在图 5.8 所示的行为统计模块对

应的中层测试表 BehTest.STD 中，第二条用于选择行为统计选项的测试记录，定义一个名为^means 的参数变量，数据驱动引擎从数据文件里分别读取^means 的五个值进行测试。这样在每个模块对应的中层测试表中都使用这个参数变量就能测试各模块是否能够按不同的统计方式进行正确地统计。

	A	B	C	D	E	
1		;BehTest.STD				
2		;This test will be process by SuiteDriver in the WRAFS				
3						
4		;Record Types (RT):	;Description:			
5		;C = DRIVER COMMAND	Driver to perform some function other than a test step.			
6		;T = TEST STEP	A test to perform.			
7						
8	;RT	Steps	ARG1	ARG2	ARG3	ARG4
9	T	SelectBehFirst	^type="專利"	^stage="提利取得前"		
10	T	SelectBehStat	^caseType="發明專利"	means	^outType="折線圖"	^dat
11	T	ConfirmBehStat				
12	T	BehChart	^chOutType1="長條圖"	^chOutType2="圓餅圖"	^chOutType3="雷達圖"	

图 5.8 BehTest.STD

Figure 5.8 BehTest.STD

此外，“用户登录”测试用例集需要使用正确和错误的用户名、密码来登录系统，从而验证系统能否给出正确的响应。因此在“用户登录”测试用例集对应的中层测试表 LauchAPP.STD 中也使用了两个参数变量^userName 和^Password，用于接收不同的用户名和密码。综合上述分析，数据文件的内容如表 5.10 所示。

表 5.10 数据文件格式

Table 5.10 Data File Format

VarName	DataNum	Data1	Data2	Data3	Data4	Data5
userName	5	abc	kezhang	patent	aa	dims
Password	5	abc	kezhang	patent	aa	dims
means	5	日	周	月	季	年

5.3.6 设计检测点

自动化功能测试的实质是测试工具自动执行被测应用程序，记录应用程序实际运行结果，比较其与期望结果是否相同，从而判断被测应用程序是否符合系统功能需求说明。因此在测试设计中需要利用 WinRunner 在控制脚本中插入检测点，测试工具执行到此处时会自动将运行结果与期望结果进行比较，比较的结果会在测试结果中显示出来，供测试人员分析测试情况。对于 Web 应用程序功能测试主

要检测网页上的内容是否正确显示，例如检测表格、单元格或图片的内容，检测链接是否正确等。DIMS 系统设计如下检测点。

1．用户使用错误的用户名和密码进行登录，系统进入出错页面，检测出错页面是否显示用户无权访问系统的错误信息。将光标设置在控制脚本中需要插入检测点的位置，选择菜单栏 Insert→Bitmap Checkpoint→For Screen Area，此时光标变成十字形，拖动鼠标在被测页面上截取期望看到的位图，如图 21 所示，然后点击鼠标右键完成操作，WinRunner 会自动在控制脚中添加如下一条语句：

win_check_bitmap("錯誤", "Img1", 1, 407, 276, 433, 39)

该语句中参数 Img1 即为图 21，测试工具会在实际页面上寻找是否有与图 5.9 相同的位图出现。

图 5.9 登录错误提示
Figure 5.9 Hint of Login Error

2．用户使用正确的用户名和密码进行登录，系统进入索引页面，检测索引页面能否根据用户所属群组，显示该用户权限范围内的统计模块链接。DIMS 系统用户分为四个群组：局长、科长、专利组、个人。用户群组及其对应的权限如表 5.11 所示。

表 5.11 用户群组与权限对应表
Table 5.11 Corresponding Table of User Group and Popedom

用户群组	权限范围
局长	所有模块
科长	规费统计模块，行为统计模块，专利案件审查过程及结果统计模块，专利申请案件统计模块
专利组	专利案件审查过程及结果统计模块，专利申请案件统计模块
个人	绩效考核模块

在测试数据中，我们分别设计了四组属于不同用户群组的登录帐号，进行验证。例如用户名和密码均为 dims 的用户属于局长级用户，他可以操作所有统计模块，因此系统索引页面显示所有统计模块的链接，如图 5.10 所示。

图 5.10 索引页面
Figure 5.10 Index View

选择菜单栏 Insert→GUI Checkpoint→For Object/window，此时鼠标变成手形，鼠标选中被测页面需要检测的链接对象之后，点击右键完成操作，此时测试工具自动在控制脚本中添加 GUI 检测点代码，测试工具运行到检测点时就会判断实际链接的 URL 与期望链接的 URL 是否一致。局长级用户需要检测的 GUI 检测点如下。

```
obj_check_gui("績效考核", "list2.ckl", "gui2", 1);
obj_check_gui("規費統計", "list3.ckl", "gui3", 1);
obj_check_gui("行爲統計", "list4.ckl", "gui4", 1);
obj_check_gui("專利案件審查統計", "list5.ckl", "gui5", 1);
obj_check_gui("專利申請案件統計", "list6.ckl", "gui6", 1);
```

