软件工程技术丛书

测试系列

Just Enough Software Test Automation

软件测试自动化

能提高软件质量并减少上市之前时间的实用自动化测试技术

本书是一本从测试开发人员和用户角度考虑的实际可用的指导软件测试自动化的书。两位优秀的软件测试顾问讲述了在真正的测试自动化基础设施设计和实施中能够做的和不能做的工作——还有一些实际的建议告诉读者现今最流行的自动化测试方法所能完成的和不能完成的工作。其内容涵盖：

- 设定现实的预期：了解何时进行自动化与什么可以进行自动化
- 对自动化测试进行计划
- 实现控制同步数据驱动测试（CSDDT）框架，这是一个已被证明可以简化并加快测试速度的方法
- 使用结构化的测试脚本以简化测试脚本的维护并提高重用性
- 自动化单元测试、集成测试、系统/回归测试
- 管理自动化测试过程以优化效率

本书还包括一个完整的自动化项目计划的例子，其中包括完整文档、实现、自动化环境、角色、责任等等。

http://www.phptr.com/mosley这个站点是一个FTP链接，其中有本书中所描述的所有方法在自动化测试项目中应用所需要的信息和工具资源。

作者简介

DANIEL J. MOSLEY 是客户机－服务器软件测试技术的创始人。他也是《The Handbook of MIS Application Software Testing》和《Client Sever Software Testing on the Desktop and Web》两本书的作者。Mosley是一位CSTE（认证软件测试工程师），他是质量保证研究所（Quality Assurance Institute）的一名高级顾问和研讨班主管。他的著作还有《TEST-RxTM Methodology》。

BRUCE A. POSEY 的特长是使用SQA套件和Rational小组测试开发和实现数据驱动、基于框架的测试脚本。他有将近30年的IT从业经验，曾在MasterCard、Deutsche Financial Services、SBC和其他杰出的公司做过多种工作。现在他是Archer Group的董事和首席顾问，该公司的业务主要是软件测试和培训。

适用水平：中、高级

封面制作 锡彬

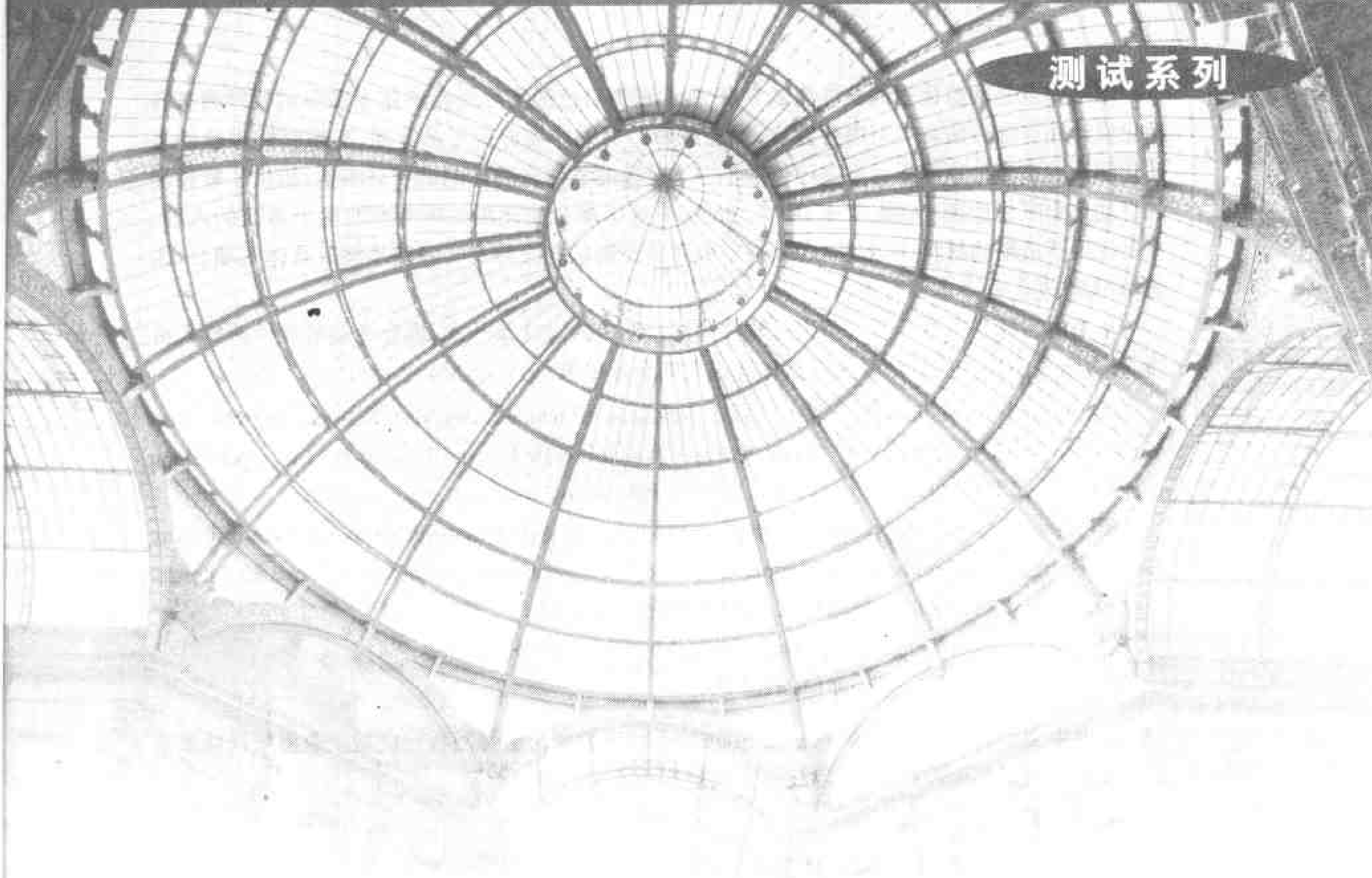
ISBN 7-111-12818-4
9 787111 128182

华章图书

网上购书：www.china-pub.com
北京市西城区百万庄南街1号 100037
读者服务热线：(010)68995259，68995264
读者服务信箱：hzedu@hzbook.com
http://www.hzbook.com
ISBN 7-111-12818-4/TP · 2874
定价：29.00 元

软 件 工 程 技 术 丛 书

测试系列



软件测试自动化

Just Enough Software Test Automation

（美） Daniel J. Mosley
Bruce A. Posey 著

邓波 黄丽娟 曹青春 等译

机械工业出版社
China Machine Press

本书作者是两位优秀的软件测试专家，他们根据自己的实际经验讲述了实际的软件测试自动化设计和实施方法。本书内容翔实，概念清晰。书中详细分析了从计划、实施到管理软件自动化测试的整个处理过程。本书还包括了一个完整的软件自动化测试项目实例，给出了软件测试自动化中项目计划、文档、实施、环境、角色、责任等等自动化测试应考虑的各个方面的内容。本书使用了最新的材料，时效性很好，书中所列各种测试方法都有实际的步骤及具体的项目，因而操作性较强，是一本不可多得的好书。

本书的目标是指导开发人员进行软件测试自动化的设计及开发，适合专业的软件测试人员阅读，对于关注软件质量的开发人员也有很大的参考价值。

本书版权登记号：图字：01-2003-1001

图书在版编目(CIP)数据

软件测试自动化/(美)莫斯里(Mosley, D. J.)等著；邓波等译. －北京：机械工业出版社，2003.10

(软件工程技术丛书　测试系列)

书名原文：Just Enough Software Test Automation

ISBN 7-111-12818-4

Ⅰ.软…　Ⅱ.①莫…②邓…　Ⅲ.软件－测试－自动化　Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字 (2003) 第 068433 号

机械工业出版社(北京市西城区百万庄大街 22 号　邮政编码　100037)

责任编辑：李　英

北京昌平奔腾印刷厂印刷 · 新华书店北京发行所发行

2003 年 10 月第 1 版第 1 次印刷

787mm×1092mm　1/16　·15 印张

印数：0 001-5000 册

定价：29.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线电话：(010)68326294

译者序

在IT产业迅速发展的今天,硬件在质与量方面的飞速发展给人们留下了深刻的印象。相对而言,软件在量的方面同样发展迅速,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疼的问题,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。面对这种挑战,人们提出了很多软件过程方法,力图通过严谨的开发过程保证软件的质量。这些努力当然是创造高质量软件产品的有效方法,但对软件进行测试仍然是保证软件质量最重要和最有效的方法。

软件规模的扩大同样给测试工作带来了新的问题,手工测试的速度太慢,效率太低。因此人们想到了自动化测试,想通过脚本程序的运行让测试工作自动进行。然而哪些测试内容可以自动化,什么时候进行自动化,这些都是常让测试人员困惑的问题,因为测试自动化是一种新的测试方法,没有什么经验可循。

本书的两位作者都是软件测试自动化的专家,在业界最早进行了数据驱动自动化测试的实验并取得了成功。在本书中,他们通过深入浅出的论述和实际的例子向读者介绍了软件测试自动化方法。不同于泛泛而谈,本书中介绍的方法都是以实际案例为基础进行论述和分析的,因面具有极强的操作性,相信各位读者看完全书后也会有自已的体会。对于读者而言,本书致力于讨论软件自动化测试的以下问题:

- 实施自动化测试时如何考虑测试过程?
- 什么类型的测试可以自动化实现?
- 自动化测试的时机如何确定?
- 自动化测试的内容是什么?
- 实施自动化测试时如何在时间和代价约束下考虑自动化测试的有效程度?

本书的目的是尽量提供对软件自动化测试概念和实施过程的清晰认识和理解。

本书共有10章:第1、2章介绍了软件测试自动化的概念及其应用场合;第3章讲述了自动化测试的准备工作,包括测试需求和测试数据的定义;第4章介绍了自动化测试脚本的相关知识;第5、6、7章分别详细讲述了单元测试、集成测试、系统/回归测试的具体方法和内容;第8章深入介绍了CSDDT的知识;第9章的内容为手工测试与自动化测试的结合;第10章讲述了自动化测试的管理。此外本书的附录还包括关于软件测试自动化的网上讨论、自动化实施的方案模板。这些对于读者进一步了解学习和实际应用都会有所帮助。

我们相信,本书不仅对软件测试人员提供有效帮助,对关心软件质量和软件过程控制的开发人员也有非常大的参考作用。希望本书的翻译出版能为提高国内软件自动化测试水平作出一点贡献。

全书由邓波、黄丽娟、曹青春、崔晓斐、王梅蓉、李昂、金旭、赵宏智、卢世凤、齐悦、郝敏、曹振南、韦华颖、刘学敏等进行翻译。本书最后由邓波统稿。由于时间仓促，且译者的水平有限，在翻译过程中难免会出现一些错误，请读者批评指正。

译　者

2003年5月

前 言

迄今，市面上已经有了很多关于测试自动化的书籍。它们所提倡的软件测试自动化方法各不相同，级别也有所差异。当然也有人尝试过将软件测试自动化的生命周期用最大众化的方式描述出来(参看第2章中Dustin的论述)。传统上，软件人员通常的做法是尝试用假设的模型来描述我们使用的过程。有些时候这种方法行得通，但有些时候它是行不通的。其问题在于没有经验性的证据能够证明在这些模型上使用的方法是否能够在现实世界中工作。对于大多数受到推荐的软件测试及软件开发方法来说，其基础只是轶事逸闻和项目管理的狂热，而这些是由那些所谓的信息系统专家发起并被首席信息官(Chief Information Officer, CIO)们推动发展的。

我们不相信自动化测试的生命周期。这只是一个人工构造的概念，而且我们发现它没什么用处。我们也不相信软件测试生命周期，我们所真正相信的是软件测试是属于软件开发过程中的一组紧密相关的活动。我们还相信好的软件测试需要有特定规范的项目管理以及一套它自己的操作技巧。测试也需要一套测试工程师在进行测试活动时能够依赖的工具。这些工具可以是与测试相关的产物，例如测试工程师可以依据的打印出来的测试场景，或者是在执行测试时工程师可以填写的打印出来的测试日志。

我们并非是说必须等到有了诸多经验证据之后才能接受并使用IS权威们所鼓吹的工具和技术。我们的意思是我们必须对这些工具和方法进行评估并去其糟粕。真正行之有效的经验是：当从业者(你和我们)尝试使用这些技术时，能够知道哪种技术可行而哪种不行。

本书第一作者Dan Mosley探讨了他在1985年使用的第一个自动化测试工具，那时的技术还相当简陋。他给本科生教授最早的软件测试课程时获得了该产品的评估版拷贝(圣路易斯的华盛顿大学，1985－1992)。在20世纪90年代中期，他与当时在Software Quality Automation (SQA)公司、现在在Rational Software公司(该公司后来吞并了SQA及其产品，其业务主要是提供自动化软件测试工具以及软件测试工程师所需要使用的东西，而译者现在得知Rational Software公司业已被IBM购买，但仍保持Rational品牌)的Eric Schurr通了长途电话，作了一次长谈。

他们讨论了一个好的自动化工具应有的功能和应该包括的特征。通过与Eric的联络，Dan获得并使用了SQA(现在是Rational)的自动化测试工具，那是在早期的1.0版后面出来的版本。现在最新的版本(写本书时)是"Rational Suite TestStudio 2002"。Dan对这个产品的使用经验表明，测试自动化不是对草率的测试工作的简单仓促的修正。此外，他的经验还证明了自动化测试不能代替手工测试。Glen Myers在20世纪70年代后期提出了软件测试的基本概念。直到今天，他的《Art of Software Testing(1979)》仍被认为是关于软件测试的最早著作。在我们进行手工测试以及构建自动化测试基础设施时我们仍要遵循他的一些建议。

我们对现在测试自动化实践的主要意见是其缺乏前期的测试计划和测试设计活动。我们重复地犯着那些软件开发人员早已成为痼疾的经典错误——我们在还没有对问题做合适的分析以及对测试基础设施进行设计时就开始了测试工作。这令我们想起了几年前的一个卡通片——某

位经理对程序员说:"你开始编写代码吧,我会去研究到底要做什么。"那些"测试"错误东西的自动化测试所带来的结果只有一个:不合适的测试更迅速地被执行完了。

任何自动化测试的最终目标都应当是与一套测试需求相对应的一套有计划的测试,这些测试需求反过来也能在自动化测试中体现出来。此外,测试工作的中心是测试数据,而不是测试脚本。这就是为什么有那么多人鼓吹将数据驱动自动化测试作为自动化实现的框架。其基本前提是数据应该驱动测试的进行并且应该体现被测程序(Application Under Test,AUT)的特征。测试脚本仅仅是数据传送工具。只有当设计出好的测试数据后,自动化测试才会有效。

自动化测试框架的另一个可操作目标是让测试脚本的维护量减至最少。各测试工具供货商多年主打的传统"捕获/回放"模型导致了高得异乎寻常的脚本维护量,因为测试数据在测试脚本程序中是以硬编码方式实现的。Mosley 开发自动化测试脚本的最初经验明确地表明严格使用"捕获/回放"是行不通的。此外,工具内建的测试用例除了测试应用程序的图形用户界面(GUI)外,实际上是没有用处的。真正的功能测试需要测试工程师编写对 AUT 进行深度探查的测试数据。GUI 测试能够用自动化测试做,并且用最少的工作量就能实现它的自动化。在实际测试中,我们通常用一个单独的测试脚本处理 GUI 对象,它会对所有 GUI 对象的属性进行基准和确认测试,这个测试脚本在每一个 GUI 显示屏都会执行。

自动化功能测试需要复杂全面的测试数据对 AUT 进行测试。这些测试数据必须能够重复产生演示重要系统特征的测试场景。因此,与其他测试相比,自动化功能测试更加复杂也更加困难。它要求测试工程师写出测试脚本的主要部分而不是记录测试。它还意味着要设计有效的测试数据。理解要测试什么(有一套书面的测试需求)并且设计出验证这些需求的数据是编写有价值的自动化功能测试的关键所在。

理解如何对测试结果进行验证与知道测试什么同等重要。自动化测试验证也具有数据依赖性。作为最主要的验证方法,自动化测试经常将基准数据捕获并存储起来,稍后与回归测试中捕获的相同数据进行比较。更全面的测试会在测试执行之前、之中和之后对数据库记录进行访问和操作。

功能强大的自动化测试框架必须提供测试计划、测试设计、测试构建、测试执行和测试结果验证的工具。一个有效的测试基础设施是建立在集成的中央知识库之上的,测试相关产物可以存储在这个知识库中并能够重用。这个基础设施还应提供可以定制的报告功能。

我们两人在合作此书之前就有过合作交流的经历。1996 年我们碰到了一起,一同做出了到那时为止第一个真正成功的自动化测试项目。从那时起,我们在很多测试自动化项目中都相互合作。我们逐渐知道了实施成功的自动化测试项目需要什么,以及在 IS 研发和测试组织中推广自动化测试需要什么。我们知道什么可行,同样也知道什么不可行。

经过共同努力,我们实现了一个数据驱动自动化测试的范型,在此之前我们甚至还没有听说过"数据驱动测试"这一现在已经泛滥的业界名词(我们不知道 Richard Strang 在 1996 年 STAR 会议上发表的文章;参看第 1 章)。在他人刚开始探讨及写作相关文章时,我们已经率先对数据驱动方法进行了实现和完善。当然,对于任何新的热门技术来说,它并不是真正全新的,仅仅是被重新发现了。数据驱动测试也不例外。Berzer 在《Software Testing Techniques(1983)》一书中就描述过"基于数据驱动的测试设计"。该书出版的时间是大型机时代的晚期以及 PC 革命的早

期、所以它的思路是关于大型机应用程序测试的。他提出的这种过程不是为了测试数据库,而是为了使用数据库结构生成测试数据。他认为该方法"最适合验证系统规约中所规定的功能需求"。通过很简单的步骤就可以将这种方法扩展,以包括那些基于被数据库表结构所支持的业务规则的测试。将测试 GUI 对象和它们行为的数据加上,你就得到了数据驱动测试。

这一时期我们还进行了结构化测试脚本的编写(也称作基于框架的测试)。这也不是什么新的技术。测试脚本就是使用修改过的普通编程语言(VB 或 C)编写的程序。它们的不同之处在于它们有一些为了软件测试任务而改造的特征。有很多文献都讲述了"结构化程序设计"的概念,例如功能分解、模块结合、模块耦合及模块(功能)重用。由于测试脚本包含的程序是要对另一个程序进行测试,所以它也要遵循被测程序所遵循的设计和构造规则。因此,结构化程序设计概念也适用于自动化测试脚本。

因为自动化测试脚本也背负着与其他软件编程相同的包袱,所以它们也有逻辑错误、死循环、硬编码变量等等问题,它们能够由过程和数据一同在测试脚本中实现。所有这些加起来就增加了相关测试套件的维护开销,就像测试脚本所测试的软件系统需要相应的维护开销一样。创建结构化的基于组件的测试脚本并让这些脚本与其所执行的数据相分离是创建有效软件测试自动化基础设施的惟一途径,这样可以达到最佳的测试精度并将测试维护减到最少。

最近还有人在开发一种高级的自动化语言,通过使用这种语言,非技术人员例如商业及产品分析人员或系统消费者也可以设计并实施自动化测试。这种尝试被认为是自动化测试发展的下一个阶段。然而迄今为止,我们还没有看到一种足够简化的测试脚本开发方法真正能够得到应用。只有在研究出一套可以支持公共的类 Java 的测试脚本语言的通用脚本库后,我们才会发现这样做的意义。然而,迄今的多数框架所支持的是精心设计的高级命令语言,而不是面向对象的语言。此外,支持库在提供给脚本编写者的功能方面还不够成熟。为了完成他们的测试需求,各机构必须在现存库的子例程和功能之上添加额外的代码。

由于我们是从业者,所以本书的目标是从开发者和用户的角度提供关于测试自动化的有用建议。这其中包括在设计和实现测试自动化基础设施时应该做什么的实际建议和不应做什么的一些告诫。此外还包括对现在流行的测试方法可以和不可以为你的测试做哪些事情的一些建议。

我们的例子是在 Rational Suite TestStudio 平台上开发的,但是我们认为它们可以很容易地被移植到其他自动化测试平台上使用。此外,本书还有一个 FTP 支持站点(www.phptr.com/mosley)。这个站点中既包括 ArcherGroup 的 CSDDT(Control Synchronized Data Driven Testing,控制同步的数据驱动测试)方法的模板文件,也包括 Carl Nagle 的 DDE(Data Driven Engine,数据驱动引擎)方法(适用于 Rational 环境)的模板文件,还有 Keith Zambelich 使用 Mercury Interactive的 WinRunner 自动化测试工具的完全数据驱动方法(Totally Data-Driven),该方法的基础是 Zambelich 的"测试计划驱动(Test Plan Driven)"框架并使用了他为WinRunner开发的工具包。通过使用这些资源,你可以轻松地立刻开始进行数据驱动自动化测试。

目录

第 1 章 什么是测试自动化

本书并不论述怎样选择和实现自动化的测试工具包，因为目前有很多关于软件测试自动化工具选择的文章和书；本书也不是关于软件测试自动化的介绍性书籍。本书面向对测试自动化有经验的读者，面向那些在测试自动化实际应用方面遇到严重问题的读者。当然，在你所在的领域，测试自动化可能有也可能没有成功的实现，但无论何种情况，你肯定曾经遇到过测试自动化在操作上、政治上、文化上的缺陷。你需要的是一本指导性书籍，它可以提供一些实用的技巧、建议以及经过检验的方法。如果你想要的是这些，那么本书将适合你。

1.1 请拒绝新模型

“注意：不要新模型！”这句话反映的观点我们完全同意［14］。如同在前言中提到的，目前已经有太多的软件测试过程的模型［6、10、11］和自动化软件测试过程的模型［4、7、8、9、12、15］，包括软件测试自动化生命周期模型［2］。尽管这些模型是完全正确的，并且在讨论软件测试和测试自动化时，某些方面是起作用的，但它们对实际从业者作用甚微。

卡耐基梅隆（Carnege Mellon）大学的软件工程研究所（Software Engineering Institute，SEI）已经建立了软件测试管理关键过程域（Key Process Area，KPA），它对于达到软件过程能力成熟度模型（Capability Maturity Model，CMM）［11］第二级——可重复级（Repeatable）是必须的。软件成熟度模型具有普遍指导作用，但是它并不能从严格意义上定义

一个对测试工程师有用的过程。使用该模型时，会给测试管理者一种既亲切又模糊的感觉，但实质上测试过程活动并没有真正反映该模型。软件测试自动化生命周期模型也是如此。我们并不相信生命周期，相反，我们相信控制工作流程的过程。每一个测试团队都有其工作过程，有些时候它是一个混乱无序的过程，有些时候它是一个有组织的过程。

Krause 为自动化的软件测试提出了四级成熟度模型［3］。在该模型中，他将软件测试成熟度模型［1］和 SEI（软件工程研究所）的软件过程成熟度模型［4］（演变为 CMM）联系起来了。四级成熟度模型中的四级分别为：附属级自动化（Accidental Automation），初始级自动化（Beginning Automation），主体级自动化（Intentional Automation），优化级自动化（Advanced Automation）。尽管这个模型从概念上描述了测试自动化，但实际上它并不能促进测试自动化的实现，它仅仅是描述作者在一些典型测试组织中注意到的一些问题。

Dustin、Rashka 和 Paul 合作公布了自动化测试生命周期方法学（Automated 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［2］。它定义了一种四阶段方法学：自动化测试的决定；自动化测试的介绍；测试计划、设计和开发；自动化测试的执行和管理。

尽管这个模型从管理和控制角度来说是有效的，但是从测试自动化工程师的观点来看，它并不实用。对此，Powers 提出了一些实际建议。这些建议将对那些负责构建和实现测试自动化框架的软件测试工程师很有帮助。他从一般意义上讨论了编程风格、命名规则以及用于编写自动化测试脚本的其他一些惯例［9］。

该书［9］还综合论述了数据抽象的原则。对自动化的软件测试来说，数据抽象是数据驱动方法的基础。在该书中，Powers 还讨论了如何定义数据以及测试脚本如何使用数据的编码的可选方法。他认为“原则就是应尽量少地依赖于变量和常量的字面值，而尽量多地依赖于它们在测试中的意义、角色或用法”。他还提到“产生数据的约束”，他对此定义如下“……这种数据抽象最简单的形式就是用已命名的程序常量来代替字面值”。同时也提到“产生数据的变化”，并举例如下“……数据抽象的原则要求程序员用程序变量来代替字面值，如用‘Sfullnme’来代替‘John Q.Private’，并且在程序中只能给该变量赋值一次。字面值只出现一次意味着要修改该变量的值，只有一个地方可以对它进行编辑。”［9］

Powers所做的论述可以立即给人这样一种印象：当讨论到自动化测试脚本的维护时，数据抽象可以带来益处。他进一步建议将这些值存放到知识库中以便于让测试脚本代码访问，"所需的只是可从中取值的知识库，以及可以实现从知识库中取值的编辑机制。"[9]

以下将讲述Strang的数据驱动自动化测试方法的原则。他的方法使用脚本框架从测试数据知识库中读取所需要的值，这需要用到包含了输入以及它的预期操作的数据文件。他的方法使数据抽象不仅存储字面值，还存储预期的结果值。这种方法既适合于手工数据生成，也适合于自动化的数据生成。而且测试脚本也必须以能够将错误结果和正确结果区分开的方式来编写。[12]

在本书的第7章和第8章讲述数据驱动方法时，会再一次提到Powers和Strang的工作。Archer Group的控制同步数据驱动测试（Control Synchronized Data Driven Testing，CSDDT）是应用这里所讲述的概念的一种方法的例子。

Rational软件公司已经颁布了Rational统一过程（Rational Unified Process，RUP），它包含了具体测试阶段，用于支持它的自动化测试工具包[10]。即使你不是Rational用户，测试过程信息也会给测试（甚至是手工测试）提供良好的基础。RUP本身包含了软件开发所有方面（而不仅仅是测试）的过程文档。而且它的价格并不高——RUP的光盘只卖1 000美元。RUP测试方法最重要的一个方面就是它可以用来支持数据驱动的自动化测试框架。这是我们过去使用它以及本书介绍它的主要原因。

1.1.1 生命周期不是过程

到目前为止，本书中所引用的作者和其他业界权威所采用的测试方法存在着和所有生命周期模型一样的问题，那就是——它们是面向管理层的，而不是面向从业者的。而且，生命周期方法对操作过程（也可称之为自动化的测试过程）起的作用极小。其他的一些方法，如数据驱动自动化测试，正如一些作者批评的那样，虽然提出了很多方法和技巧，但却难以应用到日常的自动化测试活动中。这种直线思维所产生的模型同样给测试管理者一种又亲切又模糊的感觉，正如上面所提到的测试成熟度模型一样。

尽管生命周期模型致力于成为经验模型，但它在自动化测试上的表示并不是在演绎基础上发展起来的。它是基于归纳推理的一种理论，归纳推理又多半以多个案例的证据为基础，正如信息系统（IS）文献所倡导的大

多数模型一样。另一方面，非管理性技术是基于演绎推理的，非管理性技术是操作性的，而不是管理性的，它应用于自动化过程的具体任务。数据驱动测试是非管理性技术的一个例子。这些技术是经过从业者不断尝试和受挫逐渐形成的——这是一种沿用至今的传统工程方法。

1.1.2 工具不是过程

关于CSST技术公司的一个小规模调查结果表明：40%（258份返回问卷中的102份）的人认为软件测试方法或过程的实现最大程度地促进了他们的测试工作；24%（63份）的人认为改进的软件需求文档是最重要的促进因素；19%（50份）的人认为软件测试标准的实现是最重要的方面；10%（25份）的人认为改进的测试计划才是最重要的因素；只有7%（18份）的人认为花更多的时间来测试将会促进他们的工作。

购买软件测试工具包并不意味着实现了软件测试过程。过程是一系列的步骤，通过这些步骤得到所要达到的目标或生产出产品。这些过程实现测试活动，测试活动导致测试执行和测试产物的生成。自动化的软件工具支持现存的测试过程，并且，当过程是混乱过程时，它能强加一些必需结构到测试活动中。软件测试工具实现失败的一个主要原因就是在购买工具前几乎不存在任何测试过程。

当我们设计和创建自动化测试时，甚至不曾考虑到过程。我们考虑到的只是：任务、进度、人员的分配。对我们来说，软件测试自动化应该达到帮助我们有效工作的程度。如果我们没有商业化的软件测试工具可用，我们也会创建自己的工具或者是使用客户日常在工作站上装载的桌面工具。

图1-1描述了一个测试过程，这个测试过程松散地基于RUP测试方法[10]，它是为我们的一个客户定义的。我们的这种过程方法不同于Rational公司颁布的RUP，因为我们将测试脚本设计视为测试实现的一个部分，然而在RUP中这被认为是测试设计行为。我们之所以不同于RUP是因为我们相信测试设计所需要的技巧中并不包括以前的编程经验，但测试实现需要以前的编程经验。

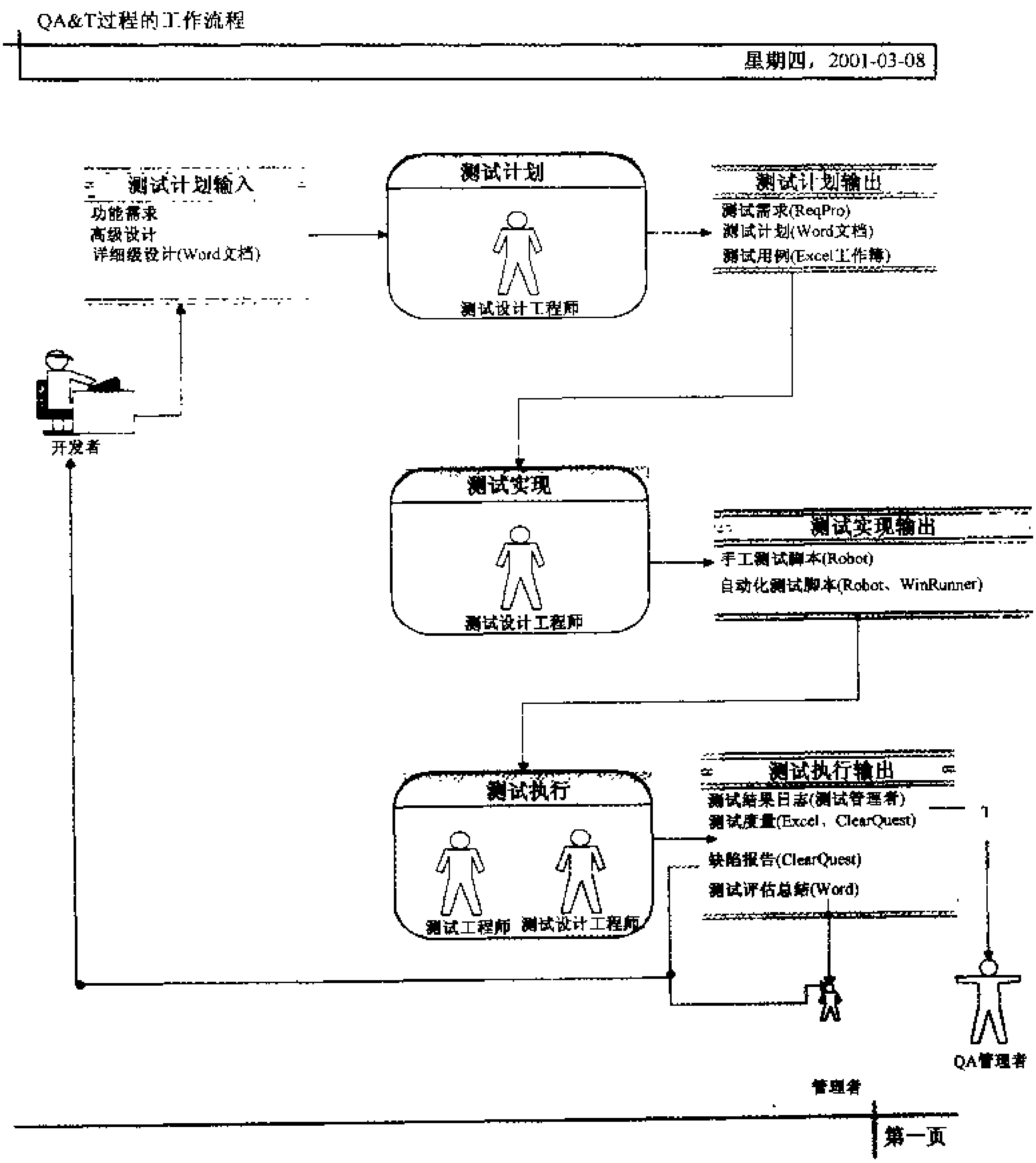


图 1-1　质量保证和测试（QA&T）过程

1.2　自动化需要达到什么程度才足够

这个问题在自动化测试工具发展的最初阶段就有人问了。工具销售商已经给我们提供了一个观点，并且业界专家也给我们提供了各种不同的观

点。最初，销售商提供基本的捕获/回放工具，这些工具已逐渐发展成了一些复杂的高度集成的测试套件。他们似乎想让从业者来决定基本的捕获/回放模型之外的一些东西。测试自动化方面的专家写过很多的文章和书籍，他们引用成功的以及失败的自动化测试来做研究，最后在我们必须做什么上稍微达成一致意见，但是就我们如何来做并没有任何进展。在这一节，我们将给出我们关于如何做测试自动化的观点。我们认为业界就该做什么已经争论太久。我们一直拥有一个能使用的自动化框架工作原型，直到工具套件达到一个新高度以及直到它们更复杂。

为了知道自动化程度需要达到什么程度才足够，我们必须了解这些领域：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。测试工具和测试过程是不相同的。工具是用于促进测试过程的。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，工具自带的内建程序可以被理解为过程。然而，它们往往也是不完整的，不能正确反映过程。最好的软件测试工具是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。

什么测试类型能够自动化呢？它们包括：单元测试、集成测试和系统测试。对自动化系统测试进一步分类如下：安全测试、配置测试和负载测试。自动化回归测试贯穿于整个开发过程的单元测试、集成测试和系统测试，并使用最大和最小发布版本的系统产品分别测试。

我们应该考虑测试过程的哪些方面呢？一般包括以下几个方面：测试计划、测试设计、测试构建、测试执行、测试结果的捕获和分析、测试结果验证和测试报告。还有一些活动是和测试活动紧密相连的，它们包括问题（缺陷）跟踪和解决、软件配置管理以及软件测试度量。总之，测试过程的这些活动是密不可分的，就好像软件开发过程一样，由好的项目管理技术粘结在一起。

所有领域的自动化水平应该达到这样一种程度，它能够根据时间和成本适应于你的组织。你实现的自动化程度越高，你的测试过程就越好越有效。这种观点总是对的，只要你的工具是适合的，并且被正确地实现。在这里，实现（implement）是指一个集成的测试自动化框架已经被创建并在使用中。

1.3 测试过程的各方面

让我们依次了解测试过程的各个组成部分。

1.3.1 测试计划

让我们从测试计划开始对测试过程的讨论，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。所有这些部分就构成了一份正式的测试计划，也可以独立开发这些部分并在合适的时间使用。

测试计划的传统思想是关于测试过程中谁来测试，对什么测试，何时测试，何地测试，怎么测试，测试持续多长时间。由于使用 Rational RequisitePro 工具，我们已经对于什么是测试过程以及怎么使用它的一些想法进行了调整。我们可以将软件需求文档引入到工具中，如 Rational RequisitePro，然后再直接从这些工具开发测试场景，这样得到了测试计划，而这些测试计划是使用 RUP 测试计划模板（我们已经广泛修改过，见图 1-2）构建的。从测试计划场景，我们可以得到测试需求。它们可以在测试计划文档的场景表中直接创建，或者它们也可以使用 RUP 模板在独立的测试需求文档中产生。由这些文档，我们可以得到测试需求视图，并且将其输出到 CSV（Comma Separated Values，逗号分隔值）文件。然后我们可以在测试过程中用微软的 Excel 打开该测试需求，这样我们就可以根据测试结果在线修改测试需求并把修改后的测试需求导回 RequisitePro 中。以上介绍了如何来定义我们的手工测试过程。尽管我们还没有将其完全实现，但是我们已经试用了它而且它也工作得很好。

我们将测试计划视为从软件需求中抽出来的工作文档，并且是和测试需求和测试结果相联系的。在测试阶段，它是一个动态文档。对于测试计划，旧观点认为它是计划性文档，它迫使测试者去考虑在测试过程中要做什么。从这个观点来说，测试计划就成了一个一旦测试阶段完成就束之高阁的文档。由我们的经验可知，几乎没有人会在测试执行阶段再参考测试计划。而事实上，在我们曾经工作过的几家公司，测试计划往往在测试完成后产生。然而，若采用我们的方法，测试计划不但在测试之前产生，而且它会随着软件需求的更新而更新；随之，更新也会被反映在测试需求中，这些测试需求将在测试阶段中用到。测试计划标准是基于包含在 RUP 中的模版的修订版，它是伴随 Rational Suite TestStudio 一同发布的。

以下我们将讲述如何在一个客户组织中定义手工测试过程。我们也已经试用了这个测试过程并且它工作得很好。在各个测试计划场景，我们需

要建立测试用例需求．测试用例需求直接建立在测试计划文档的场景表中以及分开的测试需求文档中（需求表）。由这些需求材料，我们可以建立测试需求视图并可以将其导出到 CSV 文件。我们往往用微软的 Excel 来打开 CSV 文件用于手工测试。我们利用这些信息来指导手工测试执行过程，并且我们可以用测试结果在线地更新 CSV 文件。然后我们可以把修改过的文件导回到自动化工具中用于测试结果分析和报告。