3．验证各模块“用户确认页面”能否正确显示用户在前几级页面选择的统计选项信息，如图 5.11 所示。统计信息在页面上实际是按照表格的形式组织的，插入 GUI 检测点，检测该表格的 TableContent 属性，即可验证内容是否为用户先前所选的统计信息。如上述方法一样，在需要插入检测点的地方选择菜单栏 Insert→GUI Checkpoint→For Object/window，用鼠标选中确认信息所在的表格，双击鼠标弹出检测点属性对话框（如图 5.12 所示），列出所选 GUI 对象可供检测的属性，选择 TableContent 属性，双击 Expected Value 即可编辑需要检测的表格的期望值。完成之后就会在插入添加如下语句 obj_check_gui("确认信息 ", "list1.ckl", "gui1", 1)。确认信息为检测表格的逻辑名，list1.ckl 和 gui1 文件存放期望的表格内容。

您的輸入資訊如下：

統計對象： 專利
統計階段： 權利取得前
案件類型： 發明專利分割案
統計方式： 周
起始日期： 2000-01-01
截止日期： 2000-01-10
輸出模式： 折线圖

是否需要更改?

图 5.11 统计确认页面
Figure 5.11 Confirm View

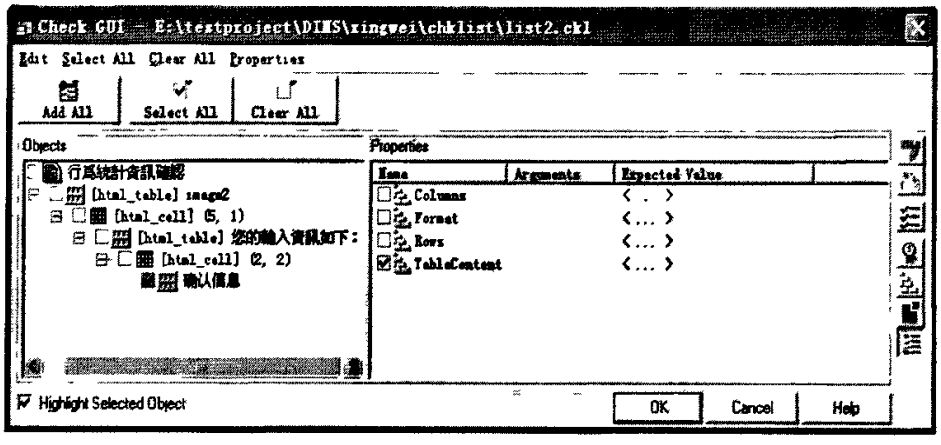

图 5.12 表格组件检测点
Figure 5.12 Checkpoint of Table Component

4. 验证各模块“统计结果显示页面”，切换不同的图形输出样式能否显示正确的图形，如图 5.13 所示。在控制脚本中插入位图检测点，判断是否出现指定样式的统计图形。

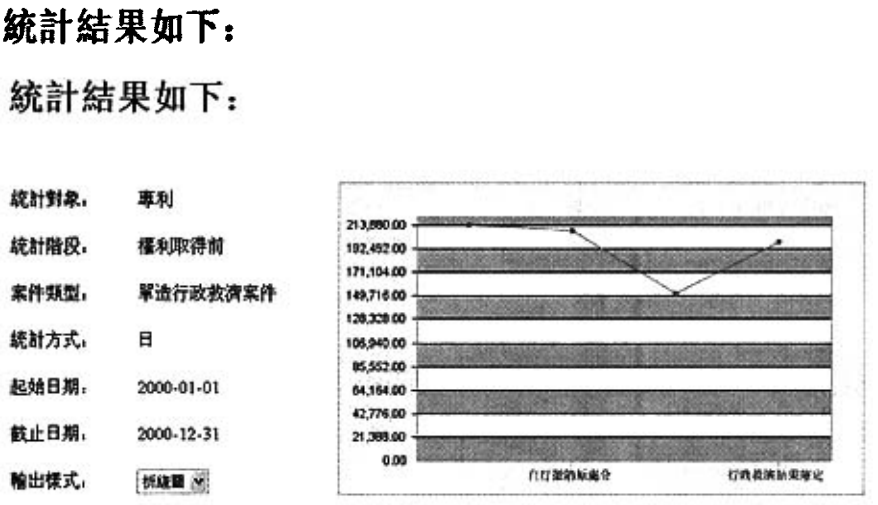

图 5.13 统计结果页面
Figure 5.13 Result View

5.4 自动化测试实施

完成上述各项工作之后，就可以执行自动化测试了。在执行自动化测试之前，首先要确定测试工程所需的文件按照图 5.14 所示的结构进行组织，所有文件存放在名为 DataPoolCycle 的根目录下。Cycle 文件夹放高层测试表的.CDD 文件，Suite 文件夹放中层测试表的.STD 文件，Step 文件夹放低层测试表的.SDD 文件，Data 文件夹存放数据文件，Logs 文件夹用于存放测试日志，Test 文件夹用于存放测试结果。DIMS 文件夹存放插入检测点时捕获的期望数据，WinRunner 所需的.gui 文件和测试框架所需的.map 应用映射文件直接放在 DataPool 根目录下。