图 1-2 是 RUP 测试计划模板的修订版的所有内容的文档表（Table of Content，TOC)。TOC 已经被简化，测试类型减少到只包含功能测试，业务周期测试，安装和配置测试，以及用户接口测试。

1．介绍
- 1.1 目的
- 1.2 背景
- 1.3 范围
- 1.4 项目鉴别

2．测试场景

3．测试策略
- 3.1 测试类型
  - 3.1.1 功能测试
  - 3.1.2 业务周期测试
  - 3.1.3 用户接口测试
  - 3.1.4 性能测试
  - 3.1.5 负载测试
  - 3.1.6 压力测试
  - 3.1.7 容量测试
  - 3.1.8 安全和访问控制测试
  - 3.1.9 故障切换和恢复测试
  - 3.1.10 配置测试
  - 3.1.11 安装测试
  - 3.2 工具

4．资源
- 4.1 工作人员
- 4.2 系统

5．项目里程碑

6．提交文档
- 6.1 测试模型
- 6.2 测试日志
- 6.3 缺陷报告

7．附录 A：项目任务

图 1-2 修订后的 RUP 测试计划 TOC

测试计划的目的是收集从需求/设计文档中得到的信息，并将这些信息表现在测试需求中，而测试需求将在测试场景中得到实现。测试场景是测试计划的一部分，它直接提供给测试条件、测试用例、测试数据的开发。

有一些桌面工具可用于自动化测试的计划和项目的管理，比如：微软Office和微软Project。举例来说，在微软Excel电子数据表中生成的检测表可用于风险的评估和分析，并且可用于生成测试需求文档；微软Project可以用于产生项目计划；微软Word可用于生成正式的测试计划。这些文档是测试计划的相关产物。和软件开发产物需要配置管理一样，测试对象也是如此。

1.3.2 测试设计

测试设计包括：确认基于先前特定测试需求的测试条件，发展这些条件下所有可能的功能变体，推测每一个变体在被测试的应用程序(application under test，AUT)运行时的各种预期行为，并在设计过程中执行手工测试，手工测试是先于自动化测试的。手工测试允许测试设计工程师来验证自动化测试中用到的测试数据是否正确和合适。并且手工测试可以让测试设计师确认自动化测试中可能漏掉的一些错误。测试设计还包括设计输入到AUT中的测试数据的布局。从数据驱动的角度来设计时，同样的数据能够控制自动化测试脚本的运行。最后，还必须设计测试脚本。

设计测试和测试数据是测试过程中最费时的一部分，但也是最重要的一部分。如果测试并没有测试到那些需要测试的部分，那么测试可以说是无效的。同样如果测试数据并没有反映测试的意图，测试也是无效的。测试用例的设计是如此重要，所以在附录中我们专门有一部分讲述测试设计技巧以及它们的用法。

测试设计者可以使用微软Excel来设计和构建测试。如果你使用这种方法，最好将所有东西保存在一个工作簿中，其中包括一个测试条件电子数据表，一个测试数据电子数据表，和足够多的详细测试电子数据表，这些详细测试电子数据表是用来描述与每一个测试相关的环境方面的活动和测试前后的一些活动。在手工测试中，必须建立测试日志，因为在测试执行中要用到测试日志（在线使用，而不是打印出来的测试日志）。有一些集成的测试套件可以选用，它们采用一种叫作数据池（data pool）的机制来支持测试数据的设计和产生。所有测试用例中的数据被存成CSV文件，这些文件可以被自动化测试脚本读取和解释。而这些数据既可用于手工测试，又

可用于自动化测试。

1.3.3 测试实现

测试实现可以被分为以下几个部分：测试构建，测试执行，测试结果的捕获和分析，以及测试结果验证。我们将分别介绍这几个部分。

1. 测试构建

测试构建沿用了测试设计中使用的那一套工具，使之容易了一些。测试数据建立在一个电子数据表中，这个电子数据表和测试条件同属一个业务手册。这些数据可以导出到 CSV 文件，CSV 文件将在测试执行中用到。如果测试是通过自动化框架来执行，那测试构建还包括编写测试脚本。自动化测试脚本是软件程序。它们有自己的适用于软件测试事件的编程语言或语言扩展。脚本语言通常嵌入在捕获/回放工具中，该工具带有源码编辑器。语言的风格随着开发商的不同而不同，而且随着相关语法和语义的不同，所以使用特定产品的困难度也不同。另外，一些开发商的脚本语言和他们的记录工具比其他开发商的更健壮。

可用的命令（命令是语言的一个部分）越专门化，那么测试工程师就能越好地控制测试环境和 AUT。专门的测试成为语言的组成部分，类似于命令，当执行的时候测试具体的项目——比如图形用户界面（GUI）对象的属性和数据以及窗口的存在性——并且需要进行文件比较。一些内建的测试用例非常有用，大多数在 GUI 测试方面功能十分强大。但是它们对功能测试就不那么有效了。我们可以通过执行外部测试数据（这些测试数据反映了测试需求）和验证测试结果来实现很多测试。这些数据控制了测试

3．测试结果的捕获和分析

对于手工测试，测试日志必须在线发展和实现。至少，它必须是一个与测试条件和测试数据存放在同一个业务手册中的电子数据表。理想情况下，测试结果数据库将被用作永久存储的中心知识库。当使用捕获/回放工具（比如 Rational Robot 或 Mercury Interactive 的 WinRunner）来执行测试时就是这种情况。测试结果被捕获，并将测试结果写入知识库，以后便可以查看测试结果并打印出报告。

Bruce已经开发了一种自动化的手工测试脚本（使用 Rational Robot 和 SQABasic 语言）用于手工测试中的测试结果捕获和分析（这个工具我们将在第 9 章详细讲述）。它显示手工测试脚本，就像讲词提示装置那样，在测试执行过程中脚本的每一步都可以被选择。当执行完毕，测试将以通过、失败和跳过三种状态记人日志。该手工测试脚本基于多年前由 Phoenix Arizona SQA 用户组出版的手工测试脚本。这些测试脚本已被重写，并作了不少改进，在测试日志中添加了客户评论。在支持本书的 FTP 站点上可得到这个测试脚本。它还可以容易地用 SQABasic 之外的语言改写。

4．测试结果验证

测试结果验证既可以手工完成，也可以通过自动化测试完成。手工目测结果并做对它的正确性做主观评估是验证测试结果的一种方法。进一步的，可以拿测试结果与预期的输出数据值做比较，也可以拿测试结果与数据库各行中基于事务测试产生的值做比较，还可以与存储文件和报告做比较。

测试结果验证是适合自动化测试的任务。测试结果在测试日志中被捕获，测试日志是存放在测试知识库中的。测试日志中存放着以前测试的结果，这些测试结果可以作为基线行为来与目前的测试结果做比较。对手工测试来说，测试日志可以是微软 Excel 形式的工作簿，它为每一次新的测试迭代产生新的电子数据表。但它并不关心测试结果如何被存储。而重要的是要能够做比较。如果没有建立基线，那么对测试结果是通过状态还是失败状态的评估就只能依靠猜测了。可以定义和存储基线行为是自动化测试的重要优势和有力论据。

商业上可用的自动化测试工具套件提供各种自动化验证方法。比如，Rational Robot 使用 SQABasic，它提供一些特定的测试用例，这些测试用例被称为验证点（verification point），可以用于捕捉 AUT 特征；它们也用作以后回归测试的基线。

5. 测试报告

测试报告是测试过程中必不可少的一部分，因为它记录了测试结果及其分析。需要有两个层次的报告——一个是总结性的报告，这是提供给被测方中层和高层管理者以及客户的，另一个是详细报告，经过编辑和整理，作为反馈文档提供给开发小组成员。

这些报告都应该有标准的表现格式，能够被编辑并且可以转换成每一个单独的测试项目报告。我们曾经使用过 RUP 文档中的报告模板，当然你也可以创建你自己的模板。必须生成报告的两个版本——总结性报告和详细报告。你也可以在互联网上找到测试报告的模板和例子。

在报告中很重要的一项就是缺陷跟踪信息。缺陷跟踪报告可以使用工具（如 Rational ClearQuest、微软 Excel）独立产生。缺陷信息也必须分别包含在总结性的测试评价报告和详细的测试评价报告中。缺陷跟踪信息必须包含一个已知缺陷列表，这个列表中的缺陷还没有被解决而且在软件发布之前应包含在软件中。列表中的信息应该以严重程度来分组。这类信息有助于作出智能的发布决策，并且在软件正式投产后有助于用户对产品的使用。

1.4 辅助工作

测试是一项团体工作

因为软件测试是小组合作完成的，所以它需要一些工具来支持和提高小组成员之间的交流，也需要工具来提供一个集成的界面，使得小组成员可以分享对测试过程活动及其产物的共同观点。在测试过程的所有阶段的一个突出问题是测试产物的控制和存储。对测试产物的自动化配置管理是能够提供最多反馈给测试过程的领域之一。有很多的文档和可执行文件必须是所有测试小组成员可得的，小组成员可以频繁地并行使用这些文档和可执行文件。并且在多个成员使用同一个交付物时，又必须保证小组成员对它的修改不会同时进行。而且还必须有一个中心知识库来存放测试产物以便测试小组成员能够公用。

我们曾经在很多这样的测试项目中工作过，在这些测试项目中并没有中心存储，而且小组中的各个成员只能在他们的本地机器上创建和更新文

档。我们指定了一些目录用于存放特定的可交付文档，并且申明每个人都可以在这些共享的公共目录下存放他们的工作。这个解决方案总比没有解决方案强，但是它还是不能为版本提供对这些文档的有效访问控制。

测试管理和需求管理是测试过程中最重要的。测试管理可以借助于像微软 Project 这样的工具来实现。它通过设置测试里程碑提供任务识别、资源管理和过程评估功能。需求管理工具也是必须的，这是因为在开发过程中软件需求必须被记录和不断更新，同时随着开发活动和测试活动的进行测试需求也必须被记录和不断更新。

RequisitePro 是我们曾经选择的一个需求管理工具，因为它集成了软件需求和测试需求规范。并且它的测试需求表可以导出到微软 Project 中用于指导和监控测试过程。还有一些其他的需求管理工具可以选择，其中一些还集成了测试工具套件。本书并不准备详细介绍评价这些需求管理工具的方法，但有两个方面是必须考虑的。第一，产品是否集成了测试工具套件？第二，如果没有集成测试工具套件，那它是否提供开放的应用编程界面（API）用于创建自己的集成代码？

其次是软件配置管理。目前，有一些产品可以用来实现对测试产物的配置管理。这些产品包括微软 Visual SourceSafe、Rational ClearCase 和 Merant 的 PVCS 等等。重要的是所有的测试产物应当存储在自动化配置管理数据库中。在所有的测试活动中，你选择的特定配置管理工具和你使用的其他工具（用来支持测试过程活动）能够互相通信也是很重要的。如果你选择的配置管理工具不能做到这一点，那它必须提供和开发 API 来创建你所需的软件桥梁。

软件测试度量是测试评价报告的必要组成部分。它包括缺陷度量、覆盖度量和质量度量，还包括很多有用的缺陷跟踪量度。一般，缺陷度量可以分为：缺陷密度度量和缺陷老化度量。缺陷密度度量包括：每天/每周打开/关闭缺陷的次数、与特定的软件/测试需求相关的缺陷数目，列于应用对象/类上的缺陷数目，与特定的测试类型相关的缺陷数目等等。缺陷报告应该实现自动化，这可以通过使用 Excel 工作簿实现，因为 Excel 能够将电子数据表中的数据总结成图表和图形。缺陷报告也可以通过其他工具（比如 Rational ClearQuest）来达到自动化实现。

测试质量度量是缺陷度量中的一个特定类型。它包括 [8]：

- 缺陷的当前状态（打开，被修复，关闭等）
- 缺陷的优先级（对解决缺陷很重要）
- 缺陷的严重性（影响到终端用户、组织、第三方等等）
- 缺陷来源（导致这个缺陷的最初错误，即需要被修正的组件）

覆盖度量体现了已执行测试的完备性（completeness）表示。它既包括基于需求的覆盖测度，也包括基于编码的覆盖测度。对这些度量的举例请看第9章参考文献［6］和参考文献［10］的在“Key Measures of Testing”下的概念部分。

1.5 测试自动化组的范围和目标

1.5.1 范围

测试自动化组的目的是开发自动化的测试支持工具。这个工作组必须负责设计并实现数据驱动的自动化测试框架。他们还应该设计和构建用于回归测试目的的自动化测试套件。图1-3描述了自动化测试的基础设施，这是CSST技术有限公司为一个知名公司设计的。

为了支持自动化测试脚本的开发和与所有级别测试有关的维护，必须专门地来部署测试自动化的框架。这个框架必须既支持单元测试，又支持集成测试，还必须支持系统/回归测试。但这并不意味着其他不在这个范围内的领域就不能利用这个测试自动化框架和工具套件的优势。其他的一些可能对使用测试自动化框架和自动化工具套件感兴趣的部门，应该帮助自动化小组并与自动化小组协调部署工作。自动化工作重心应放在某个特定领域的部署上。

所选择的方法应该能够覆盖自动化测试的所有工作，而这些工作是由自动化工具组来完成的。手工测试活动可以作为自动化测试的先导。手工测试的目的是手工测试应用程序的所有特征，同时，在测试的过程中开发一些测试条件和测试数据，这些测试条件和测试数据可以用回归测试的自动化框架来实现。

举例来说，数据驱动的方法可以通过结构化测试脚本来实现，结构化测试脚本利用了存放在库文件中的函数和过程。这样做有两个目的，首先是为了将测试数据从测试脚本中分离出来，其次是为了开发可重用的测试

脚本组件结构。达到这两个目的就可以大量减轻自动化测试脚本的维护负担。

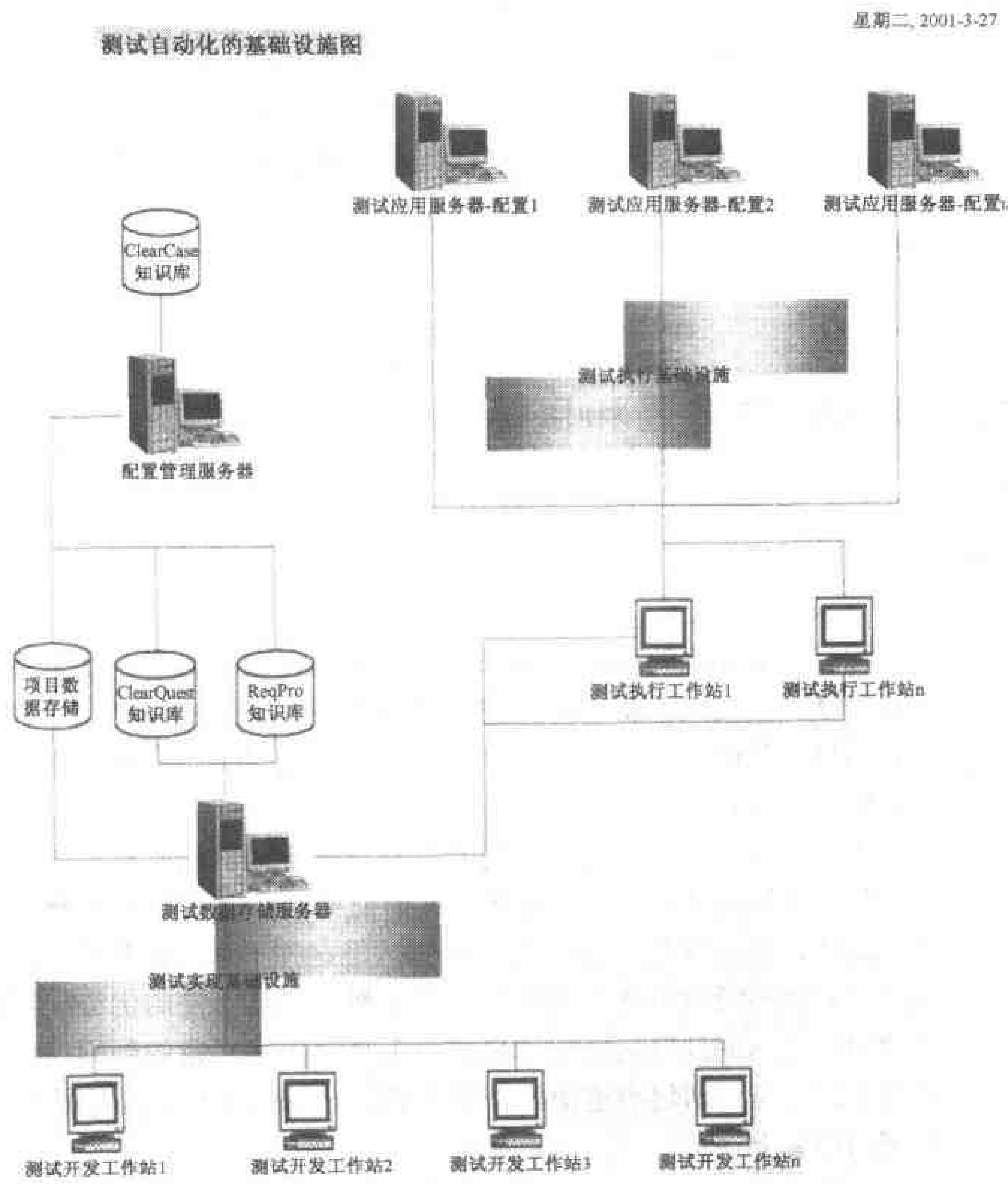


图 1-3 自动化测试基础设施的一个例子

1.5.2 自动化测试框架的假设、约束条件和关键的成功因素

以下是要用到的一些假设。

假设　以下假设形成了测试自动化策略的基础。

- 集成工具套件必须是主要的测试管理、计划、开发和实现的工具。
- 工具套件必须用来指导和控制测试的执行，用来存取测试产物，并且用来捕获/分析/报告测试结果。
- 工具套件必须包括一个可选工具用于缺陷跟踪及其解决。
- 工具套件必须包括测试需求管理组件。
- 工具套件必须包括一个可选的配置管理工具。
- 配置管理工具能够对手工测试和自动化测试的产物进行配置管理。
- 所有上述工具必须与桌面工具结合，比如微软 Office。
- 正确的自动化测试工作区必须在测试服务器上产生，测试服务器是和开发服务器相分离的。
- 测试工程师需要的桌面－脚本－开发配置必须被定义并且被实现。
- 必须遵循测试标准，测试标准以文档记录下来。

约束条件　如果不注意这些约束条件，自动化测试工作将难以成功。

- 自动化工具集资源必须独立于任何手工测试集。
- 自动化测试小组中可能没有足够多的工作人员。
- 对于自动化工具的使用来说，软件开发小组的协调水平和管理水平可能太低。
- 在创建可测试应用中，可能与开发者缺乏协作和信息交流。
- AUT 主要版本和 AUT 客户定制版的发布进度安排太紧。
- 由于 AUT 中 GUI 的更新，会产生不定因素。
- 对工具的使用可能有共同的要求。

重要的成功因素　以下所述的重要成功因素是基于一套测试自动化的指导规则，这套指导规则是由 Nagle 制订的 [7]。

- 测试自动化必须作为一项主要业务工作来做，而不是次要业务。
- 测试设计过程和测试自动化框架必须作为两个单独的实体来开发。
- 测试框架必须独立于应用程序。
- 测试框架必须容易扩展、维护和增强。
- 测试策略/设计词汇必须独立于框架。
- 测试策略/设计必须对测试者隐藏测试框架的复杂性。

策略目标　这些目标是以上面所列出的重要成功因素为基础的。

- 实现策略应允许测试手工开发和执行（初始测试周期）以及通过自动化框架开发和执行（回归测试周期）。
- 测试设计和测试实现相分离，这样就可以使测试设计者集中于开发测试需求、测试计划以及测试用例设计，而测试实现者集中于建立和执行测试脚本。
- 实现一个技术性测试者和非技术性测试者都能使用的测试框架。
- 使用这样一个测试策略，这个测试策略可确保测试用例包括要执行的导航和执行步骤，要使用的输入数据以及都在输入数据源的一行或一条记录中的预期结果。
- 实行集成方法，该集成方法利用了关键字驱动测试、数据驱动测试和功能分解测试的最好特征。
- 实现独立于应用程序的测试自动化框架。
- 以文档记录并公布该框架。
- 为每个应用发布版本开发自动化建立确认冒烟测试的工具。
- 为每个应用发布版本开发在不同环境下能自动安装的实用程序脚本。
- 开发自动化回归测试用于：
  * GUI 对象和事件
  * 应用程序功能
  * 应用程序特定特征
  * 应用程序性能和可扩展性
  * 应用程序可靠性
  * 应用程序兼容性
  * 应用程序性能
  * 数据库验证

1.6 测试自动化框架的产物

1.7 测试工具分类

为了支持自动化测试框架，需要很多不同类型的自动化测试工具和手工测试工具。Marick 根据在测试过程中何时以及如何使用，将它们分类如下 [5]。

测试设计工具。测试设计工具用于为软件测试活动做计划。它们可用来产生测试产物以驱动后续的测试活动。

静态分析工具。用这些工具来分析程序，而不用执行程序。审查（Inspection）和遍历（walkthrough）就是静态测试工具的例子。

动态分析工具。通过执行软件来对它进行测试。

GUI 测试驱动和捕获/回放工具。这类工具运用大规模记录功能来自动化测试使用 GUI 的应用程序。

负载和性能测试工具。这类工具为自动化的压力和容量测试模拟不同的用户负载条件。

非 GUI 测试驱动和测试管理器。这类工具实现了对不允许测试者通过 GUI 进行交互的应用的测试自动化。

其他的测试实现工具。各种各样的工具都有助于测试实现。这也包括微软 Office 工具套件。

测试评价工具。用来对测试工作的质量进行评价。

附录 B 是自动化测试术语以及定义的列表。

1.8 小结

你的工作就是监视每一个测试阶段，并在给出你的组织的自动化工作的范围、目标、目的的情况下，决定各测试阶段使用的测试工具（手工或自动化）种类。你将会发现，你必须做一些妥协和折衷，而且理想的测试自动化框架仅仅是理想。你最终实现的是工具和技术的相互混合，而这些工具和技术是适合于你的需要的。

1.9 参考文献

1. Bender, Richard. *SEI/CMM Proposed Software Evaluation and Test KPA*. Rev. 4, Bender and Associates, P.O. Box 849, Larkspur, CA 94977, April 1996.
2. Dustin, Elfriede, Jeff Rashka, and John Paul. *Automated Software Testing*. Addison-Wesley, Reading, MA, 1999.
3. Humphrey, W. S. *Managing the Software Process*. Addison-Wesley, Reading, MA, 1989.
4. Krause, Michael H. "A Maturity Model for Automated Software Testing." *Medical Device and Diagnostic Industry Magazine*, December 1994.
5. Marick, Brian. "Testing Tools Supplier List." *www.testingfaqs.org/tools.htm*
6. Mosley, Daniel J. *Client-Server Software Testing on the Desktop and the Web*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.
7. Nagle, Carl. "Test Automation Frameworks." Published at *members.aol.com/sascanagl/FRAMESDataDrivenTestAutomationFrameworks.htm*
8. Pettichord, Bret. "Seven Steps to Test Automation Success." Rev. July 16, 2000, from a paper presented at STAR West, San Jose, November 1999. Available at *www.pettichord.com*
9. Powers, Mike. "Styles for Making Test Automation Work." January 1997, Testers' Network, *www.veritest.com/testers'network*
10. Rational Software Corporation. *Rational Unified Process 5.1, Build 43*. Cupertino, CA, 2001.
11. The Software Engineering Institute, Carnegie Mellon University. "Software Test Management: A Key Process Area for Level 2: Repeatable." Available in the "Management Practices" section of *www.sei.cmu.edu/cmm/cmm-v2/test-mgt-kpa.html*
12. Strang, Richard. "Data Driven Testing for Client/Server Applications." Fifth International Conference on Software Testing, Analysis and Reliability (STAR '96), pp. 395–400.
13. Weimer, Jack. *Manual Test User Interface Program*. Touch Technology International, Inc., *www.touchtechnology.com*. Available from Phoenix, Arizona, SQA Users Group. This code is free and can be passed on to anyone who wishes to use it, provided the copyright, credits, and instructions stay with the code. *www.quanonline.com/phoenix_sqa/tips.html*
14. Wiegers, Karl E. "Read My Lips: No New Models!" Whitepaper, Process Impact, (716)377-5110, *www.processimpact.com*
15. Zambelich, Keith. "Totally Data-Driven Automated Testing." Whitepaper, Automated Testing Specialists (ATS), *www.auto-sqa.com/articles.html*

第2章 了解何时以及对什么进行自动化

2.1 概述

Zambelich 关于测试自动化说过这些话：

> 已有相当多的测试专家令人信服地重复实施着软件测试过程自动化的工作。参加软件测试的大部分人都认为测试过程的自动化不仅是我们期望的，而且实际上也是当前市场必须的要求。[3]

开发自动化测试框架是一个费钱而且费时的项目。这个框架的创建和实现必须在 AUT 交付 QC（Quality Control，质量控制）之前完成，因此得在该项目的生命周期的早期阶段构建并测试。然而，一旦使用该框架，你可能会发现它不适合你的组织所有的软件测试需要。了解何时使用这样一个框架是测试自动化重要的部分。我们已经与自动化测试团体的其他成员在在线讨论中热烈讨论过这个问题。我们讨论的部分内容是给定开发该框架所花费的工作量，数据驱动测试的价值何在。该讨论的部分内容发布在 1999 年 5 月 17 至 21 日 SQA 用户组网站上。尽管很多人参加了讨论，这里只摘录 Elfriede Dustin、Carl Nagle 和 Dan Mosley 的谈话。完整的在线讨论请见附录 A。

Elfriede：

> 我同意 Mark 的观点，数据驱动测试会有普及的一天，虽然现在还未到时日。我在使用数据驱动测试时使用的是“测试

数据”（请看 www.autotestco.com，那里有一个例子描述了我们如何使用 Robot 工具对两千年数据进行测试），但是我们很少在数据驱动测试中使用“控制数据”。原因在于实施起来很麻烦，并且只有当测试能在后续版本中重用多次时这些努力才算值得。

Ed Kit 在 STAR 做完讲座之后我曾经咨询过他，他也认同我的观点，即这种方法至少要重用 17 次后才能使回报与付出相抵。（没错，这是他给出的数据。）

较早时候，我在先前工作中的一位合作者刚好接受了这种数据驱动测试的训练。此人花了 3 周的时间实施数据驱动方法。有很多漂亮的表格，里面有各种命令/控制以及要读取的数据。但是最后大家归结出如果使用简单的记录以及回放和脚本的修改，将会更有效率，因为这种测试无法重复使用。在此案例中这种尝试属于浪费时间。你将不得不依赖于你自己的判断，并请记住在测试自动化中使用复杂的数据驱动框架有时是得不偿失的。

Carl：

我对你的各种观点都比较同意，但是我认为如果不是有意进行重用，那么再多的自动化方法也是不合算的。实际上，即使是经过第 17 次重用才划算的说法听起来也太好了。根据每晚进行的构建验证，应该是 17 个工作日（或更少）并全在同一个版本中！

Dan：

我恐怕不能同意你（Elfriede）的观点。数据驱动的确需要你所说的前期投入，但是在软件创建这个级别的回归测试中它的回报是很大的。我自己经历过并且见过。我们曾经必须为一个财务软件测试 100 多个交易画面（每一个都是一个窗口）。我们开发了超过 7000 个的数据驱动测试，其中每一个大约要花费 3～5 天进行编制和调试，但是这些测试在每一次软件新版本创建之间重复运行只要 1～2 个小时。我们通常每周都会收到一个新的版本，而我们每周都能够重复使用那 100 多个测试脚本和 7000 多个测试记录，并且我们可以按时完成工作。

从这些讨论中可见，关于自动化测试的价值，特别是数据驱动方法的价值存在不同的意见。主要问题是：何时实现自动化测试框架是有意义的？对于较小的测试项目，这种工作就不值得了。即使对于较大的测试项目，

如果测试不在回归测试套件中重用，测试自动化也似乎不明智。

另一个解决这个问题的方法是看要被自动化的测试类型。对于单元和集成测试，自动化是必不可少的。为什么？因为如果没有自动化，用于测试的构建版本质量就会很差。结果是本应在开发中发现的缺陷，留给系统测试者了。在当前的 Web 开发领域，Java 是可选的语言，且使用面向对象方法。开发者使用测试工具比如 JUnit 把测试用例作为类嵌入，这些类可与其余代码一起保存，并且在构建过程中集成前被用/重用来测试 Java 对象。这样做可以清除许多当前留给系统测试的错误类型。此外，自动的集成级冒烟测试应该在每个构建版本进入系统测试前准备和执行。第 5 章更深入地讨论了这些问题。

Hancock 认为：

> 测试自动化是一项投资。最初的投资可能很多，但投资的回报也很丰厚。自动化测试运行超过 15 次以后，测试就是免费的了。[1]

他将自动与手工测试的比较看作是基本的成本—收益分析。他引用了 Kaner 的假设，构建、验证和存档自动化测试要花手工测试 3～10 倍的时间 [2]。Hancock 在他的测试自动化的可能回报的例子中用 15 倍作为最坏的情况，对可能的回报做了保守估计。

为了计算投资回报（Return on Investment，ROI），Hancock 说一定要确定“次数”；也就是说，测试集需要执行多少次——要将平台数、操作系统数，以及与软件相容的外语数目相乘，也要与测试要运行的构建数/版本数相乘 [1]。

如果你用 Hancock 的 ROI 方法，单元和集成测试的自动化比系统测试的自动化更值得，因为系统测试比其他两种测试类型重用的机会少。如果系统级测试没有在一个重用程度很高的自动化回归套件中实现，系统测试自动化就不会有太高的 ROI。而单元和集成测试的 ROI 就高得多。

很少有公司完全意识到自动化测试的价值。他们想拥有，也知道他们需要，但最后他们不能让上层管理者信服其成本收益。

2.2 何时自动化系统测试

有很多附加因素决定何时进行自动化。以下分析是基于 1996 年以来在多个公司做了多个项目的实际经验。

2.2.1 自动化的时间总是第一因素

建立自动化测试项目比建立手工测试项目花费的前期时间多。自动化在开始阶段没有捷径。你必须执行所有手工测试项目要执行的相同步骤。自动化应在所有测试用例根据完善的需求（这是乐观的想法）定义好之后，并在 AUT 的一个构建版本可交付之后进行。

此外，被自动化测试的应用程序特征必须有效。如果需求或应用程序特征不能 100%有效，就不能根据它来创建有效的完整的自动化测试。自动化测试的基本目的是一旦需求确认后，验证后继构建版本或 AUT 的修改版是否正确实现了需求。而且，因为自动化测试减少了执行回归测试所需的时间，这是它最被认可的好处。

如果你做一个进度很紧迫的项目，项目管理有非常紧张的交付进度表，你就不要考虑自动化了，除非专门为它分配了充分的时间。

2.2.2 一个极端的例子

让我们看看要交付的 AUT 构建版本的数目和复杂度。如果你只是要得到一个普通规模和复杂度的项目的一个构建版本，进行测试自动化可能就没什么好处。实际上如果只在维护期重用一次（这样情况很少见），测试自动化就根本不合理。即使项目本身相对复杂点，如果重构建版本的数目和测试的交付物只限于修补，也不值得花时间进行自动化。

如果该项目将重复地被交付测试，而新的特征集将在多个测试间隔交付，且特征集很复杂，自动化测试可能会有很大益处。普遍接受的看法是自动化测试要花费执行手工测试的 3～4 倍的时间。如果你能在项目早期预先判定会有超过 3～4 个重要的可测试构建版本交付给你，那么你的项目就可选择自动化测试。

没有奇迹的组合或精确的公式能决定何时应该和不应该执行自动化。以下几点可供考虑。

- AUT 中的特征集本身是简单还是复杂？
- AUT 中的特征集在开发过程和阶段中是否会改变很大？
- AUT 中的特征集当前是否按需求所述工作？
- AUT 中的特征集是否需要大量的数据组合来确认所有业务规则？

- 自动化测试使用的测试工具是否能与用于测试目的的特征的所有必要属性交互?（例如，它是否能像用户一样交互? 我们是否能从 GUI 和子对象获得所有必要数据?）

如你所见，这些问题很复杂且难判断。总的来说，这里有一些实践准则，可用来决定是否进行自动化测试。

1）如果 AUT 不复杂且不太大，不要自动化。

2）如果你只将接收几个（3 个或更少）构建版本，不要自动化。

3）如果一个特征不是 100%有效，不要对它执行自动化测试，不管该 AUT 的规模或复杂度如何（你可以为它制定计划，但不要创建真的自动化测试脚本，除非该特征能够完成并 100%有效）。

4）如果开发周期的时间表很紧，每次交付间隔时间很短，你就没有时间自动化。

5）如果一个特征不能通过自动化测试达到 100%准确测试，就不要进行自动化了，除非它能节省大量的手工测试时间。这并不意味着特征必须要 100%的测试。注意软件测试几乎不可能覆盖到 AUT 的每个特征。不要妄图达到这个目标。你执行的测试不应该是主观的。结果应是可预见的，而且应该能指出通过或失败的条件。

2.2.3 一个定量的例子

AUT 的一个特征集要花 6 小时进行手工测试。如果执行这些测试要花 6 小时，那么使用自动化测试工具记录测试最少要多花 6 小时（很可能更多，因为你会在手工测试和测试数据/测试脚本创建之间互换）。普遍接受的看法是要另花 12 个小时来组织和建立自动化测试脚本。情况就是这样，这个特征集的自动化测试前期会花 18 个小时，是执行手工测试时间的 3 倍。一般来说，重新运行一个测试脚本只要手工测试执行时间的 1/10。测试运行接近于机器速度，通常只受应用程序反应时间及任何插入测试脚本的延迟（比如模拟用户思考时间）的限制。

据说，自动化测试执行时间大约是 36 分钟。因此，在第 4 次对该特征集执行测试时，自动化测试开始比手工测试节省大约 5.5 小时测试时间，而且此后每次对这个 AUT 的特征集执行回归测试时都能节省同样的时间。自动化前期要花费较多时间，但在每次回归测试执行时都能迅速回报。不仅能节省时间，而且因除去了人为交互因素而使得测试执行的准确性更高。

2.3 对什么进行自动化

如果时间允许，对整个项目进行自动化测试。实践表明平均可对整个项目的60%进行自动化。你的目标应该是自动化测试AUT的关键路径。首先，对目标用户将要执行的基本功能进行自动化。如果时间允许，慢慢地但要确定地加入该应用程序不太关键的部分。一定要开发测试覆盖矩阵，一个轴显示需求，另一个显示已开发测试，以便始终可以清楚地看到什么进行了自动化，什么没有进行自动化。

项目早期不必对诸如登录、用户配置或其他选项之类进行自动化测试。不要对状态栏、帮助显示或直到项目后期开发才注意的应用程序其他部分进行自动化测试。在测试开发早期为这些项目的自动化策略制定计划，但直到对它们集中开发时才执行这些计划。按照常识，这些项目的自动化测试要在开发结束或完成时进行。

Hancock说过"对什么进行自动化是有意义的"。这个问题的答案因项目和环境的不同而不同。你必须为你的测试组织开发评估策略，用来决定对什么进行自动化。我们相信我们的提示会对你的决定有所帮助。

2.4 关于创建测试脚本的一点注意事项

当你在开发自动化测试时，会比运行已完成的测试脚本时发现更多缺陷（因为你必须首先执行手工测试来验证你的自动化测试是否正确工作）。这是正常的。自动化可用来在被测特征测试并交付之后，揭示那些偶然或不注意产生的缺陷。你用什么速度可以发现这些新引入的缺陷是你目前面临的问题。自动化测试是你的保障，因为它就是用来验证曾经正确工作的部分仍然还在正确工作。

2.5 小结

本章包括了关于是否进行自动化以及何时、对什么进行自动化的实践

性的建议。这些建议是根据我们作为员工和顾问时的实践经验提出的。

一般的建议以及关于单元和集成测试的建议，是根据本书合著者 Daniel Mosley 的经验和他在基于 Web 的 Java 环境中所做的测试。这些建议也可用于非 Web 的开发环境，比如 VB。

系统测试自动化的提示，是根据本书合著者 Bruce Posey 的工作，他的专长就是测试脚本的设计和构建。

2.6 参考文献

1. Hancock, James. "When to Automate Testing A Cost-Benefit Analysis," June 1998, Testers' Network, *www.veritest.com/testers'network*
2. Kaner, Cem. "Improving the Maintainability of Automated Test Suites." *Software QA* 4, no. 4, 1997. (Also published in *Proceedings of the 10th International Software Quality Week*, San Francisco, May 1997.)
3. Zambelich, Keith. "Totally Data-Driven Automated Testing." Whitepaper, Automated Testing Specialists (ATS), *www.auto-sqa.com/articles.html*

第3章
从头开始：定义测试需求、设计测试数据

3.1 软件/测试需求

在 www.csst-technologies.com 网站进行的非正式调查中，我们问了这样一个问题："作为一名软件测试人员，你认为什么将最大程度地减轻你的工作?"答案如下：

改进了的系统需求文档	33%（28份）
软件测试方法/过程	31%（27份）
软件测试标准	16%（14份）
改进了的测试计划	13%（11份）
更充分的测试时间	7%（6份）