测试从 StartCycleTest 控制脚本开始，在此脚本中定义了几个重要的变量，用于指定测试执行时所需的各类文件的路径，例如 PROJECTDIR 变量指示测试工程文件存放的路径，即 DataPool 文件夹的路径，TDPATH 变量指示数据驱动引擎及组件函数库文件的路径，CyclePath、SuitePath 和 StepPath 变量就分别指定对应层级的测试列表文件存放的路径。因此在执行测试之前，要确保给这些变量赋予正确的值才能保证测试的正常执行。此外 DIMS 是基于 Web 的应用程序，在执行测试之前还应启动与应用程序相关的服务器，例如应用程序服务器、数据库服务器等。

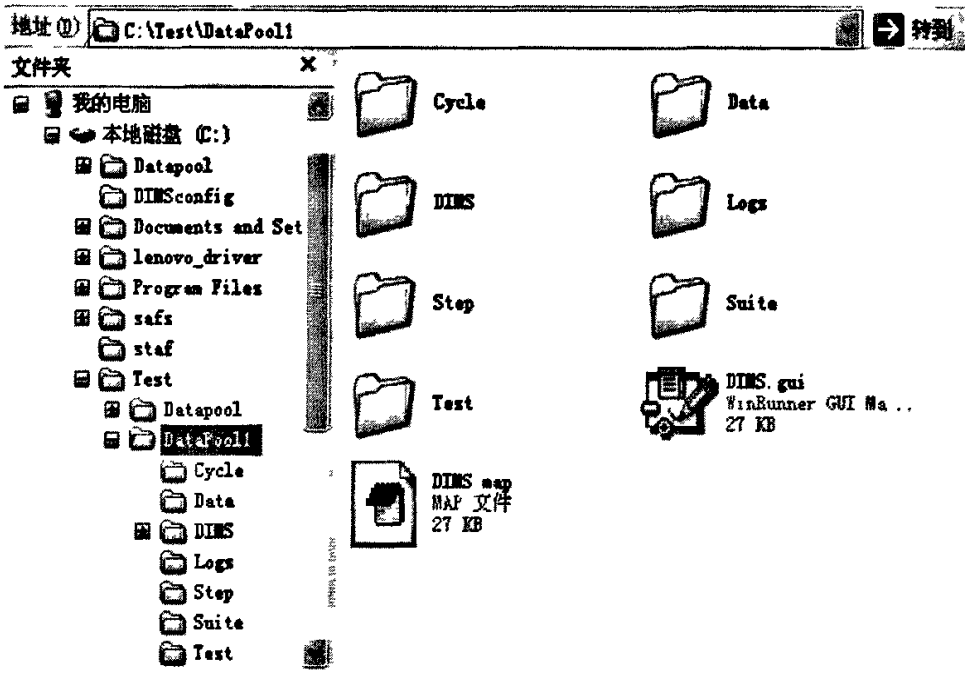

图 5.14 测试工程文件组织结构
Figure 5.14 Structure of Test Project File

一切准备工作就绪之后，首先启动 WinRunner，启动的过程中要加载 WebTest 插件才能对 DIMS 系统进行测试。然后运行 StartCycleTest 脚本，测试就开始执行了。测试执行过程中，控制流会根据模块化测试列表中的记录类型在核心数据驱动引擎之间转换，依次执行测试列表内的每一条记录，直到所有记录执行完毕，测试结束。采用此测试框架一旦测试开始执行，就不需要人为干预，测试完全自动化模拟人的点击鼠标对系统进行操作，若在测试过程中出现实际输出与预期输出不一致的错误，测试并不会中断，WinRunner 会将错误记录在日志中，跳过错误继续执行，执行完毕后，WinRunner 会自动生成测试结果报告，指示测试成功或失败，以及失败的原因。

5.5 自动化测试结果分析

WinRunner 执行测试的过程中会将测试过程中的实际输出和预期输出进行比较，判断被测应用程序是否满足测试要求。测试完成之后会自动弹出测试结果对话框，如图 5.15 所示。