显然，软件测试工程师所需要的最重要的两个改进是改进系统需求文档和软件测试方法。软件开发经理却没有如此明显地认识到这个问题。

在经济形势良好时期，大公司聘用了相当数量的质量保证和测试（Quality Assurance & Testing，QA&T）人员，此外也从外面聘请顾问。然而，对于质量保证和软件测试行业来说，2001年形势非常不好。2001年经济环境影响了雇佣模式，暴露出在经济良好时期隐藏起来的问题。不仅如此，雇主们为了获得最大的经济利益而采取的种种做法使暴露出的问题更加严重。雇主用相同的人员既做测试计划、测试设计又做测试执行。这就导致了测试工程师、测试设计工程师和测试执行工程师角色的混淆。

两方面因素导致了对这些工程师工作职责的误解。首先是对软件测试过程缺乏了解以及自动化是如何协调的，其次是对整个测试过程中测试工

程师角色的更替缺乏了解。现在已经是专业化时代，软件测试也不例外。测试工程师可以分为测试计划、测试设计和/或测试执行三个角色。每个角色都需要不同的手工和自动化测试技能。测试计划人员和测试设计人员可以并行工作或者可以是同一个人；他们在定义和编写测试产物方面拥有相似的能力。然而，测试执行人员是不编程的手工测试人员或者是做自动化测试的程序员，后者编写和执行自动化测试脚本。这两者都要有取得测试结果和分析测试结果的能力。如上所述，由测试设计人员在设计活动中担任手工测试人员是合理的。因此，测试设计工程师必须是“通才”，既能做测试计划和设计，也能做测试执行。而测试执行工程师必须是编写和执行自动化测试脚本的“专才”。测试执行工程师是测试脚本设计的专家而不是测试/测试数据设计的专家。

关于测试的自动化方面，过去的十年在本地的许多公司中，我们一直致力于实现测试自动化，我们强调自动化测试脚本的智能并不在自身而在于测试数据。测试数据必须包含基于软件需求和设计说明的灵活测试(smart test)。智能（intelligent）一词是指测试数据发现已知/预期应用程序缺点的能力。在测试执行之前，测试设计人员必须使测试数据具有智能，只要测试数据被应用在手工测试或自动化测试脚本中。

测试数据必须包含脚本中用于导航 AUT 的值。在测试脚本中或脚本调用的函数/子例程中硬编码数据值，是仅有的使测试脚本灵巧的其他方式。传统的捕获/回放自动化方法中就会用到这种方式。在记录过程中，需要收集两种类型的测试数据。一种是数据点（date point)，用于设置测试上下文和导航 AUT；另一种是数据值，用于提交给应用程序来测试程序的功能、性能和安全性等。如果把所有的这些数据都植入到测试脚本中，那么维护测试脚本将变得非常困难。

除非自动化测试脚本能够执行智能的测试数据，否则自动化测试脚本并不一定会更好。而且为了使手工测试也很灵活，把自动化测试的测试数据设计开发与手工测试相结合是也是很重要的。如果一台测试机器上运行着应用程序，一台电脑上运行着微软 Excel，那么使用需求和设计文档来设计测试数据是个很好的方式。当明确了测试需求，就可以用 Excel 电子数据表为需求编写测试条件和环境要求。接下来在测试机上手工执行测试数据。如果得到了预期的结果，在电子数据表上编写文档，然后转向下一个测试条件/要求。如果采用这种测试方式，必须指明测试数据、编写测试前环境变量文档以及测试后清理文档以保证每次测试的独立性。最后，以一种自动化测试脚本能使用的格式导出测试数据。

测试设计人员也必须指明测试先决条件、AUT 的导航、被测试的方面

和所必需的测试后清理。为了编写出有效的自动化测试脚本，这些信息对于测试执行人员（脚本编写人员）都是必须的。测试设计人员必须使用需求说明文档和设计文档确认潜在的测试条件（有缺陷倾向的应用程序领域）和预期的结果。测试设计人员也可使用其他可用的描述应用程序的信息。

经常在软件质量自动化（Software Quality Automation，SQA）用户组和数据驱动引擎（Data Driven Engine，DDE）用户组的人员都知道我们提倡使用数据驱动框架。本书作者 Bruce Posey 已经开发了他个人风格的范例，并命名为控制同步数据驱动测试（Control Synchronized Data Driven Testing，CSDDT）。SAS 协会的 Carl Nagle 对此有着不同的观点，他用 Rational Software 公司的 SQABasic 语言平台和微软 Excel 开发了 DDE 模型。尽管两种方法都是用 Rational Software 公司的 Test Studio 工具套件开发的，但是，他们与其他的商业工具套件，例如由 Mecury Interactive 和 Seque 提供的工具套件是兼容的。

两种方法的根本不同点在于 CSDDT 存储了测试上下文、导航规则和特殊的测试程序，特殊的测试程序作为子例程在测试脚本中可以由主程序直接调用；然而在 DDE 模型中这些任务是由包含引擎的专门的测试脚本库集合完成的。在某种意义上，DDE 是测试自动化的中间件。第二个不同点在于 DDE 是由测试表集驱动的，测试表是用 Excel 电子数据表开发的，表中包含特殊的命令指示引擎去哪里，测试什么；在 CSDDT 方法中，这些信息预置在每个测试记录的实际测试数据中，测试记录保存在 CSV 文件或数据池中。

两种方法都存在的问题是它们都依赖于测试数据和这些测试数据的外来信息执行实际测试。如果测试数据命中率低，那么构建的测试是无效的。使自动化测试集中的惟一方法是从软件开发过程产物中开发测试数据，开发过程产物可以在测试计划/设计人员那里得到，或者让系统分析员和设计员来做自动化测试。

除了能在手工测试方面找到一两个就业机会外，IT 行业当前惟有的测试就业机会就是编写自动化测试脚本。只聘用测试执行人员意味着公司让测试工程师在测试过程中身兼两职，即测试设计工程师和测试执行工程师。这种情况在经济下滑以前就出现了，但是，测试设计人员的就业机会已经不存在了。尽管公司招聘广告中要求求职人员在测试自动化方面具有广泛的背景，但实际上只需要能够编写测试脚本的人员。现在，编写测试脚本是一个很好的职业，但我们遇到的许多工作人员都是喜欢通过编写和执行测试脚本来完成测试，而不喜欢对测试做计划和设计。结果，他们做的测试也就不能带来什么益处。

我们是这样划分本书作者的工作的：Dan 的个人工作偏好是做测试计划和设计；Bruce 是做测试执行。当我们为当地的 IT 组织提供咨询时，我们提供的服务是测试需求确定和文档（Dan），测试数据设计和构建（Dan），测试执行（Bruce）。从 2001 年人夏开始，测试设计咨询职位实际上几乎不存在。所有的求职面试主题仅集中在测试脚本的编写，重点在于求职人员是否能快速的开发出一套自动化测试脚本，这些脚本能提供自动的回归测试。根据 Kit 对自动化测试工具（捕获/回放工具）发展划分的定义［9］，我们正处在第三代。

例如，如图 3-1 所示，这是 Dan 被面试的一家财富 500 强企业对软件测试工程师职位的描述。在那个面试中，我们更加认识到实际上只存在两种职位，而且都是给测试执行人员的而不是测试设计人员。从图 3-1 中的描述，我们清楚地看到写这个文件的人根本不懂得应该由谁来做什么测试活动、测试什么时候开始。因为该描述只招收一个人，这个人既要有测试设计工程师的技能又要有测试执行工程师的技能。他们要求招聘来的人进入公司就开始编写测试脚本。显然，我们可以看到在这种情况存在的问题。他们忽略掉或者随便地编写测试计划和测试数据，这样破坏了测试过程。工作描述中惟一相关的部分是要求应聘者具有编写手工和自动化测试脚本的经验，具有 SQL 语言方面的知识，因为将来要求他们编写脚本。其余是关于测试计划和测试设计的而不是测试执行的。

测试脚本编写人员（执行工程师）为什么需要具有 CMM 的软件开发的知识？这是没有道理的。这些知识会帮助测试经理寻求测试过程的标准化和构造方法以及管理测试过程，但是对于测试执行人员来说却毫无用处。不必说了，Dan 没有得到任何一个机会。一个原因是他向面试人员解释说，如果忽略了自动化测试开发中一个很重要的方面——智能测试的设计，那么这样的自动化测试最终将失败。我解释了为什么以及在那个领域将提供什么样的服务，但是没有起到任何作用。他们决定在没有做计划和设计的情况下就开始编写测试脚本，他们这样做了，但聘用了 Bruce。然而只有测试计划和设计才能保证自动化测试执行的成功。

需要指出的一点是：编写一套自动化测试而没有首先设计测试数据无异于随机测试或者根本就没有测试。根据我们在这个领域的经验以及为多家公司在这方面提供服务的经历来看，这是 QA&T 管理的失误造成的。什么时候这些人将开创有效的软件质量控制过程以及怎样做才能实现这一目的？如我们前面说过的，我们已经在许多 IT 组织和 QA&T 部门工作过，但其中也有例外，我们也看到了测试自动化已经被正确地实施了。

信息技术组： TCCQA
发布日期： 10/2/01
开始日期： 11/1/01
任务期限： 12 个月
状态： 公开
工作地点： St. Louis

职位描述：
XX 公司软件系统开发部门关键组成员之一。

职位责任：
作为软件测试员，合格的应聘者进入公司的主要职责是保证我们的系统在发布之前达到质量标准。本职位涉及项目的整个生命周期，要保证完成验证需求，完成软件的集成、系统测试。

职位要求： 具有至少 3～4 年的软件测试经验。

在系统以及集成测试、测试计划开发、测试过程和技巧以及手工和自动化测试脚本编写方面的具有丰富的经验和知识。

熟悉使用 Rational Suite TestStudio 进行自动化测试，Rational RequisitePro 进行测试需求跟踪。熟悉使用微软 Office 编写测试计划和手工测试脚本。

较强的技术敏感性，较强的解决问题能力和决策能力，良好的沟通能力，处事灵活，较好的团队合作精神。

其他技能：
具有 CMM、SQL、Oracle、以及 Team Share 的进程知识优先录用。

雇佣职位编号 0/1

图 3-1 本地财富 500 强公司之一的软件测试工程师职位描述

随着预算的缩减，没有聘用测试设计人员来设计智能测试，这些经理就不能指明测试脚本要针对的目标。想想阿富汗战争，如果在陆地上的着弹点观察员不引导高科技的激光炸弹，炸弹将不会命中恐怖目标。所有的炸弹将会无效，除非军队采用地毯式轰炸，这种方法曾在二战中使用过。这种方法需要在广泛的目标上投下数以吨计的炸弹，而现在通过使用少量的所谓灵活炸弹就可以达到相同的结果。地毯式轰炸需要更多的工作和资源；相同的道理，随机测试也是如此。在工作职位的描述中（图 3-1）可以

看出，管理者需要测试人员进入公司就开始自动化随机测试。

测试应用程序时，如同激光炸弹要瞄准特定目标，数据驱动测试要瞄准特定的测试条件。激光炸弹的准确率是空前的，对目标的毁坏率非常高。数据驱动测试的缺陷查找率也可以如此表述。运行非智能测试（这些测试不是基于这样的知识：应用程序怎样运行和应用程序在什么地方可能出现缺陷）的自动化测试套件与地毯式轰炸很相似。使用了更多的测试，但测试的缺陷查找率较低。统计表明，不到三分之一的随机测试能在 AUT 中发现一个问题。

采用数据驱动方式（灵活测试）的一个原因是软件测试的经济因素。发现缺陷的测试越有效，测试设计和构建的投资回报（ROI）就越大。灵活炸弹与用于地毯式轰炸中的炸弹相比，投资回报与此相似。灵活炸弹前期花费高，但是能命中更多的目标；普通炸弹造价较低，但需要更多的炸弹才能取得与灵活炸弹相同的结果。设计智能测试最初比一开始就着手编写测试脚本花费得多，但是当系统发布后就可以看到真正用在软件质量方面的花费。软件产品中存在的缺陷越少，软件质量就越高，维护的费用就越低。

这里的教训——如果有的话——就是：即使在经济形势不景气时期，QA&T 经理也必须明白测试自动化没有捷径可走，放弃聘用有才能的测试计划和测试设计人员，无异于冒险，在软件向客户发布后可能需要更高的花费。

3.2 需求收集和测试计划自动化

明确和定义软件需求通常是很困难的一项工作。需求管理被看作是软件开发成功与否的关键［9、18］。过去的几年里，美国国家标准化组织（ANSI）与电气电子工程协会（IEEE）共同定义了如下标准：

- ANSI/ IEEE 830－1948：软件需求说明
- ANSI/ IEEE 830－1998：软件需求说明的推荐实践
- IEEE 标准 1362－1998：（结合 IEEE 标准 1362a－1998）IEEE 信息技术——系统定义——操作概念（ConOps）文档指南

前两条是软件工程师要遵循的需求说明的技术标准，第三条是从用户角度出发定义软件需求。

对于测试需求存在的意见主要集中在两方面，这在本章之前已提到过。为了尽量满足这两方面的意见，提出用于明确测试需求的标准方法花了很长时间。这一方法规定了把软件需求转换为测试需求的形式，且使此过程是可重复的（CMM 2）。尽管 ANSI/ IEEE 标准有一定作用，但 Gerrard 坚持认为大多数需求文档通常是“组织混乱、不完整、不准确、不一致”的[9]。他还认为大多数文档记录的需求是“不可测试的”，因为这些需求采用的形式不恰当，不是“测试人员”所期望的，对于测试人员来说这些形式是很难理解和测试的。这就需要测试工程师自己把那些需求转换为可测试的需求。

目前没有标准的文件指导软件测试的需求说明（或指导如何对已有的软件需求说明进行转换）。把已有的软件需求说明细化为软件测试需求是很困难的。为了正确测试软件组件，测试工程师必有非常细致和非常明确的信息。当然，对于测试和测试数据说明存在不同的级别和方法（黑盒、灰盒、白盒等软件测试方法）；这些概念的详细说明和和应用可参看［14、15、16］。测试需求实质上依赖于测试工程师的观点。测试工程师的观点也取决于软件需求说明包含的级别深度和细节。因此，从黑盒与白盒角度明确测试需求是可能的。

在许多情况下，没有可以转换为测试需求的软件需求说明。如果是这样，测试工程师就有两个选择。一种方法是试着自己获得测试需求，另一种方法就是告诉项目经理无法测试软件，因为没有测试需求文档。但是，在现实的世界中，许多软件需求说明书是在软件构建完成后才编写的。如果是这样，在测试开始之前或测试过程中，测试工程师必须自己找出测试需求。

为什么测试需求对于测试过程如此重要？首先，测试需求是必须的，因为测试工程师要预测测试所期望的结果。Myers 在他的倾力之作《软件测试的艺术》（《the art of software testing》）中对此做了恰当的表述：“没有期望就没有惊喜。”［16］Myers 的意思是说如果认为软件所做的事是不正确的，那么在这之前就要对软件的行为有一些认识。他还写道：“如果测试用例的预期结果没有被预先定义，那么有种情况就会出现，这种情况就是一个似是而非的结果（不一定是错误的）就会被当作正确的结果。因为我们的眼睛看到了我们想要看到的东西。”没有测试需求，就没有办法预先定义软件的预期行为。

其次，除非预先定义了应用程序的行为集合，否则不能进行测试结果的验证。这是因为测试工程师必须验证每一个测试的结果。仅仅知道期望什么样的结果是不够的，所以我们必须捕获 AUT 的实际行为，把这些行为与所期望的行为进行比较，所期望的行为是基于软件需求说明定义的。没

有预先指明测试需求，是没有办法做预测和验证的。这是软件开发经理要考虑的一个主要问题，他们需要知道测试结果是如何被验证的，他们想看到测试结果的书面证明。验证测试结果有许多法。例如，只观察 AUT 的行为就足够了，但是开发经理需要更多的证据。在这些方面测试需求起着至关重要的作用。

测试需求就是测试目标。这就是当执行专门的测试时，测试工程师想要完成的任务。除此之外，反映软件需求说明中定义的 AUT 特征也是测试需求的目标。测试需求是软件需求的下一个步骤，因这它必须能够被证明所以它必须是可测量的（measurable）。“可测量”是指测试工程师能根据测试需求所期望的结果定性或定量验证实际的测试结果。为了达到这个目的，测试需求必须被进一步细化为测试用例需求。这些低级别需求包括了比软件需求和高级别测试需求更基本的细节。测试用例需求描述了具体的测试条件，在这些细节中，从每个测试用例需求到实际执行的每个测试数据记录有着直接的关系。软件需求与测试需求有如下关系：一对一（一个测试需求对应一个软件需求）、一对多（一个软件需求对应多个测试需求）、多对一（一个以上软件需求对应一个测试需求）。与此相类似，测试用例需求与测试需求有如下关系：一对一（一个测试条件对一个测试需求）、一对多（一个测试需求有许多测试条件从而有多个测试用例需求）、多对一（一个以上测试用例需求对应一个测试条件）。在很多情况下，也可能出现多对多关系，这种关系使测试很复杂，得出的结果也很难解释，所以应避免出现这种关系。如果遇到这种情况，考虑使用分解的办法，把测试需求分解成一个或多个不太复杂的测试需求。

测试需求也必须与手工或自动化脚本相结合。例如，一个测试需求可能产生 50 个测试条件，这些条件说明了基本测试数据的功能的变化。那些测试数据以测试数据记录的形式保存在文本文件中。自动化测试（数据驱动测试）脚本可以为 AUT 导航和读取数据，然后把每个数据记录插入到合适的 GUI 屏幕中，AUT 把记录存放到数据库。为了保证测试覆盖度量，给测试需求附加上测试脚本是很重要的。

3.3 从软件需求到测试需求再到测试条件：一个自动化方法

在自动开发工具套件如 Rational Enterprise Suite 中，需求管理工具指明了软件需求的许多不同类型。自动化测试工具套件，如 Rational Suite

TestStudio 1.0/1.5 为测试工程师提供了一项功能，使他们可以把通过 Rational RequisitePro 输入的软件需求转换为测试需求用于设计和构建自动化测试脚本。

Rational 开发/测试工具套件是依据内部过程——Rational Unified Process（Rational 统一过程）设计的。RUP 描述了几个通用的软件需求类型 [20]：

功能需求——定义系统的输入输出行为。

易用性需求——定义用户界面、文档和培训资料的美观、一致性。

可靠性需求——定义故障和可恢复性。

性能需求——定义响应时间。

支持需求——定义测试能力和维护能力。

除了这些，RUP 也描述了几个面向软件构造的需求类型，其中也考虑了物理约束 [20]：

设计需求——设定软件设计的限制。

实现需求——设定软件构造的限制。

接口需求——设定软件与其他系统或用户等交互过程中的限制。

物理需求——设定在软件实现时的限制，如硬件类型等。

此外，Rational RequisitePro 还提供了另外一项功能，可根据专门的项目需要指定客户需求类型。至于测试需求，RUP1.0 声明如下：

> 测试用例是验证系统实际运行情况的方法，因此测试用例应该像需求一样来进行跟踪和维护。我们引入需求类型的概念来区分不同的需求。[20]

Rational RequisitePro 有两个默认需求类型与测试有关。这就是测试需求类型（Test Requirement，前缀为 TR）和测试用例需求类型（Test Case Requirement，前缀为 TCS）。两种测试需求都是既可以通过 RequisitePro 界面进入也可以通过 Test Manager GUI 进入。也可以使用关键字查询和所包含的算法，从微软 Word 或标准的文本文档收集需求。

一旦进入，需求被存储在相关的项目和测试知识库（test repositories）中，可以看到需求呈现为一种父子层次。我们会想到，测试需求与测试用例需求之间存在着某种关系。测试需求具有子需求，也就是相关的测试用例需求，就体现了这种关系。反过来，测试需求也可以作为子需求连接到其他的需求类型中。

Rational RequisitePro 和 Rational TestManager 等产品可以组织和连接测试需求；还可以跟踪测试需求的修改，作一些针对需求的一般维护。然而这些产品不能指明那些需求的内容；也就是说，测试脚本的设计、测试数

据的设计和测试结果的验证等细节是不能预先指定的。测试工程师必须应用他们已有的技术和经验来开发测试需求的这些细节。为了完成这些细节，他们必须为测试需求指明特定的细节，甚至为测试用例指明更详细的细节。根据测试目的，我们可以把测试需求划分为黑盒测试，测试用例需求划分为白盒测试。

测试需求说明至少要包括以下部分及其子部分：

1）测试目的

2）相关的风险因素

  a）源于项目风险评估

  b）源于预期的用法简介

3）相关的优先级别

测试需求的信息来源包括：软件需求说明书；高级别的项目风险评估分析结果；基于评估风险的测试需求的优先级；而且，如果能得到的话，还包括软件用法简介，它是给未来的用户设计的，用于指明应用程序的功能（这个信息只能使用已有的手工或自动业务系统，通过观察用户的行为得到，但不一定已准备好可用）。

测试目的应该包括黑盒测试和白盒测试说明，描述每个需求的测试目标。

测试用例需求说明应该包括以下部分及其子部分：

1）测试前设置

2）测试条件

  a）有效条件

  b）无效条件

3）测试数据

  a）用于有效条件

  b）用于无效条件

4）测试条件的预期行为

  a）对于有效条件

  b）对于无效条件

5）测试执行

  a）手工

    I. 相关测试脚本

  b）自动化

    I. 附加的测试脚本

6）测试结果验证过程

7）测试后清理

测试前设置限制说明应该包含所有的正确地执行测试所需的环境条件描述。这包括数据值说明，就是保证程序有效性所需要的测试数据（测试输入记录）或用于导航/控制的测试脚本。例如，激活一个已被删除的用户账号的测试是不能执行的，除非用户表中包含至少一个已被标记为删除的用户记录。如果要操作的用户账号不存在，那么测试设置的一部分工作就是选择一个已存在的用户记录然后把它的状态设为已被删除。一般情况下，在测试执行之前就必须建立几个这样的交互。表 3-1 和 3-2 的信息经常被用于决定测试前要设置的数据关系。

表 3-1 GUI 编写示例

GUI 字段	数据库列	编辑	错误信息	错误覆盖 (Y/N)
活动日期	ACTIVITY DATE	不能为空 必须是数字的 必须大于零 必须是有效日期	G001A G001B G001C G001C	Y Y Y Y
发票日期	INV_NOTE_DT	不能为空 必须是数字的 必须大于零 必须是有效日期	G005A G005B G005C G005C	Y Y Y Y
发票号	INV_NBR	不能为空	G006A	Y
Mfg. 代号（业主）	MFG_NBR	不能为空	G007A	Y
计划编号	PLN_NBR	不能为空 必须是数字 必须大于零	G008A G008B G008C	Y Y Y

表 3-2 服务器编写示例

GUI 字段或数据值	数据库列	编辑	错误信息	错误覆盖(Y/N)
活动日期 发票日期	ACTIVITY_DATE INV_NOTE_DT	如为欧洲日期型则更改格式 如果发票月份=活动月份则不编辑月份 如果活动月份=01 则发票月份必须为 11 或 12	 R004C	 Y
		如果活动月份=02 则发票月份必须为 01 或 12	R004C	Y
		否则活动月份必须大于发票月份并且(活动月份-1)或(活动月份-2)等于发票月份 注意：发票日期不能跨到未来月份。例如：如活动日期是 5/15，发票日期可以是 5/20，但如活动日期是 5/30，发票日期不可以是 6/1。错误信息是编写不一致	R004C	Y
		发票日期必须是有效日期	G005C	Y
		发票日期不能大于活动日期+5 天	R004A	Y
		发票日期不能比活动日期早 60 天	R004B	Y
计划编号 发票日期	PLN_NBR INV_NOTE_DT	如 TPLAN 表中找到此计划： 如果 PLN_DISC_DAYS=0（输入日期） 将发票日期与 PLN_DISC_DATE 比较 如发票日期>PLN_DISC_DATE，另加一条错误信息"日期是"后面写上 PLN_DISC_DATE 否则编写通过	R198 plus	Y
发票号	INV_NBR	TINVOICE 表中不存在发票除非符合下列条件： 如果发票存在，并且 状态代号='8'， 当前结算为 0， 这样是没有错误的	R005	Y

（续）

GUI 字段或数据值	数据库列	编辑	错误信息	错误覆盖（Y/N）
Mfg. 代号（业主）	MFG_NBR	必须存在于表 TMANUFAC 中 不能标记为删除即 MFG_DEL_CODE = ‘D’ 注意：删除重新购买代码为 ‘C’ 的第二版。TC93 在标记一个 mfg 为已被删除时将 MFG_DEL_CODE 设置为 “D”	R006 G0071	Y N
计划编号	PLN_NBR	计划必须存在于 TPLAN 表中 不能标记为删除，即 PLN_DEL_CD= “D”	R007 R335	Y Y

关于测试用例说明中的 2（测试条件）、3（测试数据）、4（测试条件的预期行为）部分，所需的信息来源广泛。结果使得查找、辨别和组织这些信息也变得更加困难。我们在很多公司做过顾问，看到过一个并不多见的好文档，它包含了编写好的测试用例说明所需要的所有信息。它实际上是可移植到客户机—服务器环境中的主机应用程序特征的设计规范文档。它包括应用程序功能特征的书面描述和表格化描述，这些可被转化为测试条件和测试数据，对于测试结果的验证也有帮助（当编写完这样的文档后，我们能用这些信息来开发和实现数据驱动的自动化测试系统）。文档的格式并不重要，但是内容（表格）可以保存到任意 WORD 文件中，用作测试用例需求的文档资源。这就是测试需求注释（Test Requirement Note，TRN）文档。

在实际应用过程中，一个 TRN 文档对应一个处于测试中的应用程序功能特征/GUI 界面，它包括与测试用例说明文档相关的很多部分。“edits”部分列出了全局的、相关的和文件编辑；编辑所需要的字段（单一的，非交叉的字段编辑）；编辑正确的数据类型（字符、数字、日期），这些编辑在客户端完成（GUI）；还包括其他在服务器端的编辑（包括交叉字段确认和从数据库读取数据）。

表 3-1 是一个如何说明客户机—服务器编辑的例子。其中包含的信息在执行测试数据时，可用于构造测试数据来对 GUI 这一级上的数据验证，也可用于指导自动化测试脚本的编写。表 3-2 说明了服务器端执行的业务规则

的编写。表 3-1 和表 3-2 包含了细化测试条件所需要的信息，从这里也可以开发出测试数据。表中还包含了在什么样的条件下会产生并显示什么样的出错信息。这些表提供了明确和编写测试需求所必须的所有信息，以及开发（分解测试需求）测试条件的信息，这就是测试用例需求。然后，测试用例需求可用于开发测试数据。

关于测试结果验证，必须要决定能够接受什么级别的验证和采用什么方法来完成验证。表 3-3 定义了 CRUD（建立、读取、修改、删除）矩阵，矩阵指明了每个数据库表，也定义了由被测试的应用程序特征决定的访问类型。这些信息是很重要的，因为这些信息经常被直接用于选定表中专门的行/列来验证测试结果。这些信息也是指明表中的值所必需的，测试之前的设置过程中这些表中的值也必须被修改。

对于手工测试，测试结果的验证由测试工程师来完成；对于自动化测试，测试结果的验证经常通过执行自动化测试脚本来完成。后者效率更高也更有效。这也需要应用程序的信息，而且这些信息不能在传统的软件需求说明中得到。测试工程师必须查找设计说明文档来得到这些所需信息。最准确的验证方法是用数据库表保存一组所期望的交互行为，然后到表中验证那些行为是否发生。

表 3-4 中的数据库访问包括了调用数据库中的每个表；列出了每种操作（检索、保存等）；包括了参数，预期的返回值（例如，状态——查到或没查到，行计数）；执行操作的原因；数据库表的列名。表 3-4 中的信息便于自动化结果的验证。

表 3-3 CRUD 矩阵示例

数据库表名	创建	访问	修改	删除
TPRODUCT	X			
TPLAN	X			
TCOMPANY	X		X	
TINVOICE	X	X		
Config File	X			
CMFLOG File				X
Help File	X			
TCSERROR	X			

表 3-4 详细的数据库访问示例

编号	表	原因	参数	返回	访问顺序
4	TPLAN	文件编辑	OFC_NBR， PLN_NBR， PLN_DISC_DAYS = '0'	状态（查到/没查到）和列： PLN_DISC_DATE	N/A
5	TINVOICE	存在性检查和文件编辑	OFC_NBR， OFC_DL_NBR， INV_NBR	状态（查到/没查到）和列： INV_CUR_BAL INV_STAT_CD	N/A
7	TPLAN	存在性检查	OFC_NBR PLN_NBR	状态（查到/没查到）	N/A
11	TPRODUCT	存在性检查	PRD_CD	状态（查到/没查到）	N/A
12	TCOMPANY	文件编辑	OFC_NBR 的前两个特征	状态（查到/没查到）和列： DATE_FORMAT_CD 美国日期型 格式 = MDY 欧洲日期型 格式 = DMY	N/A

对于验证还需要描述的另一部分是输出处理。这包括日志文件、输出文件、报告、HTML 页面和显示 GUI 字段的值。表 3-5 说明了可能要显示在指定 GUI 界面的输出值。

表 3-5 GUI 输出示例

列名	默认 GUI	用户输入	程序修改	输出值	GUI 的顺序
ACTIVITY_DATE	当前系统日期，美国或英国日期格式	X	N/A	来自屏幕 美国日期格式	1
TRAN_CODE	01	N/A	N/A	GUI 默认	5
OFC_DL_NBR	N/A	X	N/A	用户输入值	4
INV_NBR	N/A	X	N/A	用户输入值	2
CREATE_SW	N/A	N/A	N/A	始终为空	3
INV_NOTE_DT	N/A	X	N/A	来自屏幕 美国日期格式	6
PLN_NBR	N/A	X	N/A	用户输入值	8

（续）

列　名	默认GUI	用户输入	程序修改	输出值	GUI上的顺序
INV_ORG_AMT	N/A	X	N/A	用户输入值	7
INV_CURT_OVRRID_CD	'N'	X	N/A	用户输入值	9
APPROVAL_ID	N/A	X	N/A	用户输入值	10

在测试用例需求说明中还要指明的其他部分包括以下内容：

1）安全性。

2）日志审核。

3）错误处理——用于GUI、业务规则、数据库错误。

4）帮助处理。

5）CONFIG文件逻辑。

最后要说明的部分是单元测试数据。详细描述在单元测试中确认的任何特定数据需求，这些需求可以用于软件的集成、系统和用户验收测试。描述内容包括列表或指向描述专门数据所需要的文档。

只要执行了测试后清理，在测试前设置和测试执行过程中操作过的每一项都要会被重置回它的初始状态。这一步应该被作为测试用例需求说明的最后一部分来阐述。

3.4 需求管理与可跟踪性

需求可跟踪性是需求管理的一部分。需求管理还包括处理测试需求。人们需要根据成本评估需求管理程序应该做什么工作。根据这些，需求管理应用程序需要提供或支持以下特征：

1）能够接收输入和查询产品生命周期中下列各部分之间的关系——

a）业务规则

b）业务模型

c）需求

d）使用用例

e）其他面向对象或基于组件的设计图

f）测试用例

2）能够与下列工具无缝集成——

a）OOAD（Object－Oriented Analysis and Design，面向对象的分析与设计）或者基于组件的分析与设计应用程序。

b）缺陷与测试用例管理应用程序。

3）能够为所选择的软件版本建立和比较基本需求。

4）能够根据需求类型添加、修改、删除属性［3］。

各种相关的评论也强调了可跟踪性的重要。

从RM（Requirement Management，需求管理）应用程序传输数据到测试管理工具，从而创建测试用例的能力得到软件开发组织的好评。减少编写测试用例的时间并且保证所有需求都与被跟踪的测试用例相关联，那么这可以看作是降低测试的部署时间［3］。

对于开发和使用软件测试度量来说，把软件需求与测试需求相关联的能力是至关重要的。这使得测试组能够报告被测试过的需求数量（百分比）。也使得测试工程师能够指出自动化测试执行的数量和质量。例如，可以把自动化测试的百分数与非自动化测试的百分数相比较。在这样的测试中，测试数据的数量用测试条件来表示，也可以用于写报告。

还有另一个重要的特征，那就是把测试需求嵌入到其他测试文档例如测试计划中的能力。Rational RequisitePro 就是这种需求管理工具的例子，Rational RequisitePro 可以建立需求跟踪矩阵，在矩阵中，每个需求都被链接或嵌入到用微软 Word 编写的测试计划中，也可以链接或输出到微软 Excel 电子数据表中。

提供这些功能的工具加强了结构化的手工测试。建立的用户需求矩阵仅包括了进行测试所必需的字段和可选的记录测试结果成功/失败字段。把这个矩阵输出到微软 Excel，执行手工测试的测试人员就可以打开和使用存储在微软 Excel 中的矩阵信息。根据测试计划中的指示，测试人员就可以执行手工测试并把结果记录到测试需求矩阵电子表格中。执行完测试，电子数据表再输入到 Rational RequisitePro，这时矩阵就包含了每个测试需求的测试结果。这样，测试结果分析以及测试报告都变得简单得多。

3.5 功能测试数据设计

我们知道，测试设计工程师具有设计测试用例技术的基础知识是很重要的。如果你已经熟悉这些技术并且不需要复习它们，那么你可以跳到下一章开始。如果你是第一次接触它们，你需要读一些综述并复习一下 Myers

[16] 和 Mosley [15] 的著作，这两个参考文献包含了如何使用这些技术的例子的细节。

3.5.1 黑箱（基于需求的）方法

因果图 这一方法涉及到区分在需求文档里概括的特殊因果关系。原因（cause）是存在于系统中的用来将特殊系统行为解释成结果的一些条件。结果（effect）是指在处理过程中或作为处理结果的系统输出中临时存在的各种状态。原因和结果可以转化为用来创建测试用例的因果图。

等价划分 这一方法利用系统需求去区分不同类型的系统输入（在输入域中)。每一个输入类型被定义成为一个等价类。规则被导出来管理每一个等价类。这些规则被用作创建测试用例的基础。

边界分析 这一方法竭力区分每一个等价类的边界条件。这些条件用来创建包含输入值的测试用例，这些输入值往往在每个等价类边界上或者在等价类边界上下。

错误猜测 这一方法利用测试者的经验和直觉，在用其他方法得到的测试数据之间插值来获得测试数据。

3.5.2 灰箱（基于需求和代码的）方法

决策逻辑表 这一方法着重于等价类的结合。它可从包含复杂决策逻辑结构的任何需求内容中创建测试用例。这些复杂的决策逻辑结构在系统的处理过程中往往会产生 If/Then/Else 的逻辑分支。

3.5.3 白箱（基于代码的）方法

基本测试 这一方法设计的测试用例往往遍历每一个程序模块的每一个控制流。它根据模块内部的分支在控制流出现的时候定位它。在 Visual Basic 中，分支指 If/Then/Else/Endif 语句、GoTo 语句、For/Next 循环、While/When 循环以及 Select Case 语句的结果。源代码或者程序流程图被用来确认每一个模块中分支点。通过这些信息可以构建用于设计测试用例的控制流图。

测试的目标是搞清逻辑路径的基本集。基本集由一套控制路径组成，这些控制路径包含了整个模块所有路径共同的路径元素。完全的模块路径集可以通过分解基本路径并用不同的组合重新组合它们来得到。每一个不同的组合都说明了一个模块逻辑路径。

每个模块的逻辑路径都是有限的，但是这一数量往往是很高的。对于只测试基本路径的情况，这是因为：如果对所有路径相同的构造块已被测试过，你就不需要测试模块的所有逻辑路径。这导致了用于测试每个模块的测试用例数量的显著减少。

3.6 基于需求的方法

3.6.1 需求驱动的因果测试

Elmendorf 把因果图方法描述为“受过训练的基于规范的测试”［4、5]。在 Elmendorf 工作的基础上，Myers 把因果图定义为“将自然语言的规范翻译了的一种形式化语言”。因果图是一种“综合逻辑网络”。与标准电子符号相比，它使用类似的符号，但比标准电子符号相对简单。更精确地说，它通过布尔图将功能说明书的语法内容描述为原因（输入）与结果(输出）之间的逻辑关系。

由于因果图是一种黑箱技术，所以它可以与类似 Desk Checking 和 Walkthrough 的评审过程结合用于开发过程的早期。由于产生的测试用例可用于从单元测试到系统测试的所有测试级别，所以它是一种通用的方法。

因果图　因果图是复杂的叙述性软件描述的模型，例如数字逻辑环。它们可以很容易的用于设计功能测试用例［4］。每一个环代表一个在说明书中画出的语义。在因果图中，语义信息被翻译成了有限项决策表(Limited Entry Decision Table，LEDT)。这些表可用来构造实际的测试用例。有限项决策表是一个二元真值表，每个规则代表在整个程序段中的一条逻辑路径。

使用和理解因果图惟一需要的预备知识是布尔逻辑运算符。我们最常用的运算符是 AND、OR 和 NOT，然而，NAND 和 NOR 在某些情况下也是需要的。共包括四种基本符号和两种很少使用的相反形式。在 Myers [16］和 Mosley［14，15］的著作中可以看到因果图的全面的例子。见表 3-6～3-10。

表 3-6 逻辑与（AND）真值表

A	B: 1	B: 0
1	1	0
0	0	0

注意：参与与运算的变量是 A 和 B

表 3-7 逻辑或（OR）真值表

A	B: 1	B: 0
1	1	1
0	1	0

注意：参与或运算的变量是 A 和 B

表 3-8 逻辑非（NOT）真值表

	B: 1	B: 0
A	0	1

注意：参与非运算的变量是 A 和 B

表 3-9 逻辑与非（NAND）真值表

A	B: 1	B: 0
1	0	1
0	1	1

注意：参与与非运算的变量是 A 和 B

表 3-10 逻辑或非（NOR）真值表

A	B: 1	B: 0
1	0	0
0	0	1

注意：参与或非运算的变量是 A 和 B

等价 定义了一种状态，如果 X 节点真，那么 Y 节点也真。布尔语句如下：if X=1，Y=1，else Y=0

与 定义了一种状态，如果 X 和 Y 都为真，那么 Z 也为真。布尔语句如下：Z=1 only if X=1，Y=1，else Z=0。

或 定义了一种状态，在 X 或者 Y 有一个为真的时候，Z 也为真。布尔语句如下：Z=1 if X=1or Y=1，else Z=0。

非 定义了一种状态，只有在 X 是假的时候，Y 才会为真。布尔语句如下 Y=1，if X=0，else Y=1。

与非 定义了一种状态，如果 X 和 Y 都为假，那么 Z 为真。布尔语句如下：Z=1 only if X=0 and Y=0，else Z=0。

或非 定义了一种状态，在 X 或者 Y 没有一个为真的时候，Z 为真。布尔语句如下：if X=1 nor Y=1 ，Z=1，else Z=0。

注意：使用负逻辑会导致逻辑组合具有不必要的复杂性，在可能的情况下要尽量避免。如果遇到与非或者或非的情况，在设计因果图前应该正向重新定义逻辑。

从因果图的观点来看，由于语义或者语法的约束，有些原因的组合是不可能的。此外，某些结果可以掩盖其他的结果。当这情形发生，应该在

因果图中标出。因而，约束符号必须与基本的因果图符号一起使用（如图 3-2）。

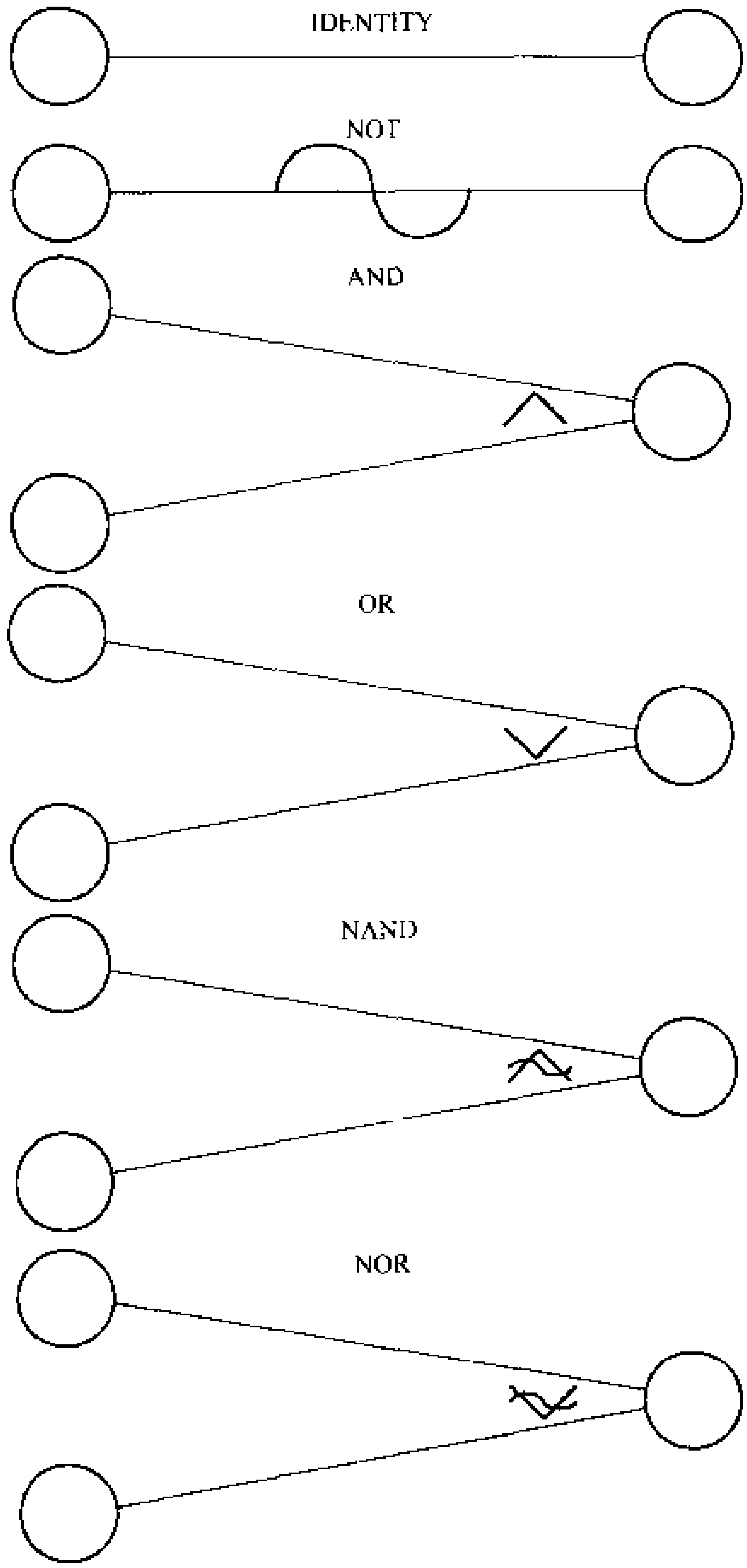


图 3-2 因果图基本符号

1. 原因中的约束

排他性约束 定义了一种状态，原因 X 和原因 Y 不能同时为真。If X

=1，Y=0；if X=0，Y=1；但是原因X和原因Y可以同时等于0

包含性约束 定义了一种状态，原因X或原因Y有一个总是为真。If X =0，Y=1；if Y=0，X=1。原因X和原因Y可以同时为1，但是状态X=0且Y=0不可能出现。

必要性约束 定义了一种状态，当X为真的时候，Y也必须为真。If X =1，Y=1。状态X=0，Y=0和Y=1，X=0也可以出现。

惟一性约束 定义了一种状态，有且只有X或Y中的一个可以为真。If X=1，Y=0；if Y=1，X=0。原因X和Y不能同时为1或同时为0。

2. 结果中的约束

掩码标记 定义了一种状态，如果结果V为真，那么结果Z一定为假。If V=1，Z=0。

总之，包含性、惟一性和必要性约束用于逻辑与操作，排他性约束用于逻辑或操作（见图3-3）。

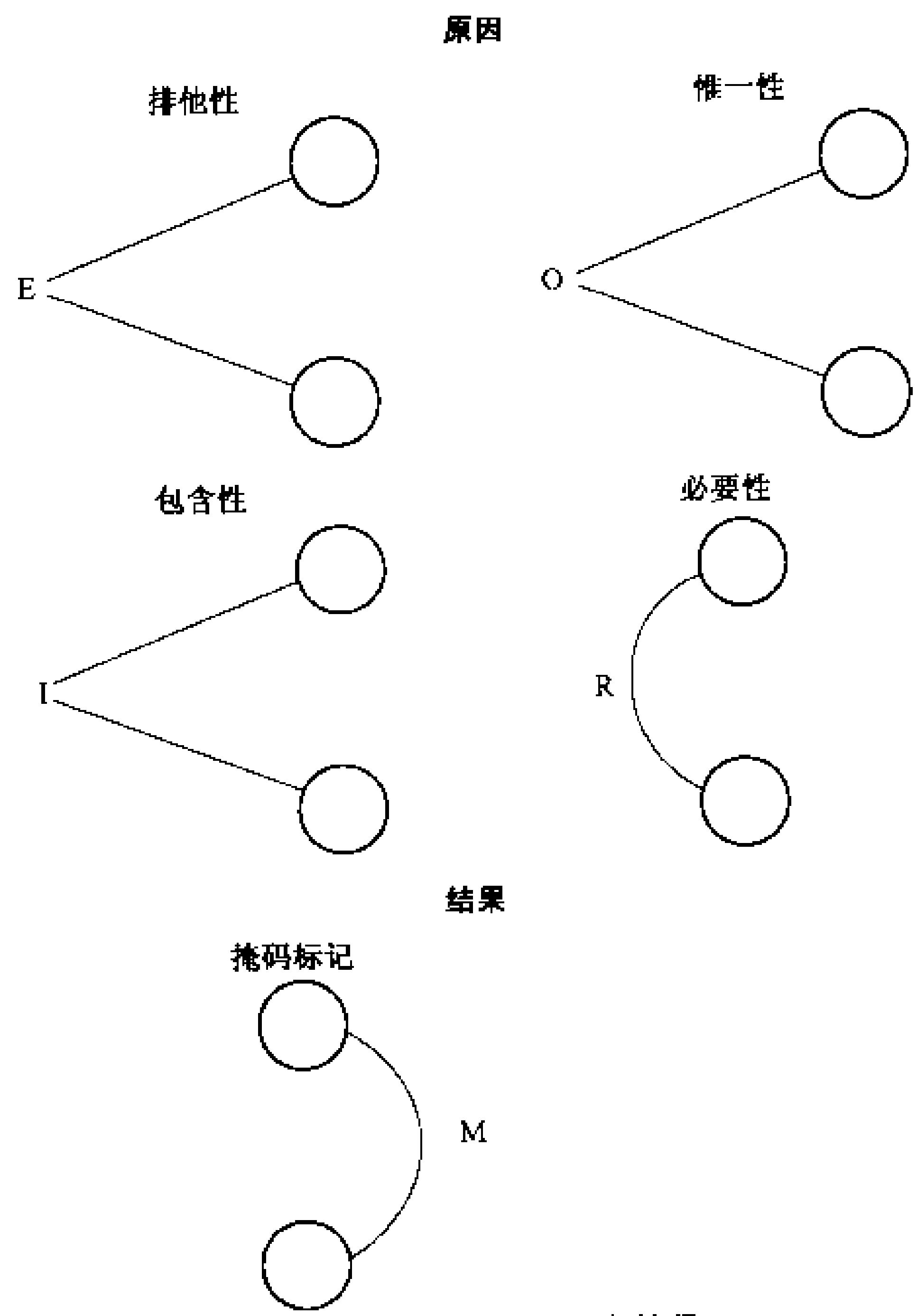


图3-3 因果图——约束符号

设计测试用例　以下是通过因果图导出测试用例的过程，这一方法来自于 Myers 的工作 [16]。

1) 将说明分解成可工作的块。不要试图为整个说明创建一个因果图。一个大的说明往往太复杂，所以必须分解成更易理解的小（低复杂度的）的块。

2) 在每一个说明块中区分原因和结果。

原因是惟一的输入条件或输入条件类（等价类）。

结果是输出条件或者是系统变换（系统数据库的替代）。

3) 将每一个段中的语义关系转换成在因果图中联系原因和结果的布尔关系。

4) 用影响原因和结果的约束限制标注每一个图。

5) 跟踪二元条件状态（在图的每一个节点中被理解为 true-false，或者 0－1）并确认每一个连接原因和结果的惟一二元状态组合。

画一个有限项决策表来概括所有可能的条件－状态组合。

在表的条件栏里列出原因，在行动栏里列出结果（参见用于定义这些术语的决策逻辑表的讨论）。

在表的条件输入象限中，描述每一个条件状态（原因的）组合。

将表的输入分解到规则中去，每一个对应因果图中的条件状态的一个惟一组合。

在列中用 X 标记出与特殊结果相关的条件状态组合，这表明条件状态组合（规则）紧挨着被调用的结果。

6) 将决策表中的每一列（规则）转化成测试用例。

在因果图（参见前面第 5 步）中区分条件状态的惟一组合是个相当困难的任务。而且，它还不像下面章节要讲到的决策逻辑表（Decision Logic Table，DLT）那样可以进行完备性检查。但是，我们可以从结果向前追述，通过使用基本输入值的中间输入值的每一个可能组合，可以产生一组清晰的条件状态。这一过程也是来自于 Myers [16]。

1) 一次操作一个结果（输出）。

2) 将结果置于真（1）状态。

3) 沿着图形向后追述，确定所有导致结果为真的原因（输入）的组合。由于原因上的约束，可能的组合将会减少。

4) 按照前面的方针，在决策表中为每一个组合创建一列。

5) 为每一个组合的所有其他结果定义状态（true 或 false，1 或 0）。

3.6.2 等价划分、边界分析和错误猜测

等价划分和边界分析是互补的黑箱测试用例设计策略，它们在开发生命周期的早期是很有用的。它们是将书面的说明转换成基于功能的测试数据的技巧。在功能说明中描述的输入约束被用来定义输入和输出类别。等价划分只是描述了输入类别，而边界分析即定义了输入类别也定义了输出类别。

这两种技巧产生了两种基本输入类：有效输入类和无效输入类。在大部分例子中，用一个单独的类来描述有效输入，而用好几个类来描述无效输入类型。每一个类的测试用例代表被创建并被添加到了测试数据集中。

Myers 为区分等价类建立了一套方针，并为构建覆盖每个类的测试用例建立了一套规则。一套额外的规则指导了等价类边界的测试用例的创建[16]。

定义等价类 定义等价类在某种程度上是实验－错误过程。它是建立在大量的直觉和过去经验的基础上的，并且是来源于你以前做的或者正在做的测试工作。下面的一些一般指导方针可以加速这一过程（也是来自 Myers [16]）。

1) 对于指定可能值（连续输入）范围的输入描述，需要识别出一个代表指定范围内的值的有效等价类。还需要识别出两个无效等价类—— 一个代表在范围之上的值，另一个代表在范围之下的值。

2) 对于不同处理而定义的一组值的输入描述，需要识别出每一个值的一个有效的等价类和另外一个代表不在集合内的值的等价类。

3) 对于输入数据的类型（例如：数值型和 Alphebetic 型），需要创建一个代表正确数据类型的等价类和至少一个代表不正确数据类型的等价类。

4) 对于具有特殊的强制性条件（例如：对于 PART-NUMBER，在 PART-NUMBER 的第一个位置必须是字母）的混合数据类型（例如：Alphanumeric 类型），需要识别出一个满足条件的等价类和一个不满足条件的等价类。

5) 回顾等价类，寻找可进一步细分类别的实例。这些类只有当值在用不同规则处理的类中被发现的时候才能被进一步细分。

Myers 认为创建一个具有如下三列的表是很有用的：左边一列分别列出所有的外部输入限制；中间一列用来描述有效等价类；右边一列放置无效等价类。为表中的所有等价类随意编号。按这种方式组织和编号等价类简化了测试用例的构建 [16]。

从表格化的等价类中构建测试用例 用于构建测试数据的规则应该以节省测试为基础。它们预备最少数量的测试用例，这些测试用例以我们可有效测试的合理置信度被创建和被安全执行。创建更少的测试用例是可能的，但是由于错误掩盖（error-masking）现象，这些失败的测试用例会降低测试有效性。

错误掩盖现象在一个测试用例包含多于一个的无效值时候会发生。在测试中，软件独立地评价每一个值，所以在第一个无效值被发现的时候，评价过程将被停止。如果发生这种情况，剩余的无效值将不能得到处理，这就意味着我们不能确定它们被处理时会发生什么。这一现象最好的例子是（在一些程序设计语言中）条件语句中的与操作（一种布尔操作）。如果第一个条件是 false 或 off 状态，那么并不检验第二个条件。问题就是我们无法确认关于第二个条件的系统行为是什么，除非第二个条件被评价了。

Myers 启发式地定义了测试用例构建指南的简化版：

1）所有的有效等价类可被包含到一个测试用例中。

2）每一个无效等价类必须由一个单独的测试用例表示。

3.6.3 为等价类定义边界条件

Myers 将边界条件定义为等于、大于或小于等价类边界的值［16］。直接的观察结果是这样的，如果使用如前所设定的规则设计等价类的，那么大于或小于等价类边界的值已经被识别了。因此，我们将创建的惟一新测试用例是那些代表每一个边界的值。因此，边界分析在执行顺序中对于等价划分来说是次要的。此外，在两个技巧之间还有一个主要的不同：边界分析还可用于输出域。

下面的相关指南同样也是来自于 Myers 的工作［16］。

输入域

1）对于连续输入，编写代表取值范围内的最高和最低有效值的测试用例。

2）对于离散输入集，构造代表集合（例如：一顺序输入文件）中的第一和最后一个元素的测试用例。

输出域

1）对于连续输出，构建的输入测试用例应该使产生的输出结果是输出范围的最高值和最低值。

2）对于离散输出集，构建的输入测试用例应该能够确保第一个和最后

一个输出元素能够被处理（例如：确保输出报表的第一行和最后一行的细节能够被打印出来）。

3.6.4 错误猜测

错误猜测是利用直觉和过去的经验对测试数据集进行插值的过程。它是没有规则可遵循的。测试者只能用肉眼扫描数据集，发现丢失的条件。两个熟悉的易错的例子是除数为零和计算负数平方根。这些都会产生系统错误和错误的输出。

经验可使我们易出错的其他例子是变长表的处理、奇数和偶数人口中值的计算、循环主文件/数据库更新（不正确的使用复制键，不匹配键等）、存储区域交叉、重写缓冲区、忘记初始化缓冲区大小等等。我敢保证你还能想出很多关于你的硬件/软件环境的独特情况和使用特殊的程序设计语言的例子来。

错误猜测与等价划分和边界分析有着同样重要的作用，它弥补了这些功能的内在不完备性。等价划分和边界分析互补，而错误猜测方法有效补充了这两种技巧。图 3-4 列出了这些方法的控制结果。

等价划分/边界分析表 1
GLOBAL_TBL 测试条件

输入条件	有效等价类	无效等价类
Connect_Type（连接类型）	字符“P”“I”“F”“D”	除了“P”“I”“F”“D”
		空
Start_Time（开始时间）	HHMM 值 24 小时	空
		非 HHMM 值
End_Time（结束时间）	HHMM 值 24 小时	空
		非 HHMM 值
Pause_All（暂停）	“N”“Y”或空格	非“N”“Y”或空格
Stop_All（停止）	“N”“Y”或空格	非“N”“Y”或空格
Poll_Increment（登记增加）	秒为单位整数	非整数
Redial_Increment（重拨增加）	整数值设为 900 秒	少于 900 秒的值
		大于 900 秒的值

图 3-4 黑箱测试用例设计图示例

		负值
		非整数值
Site_Loop_Time（站点循环时间）	整数值设为 60 秒	大于 60 秒的值
		小于 60 秒的值
		负值
		非整数值
Connection_loop_Time（连接循环时间）	整数值设为 60 秒	大于 60 秒的值
		小于 60 秒的值
		负值
		非整数值
Connection_Wait_Time（连接等待时间）	整数值设为 600 秒	大于 600 秒的值
		小于 600 秒的值
		负值
		非整数值

图 3-4　黑箱测试用例设计图示例（续）

3.7　混合（灰箱）方法

3.7.1　决策逻辑表

DLT（Decision Logic Table，决策逻辑表）是独特的，因为它们可以通过系统功能说明片段构建或者以程序流程图或源代码清单为基础。从测试的观点来看，这一方法又可被认为是白箱或黑箱方法，因此是灰箱类。

然而，DLT 是测试工具。那么它为什么能被包含到测试用例设计策略的讨论中呢？主要是因为 DLT 可被看作一种基本测试所不具备的路径覆盖（path coverage）方法。DLT 格式可以确保测试的完备性，确保特殊的模块中没有路径被遗漏。

模块的结构复杂性可以通过每一个决策的每一个条件的可能状态的乘积计算得到。如果引用 McCabe 的度量得到 C 值，这一结果与 C（Cyclomatic number，秩数）相比是一个复杂得多的值［11、12、17］。产生的这个值代表了条件状态可能逻辑组合的总数目，并不是独立组合数目。因此，整个表的复杂度值不同于秩复杂度［8］。知道整个表的复杂度的好

处是可以验证逻辑完备性。DLT 方法解决了白箱测试方法带来的主要批评——它不能解释遗失路径。

在内嵌的 DLT 中为每一个规则开发一个测试用例能够为特殊的程序模块创建完全的测试用例集。每一个程序模块可构造一个 DLT 图。来自每一个 DLT 的测试用例随后被融合形成了测试数据集。随后，它们再依次与用其他白箱或黑箱方法创建的测试数据集融合。

你们当中的很多人可能已经熟悉了 DLT 并想跳过这一节。然而，如果你阅读下面内容，它会更新你对 DLT 的概念。

从测试的观点来看，DLT 是保证软件可靠性的一个重要的工具 [1、14]。DLT 是一种用于辨清复杂逻辑的表格式的图。那些复杂逻辑在设计文档中已被明确规定。DLT 只能处理条件逻辑，使设计者可以容易理解很多决策步的状态。这些状态最终将以 If/Then/Else 形式出现在程序段中。DLT 的一个很明显的好处是它的决策制定逻辑避免了 If/Else 嵌套。这些嵌套在描述性语言和程序代码中是很常见的。甚至连结构化的英语描述都包括 If/Else 嵌套。

If/Else 嵌套的问题是提高了结构复杂度。我们处理复杂度的能力在达到一定水平后会迅速地下降 [13，19]。当这种情况发生的时候，我们开始把错误带到我们的工作中。决策表的目标可表述如下：

决策表的目标是把叙述减少为可容易地在 If/Then/Else 形式中实现的条件和行为集。

条件和依赖这些条件的行为代表了除了简单序列（顺序执行的程序逻辑）以外的所有东西。任何软件程序都可用顺序、选择和循环语句设计出来并使用相应程序设计语言的构造器构造出来。对于选择结构，行为依赖于被评估条件的状态（If/Then/Else 逻辑）。大多数选择结构局限于两种相互排斥的状态。但是，有些决策涉及评估带有两个以上互相排斥状态的条件。DLT 格式并不区分有限的输入条件和扩展的输入条件。它只是组织条件和行为，使正确的行为与恰当的条件状态组合相关联。

循环结构（循环逻辑）可也用条件和行为来建模。什么是循环？它是可重复的行为集。在某些情况中，行为集被不断重复直到代表出口的条件为真（While 或者测试前循环）。其他情况是行为集被不断重复直到代表出口的条件为假（Until 或者测试后循环）。任何循环都可被指定如果知道两件事情：不断重复的行为（依赖跳出标准的行为）和需要满足的跳出循环的条件（跳出标准）。因此，循环可用 DLT 描述。

DLT 格式 DLT 是具有四个象限的表格图：条件存根、条件输入、行为存根、行为输入（如图3-5）。条件以它们对处理逻辑的影响为序自顶向

下排列在条件存根中。更加全面的条件被列在了开头。例如，文件结束(End-of-File）是最全面的条件，因为一旦这个条件满足，处理将结束。因此，它应该被列在最前面。另一个例子：控制中断（control break）处理，最内部处理级应该被作为最新条件列在DLT表的条件存根中；而表示最外部级别的条件应该首先列出。

条件－状态名被放置在了条件输入象限中与它们定义的条件相应的那一级。特殊条件的条件状态数定义了条件的复杂度。

行为被放置在了行为存根中。它们以将要执行的顺序自顶向下的排列。哪个行为依赖哪个条件状态组合在行为输入象限中进行了详细说明，它们被分成了规则。

一个规则是整个表的输入部分（包括条件和行为）的一垂直列。它表示了条件状态的一个独特组合和组合发生时的执行行为。在DLT表中，规则的数量是条件存根中所有条件的条件复杂度的乘积的函数。一个DLT表包括三个条件，每个条件有两种可能状态，那么它就有8个规则（2×2×2＝8)。值8还代表了表复杂度。表复杂度是表的所有逻辑路径数目的测度。

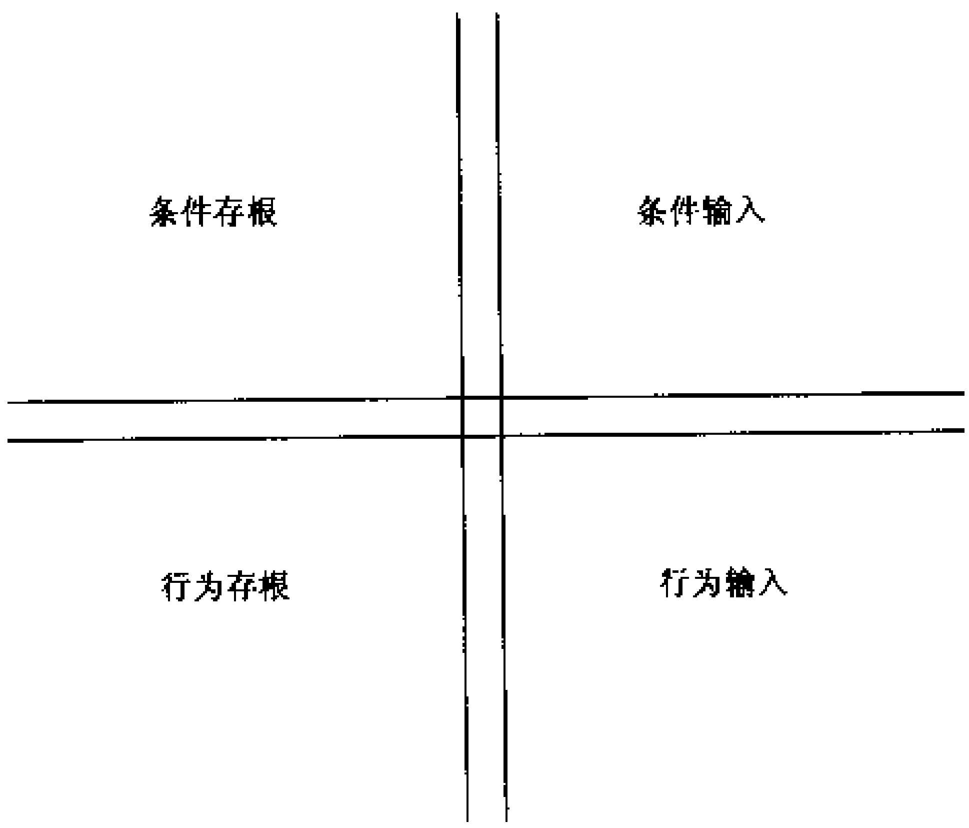


图3-5　决策表格式

列举规则

步骤一：决定DLT表中的规则，将表的输入边分成与条件存根中的第一个条件（最全面的）的条件状态相同数量的规则。

步骤二：将前面创建的每一个子划分分成与第二个条件（第二全面的）的条件状态同样数量的规则。

步骤三：对条件存根中的剩余条件重复步骤二。

按照这种方式设计 DLT 表将确保所有可能的条件状态组合都被考虑到。

指定行为 一旦所有的规则被列举出来，那么依赖每个规则的行为就可以被指定。不止一个行为可以对应同一个特殊的规则。如果几个行为依赖一个规则，从上到下将它们在行为存根中列出来。在已经执行了行为的对面规则的那一列做上 X 标记。

条件—状态的忽略 某些条件状态在它发生的时候，会否决其他的条件状态。这种情况下，主导的条件状态对于随后的条件状态值是不重要的，这些条件状态可以完成定义规则的状态的逻辑组合。

忽略是很重要的，因为一旦这种情况发生表可能会崩溃。在崩溃的表中，规则具有不同的逻辑重要性，但产生同样的一组行为的规则将被结合到一个规则之中。表将被描述得更加精确。从软件测试的观点来看，要为表中的每一个规则产生一个测试用例。如果规则因为条件—状态忽略面失败，那么大量的测试用例就不必要构建了。

表的崩溃并不从某种程度上改变表的复杂度。崩溃的规则解释了不止一条逻辑路径。规则复杂度被定义为规则解释的条件状态（路径）的逻辑组合的数目。

Gane 和 Sarson 为 DLT 崩溃制定了一些策略 [8]：

1）找出除了一个条件不同，而所指向的行为和条件状态值均相同的规则对。

2）用单个规则代替这个规则对，在不同的条件处使用忽略符号（～）。

3）用策略 1 和策略 2 处理任何一对符合忽略准则的规则。

完备性证明 对于给定的 DLT 的完备性证明基于这样的事实：整个表的复杂度可用条件复杂度的乘积或规则复杂度的和来表示。如果条件复杂度的乘积等于规则复杂度的和，那么表就是完全的。没有条件状态的逻辑组合被遗漏。

完备性之所以可以被证明是由于 DLT 的基本前提条件：

给定具有有限数量的条件状态的有限数量的条件，那么存在确定数目的组合。

3.7.2 DLT 作为软件测试工具

由于路径的数量是依赖于模块中决策的数目的（参见下面的基本测试

讨论），所以DLT对于基于覆盖原则设计测试用例是个很好的工具。它的优秀还在于DLT将模块中所有的决策都融入到了DLT格式中——每一个可能的条件状态组合（每一个组合就是一个逻辑路径）都被覆盖了[6、7]。再者，前面讨论过的完备性检查确保了没有路径被遗忘。

因此，每一个表都导致有限数量的测试用例被添加到了测试数据集中。由于测试用例是根据表中的每一个规则产生的，测试者可以从路径覆盖的观点确信完全测试了模块的逻辑。此外，如果内嵌的DLT可能会失败，那

设计器完备的功能包括可为表添加遗漏规则的完备性检查和可分解表到最简练形式的减少（Reduce）命令。设计器可以将完成的决策表转换成目标程序设计语言的语句和/或者英语语句。

再设计模块允许 DLT 从 C 源代码段中转化而来。它不支持其他任何语言的再设计。再设计组件使用被称为消除二义性的过程自动地消除源代码和表中的矛盾和冗余。

虽然每一个表被限制在 20 个条件、20 个行为和 10240 个规则，LogiCASE 支持嵌套表结构，这就使用户可将大的表分解为一组较小且不那么复杂的表。它通过包含其他表的 include 文件代替行为和条件实现了这一功能。

你可从 Logic Technologies 公司获得 LogiCASE 产品。联系地址是 56925 Yucca Trail，Suite 254 - A，Yucca Valley，CA 92284；电话 1 - 800 - 776 - 3818；传真 1 - 619 - 228 - 9653。

3.8 基于代码的方法

3.8.1 基本测试回顾

基本测试是一种利用控制流图来代表程序模块逻辑的白箱测试用例技巧。控制流图是图形化地描述模块逻辑路径的网络图。测试用例的创建是基于从图形中列举的一组独立路径。Thomas J. McCabe 发展了这一方法，它一般正式地被称做结构测试，而非正式叫法是基础测试。McCabe 的方法要被当作白箱测试用例设计策略讨论，而不能被当作测试方法学。要对 McCabe 的方法学有一个全面地了解，可参见他的工作成果［11、12］。

McCabe 方法的主要优点在于它将秩数作为程序/模块复杂度的测度。缺点是控制流图或者从模块编码前的流程图产生或者从构造完成后的源代码列表中产生。这种方法有两种问题：第一，控制流图本质上是合逻辑的，而流程图和源代码列表具有固有物质性或依赖于执行。第二，还没有明确的策略将包含在流程图中和源代码列表中的信息系统地转化为控制流程图中描述的逻辑细节。

控制流图构造发生在设计过程的早期，而且测试用例设计在实际设计（通过流程图）和编码之前发生。这方便了设计评审、遍历（Walkthrough）和审查（Inspection）；更重要的是程序和模块的测试被放置到了正式结构化系统设计方法学的框架中了。

3.8.2 基本测试技巧

基本测试被 McCabe 描述为通过源代码或程序流程图产生程序模块的控制流图。流程图可用来列举独立的控制路径和计算秩数 V（G）——一种过程复杂度的测度（参见 Myers 的秩复杂度测量的扩展［17］）。覆盖独立路径的测试用例的基本集随后被创建。

控制流图是一种用节点代表程序段，边代表从给定程序段到别的程序段的连接的一种网络图。边描出了从一个节点到另一个节点的控制转移，可以代表条件的和非条件的任何转移形式。节点代表了一种具有多发射边的决策。循环是一个具有返回节点的发射边的节点。区域是图内被节点和边包围的范围。每一个图都有入口和出口节点。路径是整个图中从一个节点到另一个节点穿越边的路径，该路径从入口节点开始到出口节点结束［11］。

节点是顺序执行的行为块，而边是一个块到另一个块的控制转移。If/Then/Else/Endif 和嵌套的 If/Then/Else/Endif 是有条件地转移控制到特殊行为组的语句的例子。

程序模块中的内部结构复杂性是模块执行功能数、模块的输入和输出数、模块的决策数的结果。结构化编程学说指出一个模块应该执行且只执行一个功能［21、22］。如果这一基本原则被遵循了，由功能数引起的复杂度就会最小化。结构化程序还基于如下一个前提：每个模块有且只有一个进入点和退出点。如果这样的话，由于模块接口的简化，由输入和输出引起的复杂度将会得到控制。这样决策数就成了引起模块复杂度的主要结构化维因素。

McCabe 复杂度测度是模块的决策结构的指示。它通过控制流图中决策的数量来测量过程复杂度［11、12］。秩复杂度是个有用的度量，这是因为它也代表了感知的复杂度。Miller 发现人类大脑可同时处理的最大信息量是 3 比特（用比特作为信息的单位是为了区分两个平等的可能选择）。整个可选择事物的数目是 2 的幂，它等于包含在复杂决策中的区分数目［13］。

根据他的发现，Miller 制定了 7±2 规则。由于每个选择都是可区分的，7 个选择是人脑默记的最优值。Miller 确定人类可同时处理的最多选择数是 9［13］。如果这一定理被用到程序模块的决策结构。程序必须处理以理解复杂决策的比特数是 2 次幂的函数，它代表了在决策中包含的条件的数量。一个 3 级深度的 If/Else 嵌套包括 2^3 或者 8 个选择，它低于 Miller 给出的限制。再增加一个决策嵌套就会使选择达到了 16，超出了限制。因此，秩复杂度大于 10 的模块，对人的瞬间记忆来说是难于完全理解的。所以如果 C 大于 10，那么模块就应该被重构，否则它将可能不可测试。

秩数可用三个简单的公式来计算。第一个表明秩数的公式是

$$C=E-N+2$$

式中，C 是秩数（为了简化符号和使符号更有意义，C 用来代替 V（G），2用于代替 $2P$（P 通常为0））；E 是边的数目；N 是节点的数目。秩复杂度可通过边与节点的关系函数来计算。

复杂度也可以根据控制流图中的区域的函数来计算。边不能够互相交叉，并且违反这个规则形成的区域是不合法的区域。只有合法区域才可代入复杂度公式进行计算。

基于区域数的计算公式如下：

$$C=R+1$$

式中，C 是秩数；R 是合法区域数。

秩数也可以根据图中的简单决策数来计算。简单决策是这样一个分支，它评估一个与单一条件相关联的条件状态。嵌套的条件和由逻辑运算符（AND、OR 等）连接的条件应该被看成完全独立的决策。

公式如下：

$$C=D+1$$

式中，D 是简单决策的数目。

后两个公式的优点是，它们更容易理解和使用。事实上，控制流图没有必要构建成使用第三个公式。因为程序流程图或源代码列表中的决策数是可以数得到的，并且可以被用来代替公式中的 D。

这三个公式应用于单个程序模型。得到总体程序复杂度的一个方法就是基于单个模型复杂度（减去多余节点）的总和。多余节点出现在高级模块中，比如主线模块，如果被调用的模块是作为上级模块的一部分联机调用，那么高级模块将被子图代替。

公式如下

$$C_t=C_i+2-(N-1)$$

式中，C_t 表示总体程序复杂度；C_i 表示单个模块复杂度；N 是程序中的模块数。

每个模块中的 C 值表示了在一个给定程序单元中独立路径的最大数目的上限。如果一个控制路径集构建成等于 C，那么用来测试这些路径的测试用例将充分测试模块并构成测试用例的基础集。这个基础集并不需要测试程序段中的所有可能路径，而是一个子集，所有其他路径都可以由这个子集虚构出来。

一旦秩数已知，那么独立路径就可以列举出来。要列举出基本路径集需按如下步骤：

1）用惟一的字母或数字标志所有节点。

2）从入口节点开始，使用最左边的路径遍历整个网络，直到出口节点。列出包含在路径中的所有节点，并在图上显示出经过的边。

3）沿原路返回直到遇到这样一个节点，该节点有一条或多条与它相连的未被标志的边。从入口节点开始，沿原路径到该节点（有未被标志的边），使用最左边的未被标记的边到出口节点。

4）一旦新路径横断了先前的路径，则沿后来的路径到出口节点。

当不存在未标记的边时，基本路径集就完成了。其中，路径总数必须等于 C。如果路径数不等于 C，那么模块就设计得不好，并且过于复杂。最好重新设计这样的模块。

当构建测试用例时为具有被测试条件的决策节点做注释，并标上真和假分支路径，这样做是非常有帮助的。一个测试用例至少可以测试每条独立路径。测试用例的基础集也必须至少测试每一个条件分支一次。一个测试用例就是一个输入数据记录，它为每个数据字段包含一个有效值或无效值。图 3-6 说明了这种技术。

3.9 小结

创建测试需求和测试用例需求说明文档需要标准的格式。这个格式必须包括在本章中所描述的信息的类型和级别，并且文档必须导入到 Rational RequisitePro 或者其他相似的产品，这些产品定义了测试需求并将它们和测试过程中其他产物联系起来。至少，测试需求必须和测试数据说明联系起来。对于测试执行覆盖目的来说，它们必须和执行这些数据的自动化测试脚本联系起来。

测试用例需求必须被转换成电子数据表中的测试条件。测试条件应遵循在本章功能测试数据设计部分给出的指导原则来构建。测试设计工程师应该使用黑盒（基于需求）过程来为自动化系统/回归测试开发测试数据。软件开发人员应该使用白盒（基于编码）过程来构建单元和集成测试数据，可利用像 JUnit 之类的工具。

成功的关键是要将动态测试计划文档中的测试需求和测试数据连接在一起，这个文档是使用自动化的需求收集工具如 RequisitePro（作为 Rational Suite TestStudio 中的一个组件）开发的。测试计划作是作为 RequisitePro 工程中的一个文档来生成并维护；因为与工程联系在一起，所以测试计划可以看到同样也在 RequisitePro 中的需求矩阵。

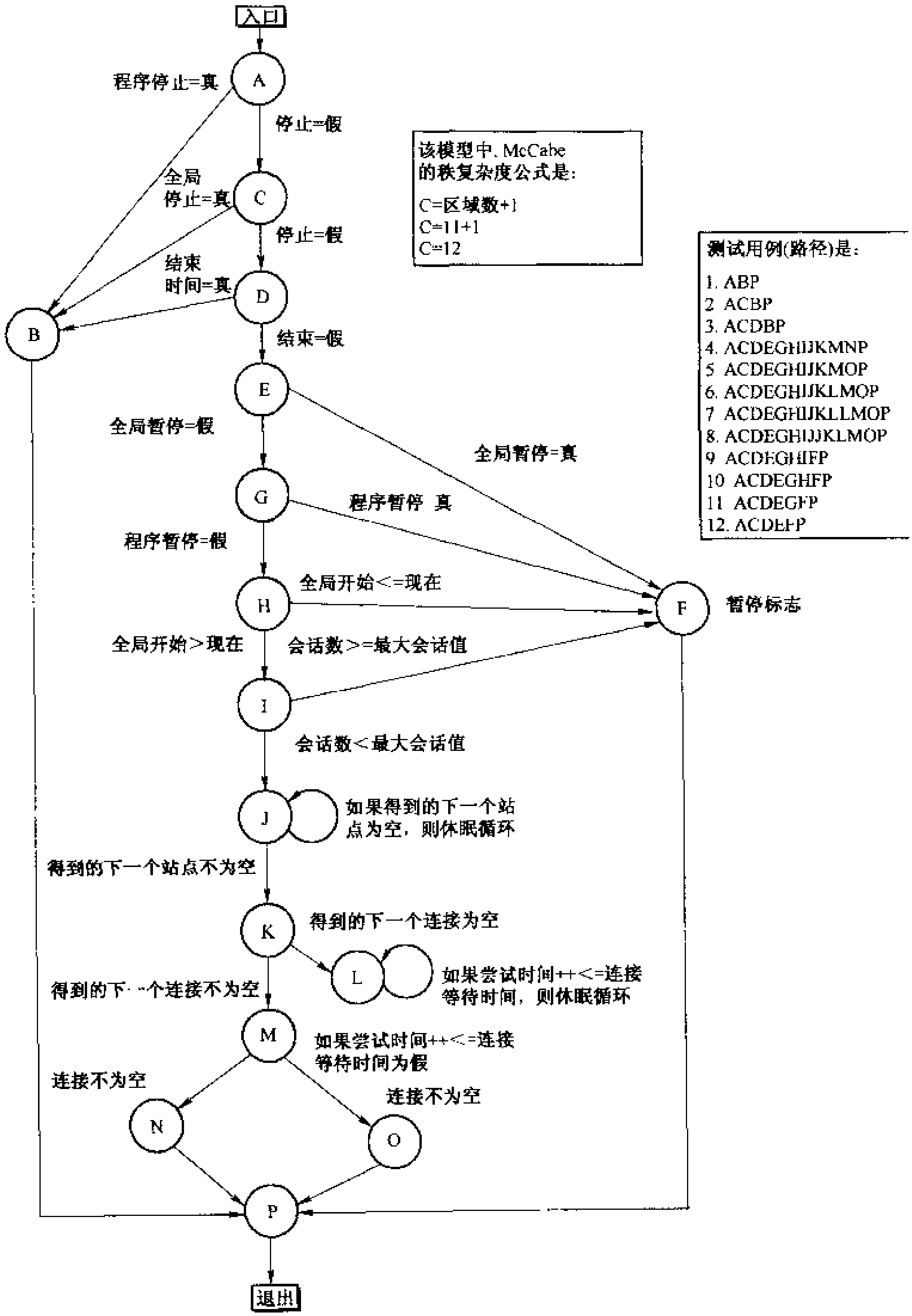


图 3-6　Arc Polling 软件的主体模型控制流程图

3.10 参考文献

1. Bartusiak, Marcia. "Designing Drugs with Computers." *Discover,* August 1981.
2. Couger, Daniel. "The Structured Tableau Design Methodology (STDM)." *Computer Newsletter,* University of Colorado, Colorado Springs, 1983.
3. Councill, Bill, and Carol Councill. *Automating Requirements Traceability* Conducting verification and validation to help you build the right software product in the right way, *www.stickyminds.com,* > Software Testing > Test Automation Zone > Articles & Papers.
4. Elmendorf, William. *Cause-Effect Graphs in Functional Testing.* TR-DD.2487, IBM System Development Division, Poughkeepsie, NY, 1973.
5. ———. "Functional Analysis Using Cause-Effect Graphs." *Proceedings of Share XLIII,* New York, 1974, pp. 577–87.
6. Franz, Don. *Information Systems File Structure and Program Design Workbook*

18. Oberg, Roger, Leslee Probasco, and Maria Ericsson. "Applying Requirements Management with Use Cases." Technical Paper TP505, Rational Software Corporation, 1998.