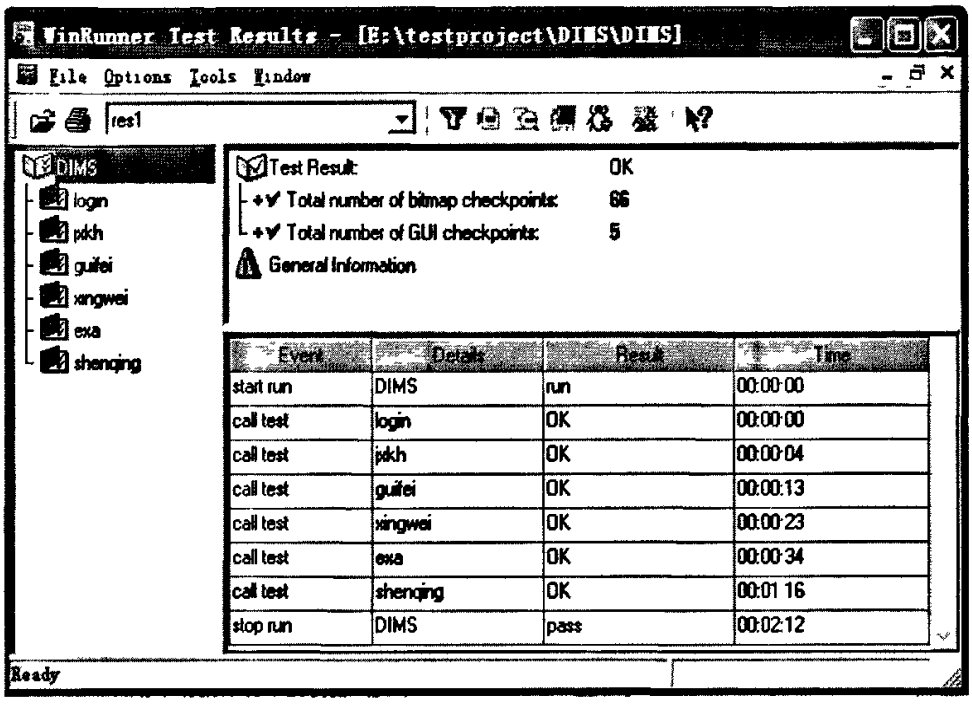

图 5.15 DIMS 总体测试结果

Figure 5.15 Whole Test Results of DIMS

图中显示了 DIMS 总体测试结果，双击左侧某一模块测试用例的图标，就能查看该模块具体执行情况。由于在控制脚本插入了检测点，因此在测试结果中会记录每一个检测点的执行情况，图 5.16 表示登录模块测试结果。右下方测试结果表中绿色字体显示的部分表示检测点通过测试，若检测点未通过测试，则该条记录用红色显示。双击检测点结果记录，可以查看检测点的期望值和实际值。例如图 5.17 显示检测索引页面绩效考核模块链接是否正确的检测点执行结果。若检测点未通过测试，可以通过图 5.17 中所示的期望结果与实际测试结果的值来判断被测程序出现怎样的错误。

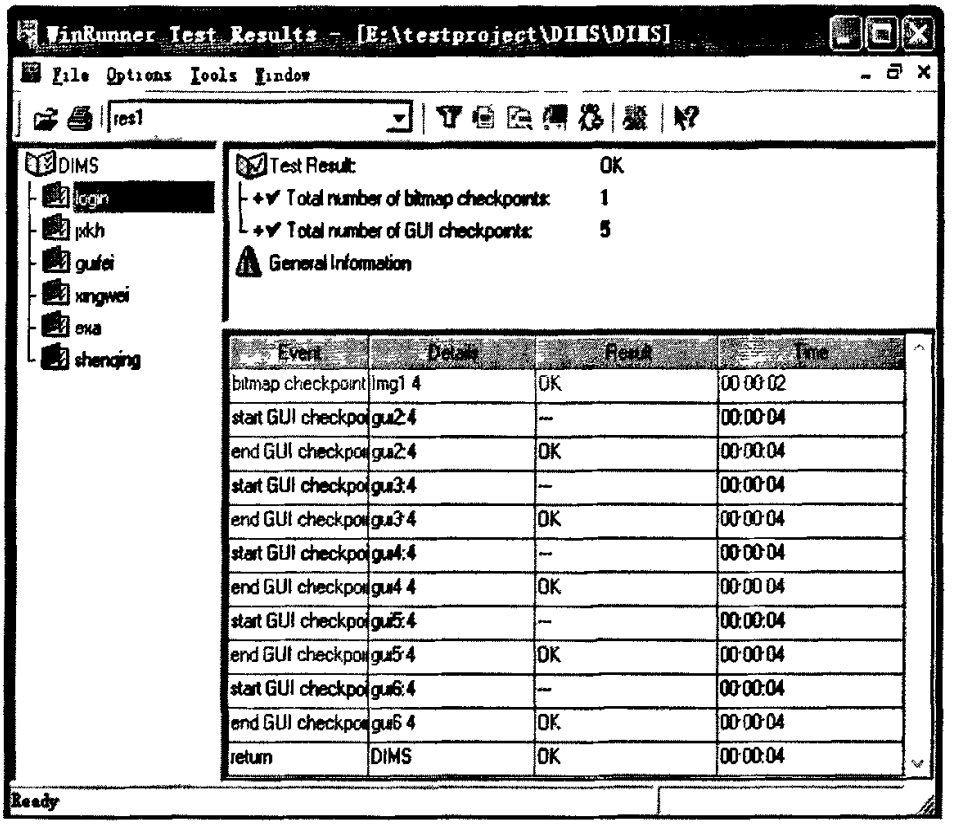

图 5.16 登录模块测试结果

Figure 5.16 Test Result of Login Module

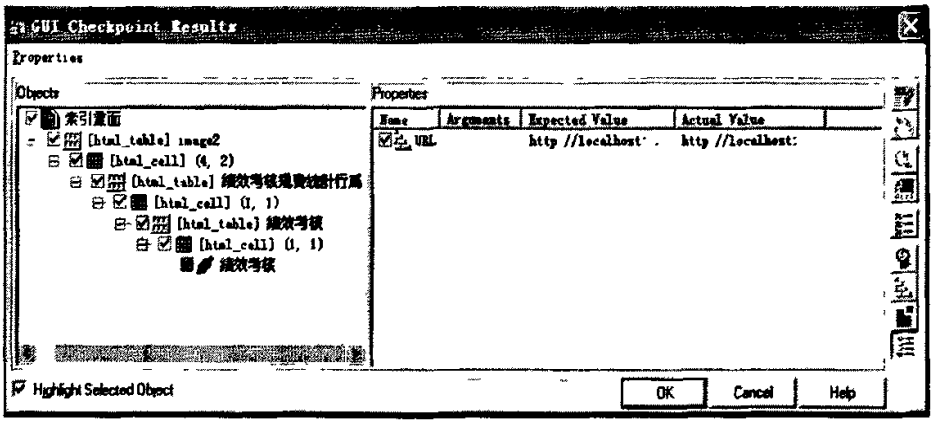

图 5.17 检测点结果

Figure 5.17 Checkpoint Result

资讯确认页面的 GUI 检测点结果也可在测试结果记录中查看，图 5.18 为行为

统计模块资讯确认页面“确认信息”表格检测点的测试结果。双击 TableContent 属性，即可查看期望值与实际值，如图 5.19 所示。

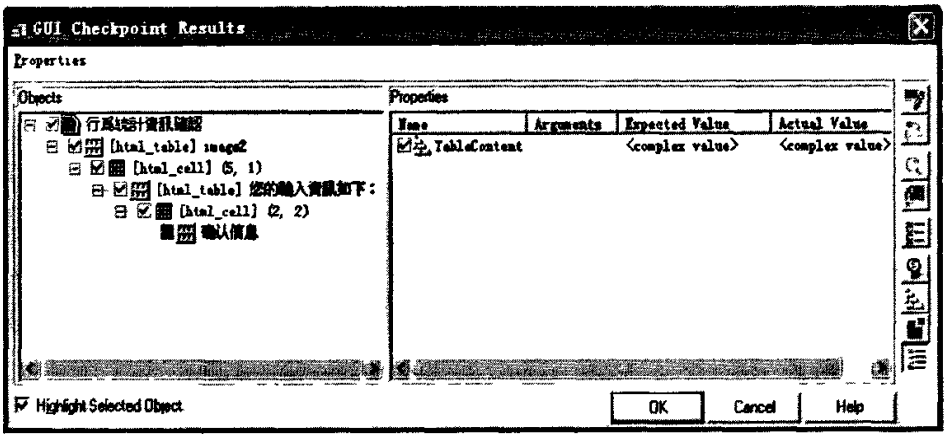

图 5.18 “确认信息”表格检测点结果

Figure 5.18 Checkpoint Result of “Confirm Information” Table

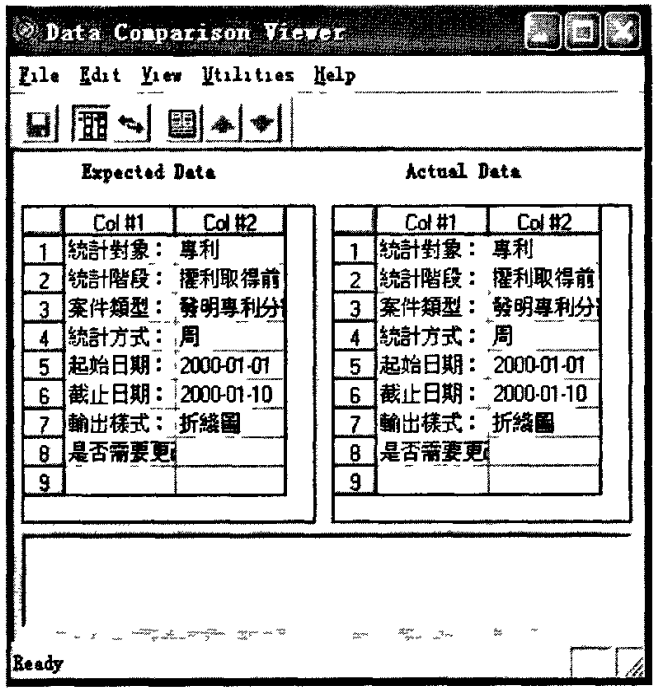

图 5.19 Data Comparison Viewer

Figure 5.19 Data Comparison Viewer

分析完测试结果，需要根据自动化测试结果编写软件缺陷报告，在报告中以明显、通用和再现的形式描述被测应用程序中存在的缺陷，并对软件缺陷进行分类，以简明扼要的方式指出其影响。软件测试人员和开发人员共同分析产生缺陷的原因，并进行相应的调整，修复软件缺陷。然后，再将修改过的应用程序提交给软件测试人员重新进行测试，确认缺陷是否被修复。待被测应用程序达到测试目标之后，再编写软件测试文档，对整个测试过程进行记录，并作出最终的评价。

通过利用软件自动化测试混合框架对 DIMS 系统进行测试，系统已经达到软件需求规格说明书的功能要求。并且给定有效的统计输入，系统能够产生正确的统计输出。给定无效的输入，系统能够向用户产生错误提示。

6 结论

本文在研究了目前几种自动化测试框架的优缺点之后提出了一种继承各个框架优点的软件自动化测试混合框架，并基于自动化测试工具 WinRunner 的 WRSAFS 数据驱动引擎实现了此测试框架，最后在实际的项目中运用此测试框架实施自动化测试。通过研究总结出软件自动化测试混合框架具有以下优点：

（1）模块化测试列表代替了传统的捕捉/回放测试脚本。过去的经验告诉我们，无法依靠传统的 GUI 自动化工具来完成测试，因为捕捉产生的测试脚本还需要测试人员修改、完善之后才能运用，这就要求测试人员需要对测试脚本语言很了解，并具备一定的编程经验；此外捕捉产生的脚本维护困难，而且生存期很短。模块化测试列表解决了上述问题，它将测试人员从繁琐的脚本语言编程中解脱出来，使他们把更多的精力放在如何设计好的测试用例上。并且模块化测试列表还将传统测试脚本中变化与不变的东西进行了分离，这种分离使得分工更加明确，并且避免了它们相互之间的影响，从而降低了测试脚本的维护成本。