19. Poole, Bernard, and Noreen Prokop. "Miller's Magical Number: A Heuristic Applied to Software Engineering." *Information Executive*, 1989.

20. Rational Software Corporation, *Rational Unified Process 5.1.1*. Cupertino, CA, 1999.

21. Stevens, Wayne, Glenford Myers, and Larry Constantine. "Structured Design." *IBM Systems Journal* 13, no. 2, 1974.

22. Yourdon, Edward, and Larry Constantine. *STRUCTURED DESIGN: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, Englewood Cliffs, NJ, 1979.

第4章
纵观自动化测试脚本的发展及测试的自动化程度

4.1 开发自动化测试脚本

可以用测试工具的测试脚本语言记录或编写自动化测试脚本。如果没有自动化测试工具，也可以用 Windows 宏录制器函数来记录测试脚本。不管选用哪中方式，只要遵循几个简单的测试用例（设计规则，都将得到更健壮、更易于维护的测试用例。Adhikari 引用了 4 个用在 Charles Schwab 中的测试用例设计原则。设计规则的设计人员要按照下面列举的那样做 [1]。我们采用了这些规则，发现在自动化测试/测试数据开发时很有效。

1）设计相互独立的测试用例

2）设计自包含式的（self-contained）测试用例

3）设计基于出发点的（home-based）测试用例

4）设计无重叠的测试用例

测试用例的独立性能够保证一个测试用例不依赖另一个用例的成功完成来运行（它不依赖于前一个测试用例的结果），它也确保了即使在无人干预的情况下，自动化测试套件也能得出结果。

科学的方法是从测试中得出来的。当科学家进行实验的时候，每次实验重复他们只测试一个条件。这样就确保每次实验结果就是改变后的条件的直接函数。如果两个或更多个变量同时改变了，就很难精确指出观察到的实验结果是由哪一个条件引起的。当然了，在科学中同在实践中一样，有时是不可能创造出理想的实验条件的。

在测试软件过程中，如果两个测试用例不独立，会出现两种情况。第一，随后的实验可能无法执行；第二，分离失败的原因是极其困难的。虽然有时候非常困难，但是设计独立的测试用例还是可能的。

软件测试的目的是识别新的错误，而调试的目的是定位并消除已知错误［5、6、7］。Fat 测试用例（胖测试用例，包含多个测试条件的测试用例）被用来识别新错误，而 Lean 测试用例（瘦测试用例，只包含一个测试条件的测试用例）被用来定位并消除已知错误。因为只需较少的非独立测试用例就可发现错误，所以用它来发现错误更好更经济，但是，独立的额外测试用例常常被用来发现和消除错误。

这已经成为一种折衷方式。如果你的目的是得到经由 Shell 程序以批处理方式执行的自动化测试用例，那么你需要独立的测试用例。但该方式主要的缺点是发现同样多的错误，要比用“较胖的”手动测试用例需要更多的测试用例。另一个缺点是：测试用例的数目越多，测试套件就需要更多的维护。简而言之，测试用例越多，测试代价越大。

自包含式（Self-contained）测试使用的是那些具有在基准数据库中被实现的测试需求的测试用例。Maggio 叙述了在测试脚本中对基本状态的需求。基本状态消除了测试用例之间的线性依赖。他指出，因为一个初始的前提条件包围每个测试脚本内部的验证过程并以后置条件结束，所以初始条件是从其他测试脚本中孤立出来的。并且，是通过还是失败都与前后应用条件程序无关。设置基本状态需要用到基于出发点的测试用例［4］。

在应用程序中基于出发点的测试用例都从同一个点开始。Maggio 认为，当一个应用程序第一次被执行的时候（前提条件）［4］，它必须处于一种特殊的状态，比如初始状态。这意味着应用程序被打开的时候所有的菜单都是可用的，而且没有打开的子窗口或对话框。这样做的必然结果就是当测试用例完成时必须使应用程序恢复到出发点（后置条件）。

构建基于出发点的测试用例非常重要，因为如果每个测试用例都是从一个已知点开始而且结束后会进行清理，那么测试用例执行失败的几率就会很小。在一套自动化测试脚本中，这样做可以确保每个脚本都与先前测试脚本一样开始于同样的条件，有助于保证测试脚本是独立的。但这并不意味着从一个测试用例得出的结果没有可能成为另一个测试用例的前提条件。

无间隙和无重叠意味着测试用例应该包含所有系统功能的各个方面，而且测试用例之间的冗余应该被消除。测试者有时候会有测试一切的倾向。他们甚至会对从来不可能发生的条件进行测试。Mosley 和 Myers 明确指出，进行全面的测试是不可能的［5、6、7］。因此在给定可用的时间和资源的情况下，测试者应该尽可能多地进行测试。

测试经济学提出了用胖测试用例识别新软件错误。这些测试用例不是随机选择的，而是根据一些准则来设计和构成的，这些准则确保了以有限的测试用例得到适当的涵盖范围。前面章节提到的那些技术（因果图、等价划分、边界分析、错误猜测、决策逻辑表和基本测试）如果以相互结合的方式使用，会形成一个既包括了所有系统功能又含有最少的测试用例冗余的测试数据库。

自动化测试应该：

- 测试应用程序做期望要做的事情（通常所说的建设性或积极的测试）
- 测试应用程序不做不期望做的任何事情（通常所说的破坏性或消极的测试）
- 测试应用程序是健壮的（例如，能够处理假的数据而不崩溃）；如果可能，特殊的积极测试结果和消极测试结果应该被确认为特殊测试脚本的预期输出。

Fuchs 是一个宁愿写自动化测试用例而不愿记录它们的人，他提供了一种创建自动化测试用例的三步方法，下面给出了具体的实施方案 [2]。

第一步：设计测试用例。Fuchs 说每一个测试用例都应该包含 1～10 个密切相关的场景。具有 10 个以上场景的测试用例应该分成多个测试用例。每个场景都应该与一个惟一的预期结果相关联。

第二步：手工运行测试。Fuchs 指出在回归测试中自动化测试是发现错误的最好方法，而在第一回合中用手工测试比较好。第一组测试用例在发现新错误方面应该是最有成效的，而回归测试应该发现较多的与软件的改进有关的错误。但是回归测试确实发现了原来没发现的错误。我们曾亲眼看到回归测试识别出了从一开始就存在于软件系统中的错误（举例来说，有一个错误在系统中存在了 20 多年，直到回归测试发现了它）。

第三步：自动化测试用例。在每个测试场景中，测试脚本应该包含以下几个部分：

- 进行设置
- 进行测试
- 验证结果
- 记录结果
- 处理意外情况
- 决定停止还是继续测试用例
- 进行清理工作

测试脚本必须运行的设置工作包括定义测试所用的共同变量、常数以及过程；也包括启动 AUT 和创建所需要的目录和数据文件。

测试工作需要模拟用户是如何使用 AUT 的。这里比较重要的一点是测

试场景必须按照它们被编写的顺序执行，因为这个顺序代表着用户怎么样来进行工作。在识别典型的用作业务目的的场景时应该从功能需求文档开始。比较好的方法是看一个人怎样实际使用这个系统，然而在大多数情况下这是不可能的，因为系统没经过测试。

验证结果涉及到检查每个用户操作中的控制参数的初始状态和终止状态。同时也意味着为那些预期的和非预期的变化检查数据库。验证业务规则以及业务规则相互作用有关的应用程序功能性的惟一方法是确认那些变化是针对相关的数据值出现的。

记录结果就是指为每个测试用例的通过或失败的状态做一个记录。这样，在测试完成后，你可以回顾测试结果，可以把不同测试执行情况的测试记录相比较。

处理意外情况包括跟踪意外事件并对它们进行恢复。可能发生的意外事件包括意外按键、意外窗口、本来应该出现却没有出现的窗口和系统级的中断。

在每个测试场景结束时需要决定是停止还是继续。作出怎样的决定取决于刚刚执行的测试场景是处于通过状态还是失败状态。在一些情况下，即使前面的测试场景失败了，后面的仍然能够执行，但在另外一些情况下，如果前面的测试场景失败了，那么后面就会输出无效的测试结果。

进行清理工作包括关闭 AUT、删除任何不再需要的目录和数据文件以及运行其他所有附属的清理工作。这通常是指将测试用例返回到出发点。

测试脚本应该置入测试套件中，并且测试套件从 Shell 脚本中执行[2]。微软 Visual Test 在线文档显示了如何设计功能区域测试套件、回归测试套件、基准测试套件、压力测试套件以及验收测试套件。我们则添加了三类测试——单元测试套件、集成/冒烟测试套件和系统测试套件。

4.1.1 单元级别的测试

- **单元测试套件** 在开发过程中，软件开发者实现和使用的单元测试套件是构建时由每个开发者在软件组件集成之前运行的可重用的测试套件。这些测试应该由环境来自动化和构建，就像 Java 环境下的 JUnit。
- **集成或冒烟测试套件** 是运行在软件构建完成之后的自动化测试套件。它的目的是双重的：第一，它应该初始化 AUT 执行的环境，比如设置环境变量、初始化数据库表等等。第二，它需要测试构建版本

的基本功能。AUT 通过集成或冒烟测试之前，不能进行其他类型的测试

- **单元级别的回归测试套件** 测试已经被测过的功能来判断在单元测试阶段中加入那些能够导致新错误的纠错、调试和改进部分到构建版本中时，这些功能是否依然正常。
- **功能区域测试套件** 应该是为我们能够在 AUT 中识别的每个功能区域开发的。第一次应该手工执行这个测试套件，以后就使之自动化，并作为回归测试套件重复使用。

4.1.2 系统级别的测试

- **系统测试套件** 执行交叉功能测试，这些测试用来模拟在正常的业务活动过程中用户与系统的交互。另外，系统测试套件应该包括安全性测试、配置测试、易用性测试等等。与功能测试一样，第一次应该手工执行系统测试。自动化功能性测试和系统测试，并把它们用作回归测试套件的一部分。
- **验收测试套件** 测试软件以判断该软件是否符合用户接受系统的最低标准。目的就是判断系统是否经受得住 beta 测试和（或）产品使用的严酷考验。
- **回归测试套件** 测试已经测试过的系统功能来判断在维护阶段中加入那些能够导致新错误的纠错、调试和改进部分到系统中时，这些功能是否依然正常。

4.1.3 特殊的系统级别的测试

- **基准测试套件** 测试系统在变化的条件比如不同的软硬件平台和不同的系统负载下的性能。在这里自动化的基准工具是绝对必需的。市场上有很多产品能够实现自动化的基准测试。在给定的资源需求下，试图手工做这些测试是不合理的；如果没有自动化测试工具，那么只能放弃这种测试。
- **压力测试套件** 在极端的条件下测试系统，比如在峰值时期繁忙的系统载入。这些测试也可能需要自动化测试工具，比如 Mercury Interactive 的 LoadRunner 或者 Rational TestManager。从网上可以下载免费的性能测试工具——比如微软 Web Analysis Stress Tool。然而，这

样的工具开发测试的能力有限，而且用它来测试大型的客户机-服务器应用程序和Web应用程序通常是不够的。和基准测试一样，如果没有工具就不进行负载和压力测试，因为这些测试需要太多的资源。

4.2 记录还是编写测试脚本

记录式测试脚本具有局限性，通常必须在正常工作之前对它们进行编辑修改。Zambelich很明确地指出了记录式测试脚本的缺点：

1）从这种方法得出的测试脚本包含硬编码（hard coded）值，如果程序中有任何改变的地方，这些硬编码值就必须改变。

2）维护这些脚本的费用是巨大的，令人无法接受。

3）这些脚本是不可靠的，即使实用程序没有改变，它们也经常无法重现原过程（弹出窗口、消息，其他一些记录测试时没有出现的事件）。

4）如果测试者产生错误的输人数据，必须对测试进行记录。

5）实用程序变化时，必须对测试进行记录。

6）被测试的是那些已经工作的事情。在记录过程中，会遇到有错误的区域（毕竟这是手工测试）。这些错误要上报，但是在软件被校正之后才能记录脚本。那么你正在测试的是什么呢?

正像在上面第三条中指出的那样，记录的测试脚本是不可靠的。许多测试条件描述了一个测试用例的功能变化。因而，必须记录同一个测试脚本的许多变化。这里的问题是每个测试脚本既包括测试过程指令，又包括硬编码的测试数据值。另外，特殊测试的每个功能变量都需要产生并执行一个单独的测试脚本。这是一个能导致大量脚本需要维护的可怕方法。每次改变或改进AUT的GUI，以及每次修改AUT的功能，记录的测试脚本都会失效。最后，不得不返回头来打开并编辑每个脚本以修改代码和适应这些变化。这意味着在工具的脚本语言中需要修改和（或）写人附加的程序语句。

有一个很好的准则可以遵循，就是：要做任何有意义的测试，你首先要精通测试脚本语言。Fuchs认为许多测试者宁可写测试脚本也不愿记录它们。Fuchs建议创建测试脚本模板文件，这个文件要包含三种类型的公共数据——头信息、安装路径和include文件。这样，测试者就可以用它们作为起始点来写测试脚本、保留相关的模板部分以及删除特殊的测试环境不需要的部分［2］。第7章和第8章中讨论的CSDDT方法应用了Fuch的与测

试脚本的开发和模板使用有关的思想。

当然了，并不是每个人都是程序员；许多测试者只有这方面的经验却没有专门的技术知识。熟悉业务的人能成为比那些有专门技术知识的人更好的测试设计师，因为他们有不同的观点。反过来，那些有编程技能的人能成为更好的脚本编写者。这就是在一个自动化测试框架（第3章讨论过）内部必须有分工的一个原因。测试设计工程师不必是程序员，而测试实现工程师却需要有一定的编程经验。

因此，典型的自动化测试的场景应该包括代表测试实现工程师的活动。他们必须能从头开始编写能够导航 AUT 并能输入测试数据的测试脚本。而且他们必须用工具的记录功能来捕获必须被嵌入测试脚本的 AUT 属性（有些自动化工具套件自动捕获 AUT 属性并将其存储为测试脚本在测试执行过程中可以参考的实用程序映像）。我们可以用这种方式创建基本的测试脚本模板。

但是，用测试工具的记录功能记录的测试脚本不适合用来测试实用程序的功能性。创建记录测试的功能变化正是问题所在。怎么才能解决这个问题呢？如果你打算用你的测试工具来测试业务对象和数据库对象以及与它们相关的规则，那么只记录和编辑测试脚本是不充分的。基于规则的确认可以发生在许多级别上，但是他们通常涉及到 GUI 级和服务器级的确认（业务规则确认）。为了测试它们中的任何一个级别，你首先必须设计能运用实用程序性能来加强这些确认规则的测试数据。然而，要记录所有在测试执行过程中可能用到的特殊数据输入场景，在测试工具的宏记录器运行的过程中，你将不得不长时间的坐在计算机旁，一边输入数据，一边等待。解决的方法就是写测试数据文件，并创建专门的测试脚本以输入文件所包含的数据。这种类型的测试脚本不能被记录，只能由精通测试工具脚本语言的测试者来编写。

为了用自动化测试工具测试实用程序功能，我们主张使用由嵌在测试数据——能将测试的功能性变化存储成测试数据记录——中的控制数据值驱动的书面脚本模板。这种方法通常称为数据驱动测试［9］，是一种及其有效的方法。它是有效地运用自动化的测试工具测试实用程序功能的惟一方法。

以基于框架（framwork-based）的测试著称的模块化测试脚本设计技术逐渐成为数据驱动测试的补充［3］。基于框架的测试实际上是一种结构化程序设计［8］，用于测试脚本设计和构建；然而，它遵循的规则不同于结构化程序设计所遵循的规则，并且远不如后者严格。

Zambelich 提出的功能分解（functional decomposition）测试脚本开发方

法论进一步增强了基于框架的测试概念。他对这种方法的定义如下所述：

> "功能分解"脚本开发方法论是将所有的测试用例减少到它们最基本的任务，并且编写可以相互独立执行这些任务的用户自定义功能、业务功能脚本、"子例程"或者"实用"脚本。

Zambbelich 将脚本任务分成了以下的几个必要区域［10］：

1）GUI 屏幕导航。

2）特殊的业务功能。

3）数据验证。

4）返回导航。

关于测试脚本开发的最基本的概念是数据与功能的分离。Zambelich 说：

> 为了实现这一任务，必须从功能中**分离**数据。这样就允许可以用**数据文件**提供输入和期望输出验证，为业务功能写自动化的测试脚本。这里使用结构化设计或模块化设计来实现层次体系结构。

Zambelich 认为，主要级别是驱动脚本，这是测试的引擎。它包含一系列对一个或多个测试用例脚本的调用，这些测试用例脚本包含测试用例逻辑。然后，逻辑调用进行测试必需的业务功能脚本。当需要的时候，实用脚本和函数被驱动、主和业务功能脚本调用。Zambelich 对这些脚本的定义如下［10］：

- **驱动脚本**　进行初始化（如果需要），然后以期望的顺序调用测试用例脚本。
- **测试用例脚本**　用业务功能脚本实现应用程序测试用例逻辑。
- **业务功能脚本**　实现实用程序内部特定的业务功能。
- **子例程脚本**　执行由两个或更多的业务脚本所需要的特定应用任务。
- **用户自定义函数**　包括一般、特定应用和屏幕访问函数。

要注意的是这些脚本类型中的任一种都可以调用函数。在第 7 章中，会详细介绍 Zambelich 的测试计划驱动（Test Plan Driven）测试方法论。

4.3 小结

我们采用的测试自动化程度方法——CSDDT——结合了模块化测试脚本设计和数据驱动测试，并增加了控制数据值和关键字处理。在第 7 章和第 8 章中会详细讨论这方面内容。

4.4 参考文献

1. Adhikari, Richard. "Planning, Testing, Teamwork: A Recipe for Quality Apps." *Software Magazine* 14, no. 3, March 1994, pp. 41–47.
2. Fuchs, Steve. "Building Smart Testware." Microsoft Technet CD, Test Technical Notes, Vol. 4, Issue 2, February 1996.
3. Kaner, Cem. "Improving the Maintainability of Automated Test Suites." A paper presented at Quality Week '97, 10th International Software Quality Week, San Francisco, CA, May, 1997. Available at *www.kaner.com/lawst1.htm*
4. Maggio, Michael D. "Automated Software-Testing Scripts." *UNIX Review*, no. 13, December 1995, p. 43.
5. Mosley, Daniel J. *The Handbook of MIS Application Software Testing: Methods, Techniques, and Tools for Assuring Quality Through Testing.* Prentice-Hall Yourdon Press, Englewood Cliffs, NJ, 1993.
6. Mosley, Daniel J., and Bruce A. Posey. *Building and Executing Effective Real-World Test Scripts with Rational Suite TestStudio 2001.* Workbook from seminar offered by CSSTTechnologies, Inc., and Archer Group.
7. Myers, Glenford. *The Art of Software Testing.* Wiley-Interscience, New York, 1979.
8. Powers, Mike. "Styles for Making Test Automation Work." January 1997, Testers' Network, *www.veritest.com/testers'network*
9. Strang, Richard. "Data Driven Testing for Client/Server Applications." Fifth International Conference on Software Testing, Analysis and Reliability (STAR '96), pp. 395–400.
10. Zambelich, Keith. "Totally Data-Driven Automated Testing." Whitepaper published at *www.auto-sqa.com/articles.html*, the Automated Testing Specialists (ATS) Web site.

第 5 章
自动化单元测试

5.1 引言

与以往的测试经验相反，单元测试是开发者必须完成的过程。证实软件能够实现被期望做的事情并且没有各种特征的遗漏是开发者的责任。单元测试的目的如下：

- 验证逻辑能够工作
- 验证所有必要的逻辑都存在

Hetzel 发布了他对单元测试的定义，如下：

> 在它们被编码的时候，测试逻辑的独立单元。我们所关心的是测试自然产生的逻辑部件或单元。这可能是函数、子例程或者仅仅是独立程序的合逻辑的清楚部件 [6]。

单元测试也被认为是小范围测试、程序测试、模块化测试以及隔离测试。单元测试是验证代码能够实现系统规格说明认为它能够实现的功能的过程。

5.2 单元测试的合理性

在开发周期中的任何阶段都可以进行测试和修正错误。然而，Boehm [2] 表明，且其他人重申过 [7、8]，发现并修正错误的费用随着开发的进展而极快地增长。如果程序开发者发现自己的错误，则修正错误就变得比较容易。

开发者不得不对他们的代码进行单元测试。他们不能将有错误的代码提供给当今的软件消费者使用。他们开发的单元在放置于系统构建过程之前必须以100%的准确度通过单元测试。单元测试一旦发现问题，开发者就必须给予它们高优先级，并立即解决，就像下面来自JavaWorld的规则一样：

> 一个项目的单元测试套件应该总是100%成功地执行。对有责任心的开发者来说单元测试中的任何错误都立即具有最高优先级。[9]

做单元测试需要的费用和工作量致使许多开发者甚至许多开发组织只做有限的单元测试或者根本不做单元测试。当开发资源甚至比在开发者应该进行单元测试时的资源还要缺乏的时候，单元测试没有发现的问题，或者被开发者忽视的问题这时就会不断的出现。单元测试中发现的问题必须被修复，然后必须在单元级别上进行回归测试。

5.3 单元测试过程

单元测试必须反复地进行。一个有存档的单元测试过程——带有培训的管理级和这个过程实现加强的简易化的文档内容——将会确保测试能够正确并及时地进行。单元测试工具的购买和实现不能保证重复过程——而有效使用工具就可以。自动化的测试工具支持特定的测试活动；因此它们需要特定的环境以便正确操作。此外，总是会有一些额外的、非自动化的、与它们相关的安装和清理设置。因此，一般很难说服开发者使用它们。

5.4 严格的单元测试方法

严格的单元测试方法意味着在编码开始之前单元的设计被存档在规格说明中。它也意味着单元测试是按照单元规格说明设计的，而且更适合在编码开始之前进行。最后，它还意味着单元测试用例的预期结果在单元测试的规格说明中是指定了的。

5.5 单元测试规格说明

单元测试规格说明是单元初始状态的声明；是每个测试用例的起始点；是单元的输入，包括被这个单元读取得任何外部数据值；根据单元的功能性描述了测试用例真正测试的内容；同时还包括设计测试用例时使用的分析方法（例如，在单元中要测试哪个决策）。规格说明还必须包括测试用例的预期输出。

5.6 单元测试的任务

进行单元测试时，开发者应该执行下列步骤［6］：

1）从黑盒的角度开发程序所要求行为的声明。

2）识别基于需求的测试用例。

3）补充基于设计的测试用例。

4）从随机选择或抽取（可选）中增加附加测试用例。

5）将测试用例组织成测试程序。

6）评审测试用例和测试程序。

7）按程序/模块/单元执行测试。

很显然，如果你是一个编写应用程序代码的开发者，那么你会更轻视这些任务。众所周知，开发者不会做充分的单元测试，因为他们认为这会妨碍他们应用程序的开发工作，这就是现实生活中不得不承认的一个事实。同时他们也认为没有足够的时间去做单元测试。

实现单元测试的任务

在开发过程开始时，开发者就应该实现上述列表中的单元测试的任务。在开始写代码之前，开发者应该开发能确认特定行为的单元测试，他们需要完全理解组件应该做的是什么。重点应该放在捕获组件的目的上，而不是放在实现上，同时要放在写单元测试来确认这种行为上。

接下来，开发者应该按照设计规格说明对组件进行编码并执行单元测试。如果存在错误，必要的时候开发者应该重写代码。当测试通过了，编

码就可以停止了。另外，程序设计者应该考虑到组件其他可能的弱点并且编写测试程序来保护它们。

如果缺陷被识别出了，应该对其进行验证．开发者必须编写测试来确认这个问题，同时必须创建缺陷问题报告并输入到缺陷跟踪系统中。如果开发者不能访问自动化缺陷报告和问题解决系统那么这是他们会悲惨地失败的领域。我们发现开发者不会亲自存档缺陷，但是当提供用户界面友好的工具和它的使用方法时，他们就会自愿地进入并跟踪单元测试问题。

最后要注意的一点时，每次开发者改变代码时，所有的测试都必须重新执行以确保原先运行的东西没有任何被破坏。这是单元测试过程中的回归测试。软件回归必须在所有级别测试中进行。回归测试迫切需要自动化——它们的自动化是最成功。

5.7 单元测试的经验法则

下列内容被启发式地应用于单元测试中。首先，设计集中在特定代码块的测试。其次，用调试工具，如那些在流行的集成开发环境（IDE）中可用的调试工具来监控测试。第三，利用相关技术，如单步步进法、变量审查法以及内存泄漏探测法。最后，操作代码和数据以使给定代码段的所有功能完全被测试。

5.8 单元测试数据

开发单元测试数据的来源包括功能需求规格说明、高低级别设计规格说明等。一些不同的测试用例类型以及它们的来源如下所示：

测试用例类别	测试用例来源
基于需求的	规格说明
基于设计的	逻辑系统
基于代码的	数据结构和代码
随机的	随机生成器
抽取的	现有文件
极值的	极限和边界条件

5.9 单元测试框架

和所有级别的软件测试一样，单元测试要求有一个在适当的位置的支撑基础设施。这个单元测试基础设施必须提供必要的信息来设计单元测试和单元测试数据。它还必须提供一个所有单元测试的相关产物（测试、测试数据和测试结果）可以永久存储的知识库。这个基础设施必须用作单元测试执行的测试装置，同时它必须捕获并且存储测试结果。ANSI/IEEE STD 1008-1987 定义了单元测试并记录了正式单元测试方法。

进行人工的单元测试是很费时间并且负担很重。这就是为什么开发者回避它的原因。当然，如果进行了所有的努力，我们就可以开发出使我们更快更好地工作的快捷方法。这甚至对通过使用微软 Excel 和微软 Word 记录测试数据等来增强的单元测试也适用。我们也已经看到用来记录和存储单元测试的 Lotus Notes 数据库。问题是需要手工工作来更新这些数据库，尤其是做单元级的回归测试时。因此，这增加了测试数据的维护。

一个更有效的方法是使用专门为单元测试设计的工具。有几种是可以通过商业渠道获得的产品，另外几种可以从网上免费得到。还有支持通用的编程环境如 Visual B、C 和 Java 的工具。这些可用工具中的多种工具提供源代码，并提供测试覆盖度量。

商业工具在功能性上是多式多样的。有一些工具非常先进。例如，Rational Software 的 Quality Architect 可以分析 Java 代码，并反向操作生成用来构造测试用例的 Rational Rose 图表。这种产品也可以执行针对应用程序的测试并捕获结果。另一方面，有一些工具就它们提供的功能来说是不复杂的，但是仍然很有用。在网上可以免费获得的 JUnit 是一个非常好的单元测试工具，但是它将设计、构建、执行以及分析测试的担子压在了开发者的肩上。一旦用 JUnit 安装了初始的测试框架，它就会支持单元级别的回归测试。这两种产品之间的不同之处不是在于它们做了什么，而是实现它们所需要付出的努力的多少。原则是你得到了你为之付出的。

用 Java 进行面向对象的开发

单元测试应该从开发者的角度去写，并且要集中在被测试类的特定方

法上。在面向对象的环境中开发单元测试，Canna 提出了下列的建议［3］：

- 为要测试的类编写代码之前先编写单元测试。
- 在单元测试中获取代码注释。
- 测试所有实现“有意义的”功能的公有方法（也就是说，不是获取和设置方法，除非他们以独特的方式做了设置和获取）。
- 将每个测试用例封装在正在测试的类的同一个包中来获得对包和保护成员的访问。
- 避免在单元测试中使用特殊域对象。

5.10 小结

如果开发者想要适应当今软件市场快速开发的最终期限，自动化的单元测试是必要的。越来越多的开发组织正在仿效微软公司并正在执行一天一个版本（build-a-day）的方法。在过去，版本日期是被设定的，然后新版本以一定的间隔比如几天、几周或几个月出现。那些与面向对象开发和语言如 Java 相关的新开发过程需要更频繁的软件版本。Java 开发者可以在网站 www.javaworld.com/javaworld/tools/jw-tools-testing.html 上得到全面的单元测试工具列表。

另外，JUnit 是由 Erich Gamma 和 Kent Beck 编写的单元级的回归测试框架。它的用途是让开发者来实现 Java 中的单元测试。JUnit 是发布在 IBM 公共许可证之下的开放源码的软件（Open Source Software），并放在 SourceForge 主页上。它现在有自己的网址 www.junit.org。我们把压缩的 JUnit 源码下载到支持本书的 www.phptr.com/mosley 中。

除此之外，下载 Windows XP 下使用的 Java 编程测试工具的链接在 www.xprogramming.com/software.htm 上。两个有名的工具是 JUnitPerf 和 JDepend。据它们的作者描述（这两个工具都是 Clarkware Consulting 公司的 Mike Clark 创建的），JUnitPerf 是所有用来测量包含在现存的 JUnit 测试内部的性能和功能可扩展性的 JUnit 测试修饰性工具的集合［5］。下载链接和测试用例的概要和示例在 www.clarkware.com/software/JUnitPerf.html 上。

Clark 这样总结 JDepend：“JDepend 贯穿了一系列 Java 类和源文件目录，并为每个 Java 包生成了设计质量度量。JDepend 允许自动化测量在可扩充性、可重用性以及可维护性方面的设计质量来有效地管理和控制包的依赖性。”下载 JDepend 和 JDepend 详细信息及使用的链接是 www.clarkware.com/software/JDepend.html。

5.11 参考文献

1. American National Standards Institute/Institute of Electrical and Electronics Engineers. *Software Unit Testing*. ANSI/IEEE Std 1008-1987.
2. Boehm, Barry. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.
3. Canna, Jeff. "Testing, fun? Really? Using unit and functional tests in the development process." *IBM developerWorks : Java technology*. Java technology articles available at *www-106.ibm.com/developerworks/library/j-test.html*.
4. Clark, Mike. JUnitDepend, *www.clarkware.com/software/JDepend.html*.
5. ———. JUnitPerf, *www.clarkware.com/software/JUnitPerf.html*.
6. Hetzel, William. *The Compete Guide to Software Testing*. 2d ed. QED Information Sciences, Wellesley, MA, 1988.
7. Mosley, Daniel J., and Bruce A. Posey."Building and Executing Effective Real-World Test Scripts with Rational Suite TestStudio 2001," Workbook from public seminar of the same name offered by CSST Technologies, Inc., and the Archer Group.
8. Myers, Glenford. *The Art of Software Testing*. Wiley-Interscience, New York, 1979.
9. Nygard, Michael T., and Karsjens Tracie. "Test Infect Your Enterprise JavaBeans™: Learn how to test your J2EE components live and in the wild." *JavaWorld* Web Page, 2001. *www.javaworld.com/javaworld/jw-05-2000/jw-0526-testinfect_p.html*

第6章 自动化集成测试

6.1 引言

你的单位谁负责集成测试？你的答案将决定实施什么类型的测试以及如何自动化测试它们。有一个这样的独立小组吗？其中既不包括开发人员也不包括系统测试人员，它的目的是在应用程序构建过程后立即测试应用程序。在一些部门中，开发人员创建了自己的构建版本，并进行集成单元测试。在其他小组中，负责完成构建方面的专家——系统构建师（build master）在应用程序交付系统测试组之前进行一些初步测试。集成测试是否真是系统构建师应尽的职责？

第二个问题：什么是集成测试？这个问题可以通过分析第一个问题的答案面找到答案。如果这个答案不反映结构化的构建和测试过程的话，那么集成测试就是一个任何人都可以做且做的效果如何都无所谓的方法。但是它需要多大程度的控制？

为了回答该问题，我们必须认识到这是一个开发过程中最易出问题的领域。我们需要由好的自动化工具支持的有效过程。无论我们在何处进行咨询，我们都可以发现两类问题：第一、在系统测试机器上通常不安装构建版本。第二、单元和集成测试没有在前面完成，迫使系统测试组进行这些原本早该已执行过的测试。

同单元测试一样，集成测试必须自动执行，否则它将不能完成。一套最小的集成测试自动化工具应包括一个缺陷跟踪系统、一个问题报告系统、一个捕获/回放工具以及一个配置管理工具。这套工具在某种意义上说必须可将配置管理（Configuration Management，CM）知识库中的构建版本、文件与测试知识库的自动化测试和缺陷库的缺陷紧密联系起来。

为什么需要这样紧密集成呢？因为许多应用程序要求问题解决方案不仅跨越构建版本而且跨越主次产品版本。Rational Software 公司的 ClearCase、ClearQuest、Robot 这些工具为集成测试提供了这种类型的控制。

6.2 什么是集成测试

集成测试是几组单元的测试。单元可以是模块、子系统或系统。传统上这些实体通常定义为过程边界或系统对系统/子系统对子系统的接口。事实上，集成测试在软件发展早期有许多不同的说法：字符串测试、接口测试等等。重要的是不混淆应用于集成测试中的接口测试的用法和用户界面(或图形用户界面）测试。集成测试用在这儿指的是多个过程、模块、子系统、系统之间的通信接口。尤其指面向对象的应用程序中的各对象彼此间进行消息传递。例如对于 Java 语言，单元在逻辑和操作上指的是子类或类级，也可被定义为类中所包含的方法。

6.3 日常构建版本冒烟测试

冒烟测试的主要目的是在系统测试环境下，软件系统构建版本安装后验证系统的基本功能/特征是否达到预期的效果。冒烟测试的第二个目的是为了帮助实现测试的主要目的，而建立和配置环境变量和数据。自动化构建版本冒烟测试可以完成上述两个行为。

构建版本冒烟测试对测试组来说格外重要，这是因为我们不能测试不能正常工作的应用程序的特征，而且对不能正常执行的功能也不能进行回归测试。构建版本冒烟测试应该作为最后安装测试的预演。Wallace、Ippolito、Cuthill 定义安装测试如下：

> 软件安装测试活动是指在启动完全顾客验收测试之前的最后步骤。安装测试的目的就是为了证明提交的是正确软件并且与安装现场任何接口相关的软件接口都是正确的 [1]。

从上述定义，我们可以给出冒烟测试的正式定义。

> 冒烟测试活动是指软件构建版本进入系统测试阶段之前的最后测试步骤。相对于发布后的软件构建版本而言，冒烟测试活动是软

件进入回归测试之前的最后测试步骤。

每个新的构建版本在交付测试组之前必须进行冒烟测试，它可以应用于新的开发、主次系统版本、具体客户的定制。基本的构建版本冒烟测试工作流程示例如图 6-1 所示；这个例子包括了下一节将要列出的测试目标，以及其他相关测试活动。

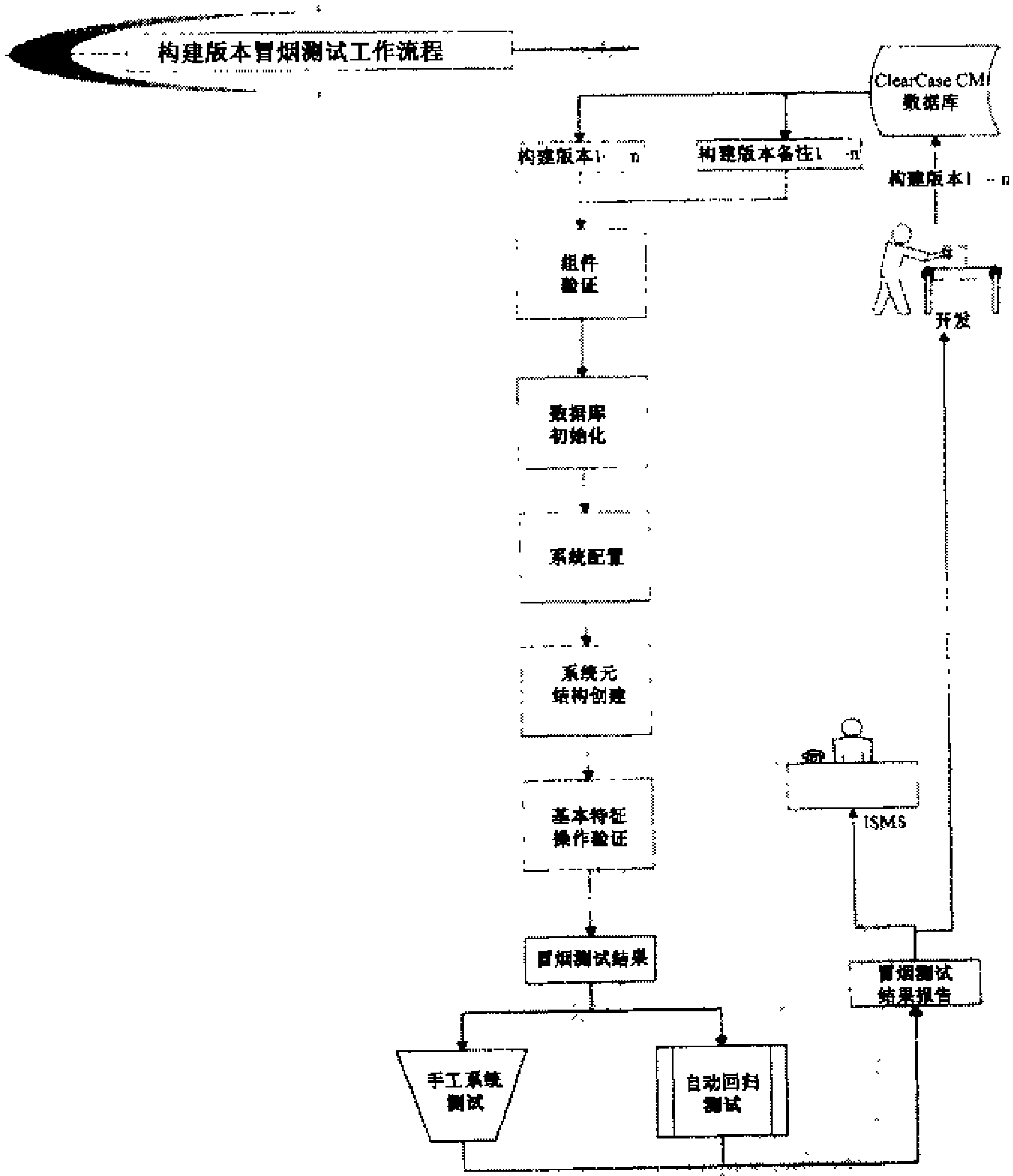


图 6-1 构建冒烟测试工作流程

6.4 构建冒烟测试的目标

- 验证所有已安装组件
- 验证没有安装多余内容
- 初始化数据库
- 配置系统变量和常量（例如：配置表设置）
- 创建和配置系统测试所必需的基本系统级元结构（metastructure）
- 验证每个特征/GUI界面的基本功能

基本特征的验证似乎与开发者所完成的单元测试相重复。尽管它是重复以前的工作，但测试原因不同。首先，单元测试是与其他系统组件相互独立执行的；其次，基本特征的验证将在开发环境中引导开发者的操作。是的，集成测试是在单元测试之后系统测试之前，将各个组件作为新创建的系统来加以测试。回归测试通常使用开发者设计的测试数据完成并在开发环境下加以执行的（在开发服务器上），而不是在测试环境下进行测试。开发服务器并不反映目标产品环境。而测试服务器及客户端被配置用来模拟产品平台环境的。将单元测试数据作为冒烟测试的测试数据集一部分是完全可接受的。

6.5 自动化构建版本冒烟测试清单

如下列表描述了构建备注应该包括的信息种类，这个列表可用于验证此类信息是否存在并且正确：

- 构建版本是否包括构建备注？
- 构建备注是否包括这个构建版本新的一系列特征？
- 构建备注是否包括安装前、安装过程中以及安装后必须顾及的各种特定考虑？
- 构建备注是否识别了在该构建版本中已经被修补的所有缺陷？
  - ➢ 初始化安装后测试机上是否是正确的组件？
  - ➢ 所有组件是否是应该安装的内容？
  - ➢ 测试数据库是否已初始化？
  - ➢ 所有测试前条件是否已识别和确定？
  - ➢ 所有特定应用程序的变量和常量是否都已初始化？

- ➢ 必要应用程序元实体是否已创建?
- ➢ 所有 GUI 界面的自动化测试是否已创建?
- ➢ 所有基本特征的自动化测试是否已创建?
- ➢ 特定特征的测试数据是否已构建?
- ➢ 在冒烟测试中单元测试数据是否可用?

- 它是否包含在冒烟测试数据集中?
  - ➢ 安装文档是否可用?
- 它是否是当前构建/发布版本中最新的?

6.6 小结

在当前的 Web 开发中常用 Java 语言，集成测试是有效软件构建过程中的重要一环。没有首先进行集成测试就无法为发布版本进行系统测试。开放源码工具如 JUnit（前一章已讨论过的）应该是在开发端安装构建后集成测试的一个集成组件。在软件开发过程更为正式的情况下像 Rational Quality Architect 这样的商业工具也可用在开发环境中。像 Rational TestStudio 这样的工具可以用于最后冒烟测试（这也是最后的集成测试)。要进行自动回归测试的小组应该指导冒烟测试。

6.7 参考文献

1. Wallace, Dolores R., Laura M. Ippolito, and Barbara Cuthill. "Reference Information for the Software Verification and Validation Process," NIST Special Publication 500-234, March 29, 1996. Available at *hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/234/val-proc.html#APPEN*

第 7 章 自动化系统/回归测试框架

7.1 数据驱动方法

Kit 定义了自动化软件测试的三个发展阶段 [1]：

- 第一个阶段是基本的 GUI 测试。使用捕获/回放工具开发自动化测试脚本的用法局限于记录在 GUI 级别上的用户操作、编辑得到的测试脚本以及重放编辑过的测试脚本。得到的测试脚本是非结构化的、无存档的，并且是不可维护的。
- 在第二个发展阶段中，脚本编写者发展了"建立结构良好、有存档的、健壮的、可维护测试"的能力。在这个级别上，测试项目成了工程项目；测试脚本包括错误捕获和恢复逻辑，其关键特征是测试脚本组件的可重用性。
- Kit 认为测试自动化的第三个阶段的特征是控制了测试资源。在这个级别上，测试设计和测试自动化被看成是互相分开的行为。

第三阶段捕获/回放用途的关键是在于测试脚本的自然属性和数量，尤其是测试脚本与测试用例之间的关系。第一和第二个阶段的捕获/回放的用户通常对于每个测试用例都有一个主要脚本。这就产生了许多脚本，所有这些脚本都需要维护。第三阶段的用户发现了一个以不同的、新奇的方式使用脚本的方法——用单个脚本处理每个测试用例。

数据驱动框架是什么？以及它在什么地方有助于自动化各个阶段的捕获/回放工具使用？关于数据驱动测试的内容是什么这个问题众说纷纭。实际上，几年前在 SQA 用户组中（www.dundee.net/sqa）进行过热烈而详尽的讨论。讨论的全文被 SAS 研究所的 Nagle 记录下来进行了综合并发布在专门进行数据驱动测试和 DDE 方法研究的网站（groups.yahoo.com/group/

RobotDDEUsers）上，详细内容可以参看附录B。这种方法是用SQA Basic语言实现的，SQA Basic语言是Rational Software公司的Robot产品的母语。对那些使用WinRunner的Mercury Interative用户来说，Automated Testing Specialists（自动化测试专家）（www.auto-sqa.com）的Keith Zambelich为支持数据驱动测试框架（类似Nagle的DDE）的WinRunner开发了一种工具包。下面将讨论这些方法，包括CSDDT框架。

那么，什么是数据驱动测试呢？它是目前通用的术语，但是大多数人认为数据驱动测试就是简单的把外部数据作为源数据输入到AUT；例如，使用数据池的情况。下面是我们给出的定义：

> 数据驱动测试就是一种数据被包含在输入测试数据文件中，并且数据控制自动化测试脚本执行的流程和动作的测试。

输入测试数据记录是从外部文件或电子数据表中读入的并且是独立于测试脚本程序开发的。这种思想被Archer Group和其他人更进一步地改善了。这里介绍的方法论也采用了使用控制数据（control data）来判断要采取的行动以及它们发生的顺序。

数据驱动测试使用存档的测试数据来驱动自动化测试过程，这些数据通常以简单的CSV文本文件形式存在。数据驱动测试可以被扩展以包括控制数据和测试数据。测试数据同时测试GUI和代表应用程序功能的服务器级数据确认规则。而控制数据通过引导测试脚本到达应用程序中合适的位置执行测试和通过指示要执行的测试或动作的类型来驱动测试脚本。

数据驱动测试脚本的特征包括下列各项：

- 使用简单的输入文本文件
- 有很高的可维护性
- 容易被非编程人员使用
- 存档正在执行的测试
- 借助占位符（placeholder）允许动态数据输入
- 用输入数据控制测试的执行

数据驱动测试的内容包括［18、26］：

1）能输入到程序中的参数。

2）用来使程序运行的操作或命令序列。

3）驱动程序的测试数据序列。

4）触发测试脚本在运行的时候产生动态数据值的占位符。

5）程序读和处理的文档。

应该使用在第3章讨论黑盒（基于需求的）和白盒（基于代码的）方法的内容中描述过的技术来设计测试数据。

数据驱动测试脚本的优点是：

1）当增加额外的测试数据时不必修改测试脚本因为测试数据会被追加到现有的文本文件中。

2）很容易修改数据记录。

3）多个测试数据记录可以开发成为函数变量。

4）需要时可以创建和使用多个输入数据文件。

我们可以确信的一点就是：数据驱动测试符合了 Kit 给出的关于第二代捕获/回放测试工具使用的定义。它符合了他给出的所有标准。

7.2 框架驱动（结构化）测试脚本

框架驱动测试将 AUT 从测试脚本中孤立出来。它提供了一套共享函数库中的函数。这些函数被看成测试工具编程语言中的基本命令。基于框架的测试脚本可以独立于 UI 进行编程。

框架驱动测试可以发生在多个级别上［8］：

- 菜单/命令级——执行简单的命令
- 对象级——执行特定事件的动作
- 任务级——关心特定的通常是重复的任务

7.2.1 开发框架驱动测试脚本

下列提示有助于开发基于框架的测试脚本：

1）为处于测试的所有应用程序特征编写测试函数。

2）为自定义控件编写函数。

3）为特定语言命令编写包装器函数。

4）为使用频繁的任务编写函数。

5）为多个测试脚本使用的大型复杂任务编写函数。

7.2.2 Archer Group 框架

在过去五年多时间里，Archer Group 花了无数时间来改进这种基于多个测试项目和客户需要的方法论。像任何软件过程一样，它逐渐包容了许

多测试场景并被应用于实际情况中。

它的目标永远是保持简单、有效，并且，最重要的是保持可维护性。在这一章和下一章中，我们将讨论一个为使用数据驱动测试而设计的测试框架。这个框架脚本可以从支持本书的 www.phptr/mosley 上找到。虽然最初是为使用 Rational 的 Robot 产品而开发的，但是它也适合于其他测试工具。

Archer Group 工作的三个主要重点是：

1）开发简单的、易于使用的、具有高度组织性的脚本框架。

2）用外部数据控制测试流程并向应用程序的 GUI 提供输入数据，而不是在测试脚本文件中硬编码数据。

3）通过使用关键字数据替换开发一种方法来使用动态输入数据。

我们采用控制同步数据驱动测试（Control Synchronized Data-Driven Testing）这个术语来代表这种类型的自动化测试。为什么要控制同步呢？除了被用来输入到 AUT 的数据以外，还把指定地点（where）、方法（how）、以及预期结果（what-to-expect）的控制信息添加到每个数据记录的前面。CSDDT 规定了

- 在 AUT 中应该何去何从；窗口、标签页、对话框等的导航；测试活动的目标。
- 每一步要干什么，要执行的动作。
- 动作执行后期望得到的结果——是否希望错误出现或正常状态。
- 用于输入或标准选择的数据。

被主测试脚本读取的输入测试数据记录应该首先包含提供记录级别的 ID/控制，或者在从输入数据文件中读取之后应该怎样处理这些记录的信息。具有跳过记录的能力是十分有好处的，因为在初始阶段它们可能不工作，但是我们不想将它们从输入文件中移除。在调试新增加的代码的过程中，很有必要在读入记录后进行单步调试脚本（读入记录后进行断点调试）。这就允许数据的动态替换，比如用关键字数据替换（例如，用当前日期以 01/25/2002 的形式替换文本 DATE）。以这种信息作为基础，输入数据记录类型可以分为以下几类：

- 好的输入记录；这可能导致预期的错误，也可能不。
- 跳过这条记录；记录是坏的或不完成的或者是用来注释下面记录的。
- 在这条记录上设置断点（在 Rational Robot 中，这可以创建用于调试目的的软件断点）。
- 执行特殊的处理，如关键字替换。
- 在到达文件尾（EOF）以前终止测试。

总之，CSDDT 指导了 AUT 的测试脚本导航，指示了要激活或者选择的窗口/标签页、子窗口、对话框，等。这些数据也规定了要执行的动作，例如：

Select/Display（选择/显示）

Insert/Add（插入/增加）

Update/Modify（更新/修改）

Delete（删除）

Save（保存）

Ok（确定）

Cancel（取消）

Apply（应用）

此外，这些数据也描述了预期结果、是否期望得到错误条件以及错误的类型。CSDDT 数据记录格式允许包括注释，这可以使数据记录可以自存档（self-documenting）。

最后但并非最不重要的是，数据记录包括了将要输入到 AUT 中的测试数据值。CSDDT 允许可变数量的输入数据字段。例如，一个记录可以有 0 个数据字段（只有一个执行命令，如 CANCEL），而另一个可以有 50 个或更多的数据字段。长度应该是动态的。

可视化在一个文本文件中主测试脚本读取和执行的每个数据记录是直接的。前面七个部分或字段（由双引号和逗号隔开，与 CSV 文件相似）包含了控制信息，其余的字段是用来输入到 AUT 中的数据。

根据测试类型，我们将一般的测试分成几组。每一种类型对应着一个单独的自动化测试类型和一个自动化测试脚本驱动的特殊类。其中有一些测试可以严格地使用记录/回放类型的测试脚本，而业务规则测试使用了 CSDDT 的方法。

- **GUI 测试**的目的是测试窗口/GUI 组件的功能。
- **属性测试**测试所有窗口、子窗口、标签页和对象的对象属性。
- **特定特征测试**（需要的话）测试自定义控件、不存在于其他类别中的特征，还可能包括特定的手工测试
- **数据库内容/初始化测试/程序**。这可能不算是测试，而是用来确认测试数据库内容有效或用特殊的测试记录来初始化数据库
- **业务规则测试**通过数据输入和数据控制选择测试客户和服务器业务规则/编辑。
- **性能和/或负载测试**（需要的话）

7.3 业务规则测试

这种测试类型是数据驱动测试方法的核心。它模拟了用户测试。它不能与GUI或对象属性测试相混合，因为这样做使测试脚本的执行速度变慢。这种测试检验了驱动应用程序特征的业务处理逻辑的核心。它包括了测试内部应用程序逻辑、应用程序组件、存储过程等。在某些情况中，AUT的测试可能需要大量的输入数据来测试所有需要的组合。CSDDT方法很适合这种测试。

7.4 GUI测试

这种测试检验了标准的GUI对象和控件。它察看了系统菜单控件、最小化、最大化、恢复以及关闭；MDI/子窗口动作、大小、移动、最小化以及在子窗口中转换；AUT菜单、工具栏、帮助系统、子窗口共同的外观和用户界面。这种测试是在[2]中图5-5A和5-5B说明的测试的基础上的改进和自动化的版本。

7.5 属性测试

这种测试类型察看AUT的命令按钮、数据显示对象、输入数据对象等。检验的公共属性包括：

- 对象的大小和排列
- 哪些对象可用和不可用
- 哪项被添加、移除或改变了（自从基线捕获以来）

检验这些属性有助于预先确定在数据测试过程中可能出现的问题。使用来自数据测试的相同代码有助于更快地识别窗口/标签页和对象。

属性测试警告：

如果使用Rational Suite TestStudio，那么这种测试可以用任何或所有可用属性来完成，这些属性是对象属性验证点能够探测到的。这种属性测试

对数据值是很敏感的，必须调整它以忽略数值，如改变日期和其他数据，否则测试者就必须确保存在相同的数据值。

7.6 输入数据测试

这种测试类型检验了 AUT 接受有效数据输入的能力，以及拒绝无效数据输入的能力（或者产生错误或者在对象的级别上掩蔽无效值）。这种测试可以结合业务规则测试类型（测试客户级和服务器级的有效确认），也可以独立开发。当单独构造时，它应该测试 GUI 级别的有效确认。它也可以被用作在负载/压力测试和可靠性测试过程中键入大量输入的方法。

7.7 格式化测试数据文件

脚本被设计成从外部输入文件读取测试数据记录然后执行以数据记录内容为基础的动作。从连续的文本文件或数据池中读取数据是最快的数据输入方法，因而是首选；然而，也可以使用微软 Excel 电子数据表（最慢且可能依赖于代码）或微软 Access 数据库。测试被执行后，要验证预期结果。(例如，使用 Rational TestStudio，这将通过 Robot 的记录验证点、窗口的存在测试等来完成。）如果出现了预期结果，那么就没有产生错误状态。从 AUT 产生的预期错误不是测试错误。如果没有出现预期的结果，那么测试错误状态就会出现并被记入日志。在测试脚本中，控制返回到测试循环的起点并读取下一个记录。测试将继续进行直到没有数据记录要处理。

测试循环是由下面的循环结构控制的：

```
DO WHILE NOT EOF            (EOF= End Of File)
     -code inside of the loop
     -read and process records
LOOP
```

只要有记录读取，测试就会继续进行，除非出现关键错误。

7.8 应用级错误

当预期结果没有出现时，例如，当发现错误的数据、没有发现正确数

据、未发现窗口或对象、改变了主要属性时，非预期的应用级错误就会出现。当发生非预期的结果如检测到非预期激活窗口时，特别是如果是错误显示窗口时，也会出现错误。如果你是一个 Rational 用户，则任何类型的测试验证点失败代表了这种错误类型。这些错误类型都被认为是测试故障(test failure)。

当发生下列情况时，关键性的测试脚本错误就会出现：

- 输入文件打不开或不能读
- 访问数据库失败
- 脚本命令失败或出现运行错误，一般是代码问题

关键性的错误一般是环境/编程错误，并不能构成测试故障。

7.9 创建外部数据输入文件

外部指的是输入文件不是测试脚本的一部分，而是包含测试数据的独立文本文件。要创建输入数据文件，首先要建立普通的 Excel 电子数据表并设置数据记录的格式。格式如下：

- 电子数据表的每一行最终会成为输入文件中的一个数据记录。
- 每一列将会成为给定记录的数据字段。

转换程序——如由 CWC 公司的 Andy Tinkham 开发并经过 Archer Group 修改过的与 Rational Robot 一起使用的那些程序——可以被用来读取 Excel 电子数据表数据以及将数据写入连续的 CSV 文件的行中。

普通的 CSV 文本文件和 SQABasic 顺序文件之间的区别在于普通 CSV 文件中的数据值看起来像

```
value1,value2,value3
```

在你能够使用 CSV 文件数据值之间，它们需要转换成你使用的测试工具所要求的格式。例如，当使用 Rational Robot 时，数据值必须采用 SQABasic 顺序文件的格式，就像：

```
"Value1","value2", "value3"
```

注意：在这个例子中惟一的区别是 Robot 要求数据值外有双引号；否则它仍然像 CSV 文件。这种格式允许在字段或值串中含有逗号。这种文件可由简单的文本编辑器建立；而在电子数据表中的数据更易于组织和读取。此外，不是必须要有转换程序，因为这个任务在微软 Excel 中就可以完成。转换程序可以用来简化过程。

在 CSDDT 方法中，前七列包含了记录的控制信息和注释字段。这是数据记录的固定部分。测试脚本代码首先读取这一部分。它的值被输入到脚本的内部字符串变量中，然后它们被检验以决定下一步操作。下面的控制字段要求代表了一个可能约定。如果需要其他可以另行设计。

```
"Rec_Type","WinCd","TabCd","ActCd","ErrCd","FldCnt","comment","data1","data2", etc.
```

表 7-1 给出了电子数据表的形式。当为输入文本文件将电子数据表的形式转换成数据记录时，这五个记录看起应该是：

```
"G","User_Preference","Font","OK","none","1","Select a font style","Arial"
"G","Options","none","Apply","none","2","Enter numeric values into the first two edit
       boxes","123","456"
"H","Customer_Info","Billing_Address","Save","Required_fld_1","2","Test address field
       inputs","null","Apt.2b"
"K","Customer_Info","Billing_Address","Save","Required_fld_1","3","Test address field
       inputs","123 main st","null","Illinois"
"X","Customer_Info","Acct_Info","Cancel","none","0","Select the cancel button"
```

- 字段 1——Rec _ Type 代表记录类型。我们实现的标准值如下所示：
  - G 表示标准的好记录。
  - K 表示好记录。预处理关键字替换。
  - H 表示只能读取不能处理的头记录或注释记录（或者简单地说，就是跳过该记录）。
  - B 表示断点记录。这会通知代码暂停以便测试者可以单步调试断点处的代码。它通常用于调试目的。
  - X 表示终止条件——在到达 EOF 之前停止处理记录。

可以为实现特殊的目的而增加其他代码。可以为特殊的处理编码异常用例，比如 K、H、B 以及 X 代码。其他的只是被简单地考虑标准记录，像 G 值。

- 字段 2——WinCd 表示窗口选择代码

这个值或字段用来指定在 AUT 种将要选择哪个窗口。这是你必须设计的与应用程序相关的代码。我们使用菜单命令和按钮的快捷键来选择窗口。例如：

F 代表 File，而 O 代表 Open。那么符号 FO 就表示 File→Open 菜单动作。

TA 表示 Tools→AutoCorrect，等。

表 7-1 格式化的测试记录示例

列名	记录类型	窗口选择码	标签页选择码	动作码	错误码	数据字段数	注释字段	数据字段 1	数据字段 2	根据需要，更多的数据字段
快捷键	Rec_Type	WinCd	TabCd	ActCd	ErrCd	FldCnt	N/a	N/a	N/a	N/a
功能	主程序用来协助程序流程	用来识别要进行测试的 AUT 目标窗口	用来识别要进行测试的目标标签页控件	识别要对目标窗口/标签页执行的动作	(只用在动作执行后期待某个错误出现时)识别相关的错误处理程序	为该条记录识别跟在注释字段后的数据字段的数目	存档记录目的的地方	输入到 AUT 中的数据值	输入到 AUT 中的数据值	输入到 AUT 中的数据值
示例记录 1	G	User_Preference	字体	OK	None	1	选择一种字体风格	Arial		
示例记录 2	G	选项	None	Apply	None	2	在头两个编辑框中输入数值型数值	123	456	
示例记录 3	H	Customer_Info	Billing_Address	Save	Required_fld_1	2	测试地址字段输入	null	Apt.2b	
示例记录 4	K	Customer_Info	Billing_Address	Save	Required_fld_1	3	测试地址字段输入	123 Main st	null	Illinois
示例记录 5	X	Customer_Info	Acct_Info	Cancel	none	0	选择 cancel 按钮			

代码中每个附加的字母或数字都可能表示菜单或次级子窗口的展开级。例如：

符号 KHL 可以表示：选择快捷键是 K 的菜单项，然后选择快捷键是 H 的子菜单项，再选择快捷键是 L 的子菜单项。

可以选择任意字母和/或数字作为符号，只要保证与输入数据中的符号以及脚本中的符号相同即可。例如，它可以是 User _ Preferences 或者 Options——只要你喜欢。

- 字段 3——TabCd 表示标签页选择码

这个码用来在指定的窗口以与窗口选择同样的方式选择指定的标签页和/或子标签页。如果一个窗口中没有标签页或子标签页，那么就使数据值为空白或者为 null，但是要保证包括了作为占位符的双引号（“G”、“WinCd”、“ErrCd” ……）或者使用关键字如 None 来指示它被特意置空以便代码能够发现它。

- 字段 4——ActCd 表示动作码

我们使用这个码来表示要被执行的动作。这些动作包括：

➢ Insert（插入）

➢ Delete（删除）

➢ Update（更新）

➢ Display（显示）

➢ Select（选择）

➢ OK（确定）

➢ Cancel（取消）

➢ AUT 需要的任何动作

- 字段 5——ErrCd 表示错误码（预期结果）

这个值被用作错误检测码。如果这个值为空或者为 ErrCd 或 None，那么没有预期的特殊错误窗口/消息或者指示器。如果特殊的代码用在这，如 Err1，那么这个代码应该寻找与这个错误有关的窗口、消息框或者小的帮助/状态条或者其他的指示器。如果这个代码等于 Err1，那么我们就要寻找与这个错误匹配的特定条件。

测试错误状态以验证不允许或无效的数据值是很有必要的。这与验证数据值有效同样重要。

- 字段 6——FldCnt 表示字段计数

这个字段指定有多少个数据字段在下一个将要读到的注释字段之后。这允许可变数量的输入数据字段，这样输入文本文件中的所有行或记录不必都有相同的长度。这个计数可以是从 0～n 的任意数。代码应该将其定义

为数值型的值。

● 字段7是注释字段

这个字段作为一个标准输入字段被插入。这是为自文档化数据文件设计的。这个字段不执行任何处理——它只用来解释和存档以及读取和删除。它必须被程序读取，以便当下一项被读取时内部文件指针能够停止指向第一个输入数据字段。在 Rational TestStudio 中这个注释能够被写入测试日志中用于记录和报告。

● 字段8～n是数据输入字段

这些数据字段被用作 AUT 的直接输入来代替用户可能键入的值或做出的列表选择。如果是在 Rational 环境下，通常的方法是通过 SQABasic 命令替换记录的值。

```
'InputKeys "x"
```

这里的“x”是你本来要键入的带有变量名和/或者字符串变量数组下标的记录值，这里的数据值是从输入文件中读取的。

从外部数据源中读取的数据一定要读到某个地方去。读取时，值被读到变量中。然后这些变量可以用于整个程序并被查看内容。接着它们的值就被用来决定怎样控制程序流程。

```
If a variable equals value x then
   do this
otherwise
   do something else
```

7.10 数据文件小结

一旦数据在 Excel 电子数据表中被创建，必须将其转换成 SQABasic 顺序文本文件或者适合测试工具环境的文件。这样做是为了提高速度并简化测试脚本代码。如果这些文件不是太大，则可以用标准文本编辑器如 Notepad 进行进一步编辑。应该建立几个不同的文件，可以为每个要测试的主要特征集建立一个文件。这样，每次你运行测试的时候就可以指定使用一个不同的输入文件。记住，只要输入数据中使用的控制代码与脚本中使用的作用于这些代码的值完全匹配，那么就能创建与测试脚本无关的输入测试数据文件。

7.11 业务规则测试的代码构造

这部分阐明了怎样建立只执行 CSDDT 的脚本。它不涉及为建立 GUI 或进行属性测试编写的脚本。想了解更多的信息可以参考第 8 章列出的脚本示例。我们采用的脚本组件包括：

- 外壳（Shell）脚本
- 主（Main）脚本/程序
- 窗口选择（Window selection）脚本/程序
- 标签页选择（Tab selection）脚本/程序
- 动作选择（Action selection）脚本/程序
- 错误检测（Error detection）脚本/程序
- 其他支持函数和包含文件

7.11.1 Shell 脚本

可选择的 Shell 脚本通常用来为主脚本设置输入条件，比如调用在测试或其他安装类型的函数之前初始化数据库的脚本。Shell 脚本也是向测试者提示要处理的输入文件的文件名和路径的地方。这样允许在运行时测试脚本可以一次一个地处理多个测试场景就像测试者指定的那样。获得要处理的输入文件的方法有多种，这取决于你使用的测试工具。在下一部分将要讲述的主脚本激发窗口文件浏览器来获得输入文本文件。

7.11.2 主脚本

主脚本或程序保存在文件名为 DDMain 的脚本文件中，它的作用是：

1) 打开输入数据文件并读取数据。
2) 控制主过程循环。
3) 检查控制字段。
4) 调用适当的程序和函数来执行不同的过程、程序，如：

a) 选择窗口 [窗口选择程序保存在文件名为 DDWindow _Select 的

可以完成数据输入然后再执行一条命令，如单击 OK 按钮、Save 按钮，甚或 Cancel 按钮。设计这个脚本部分以便它给所有的控件输入数据（对那些不需要交互的控件输入空值），然后执行最终的动作如单击 OK 按钮。

既然动作已经完成了，脚本就要判断它应该从应用程序中寻找正常响应还是错误响应。它必须事先决定好是否有预期错误状态（因为测试者有意输入错误的数据以产生错误）。要想这样做，在仍然执行 Perform _ Action 程序时就要预先检查 ErrCd（错误码）的值。

如果错误码为空或者等于"none"，那么就假定它期望从应用程序中得到正常响应，继续进行处理。寻找发生正常动作的指示依赖于 AUT。脚本可能会收到一个 Save 操作正常完成的指示（例如，状态栏信息），或者正常完成可以由窗口关闭以及增加一条新记录到表格中这个事实来指示。它也可以是没有任何事件发生的良性事件。如果没有错误信息显示，那么动作就正常完成了。不管发生了什么事情，它都必须编写脚本记录那个指示器。如果没有发生什么事情，那么就不必编写脚本。当你不预期任何事情发生却有测试错误发生时，你就得到错误信息。在 Rational Robot 中，非预期的错误窗口被检测到，那么脚本就要被设置来引起测试失败（你捕获了一个 bug!）。如果被执行的动作正常完成，则 Perform _ Action 程序返回到 Main 程序。

现在，若错误码不为空或者不等于"none"，那么脚本就会预期错误出现。这个时候 Perform _ Action 程序返回到 Main 程序。

返回到 Main 程序后，Main 程序再次检查 ErrCd（错误码）。如果错误码不为空或者不等于"none"，就调用 Process _ Error 程序；否则，Main 程序完成处理这条记录并返回到循环的起点，读取下一条记录并进行处理。

在 Process _ Error 程序中，代码用来选择检查和验证预定错误指示的存在的代码部分。这可以是显示错误文本的消息框或者是 AUT 用来告诉用户有错误产生了的其他指示。一旦错误处理完成，就返回到 Main 程序。然后 Main 程序返回到循环的起点，读取下一条记录并进行处理。

除了 Main 程序之外的四个程序都以完全相同的方式设计。设计的目标是使它们简单！

下面给出了 Main 程序代码的模板文件。

注意：脚本中的撇号（'）表示注释字段，而'$ 表示编译器指示。

```
'$Include: "Global.sbh"         'storage for global variables
'$Include: "DataDriven.sbh"     'storage for global variables specific to the Data-Driven
                                 procedures

   Declare Function ReadFields BasicLib "Global" (File_Num, Fld_Cnt, GFlds())
   Declare Function Key_Word_Sub  BasicLib "DDKeyWord_Sub" (Fld_Cnt)

   Dim Rec_Type                          'Record Type
```

```
    Dim Fld_Cnt                        'Data Field count
    Dim Comment As String
    Dim Child_Open As String
    Dim record_counter

Option Compare text                   'allows LIKE to be case insensitive
Sub Main                              'program starts here
    Dim Result As Integer
    On Error GoTo Error_Handler1
    Child_Open="No"
    current_win=""
    current_tab=""
    record_counter=0

'other ways to initialize the InfileName variable
  'InFileName="c:\sqatemplates\testdata\testdata1.txt"
  'InFileName=InputBox$("Enter the name of the input file to process.","Data Driven
                         testing",InFileName)

    CallScript "OpenFile_txt" 'open a text file for test input data this script opens a file
                         browser window
    InFileName=txt_filename 'from the openfile_txt routine

    If InFileName="" then
        Exit Sub     'the test is canceled no filename was supplied
    End if
    Open InFileName For Input As #1
    Do while Not EOF(1)   'Main process loop. Read and Process input records until there are
                         none left.
''''''''''''''''''''''''''''''
'read input test data file
''''''''''''''''''''''''''''''
        Input #1,Rec_Type,WinCd,TabCd,ActCd,ErrCd,Fld_Cnt,Comment
        if Fld_Cnt<> 0 then

            ReDim DFld(Fld_Cnt) 'set the appropriate size of the array for the number of
                                'data fields in this record

            Result=ReadFields(1, Fld_Cnt, DFld())   'pass file number, fld_cnt, address of
                         DFld
            if Result=0 then
                MsgBox "ReadFields function failed. Program terminating"
                Close #1  'close the open file
                Exit Sub
            end if
        end if

''''''''''''''''''''''''''''''''''''''''''
'log which record we are processing
''''''''''''''''''''''''''''''''''''''''''

        record_counter=record_counter+1
        SQALogMessage sqaPass, "last record read ="&Cstr(record_counter), ""

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Determine the record type
'
' H=header record, skip this do not process
' B=Breakpoint stop for program debugging
```

```
' X=Terminate prior to the end of the input data file
' G=Good record. Process normally
' K=Good record. Pre-Process for Keyword substitution,
'   then process normally
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
        if Rec_Type LIKE "H" then
            goto NoProcess
        elseif Rec_Type LIKE "B" then
            stop              'data breakpoint
        elseif Rec_Type LIKE "X" then 'terminate the program
            Close #1
            Exit Sub
        elseif Rec_Type LIKE "G" then
            Goto Start                  'good record
        elseif Rec_Type LIKE "K" then
            key_word_sub(Fld_Cnt)      'this call is used when keywords are used in the data
                                such as 'AutoDate"
        else
            MsgBox "The record type is invalid. The program is terminating"
            Close #1
            Exit Sub
        end if

Start:              'this is a label
''''''''''''''''''''''''''''''''''''''''''''''''''
'Determine if we need to open a new window or not
''''''''''''''''''''''''''''''''''''''''''''''''''
        if Child_Open="No" then
            'Result = Window_Select(WinCd) 'go open the selected window. This is executed at
                                first pass only.
            CallScript "DDWindow_Select"
            if Gen_Return_Code=0 then
            'if Result=0 then
                close #1
                exit sub
            end if

            Current_Win=WinCd   'update and save the current window selection
            Child_Open="Yes"    'we have now opened a window
        else
            if Current_Win<>WinCd then
                CallScript "DDWindow_Select"
                if Gen_Return_Code=0 then
                    close #1
                    exit sub
                end if
                Current_Win=WinCd   'update and save the current window selection
                Child_Open="Yes"    'we have now opened a window
            end if
        end if

'tab select
''''''''''''''''''''''''''''''''''''''''''''''''''
'Determine if we need to open a new tab or not
''''''''''''''''''''''''''''''''''''''''''''''''''
            if TabCd <> "" and TabCd <> "none" and TabCd <> "TabCd" and Current_Tab<>TabCd
                                then
                CallScript "DDTab_Select"
                if Gen_Return_Code=0 then
```

```
                close #1
                exit sub
            end if
            Current_Tab=TabCd
        end if

''''''''''''''''''''''''''''''''''''''''
'Perform the specified Action
''''''''''''''''''''''''''''''''''''''''
        CallScript "DDPerform_Action"
        if Gen_Return_Code=0 then
            MsgBox "Specified Action Could Not be Performed"
            close #1
            exit sub
        end if

''''''''''''''''''''''''''''''''''''''''
'Process the Error Code
''''''''''''''''''''''''''''''''''''''''
        if ErrCd="" or ErrCd="none" or ErrCd="ErrCd" then 'Error is not expected
            goto NoProcess
        else
            CallScript "DDProcess_Error"
            if Gen_Return_Code=0 then
                MsgBox "Specified Error Processing Could Not be Performed"
                close #1
                exit sub
            end if
        end if

NoProcess:        'this is a label
    Loop 'end of main loop

    'End of File has been reached, close any open child window
    Exit Sub

Error_Handler1:
    MsgBox "Error number "&Cstr(Err)& " occurred at line: "&Erl &" -- "&Error$
    Exit Sub
End Sub
-----------------------------------------------------------------------------------
```