（2）应用映射文件实现了界面元素名与测试内部对象名的分离。它在被测应用程序和测试列表之间增加了一个抽象层，将界面上的所有元素映射成相对应的逻辑对象，测试只针对这些逻辑对象进行，界面元素的改变只会影响映射文件，而不会影响测试，从而降低了测试维护的成本。

（3）模块化测试列表的分层结构实现了测试描述与具体实现细节的分离。在模块化测试列表中，高层测试表和中层测试表相当于测试描述，只用来说明测试要做什么以及期待怎样的结果，而低层测试表才涉及到测试具体实现的细节。这样做是因为测试的实现细节通常和特定的平台，以及特定的测试执行工具有着密切的联系。这种分离使得测试描述对于应用实现细节是不敏感的，而且有利于测试在工具和平台间的移植。当测试需要在不同的平台上进行，或者需要使用不同的测试工具时，只需要对低层测试表进行修改，使其符合所用的测试平台和工具即可。

（4）使用专门的数据文件存放测试所需的数据。设计测试列表时，用有意义的变量代替实际的数据，待测试执行时再从数据文件中的获取指定的数据，这样数据和测试列表可以独立维护，同时可以使同一组测试列表实现不同的测试用例，提高了测试列表的复用性。

当然由于受到各方面因素的限制，本文所提出的软件自动化测试混合框架还存在不足和需要改进的地方，主要有以下几个方面：

（1）错误处理机制不够健全，只能对简单的错误进行处理，当出现测试框架不能解决的错误时测试就会中断，需要人为干预才能继续执行。

（2）在设计组件函数库中的软件模块时，只对 WinRunner 提供的常用应用程序组件函数进行了封装，若在测试中发现没有被组件函数库处理的组件，就要修改或添加处理该组件的组件函数，使组件函数库适用范围更广。

（3）支持库是一般目的的程序和工具，它对自动化测试框架提供最基础的支持，在今后的工作中，还应增加支持库的处理能力，例如缓冲处理、数据库访问、并发控制等，从而使支持库能够为测试框架提供更多支持，增强测试框架的功能。

随着自动化测试技术在软件测试领域的逐步发展，将会有更多的自动化测试框架出现。但无论是哪种测试框架都将会按照以下几个原则设计：（1）使自动化测试框架可以为那些不是程序员的测试人员使用；（2）简化脚本开发；（3）减少维护成本。我相信经过自动化测试技术的发展，自动化工具的不断完善和自动化测试经验的积累，自动化测试替代手工测试的工作会越来越多，从而使自动化测试成为对手工测试的有利补充，更好地保证软件测试的质量。