窗口选择和标签页选择程序

这是带有占位符信息例子的 Window _ Select 模板脚本的实际代码。

```
'$Include: "DataDriven.sbh"
Declare Function Window_Select(WinCd)
Sub Main
    Gen_Return_Code = Window_Select(WinCd)
End Sub
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Window_Select(WinCd)
Window_Select=1
Select Case WinCd
    Case "w1"
```

```
            msgbox "w1 selected"
        Case "w2"
            msgbox "w2 selected"
        Case "w3"
            msgbox "w3 selected"
        Case Else
            Msgbox "Window Select not found for WinCd: "&Cstr(WinCd)&" Terminating"
            Window_Select=0  ''return value FAIL
            SQALogMessage sqaFail, "Window Select not found for: "&Cstr(WinCd)&"", ""
        End Select
    End Function
```

从文件前面摘取的这行：

```
'$Include: "DataDriven.sbh"
```

告诉编译器要包括或使用在下面的头文件 datadriven.sbh 中定义的变量。

下面采用这段最初用 Rational Robot 的 SQABasic 语言写的代码，并为了讨论的目的对其进行了一些修改。例如，让我们假定它被编码成从 3 个窗口中选择其中的一个——用户优先窗口（User Preference Window），选项窗口（Options Window）以及客户信息窗口（Customer Information Window）。我们决定使用下列窗口选择代码：User _ Preference、Options、Customer _ Info。

这些窗口选择代码要被用于输入数据文件和 Select Case 语句中，就像这里指示的那样。

```
'$Include: "DataDriven.sbh"
Declare Function Window_Select(WinCd)
Sub Main
    Gen_Return_Code = Window_Select(WinCd)
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Function Window_Select(WinCd)
Window_Select=1    'this is the return code for this function indicating success. It is
                           preset here
                    'below in the case else statement it is changed to indicate an error
                           because the WinCd
                    'could not be found.

Select Case WinCd

    Case "User_Preferences"
        Record here, performing the actions necessary to select the User_Pref window

       It is very important to insert code here that verifies that the window opens and is
                         in the expected state.
       The test fails here if the window does not open as expected.

   Case "Options"
       Record here, performing the actions necessary to select the Options window
       It is very important to insert code here that verifies that the window opens and is
                         in the expected state.
```

```
        The test fails here if the window does not open as expected.

     Case "Customer_Info"
         Record here, performing the actions necessary to select the Customer Information
                         window
         It is very important to insert code here that verifies that the window opens and is
                         in the expected state.
         The test fails here if the window does not open as expected.


   Case ... add additional cases as needed.

   Case Else
       Msgbox "Window Select not found for WinCd: "&Cstr(WinCd)&" Terminating"
       Window_Select=0  ''return value FAIL
       SQALogMessage sqaFail, "Window Select not found for: "&Cstr(WinCd)&"", ""
   End Select
End Function
```

这里验证选择正确发生是很重要的。当使用 Rational Robot 时，总是要在选择代码的结尾插入一个窗口存在验证点。对其他的测试工具环境来说，用只有你选择的工具才有的功能来开发开发相似的窗口验证测试。

如果窗口存在且处于正确的状态（可用、最大化，等等）并且准备好接受动作，那么测试的这部分就会通过，否则会发生测试用例失败，这会终止测试

7.12 使代码清晰健壮

代码必须足够健壮，意思是指当产生错误时，它应该能解决非预期值，并且总是允许正常终止。注意我们在上一节所举的例子中的 Select Case 语句。这三行简单的代码提示测试者有未知的窗口代码被读取，这可能是因为没有增加新的 Case 语句来容纳它，也可能是因为有不好的编码进入测试数据。

当使用被调用的程序和函数时，总是要构造它以便调用者可以使用 return 代码。这提示调用者函数/程序成功或者失败，并允许采取适当措施。

由于窗口已经被选择，跟着像在 TabCd（标签页码）指定的那样标签页或子窗口被选择。使用与用于 Window _ Select 代码相同类型的 Select Case 结构。

```
'$Include: "DataDriven.sbh"
Declare Function Tab_Select(TabCd)

Sub Main
    Dim Result As Integer
    Gen_Return_Code = Tab_Select(TabCd)
```

```
End Sub
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Tab_Select(TabCd)
tab_Select=1                              'initially set the return code to "Success"

    Select Case tabCd

        Case "t1"
            msgbox "t1 selected"

        Case "t2"
            msgbox "t2 selected"

        Case "t3"
            msgbox "t3 selected"


      Case Else
          Msgbox "Tab Select not found for TabCd: "&Cstr(TabCd)&" Terminating"
          Tab_Select=0                        'return value FAIL
          SQALogMessage sqaFail, "Tab Select not found fortab code: "&Cstr(tabCd)&"", ""
      End Select
End Function
```

注意：这段程序的构造方式与窗口选择程序完全相似。惟一的区别是在这个程序里我们处理的是 Tab _ Select 代码而不是 Windows _ Select 代码。仍然以刚才的例子作说明，我们假设在客户信息窗口中有两个标签页，一个是 Billing Information（账单信息），另一个是 Account Information（账户信息）。这样我们就两个 Tab _ Select 代码。我们不妨使用 Acct _ Info 和 Billing _ Info。我们重新修改后的标签页选择程序如下所示。

```
'$Include: "DataDriven.sbh"
Declare Function Tab_Select(TabCd)

Sub Main
    Dim Result As Integer
    Gen_Return_Code = Tab_Select(TabCd)
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Tab_Select(TabCd)

tab_Select=1                    'initially set the return code to "Success"

    Select Case tabCd

        Case "Acct_Info"
            Record selection of the account information tab here
            Validate the tab was selected and is now in focus

        Case "Billing_Info"
            Record selection of the billing information tab here
            Validate the tab was selected and is now in focus

        Case ……-
            Add additional cases for as needed
```

```
        Case Else
            Msgbox "Tab Select not found for TabCd: "&Cstr(TabCd)&" Terminating"
            Tab_Select=0  ''return value FAIL
            SQALogMessage sqaFail, "Tab Select not found fortab code: "&Cstr(tabCd)&"", ""
        End Select
End Function
```

标签页或子窗口一旦被选择，就该执行在 ActCd（动作码）中指定的动

```
        Case "a3"
            msgbox "a3 selected"

        Case Else
            Msgbox "Action Selection not found for ActCd: "&Cstr(ActCd)&" Terminating"
            Perform_Action=0  ''return value FAIL
            SQALogMessage sqaFail, "Action Selection not found for: "&Cstr(ActCd)&"", ""
    End Select
End Function
```

```
Window SetContext, "Caption=Window Name", ""
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys DFld(1)
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys DFld(2)
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys DFld(3)
PushButton Click, "Text=OK"
```

注意斜体文本。这里是 DFld（x）而不是 x。

这时脚本输入的数据值是从数据文件中得到的，与我们原来输入的数据相对应。在这里我们的目的是用输入文件中的值或数据来替换在记录过程中手工输入的数据和/或选择。

数据输入之后，我们就可以执行后面的动作了，如 Add（添加）、Save（保存）、Delete（删除）、Update（更新）数据库等等。这通常可以通过选择控制按钮或菜单选择如 File->Save 来完成。这种类型的通用动作要记录到子程序中，这个子程序可以通过代码调用。这样，如果做了修改则只需修复一个地方。

警告：

当菜单选择能够执行同样的动作时避免使用工具栏按钮。

动作完成后，我们就可以检查 ErrCd（错误码）值以判断是否需要寻找错误状态——错误信息或其他可以确认的错误状态。可以使用与创建窗口/标签页选择程序相同的步骤来创建 Select Case 结构。然后，将代码写进单独的程序或库文件以使 Main 程序短小并易于管理。

典型的错误处理部分的代码看起来如下所示：

```
Select Case ErrCd
    Case "Err1"
        Verify the existence of the appropriate error msg, etc. before proceeding!
    Case "Err2"
        Verify the existence of the appropriate error msg, etc. before proceeding!
    Case Else
        Msgbox "The Error code in ErrCD code does not exist. Terminating"
        SQALogMessage sqaFail, "Error Code not found for "&Cstr(ErrCd)&"", ""
End Select
```

如果发现预期的错误状态，这条记录的处理就算成功完成了。Main 程序继续处理下一条记录，直到到达文件尾或没有可读取的数据为止。如果没有发现预期的错误状态，测试就失败了（通常是测试用例）。测试被终止，并把结果输入到测试日志中；或者 Main 程序将结果记录到日志文件中，并继续执行，只要测试数据记录代表独立的测试。

测试日志应该指示最后记录到的读入包含的内容和记录号。这可允许快速调试和问题的孤立。这通过 Main 程序中很少的代码就可以完成。

Archer Group 小结

CSDDT 包含一组程序、库文件或允许使用输入文件中的数据来执行选择的程序；CSDDT 也允许数据输入，与我们键入作为输入或单击作为选择的数据相对应。如果我们改变输入文件中的数据，那么不需要改变脚本就能修改或改变选择和动作，以及输入到 AUT 中的数据。

这就是数据驱动测试框架的真正益处。一旦这项选择机制被适当应用，测试代码就被创建，对输入数据进行操作，并且我们可以使用不同的数据而不是不同的测试脚本来修改每个测试场景以创建函数变化。这是因为测试数据文件包含了一些代码，这些代码指定了在 AUT 中该去哪里执行、执行什么动作、预期是什么、要使用多少数据字段、注释（文档）和使用什么数据（输入到 AUT 中）。

7.13 Carl Nagle 的 DDE 框架

Nagle 的方法将测试自动化提高到了第三个级别上，因为它使用一个非技术性前端（在微软 Excel 格式中实现的一种表驱动方法），这个前端可以描述要执行的测试、安排测试场景以及实现执行步骤。当 DDE 分析表格时，DDE 执行自动化测试，而用户不必编写测试脚本代码。

7.13.1 DDE 综述

SAS 研究所的 Carl Nagle 为 SAS（SAS 研究所拥有对 DDE 的知识产权）开发了这个框架，但是他也将这个方法公开发布给了公众，不过带有下面的版权说明信息：

Copyright (2001) SAS Institute Inc. All right reserved.

通告：在满足下列条件的情况下，任何人可以以任何目的无偿地使用、复制、修改以及发布这个代码和它的文档：

注意：版权授予通告必须出现在所有的代码副本以及所有相关的文档中。

DDE 概念是捕获/回放方法的另一个替代。自动化测试实现失败的主要

原因是与捕获/回放方法相关联的维护负担。自动化测试脚本开发其实就是应用软件的开发，一组自动化测试脚本形成了一个测试另一软件系统的软件系统。随着被脚本测试的应用软件的不断发展，脚本自身也需要维护，这样它们才能继续工作。

维护的级别取决于实现方法。捕获/回放测试脚本要求维护投入最多，因为它们包含测试程序命令和测试数据；这些都被硬编码到测试脚本中。降低维护的关键是从脚本中移除测试数据，并使用数据来驱动测试脚本以及开发在自动化测试项目间可以共享的可重用的子例程和函数。

Archer Group 方法（前面部分讨论过）用包含两种数据类型的 CSV 文件来做这个：第一种数据类型的作用是指导测试脚本在 AUT 中的导航以及指导它对 AUT 执行的动作。第二种数据类型是实际上的测试数据，脚本用它执行指定的 AUT 测试。脚本控制数据和测试数据是在 Excel 电子数据表中建立的，并以 Rational Robot 要求的格式导入到 CSV 文件中。

DDE 方法与这个非常相似，但是被增强了，这样非技术的测试员也能够开发测试数据。测试数据开发包括驱动应用程序测试的微软 Excel 表格集合的层次构建。这些表包括三种类型的驱动表格——循环驱动（Cycle Driver）表格、套件驱动（Suite Driver）表格和步进（Step Driver）驱动表格。循环驱动表格的级别最高，从实际的测试中移出测试数据的程度也最大。图 7-1 说明了 DDE 自动化框架结构。

用 Nagel 自己的话来说对 DDE 方法最好的描述是：

> ……框架本身其实被核心数据驱动引擎（Core Data Driven Engine）、组件函数（Component Function）和支持库（Support Library）定义。而支持库提供了通用例程，这些通用例程甚至在关键字驱动框架的背景之外也有用，核心引擎和组件函数很大程度上依赖于三个要素的存在。

测试的执行先从启动测试（LAUNCH TEST）脚本开始。这个脚本通过向循环驱动（Cycle Driver）提供一个或多个高级（High-Level）测试表来激发核心数据驱动引擎。循环驱动处理这些为每个遇到的中级（Intermediate-Level）测试表激发套件驱动（Suite Driver）的测试表。套件驱动处理这些为每个遇到的低级（Low-Level）测试表激发步进驱动的中级表。而步进驱动（Step Driver）处理这些低级表时，它试图保持应用程序与测试同步。当步进驱动遇到一个指定组件的低级命令时，它会判断包含的组件的类型并激发相关的组件函数模块来处理任务［3］。

这个方法的主要优点在于：任何有 AUT 工作知识并对要测试的东西有所了解的人都可以输入循环驱动表中的信息。这意味着测试设计者可以在执行系统测试之前开发循环和套件驱动表，并提供给自动化组，然后自动

化组开发步进驱动表。通过使用 Nagle 的 VB（Visual Basic）测试产生器，测试设计者建立测试表所需要付出的努力被降到最低限度，因为 VB 测试产生器允许设计者通过 GUI 输入表数值而不是直接键入表值输入到表中。

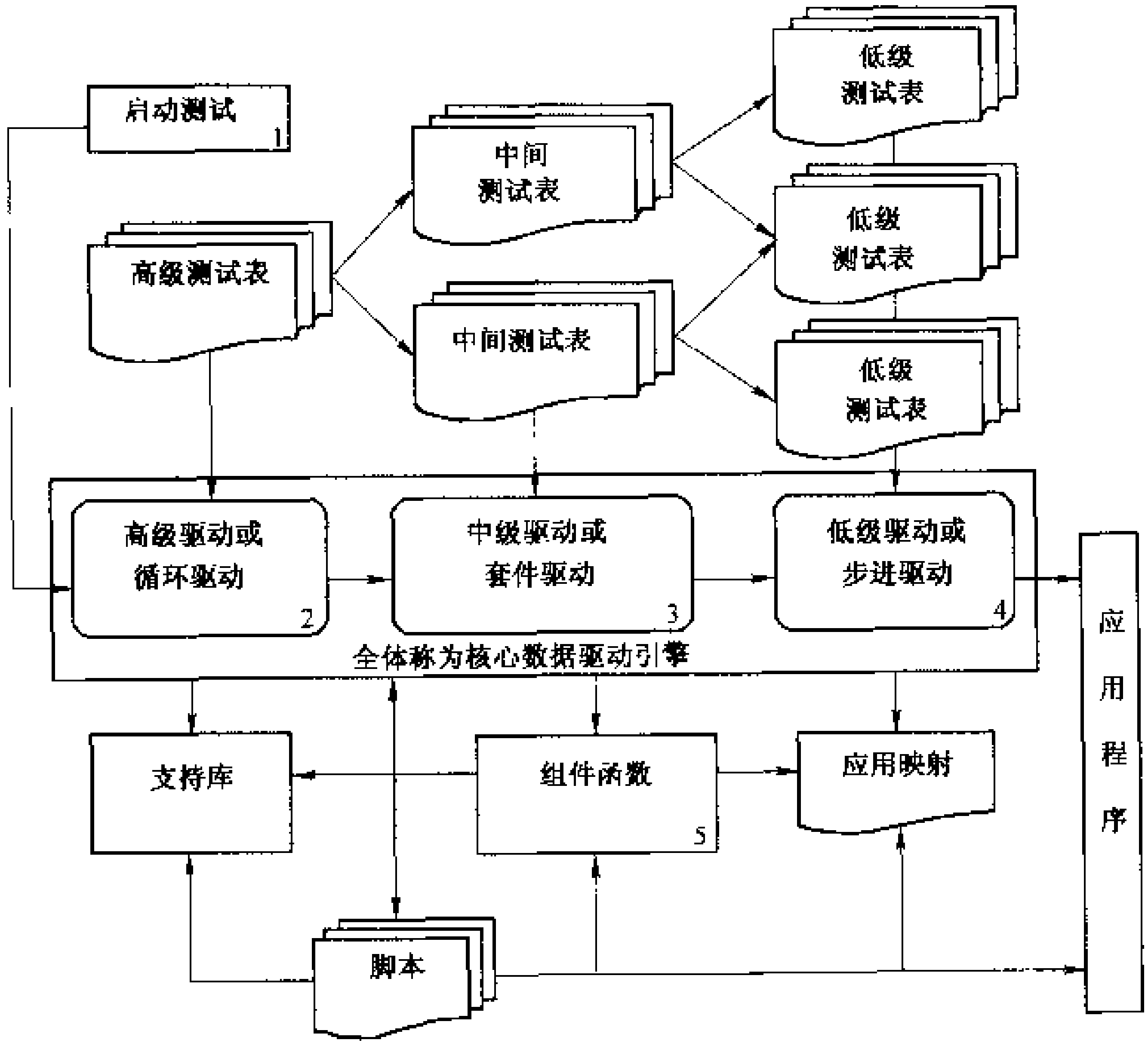


图 7-1 DDE 自动化框架（得到复制许可）

摘自于：Nagel, Carl. "Test Automation Frameworks." 图 5。Groups.yahoo.com/group/RobotDDEUsers/files/Doc/FRAMESDataDrivenTestAutomationFrameworks.htm.

注意：

通常测试设计者可以开发套件级别（Suite-level）测试列表。这样非技术性的设计者也能开发循环级别（Cycle-level）和套件级别的测试列表。然而，最低级别的步进测试表似乎就是技术性自动仪的领域了。

DDE 已经实现了。它只需要安装，然后就完全可以使用了——不需要本地用户定制就可以开始使用。开始使用时，用户只需要开发自己的测试数据。只在一种情况下，即现有的 DDE 功能不能解决给定的问题时，用户需要执行定制脚本，定制脚本不是 DDE 的一部分，但是需要的时候可以被 DDE 调用。

在 Archer Group 方法论中，控制数据包含在测试脚本中。然而，在

DDE 中，控制数据和其他数据放在包含 GUI 对象识别字符串的应用映射中。动作关键字激发组件函数，这些组件函数是对 AUT 执行动作的子程序。测试记录类型指示器的用法与在 Archer Group 中的相似。应用映射在功能上与商业自动化工具如 Rational Software 的 TestFactory 和 Mercury Interactive 的 WinRunner 创建的映射相似。Nagle 的框架包括一个应用映射工具、伴随 DDE 代码的 Process Container 工具。

下面是 DDE 记录类型的例子：

b=BLOCKID（定义输入文件内部的名字块起始）。

c=DRIVER COMMAND（执行不同于测试的实用功能的驱动）。

s=SKIPPED TEST（暂时禁用［同样被日志记录］的测试）。

t=TEST（执行的测试；可以是导航、修改数据值或验证数据）。

驱动根据记录类型将数据记录转到合适的测试脚本程序中。

测试数据可以被输入表中或被包含在表格引导测试脚本程序读取并处理数据的 CSV 文件中。套件驱动测试用例例子通过步进驱动指引 DDE 处理特殊账户的应用程序登录：

```
T,  Login, ^id=myuserid, ^pw=mypassword
```

然后步进驱动处理 Login（登录）测试表格数据以完成下面的操作：

```
t, LoginWin, UserIDBox, SetTextValue, ^id
t, LoginWin, PasswordBox, SetTextValue, ^pw
```

可以看出，开发这些测试不需要专门的技术性知识。它们是以关键字为基础的。每个记录指引测试脚本到指定的位置并根据它所含的关键字告诉测试脚本进行什么操作。

7.13.2 DDE 发展成果

DDE 安装参考文档放在 gropus.yahoo.com/group/RobotDDEUsers/files/Doc/sqabasic2000/DDEngineSetup.htm 上。在安装 DDE 和执行试验性项目之前必须要阅读这个文档。试验性项目需要持续的努力，大约需要 90 天左右。这个项目应在一个选定的 AUT 特征子集上执行。在试验阶段结束时，要对成果进行评估，并且识别出的问题域要标明位置。

接下来，剩余的 AUT 特征测试应该自动化，包括一个完整的回归测试集。开发这个测试集的实际时间应该根据特征的数目以及每个 AUT 版本的每个特征的测试用例的数目进行估计。

7.14 Keith Zambelich 提出的面向 Mercury Interactive 产品用户的测试计划驱动测试框架

Zambelich 这样描述用于 WinRunner 的工具箱（可以从 www.autosqa.com/toolkit.html 获得）：

> 本工具箱基本上由用户自定义函数（User-Defined Functions）和能使测试者更好地控制 WinRunner 测试工具的实用脚本（Utility scripts）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地使用这个工具箱，而不用受到测试工具固有的限制。[3]

这个工具箱包含一些函数，这些函数是 Zambelich 的"测试计划驱动(关键字驱动)（Test Plan Driven（Key Word Driven)）"自动化测试方法的基本基础设施。他的方法使用了测试者在含有特殊"关键字"的电子数据表中开发的测试用例文档。引用 Zambelich 的话来说就，"在这种方法里，整个过程就是数据驱动，包括功能。关键字控制这个处理过程。"

Zambelich 的工具箱现在包含下面几类实用脚本 [4]：

- "Generic（一般的）"视窗工具：用来测试任何面向对象的应用程序。
- "Language-Specific（特定语言）"工具：用来测试在特定语言环境中开发的面向对象应用程序。
- "Terminal Emulator（终端仿真机）"工具：用来测试经由终端仿真的 AS\400、ACP/TPF 以及其他大型应用程序。
- "Application-Specific（特定应用程序）"或"customized（定制的）"工具：也许需要为测试中的特定应用程序开发。

WinRunner ToolKit 允许测试者和脚本开发者完成下列任务：

- 快速产生自动化测试而不必具有广泛的 WinRunner 或 TSL（Test Script Language，测试脚本语言）知识。
- 开发"健壮的"脚本，不能由于应用程序错误、WinRunner 错误或测试脚本错误（允许"无人测试（unattended test)"）就停止运行。
- 为多个而不是一个应用程序（测试可以为任何数目的不同类型的应用程序开发而不必重新构造脚本开发过程）开发自动化的测试脚本。
- 容易开发用 ToolKit 做框架的"用户自定义"函数和工具（如果熟悉 TSL 的用法）。
- 为每个测试创建用户自定义的"测试报告"，以及为每个测试场景或

系列测试创建“总结报告”。

- 设想完全控制测试工具以及使它基本上能做测试应用程序所要求的任[illegible]

表 7-2 含有关键字的数据表示例（使用面向 WinRunner 方法的 Zambelich 的工具箱）

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）:	Screen（界面）	主菜单	验证起始点	
Enter（输入）:	Selection（选择）	3	选择支付选项	
Action（动作）:	Press_Key（按键）	F4	访问支付界面	
Verify（验证）:	Screen（界面）	Payment Posting（支付记入）	验证访问的界面	
Enter（输入）:	Payment Amount（支付数量）	125.87	输入支付数据	
	Payment Method（支付方式）	Check（支票）		
Action（动作）:	Press_Key（按键）	F9	处理支付	
Verify（验证）:	Screen（界面）	Payment Screen（支付界面）	验证剩余界面	
Verify-Data（验证数据）:	Payment Amount（支付数量）	$ 125.87	验证更新数据	
	Current balance（当前余额）	$ 1309.77		
	Status Message（状态信息）	Payment Posted（支付过账）		
Action（动作）:	Press_Key（按键）	F12	返回主菜单	
Verify（验证）:	Screen（界面）	主菜单	验证返回菜单	

7.16 “测试计划驱动”方法体系结构

- 驱动脚本
  - * 初始化（如果要求）。
  - * 调用具体的应用程序“控制器（Controller）”脚本，向它传递测试用例的文件名（从电子数据表中保存为“tab 分隔”文件）。

- “控制器”脚本
  * 读取并处理接收自驱动的文件名。
  * 与包含在输入文件中的“关键字”匹配。
  * 从后面的记录中创建参数列表（parameter-list）。
  * 调用与“关键字”相关联的“实用”脚本，传递创建的参数列表。
- 实用脚本
  * 处理来自于“控制器”脚本的输入参数列表。
  * 执行指定的任务（例如，按键或按钮、输入数据、验证数据等），如果需要则调用“用户定义的函数”。
  * 向测试用例的测试报告（Test Report）上报所有的错误。
  * 返回到“控制器”脚本。
- 用户定义函数
  * 一般的和具体应用程序的函数可以被上述任一种脚本类型调用以执行指定的任务 [4]。

使用 TestDirector（测试引向器）来驱动测试集

如果使用 TestDirector 来驱动测试集，那么体系结构就要稍微改变，因为 TestDirector 没有能力向被调用的脚本传递参数。因而，TestDirector 不能用来调用控制器脚本。我们通过创建一个可以作为 Test Director 和控制器之间的中介物的测试用例驱动脚本（Test Case driver script）来处理这个事件。测试用例驱动可以被 TestDirector 调用，然后它调用控制器脚本并向它传递所需要的 tab 分隔文件的名字。注意：这意味着每个测试用例必须有一个测试用例驱动。实际包含的代码都是完全相同的，但是每个必须带有测试用例的名字。

TestDirector
- 如果需要则调用初始化脚本。
- 为每个测试用例调用“测试用例驱动”。

测试用例驱动
- 如果需要则执行初始化（调用初始化脚本）。
- 调用应用程序特定的“控制器”脚本，向它传递测试用例的文件名（从电子数据表中保存为“tab 分隔”文件）。

从这一点上讲，这种体系结构与使用驱动脚本来驱动测试集的相同。参见图 7-2。

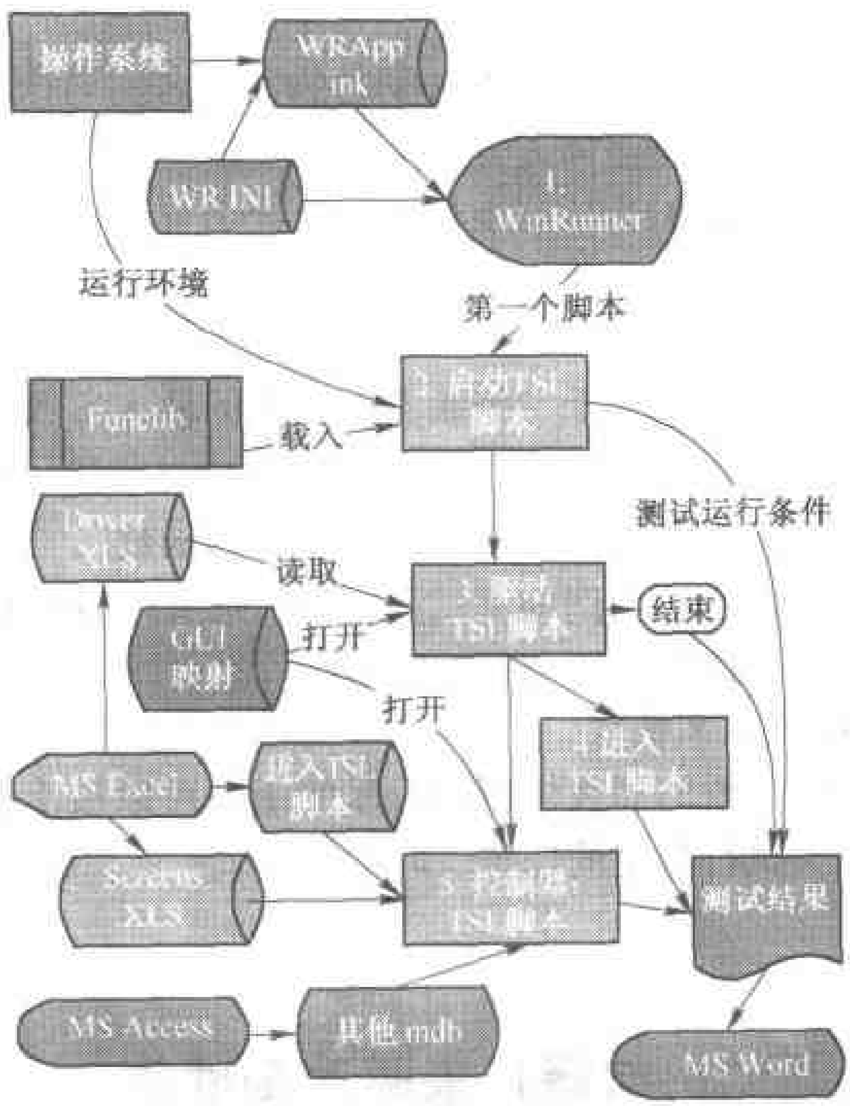


图 7-2　使用 TestDirector 的数据驱动测试执行流程

下面是 TSL 驱动脚本轮廓的一个示例。

像表 7-2 那样填写电子数据表，然后像下面这样做：

1）保存工作簿文件。

2）将电子数据表保存为“tab 分隔”文本文件。使用测试用例的名字做为文件名（如果测试用例 = TC0001A，则文本文件 = tc0001a.txt）。

3）将测试用例加到要在驱动脚本中运行的用例列表中。

当执行 **Driver** 脚本时，要做下面的工作：

- 调用 **Controller** 脚本，传递测试用例名称：ret _ code = call “Controller”（“TC0001A”）。
- **Controller:**
  - * 打开文件 TC0001A.txt 并读取每条记录。
  - * 在“**Test _ Description**”上向测试报告写测试描述。
  - * 在“**Start _ Test**”上调用特定应用程序实用脚本 **Start _ Up**（Screen（界面）~Main Screen（主界面））。
  - * **Start _ Up:**
    - ▪ 向测试报告写数据：测试的开始、时间、日期等。

- 验证我们当前的位置是主界面。
- 返回到 **Controller**。

- **Controller**:
  - * 在“**Test _ Step**”向测试报告写测试步数（Test Step Number）和描述。
  - * 在“**Enter**”，调用一般实用脚本（Generic Utility Script）**Enter**（Selection～1）。
    - * **Enter**:
    - 验证字段“Selection”存在。
    - 在 Selection 中输入 1
    - 返回到 **Controller**。
- **Controller**:
  - * 在“**Action**”，调用一般实用脚本 **Action**（Press _ Key ～ Enter）。
  - * **Action**:
    - 验证“Enter”是一个有效的键名。
    - 调用 WinRunner 函数，按下“Enter”键。
    - 返回到 **Controller**。
- **Controller**:
  - * 在“**Verify**”，调用一般实用脚本 **Verify**（Screen（界面）～ Payment Screen（支付界面））。
  - * **Verify**:
    - 调用 WinRunner 函数，在 GUI _ file 中设置界面（逻辑名）。
    - 注意：它验证 Screen（界面）显示并激活。
    - 返回到 **Controller**。
- **Controller**:
  - * 在“**Test _ Step**”向测试报告写测试步数和描述。
  - * 在“**Enter**”，调用实用脚本 **Enter**（Payment Amount（支付数量）～125.87；Payment Method（支付方式）～Check（支票））。
  - * **Enter**:
    - 验证字段“Payment Amount”存在。
    - 在 Payment Amount 输入“125.87”。
    - 验证字段“Payment Method”存在。
    - 在 Payment Method 输入“Check”。
    - 返回到 **Controller**。
- **Controller**:
  - * 在“**Action**”，调用实用脚本 **Action**（Press _ Key ～ F9）。

* **Action**：
    - 验证“F9”是一个有效的键名。
    - 调用 WinRunner 函数，按下“F9”键。
    - 返回到 **Controller**。
- **Controller**：
  * 在”**Verify**“，调用实用脚本 **Verify**（Screen（界面）～ Payment Screen（支付界面））
  * **Verify**：
    - 调用 WinRunner 函数，在 GUI _ file 中设置界面（逻辑名）。
    - 注意：在这一步中，我们要验证界面没有改变。
    - 返回到 **Controller**。
- **Controller**：
  * 在“**Test _ Step**”向测试报告写测试步数和描述。
  * 在“**Verify _ Data**”，调用实用脚本 **Ver _ data**（Payment Amount～ $ 125.87|Current Balance ～ $ 1，309.77|Status Message ～ Payment Processed)。
    * **Ver _ data**：
    - 验证字段“Payment Amount”存在。
    - 提取 Payment Amount 的内容并与输入的数据进行比较。
    - 如果数据不匹配，向测试报告报告 error-msg 显示预期的和实际结果。
    - 验证字段“Current Balance”存在。
    - 提取 Current Balance 的内容并与输入的数据进行比较。
    - 如果数据不匹配，向测试报告报告 error-msg。
    - 验证字段 Status Message 存在。
    - 提取 Status Message 的内容并与输入的数据进行比较。
    - 如果数据不匹配，向测试报告报告 error-msg。
    - 返回到 **Controller**。
- **Controller**：
  * 在“**End _ Test**”，调用特定应用程序的实用脚本 **End _ Test**（Press _Key～F3|Screen ～ Main Screen）
  * **End _ Test**：
    - 执行 End-Of-Test 函数，包含一些与 Action 和 Verify 实用脚本相同的参数。通常包含测试中的应用程序要求的附加函数。
    - 验证“F3”是一个有效的键名。

  - 调用 WinRunner 函数，按下“F3”键。
  - 调用 WinRunner 函数，在 GUI _ file 中设置界面（逻辑名）。
  - 注意：在这一步中，我们必须最低限度地验证我们返回到了起始位置（“Main Screen（主界面）”）。
  - 返回到 **Controller**。
- **Controller**：
  * 执行 End-Of-Test 函数。
  * 验证在测试过程中打开的附加 GUI _ files 关闭了。
  * 验证在 End _ Test 例程没有运行（由于遇到致命错误）这一事件中我们返回到了起始位置（“Main Screen”）。
  * 调用特定应用程序“ReturnToBaseState（）”函数以确保我们当前在起始位置。
  * 返回到 **Driver**。

驱动脚本从控制器中测试返回码（Return Code）并写“通过/失败”信息给总结报告（Summary Report）。它也决定接下来将运行的测试，并调用控制器脚本，向它传递下一个测试用例名称。驱动脚本继续测试直到不再有测试，除非出现错误使程序结束。

一般来说，不同类型的关键字文件与 Nagle 的 DDE 用来驱动测试（参见表 7-3、7-4 和 7-5）的 Excel 表是相对应的。

表 7-3 测试管理关键字

实用关键字	客户机/服务器用法	Web 用法	注　意
START _ TEST:	Start _ Up	Start _ Up（应用程序特定的）	
TEST _ DESCRIPTION:	不适用-见‘Start _ Test’		
TEST _ STEP:	不适用-见‘Start _ Test’		
END _ TEST:	End _ Test（Application-Specific）		

表 7-4 数据操作关键字

实用关键字	(Gen _ Util)	(Web _ Util)	注 意
ACTION:	Action	Web _ Action	
DO _ IF: /ELSE:			
SELECT _ LIST _ ITEM:	Sel _ List		
ENTER:	Enter	Web _ Enter	
UPDATE _ IF _ NULL:	Enter		
APPEND:	——		
EXTRACT _ DATA:	Sav _ Data	Web _ Save _ Data	
EXTRACT _ TABLE _ DATA:	——	Web _ Save _ Table	
GENERATE _ DATA:	Gen _ Data		
GENERATE _ INPUT:			
CALCULATE:	Calculate (Gen _ Util)		
CALCULATE _ DATE:	Gen _ Data	Calc _ Date (Gen _ Util)	
SET _ CONDITION:	Set _ Cont	Web _ Set _ Cond (Gen _ Util)	
SET _ DATE:	Set _ Date	Web _ Set _ Date	

表 7-5 验证关键字

实用关键字	客户机/服务器用法	Web 用法	注 意
MATCH _ DATA:	Ver _ Data	Web _ Verify _ Data	
VERIFY _ DATA:			
VERIFY _ TEXT:	Ver _ Text	Web _ Verify _ Text	
VERIFY _ TEXT _ LINKS:	——	Verify _ Text _ Links	
VERIFY _ IMAGE	Ver _ Bitmap	Web _ Verify _ Image	
VERIFY _ ATTRIBUTES:	Ver _ Attr	Web _ Verify _ Prop	
VERIFY _ ENABLED:	Ver _ Enbl	Web _ Verify _ Enbl?	
VERIFY _ VALUE:	Ver _ Value	Web _ Ver _ Value	
VERIFY _ NUMERIC _ VALUE:	Ver _ Value	Ver _ Value (Gen _ Util)	
VERIFY:	Verify	Web _ Verify	
VERIFY _ LIST _ ITEMS:	Ver _ List (Gen _ Util)		
VERIFY _ TREEVIEW _ ITEM:	Ver _ Tree	——	