参考文献

[1] 蔡军红．软件质量与测试(一)．软件世界，2002，6：141-143

[2] Edward Kit．Software Testing In The Real World：Improving the Process．1995

[3] 张丽波．基于自动化的软件测试与应用．华南理工大学硕士论文，2004．5

[4] 孙惠杰．软件测试研究以及应用．哈尔滨工程大学硕士论文，2003．5

[5] http://www.shstc.org.cn/

[6] http://www.cstc.org.cn/

[7] 中国软件评测中心，中科软件股份有限公司．中国计算机报，2003．9

[8] Bruce A. Posey，Just Enough Software Test Automation．北京：机械工业出版社，2003．10

[9] Elfriede Dustin 等著．于秀山，胡兢玉等译.软件自动化测试：引入、管理与实施．北京：电子工业出版社，2003．1

[10] 孙惠杰，杨晓红．软件测试的自动化．哈尔滨师范大学自然科学学报，2003，19(5)：47-49

[11] 张丽波．软件自动化测试的设计与实施．佳木斯大学学报，2004，22(4)：568-572

[12] Michael Kelly．Frameworks for Test Automation．SQAteste.com columnist

[13] 朱菊，王志坚，杨雪．基于数据驱动的软件自动化测试框架．计算机技术与发展，2006，16(5)：68-70

[14] Daniel J.Mosley 著．邓波等译．软件测试自动化．北京：机械工业出版社，2003．10

[15] 冯玉才，唐艳，周淳．关键字驱动自动化测试的原理与实现．计算机应用，2004，24(8)：140-142

[16] http://safsdev.sourceforge.net/Default.htm

[17] Carl J.Nagle．Test Automation Frameworks whitepaper．http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm

[18] WinRunner Tutorial，Version 7.6．Mercury Interactive Corporation，2003

[19] WinRunner TSL Reference Guide，Version 7.6．Mercury Interactive Corporation，2003

[20] Ron Patton 著．周予滨，姚静等译．软件测试．北京：机械工业出版社，2003．3

[21] 郑人杰著．计算机软件测试技术．北京：清华大学出版社，1992

[22] 张莉．软件测试方法和工具选择．成都教育学院学报，2005，19(7)：105-106

[23] 宋艳芳，苏哲明，史永辉．自动化软件测试．应用科技．2001，28(4)：24-25

[24] 张佳玥．一组软件自动化测试工具．现代通信，2001，12：27-28

[25] 朱三元，宿为民．近期软件测试工具分析．计算机应用与软件，1992．

附录 A

应用映射文件

说明：由于篇幅有限，在此只列出应用映射文件部分内容，包括 ApplicationConstants 元素和部分 Windows 元素

```
[ApplicationConstants]
DIMSSite="http://localhost:8080/dimsweb/dimsindex.jsp"
DIMSPath="D:\jboss-4.0.0\server\default\deploy\dimsweb.war"
userName="dims"
userPassword="dims"

[Browser Main Window]
Browser Main Window={class: "window", MSW_class: "browser_main_window", NSTitle:
"Browser Main Window"}

[Login]
Login={class: "window", MSW_class: "html_frame", html_name: "Login"}
userName={class: "object", MSW_class: "html_edit", html_name: "userName"}
Password={class: "object" MSW_class: "html_edit", html_name: "userPassword"}
Login={class: "object" MSW_class: "html_push_button", html_name: "login"}
Reset={class: "object" MSW_class: "html_push_button", html_name: "reset"}

[索引畫面]
索引畫面={class: "window", MSW_class: "html_frame", html_name: "索引畫面"}
績效考核={class: "object", MSW_class: "html_text_link", html_name: "績效考核"}
規費統計={class: "object", MSW_class: "html_text_link", html_name: "規費統計"}
行爲統計={class: "object", MSW_class: "html_text_link", html_name: "行爲統計"}
專利案件審查過程結果統計={class: "object", MSW_class: "html_text_link", html_name:
"專利案件審查過程結果統計"}
專利申請案件統計={class: "object", MSW_class: "html_text_link", html_name: "專利
申請案件統計"}

[績效考核首頁]
績效考核首頁={class: "window", MSW_class: "html_frame", html_name: "績效考核首頁
"}
```

achstat={class: "object", MSW_class: "html_rect", html_name: "achstat.gif"}
expstat={class: "object", MSW_class: "html_rect", html_name: "expstat.gif"}
behstat={class: "object", MSW_class: "html_rect", html_name: "behstat.gif"}
pexastat={class: "object", MSW_class: "html_rect", html_name: "appstat.gif"}
pappstat={class: "object", MSW_class: "html_rect", html_name: "appstat.gif"}
assessType_0={class: "radio_button", MSW_class: "html_radio_button", html_name: "assessType", part_value: "承辦人員"}
assessType_1={class: "radio_button", MSW_class: "html_radio_button", html_name: "assessType", part_value: "主管"}
assessType_2={class: "radio_button", MSW_class: "html_radio_button", html_name: "assessType", part_value: "專利案件階段"}
assessType_3={class: "radio_button", MSW_class: "html_radio_button", html_name: "assessType", part_value: "商標案件階段"}
assessType_4={class: "radio_button", MSW_class: "html_radio_button", html_name: "assessType", part_value: "單位"}
下一步={class: "push_button", MSW_class: "html_push_button", html_name: "下一步"}
重置={class: "push_button", MSW_class: "html_push_button", html_name: "重置"}

[績效考核單位選擇頁面]
績效考核單位選擇頁面={class: "window", MSW_class: "html_frame", html_name: "績效考核單位選擇頁面"}
achstat={class: "object", MSW_class: "html_rect", html_name: "achstat.gif"}
expstat={class: "object", MSW_class: "html_rect", html_name: "expstat.gif"}
behstat={class: "object", MSW_class: "html_rect", html_name: "behstat.gif"}
pexastat={class: "object", MSW_class: "html_rect", html_name: "appstat.gif"}
pappstat={class: "object", MSW_class: "html_rect", html_name: "appstat.gif"}
dpartment={class: "list", MSW_class: "html_combobox", html_name: "department"}
下一步={class: "push_button", MSW_class: "html_push_button", html_name: "下一步"}
上一步={class: "push_button", MSW_class: "html_push_button", html_name: "上一步"}
重置={class: "push_button", MSW_class: "html_push_button", html_name: "重置"}

附录 B

模块化测试列表

高层测试表	中层测试表	对应测试用例集名称	低层测试表名称
CycleDriver.CDD	LaunchApp. STD	用户登录	LaunchDIMS. SDD
			Login. SDD
	AchTest. STD	绩效考核	IndexAch. SDD
			SelectAchStatType0. SDD
			SelectDepar. SDD
			SelectAchStat0. SDD
			ConfirmAchStat
			AchChart. SDD
			SelectAchStatType1. SDD
			SelectCaseType. SDD
			SelectAchStat1. SDD
			ConfirmAchStat
			AchChart. SDD
			SelectAchStatType2. SDD
			SelectDepar. SDD
			SelectAchStat2. SDD
			ConfirmAchStat
			AchChart1. SDD
	ExpTest. STD	规费统计	SelectExpFirst. SDD
			SelectExpStat. SDD
			ConfirmExpStat. SDD
			ExpChart. SDD
	BehTest. STD	行为统计	SelectBehFirst. SDD
			SelectBehStat. SDD
			ConfirmBehStat. SDD
			BehChart. SDD
	ExaTest. STD	案件审查过程及结果统计	SelectExaFirst0. SDD
			SelextExaSecond. SDD
			SelectExaThird. SDD
			SelectExaStat0. SDD

高层测试表	中层测试表	对应测试用例集名称	低层测试表名称
CycleDriver.CDD	ExaTest.STD	案件审查过程及结果统计	ConfirmExaStat.SDD
			ExaChart.SDD
			SelectExaFirst1.SDD
			SelextExaSecond.SDD
			SelectExaStat1.SDD
			ConfirmExaStat.SDD
			ExaChart.SDD
			SelectExaFirst2.SDD
			SelextExaSecond.SDD
			SelectExaStat2.SDD
			ConfirmExaStat.SDD
			ExaChart.SDD
			SelectExaFirst3.SDD
			SelextExaSecond.SDD
			SelectExaStat3.SDD
			ConfirmExaStat.SDD
			ExaChart.SDD
	AppTest.STD	申请案件统计	SelectAppFirst0.SDD
			SelectAppStat0.SDD
			ConfirmAppStat.SDD
			AppChart.SDD
			SelectAppFirst1.SDD
			SelectAppStat1.SDD
			ConfirmAppStat.SDD
			AppChart.SDD
			SelectAppFirst2.SDD
			SelectAppStat2.SDD
			ConfirmAppStat.SDD
			AppChart.SDD
	CloseApp.STD	用户退出 DIMS	ExitDIMS.SDD

作者简历

教育经历：

2000年9月至2004年7月，北京交通大学计算机与信息技术学院获得学士学位，计算机科学与技术专业。

攻读学位期间发表的论文：

应用LoadRunner进行Web应用程序性能测试，《计算机科学与实践》，第四卷第十二期，2006年12月，第一作者。

攻读学位期间参与的项目：

2005年9月至2006年7月参与台湾智慧财产局决策信息管理系统（DIMS）的设计、开发与测试。

学位论文版权使用授权书

本学位论文作者完全了解北京交通大学有关保留、使用学位论文的规定。特授权北京交通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，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

学位论文作者签名：

签字日期：2016 年 12 月 25 日

导师签名：

签字日期：2016 年 12 月 25 日

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和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通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了谢意。

学位论文作者签名：[illegible]　　签字日期：2006 年 12 月 25 日