Zambelich 的工具箱也提供了一套“一般的”函数供控制器脚本使用。它们包括三个工具箱函数库（Main _ TK1、Main _ TK2、Main _ TK3）中的函数。这些编译的模块是从“ToolKit.zip”中提取出的，然后被 APP _ Init 文件夹下的脚本 AUT _ INT 放置在 WinRunner 的函数生成器类别 ATS ToolKit Functions 中。这些实用脚本被分成下图所示的类别。

本书 FTP 的网址（www.phptr/mosley）包含解释 Zambelich 的数据驱

动自动化框架的文档。他提供了一个详细描述函数和实用工具脚本的综述文档和幻灯片演示，以及一些很好的电子数据表格（参见图 7-3）样例。

Automated Testing Specialists 公司（自动化测试专家）提供了非免费的工具箱和相关结构程序。如果你正在使用 WinRunner，并想要获得这个工具箱以及实现这个框架，你可以直接与 Keith Zambelich 联系（Automated Testing Specialists, Inc., P.O.Box65564, Los Angeles, CA90065, email: keithz@auto-sqa.com）来获得 WinRunner 的工具箱。

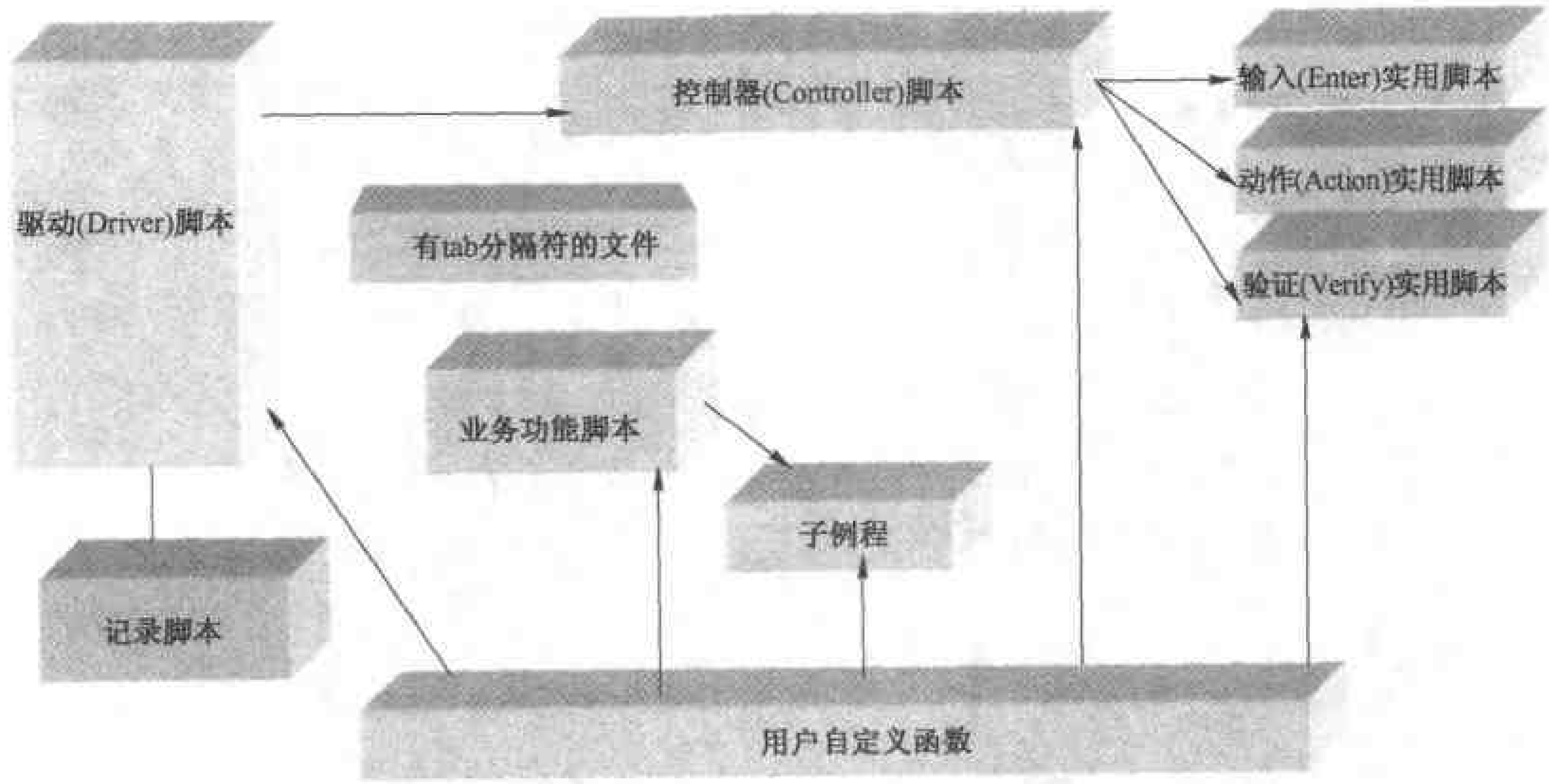


图 7-3 WinRunner 体系结构的工具箱

7.17 小结

本章讨论的数据驱动测试框架是为应用软件自动化测试提供了一般的测试平台，它突破了代表第一代自动化测试的捕获/回放范型的局限。它的出现使得用少量的测试脚本和子例程/函数来执行测试数据的大量功能变化成为可能。

7.18 参考文献

1. Kit, Edward. "Integrated Effective Test Design and Automation." *Software Development* 7, no. 2, February 1999.
2. Mosley, Daniel J. *Client-Server Software Testing on the Desktop and the Web.* Prentice Hall PTR, Upper Saddle River, NJ, 1999.

3. Nagle, Carl. "Test Automation Frameworks." *groups.yahoo.com/group/Robot-DDEUsers/files/Doc/FRAMESDataDrivenTestAutomationFrameworks.htm*
4. Zambelich, Keith. "Totally Data-Driven Automated Testing, Using Key-Word Input to Drive the Automated Testing Process." Whitepaper, Automated Testing Specialists (ATS), *www.auto-sqa.com*

第 8 章 深入了解控制同步数据驱动测试框架

到目前为止，一个接受过有关测试方法的培训、理解客户需求、对所测试的应用程序有深入理解的人是最适合执行软件测试的人。这样的人手工执行测试并在测试执行的过程中观察系统的每一处细微差别。

这种级别的手工测试非常耗时，因此费用昂贵。但是，在今天这个快速应用程序开发（Rapid Application Development，RAD）的环境中，时间和费用都是关键因素。虽然在开始的时候自动化测试并不比手工测试快，而且明显花费更昂贵，但在重复执行回归测试的时候速度就快多了。采用 CSDDT（Control Synchronized Data-Driven Testing，控制同步数据驱动测试）方法有助于节约构建有效的自动化测试套件前期所花费的时间，从而改善自动化测试初期比较耗时这种状况。

8.1 创建数据驱动测试脚本

必须再一次注意，CSDDT 只是实现有组织测试框架的一种方法。测试有效的关键还是在于创建和使用的测试数据。必须记住的是，这个框架不对自身进行确认或对象验证，它仅仅包含那些与 AUT 的窗口和对象相互作用的代码。框架代码解析输入记录并按照其上内容行动，所以说是输入记录决定了交互作用的方式，而不是框架代码。这种关系很像你和你的汽车。汽车可以载着你到各地，但它对自己什么也不能做。而你，汽车司机，就像输入的数据一样，可以控制旅行的进展——开始、加速、转弯、停止，最后在到达目的地后熄火。

执行 CSDDT 的一个好处是可以在构造框架脚本之前或者与此同时设计数据。也就是说 CSDDT 允许实现一个建造测试环境的并发进程。很多时候，可以安排 Dan 开发测试数据，而 Bruce 同时执行测试脚本。这样有助于节约测试开始前的准备时间。CSDDT 还有一个好处，就是不需要在开始测试之前创建完成整套测试。当所有用来与第一套被测试需求进行交互的组件都准备就绪后，一个测试者就可以开始测试了，另一个测试者则继续构建新的或额外的测试数据，而第三个测试者可以构建或实现与下一套窗体和对象交互的新代码。比起只采用捕获//回放来构建自动化测试集的方法来说，这种并行开发能更早地提供有效测试。

8.2 实现 CSDDT 方法

从字面上理解，数据驱动测试的主要概念是从在测试脚本中控制程序流，并且提供输入数据的输入文件中读取数据记录。简单应用可以由被记录的事件完全控制，但是即使被测试的输入数据功能变化有限，这些简单应用也是难于修改的并且可能会变得非常庞大。在 Rational Robot 中，来自键盘的击键被记录为“InputKeys”脚本命令。“InputKeys”记录命令用一个或多个命令记录了一系列击键。下面的例子显示了在向一个窗口输入某地址并且使用 Tab 键从一个输入域跳到另一个输入域时击键是怎样被记录的。用双引号括起来的是被敲入的数据。

InputKeys “123 main street, {BACKSPACE} {TAB} {TAB} Columbia {TAB} I1 {TAB} 6226 {LEFT} 3”

你能很快看出最后结果吗？{BACKSPACE} 用来擦去“street”后面的逗号，而 {LEFT} 3 用来在最后一个数字 6 前插入 3。是的，所有的击键包括错误纠正都被记录下来，这也被称为记录输入噪音。

当有大量击键被记录时，其中经常会涉及好几个 InputKeys 命令。当这些命令与其他许多命令，例如单击编辑框，可以想像以后为了更正错误而修改脚本会有多难。如果许多屏幕都需要输入数据到大量控件中，用几百行脚本命令的确可以轻松完成，但这一招也许在回放的时候非常有用，但完成的测试却有限。如果需要在只有 10～15 个输入对象的 GUI 屏幕上输入 20 套不同的数据，然后执行已记录事件，确认所有输入已被正确存储，那该怎么办呢？我们可以想像这个脚本会有多么庞大。为了能够管理脚本，我们将不得不记录每个脚本的有限功能，不得不记录大量脚本以提供合适的测试覆盖。如果应用程序有十几个不同的屏幕需要测试，那将会需要大

量的测试过程，并且这些过程必须备有大量已记录的测试用例。

数据驱动测试面临的问题在于管理脚本所处的阶段——是否已经完成、正在进行或正需要升级——以及需要多少测试人员和管理人员熟练掌握测试套件。当开发人员对屏幕的功能做了调整，或者移动、添加或删除了正在与测试交互的对象时，该怎样修改这个应用程序的下一个预发布版本的脚本呢？如果公司决定在几个月后的下一个发行版本中，应用程序中将有不同类型的输入控件，例如数据网格随编辑框的类型不同而不同，那先前已经组合在一起的测试还能被重用吗？答案是不可能。

根据我们的经验（从已发布的经验看，这也是其他人的经验），项目级自动化测试失败和终止的首要原因是最终不能正确维护自动化测试脚本。大规模 AUT 经常有改变发生，经常有新的或额外的功能合成，所以尤其会出现这种情况。

8.3 一般问题和解决方法

8.3.1 问题：数据输入

测试需要多样化的数据组合来验证业务规则、编辑规则之类。在使用 Rational Robot 进行记录时，如果必须键入所有组合，那么测试人员最好打字熟练并且有非常好的耐心。另外，所有被键入和记录的数据都是经过硬编码的数据，若要改变或修补它们，就必须编辑或重新记录脚本。

8.3.2 解决方法：使用输入数据文本文件

1) 直接好处在于如果需要额外的数据，只要加入更多的记录就可以了，不必修改测试脚本。

2) 如果需要修改数据记录，可以在原始的 Excel 电子数据表中操作，或者用文本编辑器修改文本文件（最好从修改过的电子数据表注册文本文件，这样可以保持测试数据与测试需求同步），而不必修改测试脚本。

3) 如果需要，或者想创建不同的场景，可以创建多个输入数据文件，而不必修改测试脚本。

8.3.3 问题：程序流改变

在脚本被记录后发现一个屏幕的输入会影响另一个屏幕的数据，所以需要以不同的顺序测试不同的屏幕。

8.3.4 解决方法：让输入数据做驱动

建立一套简单的代码，代表将在其中进行输入、选择、删除等数据操作的目的窗口中的不同标签页（tab）和不同的窗口。然后建立一套代码或关键字，代表某个动作，例如插入、删除、更新等任何想做的操作。当你决定所需的新动作时，你可能发现自己会把它加到列表中。再计划着创建一套代码，代表相应的一套预期的错误响应。在确定了错误响应是什么之后，你可能会有些手忙脚乱地做这些。

这些代码被包含在每个记录的开头，用来指定到哪里，到那之后做什么以及寻找何种响应。记录的其他部分用来作为输入或找到的/查询到的数据。

设计脚本是为了读取代码并且完成适当的动作，所以需要提前记录一小部分脚本命令，用来响应数据文件代码。例如，要到达某个指定窗口必须选择的菜单选项，或/和到达应用程序中指定位置需要的标签控件点击。这些代码片段被分别放入 Select/Case 语句中，这些语句负责完成一些非常有限却可重用、可管理的功能。参见《SQABasic Language Reference》中有关 Select/Case 语句的部分，这本书在 www.rational.com/docs/v2002/sqabasic.pdf（2）上可以找到，对 Rational SuiteTest Studio 的注册用户开放。

8.3.5 问题：管理应用程序改变

这种改变可能是需要不同的按钮、菜单选项、标签页点击等到达程序的某个特定位置，或者是屏幕上的对象或控件被改变了。

8.3.6 解决方法：记录或修改非常小的一部分代码

如果应用程序中某些部分被改变了，或者现在需要一套不同的菜单选项和标签页点击来到达某个特定位置，你只需要重新记录这一小部分代码。这一部分非常容易确认，并且很容易从脚本中找到，因为它是被单独存放

在小用例声明当中的。这一条对控制动作和响应的代码片段同样适用。

注意：

构建代码非常重要，因为只有这样，任何给出的 AUT 对象或控件才会只和测试脚本中的一个位置交互作用——这是一个与控件相互作用的位置，一个在改变时用来修复的位置。

8.4 设置通用的启动和结束测试条件

任何一套被记录的测试脚本都有必要总是在同一点开始和终止，这样可以确保 AUT 与它在初始记录发生时的状态一样。举个例子就是 AUT 被打开和最大化的情况以及不存在上一次手工或自动化测试遗留的打开的子窗口和数据值。如果不能返回初始状态，测试就不能重复，因此在声明自动化测试脚本前必须一直清理 AUT。当然有时你可能会依赖那些存在的剩余数据或条件（持续的应用程序），因为你在为它们做测试。做测试你也许会陷入另一种情况，即日期标志和其他数据会自动从一个测试更新到下一个中去。应该除去测试对这些项目的敏感性，这样才能保证后续测试不会失败。

8.5 修改已记录的代码以接受输入数据

修改已记录代码/脚本的关键因素是理解需要修改什么才能利用外部文件数据代替记录期间键人的数据。

一般把那些与 AUT 窗口或标签页上的数据输人控件和对象相互作用的代码放在 Perform _ Action 脚本或从 Perform _ Action 脚本调用的脚本中。参见第 7 章中的高级程序流和相关的描述。

暂且假设我们想向一个窗口输人数据，那个窗口已经被打开并且被激活了，而且正处于焦点中。同时出于讨论的目的，我们假设这个窗口或标签页仅有三个输人域用于键人数据。随后，使用 Rational Robot 记下每个域的选择，这里最好按 tab 顺序记录（假设已经完全设定好了）。如果没有使用 tab 顺序，应该按用户选择的一般顺序记录选择。在选择每个输人域后我们都将键人一个值。先键人 1，然后使用 tab 键跳转到下一个输人域，或者双击下一个输人域，键人 2……以此类推。这有助于我们在代码中轻松地确认被选字段，代码就可以被即刻更改。

8.6 非常重要的习惯

大多数情况下，在数据输入域上双击是首选的选择方法，而且比起单击对象来，双击经常更有必要。双击选择这个域中的所有数据，这样就能让新数据完全代替旧数据。单击操作可能只会将光标置于输入控件中，而不会选择现有数据，而且通常只是激活了插入模式，并不能用新数据替代现有数据，所以并不是个好方法。在某些应用程序中，若干单词或值存在于同一个编辑框中，以空格相隔，我们发现即使是双击也可能不会选中全部文本用于替换。在这种情况下，确保所有数据都被选中的方法是通过键盘进行控制，例如使用制表符、右键头或者鼠标点击。一旦光标被放置到控件中，就请记下这些键盘操作：

- Home
- Shift + End
- Delete

这样就选择了所有当前允许完全替换的数据，之后就可以删除它们了。

就像我们先前在第7章中示范的一样，包含在Perform_Action脚本中的小段被记录代码看上去可能是这样的：

```
Window SetContext, "Caption=Window Name", ""
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys "1"
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys "2"
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys "3"
PushButton Click, "Text=OK"
```

注意粗体显示的值1、2、3所在的位置。我们将用已经用过的、包含从外部源文件读取的数据的变量名来替代InputKeys命令中的数据值。假设数据被读入一个叫DFld（）的字符串数组，然后用变量DFld（x）替代已记录的键入值。这里的x代表数组的下标。虽然字符串数组的下标一般以0而非1开始，但是为了方便初级编程者，我们将从1开始。记着我们先前的讨论，每个数据记录的前七个字段首先被读取，被放进单个的、独一无二的字符串变量中。其中一个值告诉程序有多少数据值需要被读取。基于这个值，我们设置数组Dfld（）的大小与之匹配，然后利用一个小循环读取输入记录的数据字段中剩余部分。

DFld（1）包括第一个被读取的数据值；

DFld（2）包括第二个被读取的数据值；

DFld（3）包括第三个被读取的数据值；

等等，依此类推。这个循环会继续下去，因为我们需要读取和获得所有的数据字段。

现在，为了利用数组中的数据而不是我们记录的硬编码值，我们编辑的只是小段记录代码。我们记录的代码现在可能会是下面这样：

```
Window SetContext, "Caption=Window Name", ""
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys DFld(1)
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys DFld(2)
EditBox DblClick, "Label=Label name:", "Coords=x,y"
InputKeys DFld(3)
PushButton Click, "Text=OK"
```

注意文本中的粗体。这些是数据字段 DFld（1）、DFld（2）、DFld（3），而不是我们原来记录的、从字面上看是硬编码形式的值 1、2、3。

这时，重新执行脚本，输入的将会是随后读入字符串数组 Dfld（）的数据文件中的值，而不是那些原始输入值。也就是说每次改变那些将会被载入数组的输入文件中的数据，随后重新执行这个脚本片段，就能输入和利用不同的值。在用来做 CSDDT 的整套脚本中，这应该是我们与那些控件相互作用的惟一位置。如果其中一个控件发生改变，我们将不得不在惟一的位置做修复——那就是这儿。

很明显，编辑框不是我们一定要交互的惟一的控件和对象——还有许多其他要交互的控件和对象。如在控件中，有下拉选单（DDL）：

- 组合框
- 复选框
- 单选按钮
- 命令按钮
- 网格
- 菜单条

为了提供合适的选择或交互，必须确定哪些已记录代码能被修改或用数据值替换。有时必须提前决定控件的状态，就像必须提前决定复选框的选项一样，这样才能决定是否需要通过点击它或者不点击它来改变。输入数据可能会被用来指出控件所需要的状态（例如：选上还是不选）。我们应该编程测试控件状态，然后确定它是否符合数据的指定状态。如果二者相同（都指示选择），那我们将不会点击复选框。如果不同，就点击复选框，把它设成新的状态。通常我们不能将数据值直接包括进已记录代码中，就像复选框不能直接包括选项一样。

现在我们将检查另一个控件——单选按钮。从使用的本质上讲，单选

按钮是至少两个按钮的聚集体。我们将不得不按顺序记录每个单选按钮上的点击，这样才有机会选择任何一个按钮。然后我们将修改代码，让输入数据指定组中将会被选择的按钮。下面这个例子描述了此过程在 If/Else 结构中可能的发生情况。注意黑体字的信息。Dfld（1）包括测试数据，是为了观察哪个按钮会被选择。我们仅用 RadioButton _ 1 和 RadioButton _ 2 来命名选择代码。在数据值没有被直接输入到 GUI 但是被用来决定做什么或者执行哪块库代码时，可以参考这段代码来进行基于数据值的选择。

```
if Dfld(1)="RadioButton_1" then
    Window SetContext, "Name=main window name", ""
    Window SetContext, "Name=child window name;ChildWindow", ""
    RadioButton Click, "Name=rb_name1"
    Window SetTestContext, "Name= main window name ", ""

    Window SetTestContext, "Name= child window name;ChildWindow", ""
    Result = GroupBoxVP (CompareProperties, "Name=gb_mybutton_group",
           CaseID=RadioButtonProp1")
    Window ResetTestContext, "", ""
elseif Dfld(1)="RadioButton_2" then
    Window SetContext, "Name=main window name", ""
    Window SetContext, "Name=child window name;ChildWindow", ""
    RadioButton Click, "Name=rb_name2"
    Window SetTestContext, "Name= main window name ", ""
    Window SetTestContext, "Name= child window name;ChildWindow", ""
    Result = GroupBoxVP (CompareProperties, "Name=gb_mybutton_group",
           CaseID=RadioButtonProp2")
    Window ResetTestContext, "", ""
end if
```

注意，在完成选择后，我们记录了一个验证点，该验证点捕获单选按钮组的属性，确认所选按钮的确是计划选择的，并且状态与记录脚本时的状态相同。要验证所发生的动作是预期的，这种事后反复核对的做法是必要的。在脚本的任何地方完成动作后，都必须确认动作的结果合乎预期，否则就不算完成了一个有效测试！有时除了确认没有错误发生外没有其他需要验证。

在 Rational Robot 中，初次记录一个验证点时，例如为对象属性，这些属性被存放在测试工具的数据知识库中作为基准数据。之后，在回放过程中进行验证时，工具将再一次需要被测对象的属性并且会把结果与先前保存的基准数据作比较。这是一个有用的工具，但是别错误地假设初始记录值是正确的！虽然很少，不过有时候捕获的初始值的确不会如实反映一个或多个属性。在初始创建脚本的时候，测试者一定要确认当时值是正确的。因为当重新执行脚本并确认基准数据是否正确时就变得让人难以理解了。

一定要记住：在自动化任何测试之前都必须手工地或在测试工具的帮助下确认所进行的测试工作正常。

验证点确认我们的选择正确，这种说法其实并不完全正确。实际上，回放

的验证点仅能确认我们原来记录的数据和被捕获的属性仍与基准数据相同。

如果在 Perform _ Action 脚本中所做的是输入数据或者组合输入的数据、选择单选按钮等事情，那随后一般需要完成类似下面的动作：Add（增加）、Save（保存）、Delete（删除）、Update the database（更新数据库）、OK（确认）。

为了完成这些动作，选择一个控件按钮或菜单选项，例如 File→Save。一旦最后一个动作执行完，我们将不得不寻找来自 AUT 的预期正常结果或响应，或者如果我们所期望的是错误，寻找的将是错误状态。一般情况下，点击 OK 按钮或者 Save 按钮时，总会出现一些指示器通知用户动作已经完成，一切与预期相符，但是事情不会总是这样。如果有指示器，就将它捕获并放进你的测试工具中，这样在回放时就可以依靠肯定的确认来表示结果是正确的。但是，不会总是有东西捕获，并且确认发生的前提是没有错误发生。实际工作中，依赖于操作的关键程度，你可以决定执行一个过程，跳到 GUI 的后面检测动作是否正确执行。在合适的情况下，你可以通过访问相关数据库，获取正确数据，然后拿它与输入的值比较来完成检测。这是一项意义重大的任务，所以必须确定为创建这些额外的例程或函数所付出的努力是否值得，并且对在其上花费的时间是否有很好的回报。我们已经在 FTP 站点上提供了一个开发好的例子。有许多有关这方面主题的信息，其中的一位作者是 Satisfice 有限公司的 James Bach，可以从 Internet 上得到这条信息。

警告：

如果菜单选项或者弹出菜单能完成同样的任务就不要使用工具栏按钮。因为在已记录的脚本中，菜单选项更加可靠（不会经常改变）。

8.7 为通用操作创建函数——隔离命令对象

如果 AUT 有任何共同的命令按钮要交互——例如：那些在工具栏上的按钮——或者有任何菜单动作要完成，请记录下它们的选择或那些能从测试脚本集的任何地方调用的函数中的应用。这可能需要花一定的功夫去隔离交互对象，但考虑到在问题对象改变时会省去麻烦，这么做还是很值得的。再强调一次，这个过程能保证与这些控件有关的改变发生时代码中仅有一处需要修改。

8.8 继续程序流

现在动作已完成，是开始检查错误代码值，确定是否应该寻找错误状态的

时候了，这个错误状态可以是错误消息或者其他可以确认的错误状态。还记得刚才提到的窗口/标签页选项吗？我们将以和对待它们同样的方式建立 Select Case 结构。我们已将这部分代码隔离到一个脚本中去了，这样有助于保持主脚本较小和可管理。再一次强调，如同第 7 章提到的，我们将会使用像 Window _ Select、Tab _ Select 和 Perform _ Action 一样的脚本框架。

错误处理模板的脚本可以像下面这样编写：

```
'$Include: "DataDriven.sbh"
    Declare Function Process_Error(ErrCd)
Sub Main
    Dim Result As Integer
    Gen_Return_Code=Process_Error(ErrCd)
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Process_Error(ErrCd)
    Process_Error=1

    Select Case ErrCd

        Case "ERR1"
        'Process or detect error code 01 here
            MsgBox "ERR1 code"

        Case "ERR2"
        'Process or detect error code 02 here
            MsgBox "ERR2 code"

        Case "ERR3"

        'Process or detect error code 03 here
            MsgBox "ERR3 code"

        Case Else
            Msgbox "Error Code not found for ErrCd: "&Cstr(ErrCd)&" Terminating"
            Process_Error=0  ''return value FAIL
            SQALogMessage sqaFail, "Error Code not found for: "&Cstr(ErrCd)&"", ""
    End Select
End Function
```

如果找到了期望的错误状态，这条记录的处理就成功完成了。如果还没有到达输入文件的末尾，还有很多记录需要读取，主脚本就得读取下一条数据记录，继续进行处理。

如果期望的错误状态没有被找到，就会发生测试失败（通常是由测试工具记录的验证点），测试就应该终止。测试日志应该标出最后一条记录读取的数据和记录的号码，这有助于快速调试和隔离问题。实际工作中，我们可以在主脚本中用非常少的代码完成这些工作。让我们看一些 Perform _ Action 代码的例子，这些代码向屏幕输入了一些数据，选择 OK 按钮，看是否会出现预期的错误，然后退出，让 Process _ Error 脚本处理错误。

```
'$Include: "DataDriven.sbh"
Declare Function Perform_Action(ActCd)

Sub Main

    Dim Result As Integer
    Gen_Return_Code = Perform_Action(ActCd)
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Perform_Action(ActCd)
    Perform_Action=1

    Select Case ActCd

        Case "User_Pref_OK"            'processing for selecting the OK button on the User
                         Preference window.
            Window SetContext, "Caption=Window Name", ""
            EditBox DblClick, "Label=Label name:", "Coords=x,y"
            InputKeys DFld(1)
            EditBox DblClick, "Label=Label name:", "Coords=x,y"
            InputKeys DFld(2)
            EditBox DblClick, "Label=Label name:", "Coords=x,y"
            InputKeys DFld(3)
            PushButton Click, "Text=OK"
            If ErrCd ="none" then          (ErrCd equal to text "none")

              (If we are not expecting an error then record whatever the normal ending
                         status
               would be after selecting OK here, otherwise if we expect an error
               we would not process this code and we would rely on the DDProcess_Error  to
               handle it.)
            End if

        Case Else
            Msgbox "Action Selection not found for ActCd: "&Cstr(ActCd)&" Terminating"
            Perform_Action=0  ''return value FAIL
            SQALogMessage sqaFail, "Action Selection not found for: "&Cstr(ActCd)&"", ""
    End Select
End Function
```

到这里，我们已经回到了主脚本。主脚本在这个时候会检查 ErrCD 变量，发现这个变量不为空，就会确定错误是否是预期的，然后调用 DDProcess _ Error 程序进行处理。假设我们的数据文件和错误处理代码会产生预期的 Data Error 消息框，还会有相应的文本解释错误。这样，一旦我们检测到错误窗口，我们就能选择 OK 按钮来关闭它。如果我们期望一个错误，然后得到了它，那我们的测试用例就通过了。

```
'$Include: "DataDriven.sbh"
    Declare Function Process_Error(ErrCd)

Sub Main
    Dim Result As Integer
    Gen_Return_Code=Process_Error(ErrCd)
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Function Process_Error(ErrCd)
    Process_Error=1

    Select Case ErrCd

        Case "User_Pref_Data_Error"
            Result = WindowVP (Exists, "Caption=Data Error", _
                            "VP=Window Existence ")
            Window SetContext, "Caption= Data Error ", ""
            PushButton Click, "Text=OK"

        Case Else
            Msgbox "Error Code not found for ErrCd: "&Cstr(ErrCd)&" Terminating"
            Process_Error=0  ''return value FAIL
            SQALogMessage sqaFail, "Error Code not found for: "&Cstr(ErrCd)&"", ""
    End Select
End Function
```

到目前为止，我们已经完成了处理一条记录的循环。

8.9 使用多个输入记录来创建测试场景

你可能已经注意到了，在框架中没有特定的区域用来关闭窗口或标签页。这是因为根据我们测试不同应用程序的经验我们能遇到多种方式来关闭窗口或标签页。有时用Cancel命令关闭，有时选择OK或Finish按钮关闭。经常会有一个Close按钮，还有其他的方式——系统的X关闭按钮，File——>Exit菜单选项，也起同样的作用。通常，关闭窗口和标签页的方法比打开它们的方法多。所以如果在测试场景中需要将关闭窗口或标签页程序化，只需创建额外的动作代码引发关闭动作即可。就像在Window _ Select和Tab _ Select脚本中创建了小段代码打开窗口和标签页一样，在Perform _ Action脚本中也可以创建小段代码来关闭它。但是你必须同时意识到，正因为如此，你经常需要在输入数据文件中创建多个记录，用给定窗口完成全部的测试场景。例如，你可能需要创建一个记录打开一个窗口并且在输入任何新的数据之前或者与它交互之前测试它的初始状态或内容。你可能需要另一个记录来将新的信息输入到同样的窗口或标签页中，然后通过选择Apply按钮向服务器提交改变，但是保持窗口处于打开状态。接下来可能还希望再使用一个记录调用一个子例程，来访问应用程序的数据库，确认来自前一条记录的Apply动作确实存储了正确的信息。最后，但是同样重要的是你可能还要选择添加一条记录用来关闭窗口。

说了这么多，希望你现在能明白，在测试大部分测试用例的时候都需要创建和组织一系列输入记录来执行所有的必要动作。

8.10 使用动态数据输入——关键字替换

在数据驱动测试和手工测试中遇到的一个挑战是不得不处理不再有用的数据。一个典型例子是已经过时的日期值。我们使用的一个应用程序有一个业务规则：当输入某些交易时，其日期输入字段需要处于从当前系统时间算起的 7 天之内。这意味着，不论我们使用的是硬编码的脚本或者来自外部文件的数据，都必须遵守这个业务规则，定期修改日期值。矛盾很快就出现了，我们需要用另一种方法处理这种问题，解决方案是使用数据替换例程。在输入数据文件中我们放置了特殊的关键字，读取数据的脚本检测到它时，关键字会自动替换成一个替代值。这就像类似微软 Word 的字处理器完成自动更正操作一样。这个例程为其他值的替换指明了道路，所以非常有用。

在客户机—服务器应用程序上工作时，如果应用程序使用的是相关的数据库上一份产品数据的备份，其所属公司就会定期用较新的数据替换它。由于我们用来测试的数据值被新数据替换掉了，一套全新的问题就摆在了我们面前。这迫使我们在数据库中寻找出新的可接受的值用来测试，然后用新的信息替换数据驱动输入文件中的数据。这时我们又需要一个方法来自动更新或替换我们的测试数据。既然我们已经创造了一个自动改变日期值的方法，那么为与现存数据库内容相关的数据作同样的事情是很自然的。我们将使用关键字引出脚本，对数据库进行查询，找出可接受的记录与当前测试一起使用。这么一来，测试本质上就变成动态的了并且不会再对数据库内容敏感。这样问题就变成了怎样实现这个新的代码。最终我们选择使用一个二维字符串数组（就像有多行的电子数据表中的两列）。第一维（或列）包括用来与输入数据文件匹配的关键字。第二维（或者来自相同索引或行的列）包括一个获取所需数据的函数调用。新的数据值然后被载入到 DFld（）数组中，代替从输入数据文件中读取的关键字。

关键字替代例程仅在新读入的数据记录的类型值是 K 的情况下才会被调用，所以不会在每次读入新记录时候都调用这个例程。K 记录类型向主脚本表明关键字已经在输入数据记录中被替换了。检查新记录的每个数据字段，察看是否与任何预定义的替换值匹配。如果匹配，替换例程就会被调用。这就意味着一旦定义了关键字，就必须创建例程或函数来获取替换数据。我们已经提供了一些例子供使用和模仿。

8.11 使用库文件或包含文件（Rational Robot 中的 * .sbh 文件和 * .sbl 文件）

现在你应该已经注意到在每个脚本的开始有一行代码看起来像这样：'$Include:"DataDriven.sbh"或者'$Include:"global.sbh"，或者有时是两者或多者的结合。在 SQABasic 帮助文件中有关于包含文件使用方面的相当详细的信息。$Include 值是一个元命令。Rational Suite Test Studio 中的 SQABasic 帮助文件声明，元命令是一个提供给编译器有关怎样建立程序的指令的命令。在 SQABasic 中，元命令由美元符号（$）开头的注释指定。单引号（'）表明是注释，所以'$指定元命令。'$Include 表示命名文件——例如 DataDriven.sbh 或 global.sbh 或 * .sbl——实际上被用作包含文件。

包含文件典型地有两种风格——头类型和库类型。主要区别是头文件包含变量的定义，这些变量可以被包含它们的所有脚本文件使用或引用，库文件包含类似代码的函数或程序例程，它们可以被包含它们的脚本调用的。库文件包含能被编译和执行的代码，而头文件不包含这些代码，它们仅包含能被引用的变量。

在 Rational 的测试产品中，脚本语言是 SQABasic（在原始构建器之后，SQA）。因此 SQABasic 头文件（SQABasic header）的后缀是 .sbh 而 SQABasic 库文件（SQABasic library）的后缀是 .sbl。

注意不可记录 Rational * .sbl 文件内的事件。只能通过编辑器创建这些文件。包括汇编语言和 C 语言在内的许多编程语言都使用这些种类的文件。因此我们在我们的脚本程序中也使用这些文件类型。

有关使用包含文件的更详细信息，请参见测试工具的脚本语言帮助文件。

如果你创建和使用这条文件类型，就要遵循这条规则。通常以 global 命名的文件就是这样。它们应该包含所有程序或脚本可以使用的值。如果想为特定的测试集定义变量、常量等，请使用适合此测试集的名字，比如用 DataDriven.sbh 和 DataDriven.sbl 命名。

因为 Rational 的 SQABasic 语言的一些编程限制，我们大量使用 * .sbh 文件提供对类似 WinCd、ActCd、ErrCd 和 FldCnt 代码变量的全局访问。

以下是 global.sbh 文件。包括一个为关键字日期提供替换值的变量。

```
'SQABasic Header File
'Global Values
    Global Today As String
    Global Today2K As String
    Global Tomorrow As String
    Global UKToday As String
    Global NextWeek As String
    Global LastWeek As String
    Global NextMonth As String
    Global LastMonth As String
```

```
        lpstrFile As Long
        nMaxFile As Long
        lpstrFileTitle As Long
        nMaxFileTitle As Long
        lpstrInitialDir As Long
        lpstrTitle As Long
        Flags As Long
        nFileOffset As Integer
        nFileExtension As Integer
        lpstrDefExt As Long
        lCustData As Long
        LpfnHook As Long
        lpTemplateName As Long
    End Type

 Declare Function lstrcpy Lib "kernel32" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal
                          lpString2 As String) As Long

Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA" (pOpenfilename
                          As OPENFILENAME) As Long
Declare Function FindWindow     Lib "user32" Alias "FindWindowA" (ByVal lpClassName As
                          String, ByVal lpWindowName As Any) As Long  'Last parameter used
                          to be As String

Declare Function CommDlgExtendedError Lib "comdlg32.dll" () As Long

   Global Const OFN_READONLY = &H1
   Global Const OFN_OVERWRITEPROMPT = &H2
   Global Const OFN_HIDEREADONLY = &H4
   Global Const OFN_NOCHANGEDIR = &H8
   Global Const OFN_SHOWHELP = &H10
   Global Const OFN_ENABLEHOOK = &H20
   Global Const OFN_ENABLETEMPLATE = &H40
   Global Const OFN_ENABLETEMPLATEHANDLE = &H80
   Global Const OFN_NOVALIDATE = &H100
   Global Const OFN_ALLOWMULTISELECT = &H200
   Global Const OFN_EXTENSIONDIFFERENT = &H400
   Global Const OFN_PATHMUSTEXIST = &H800
   Global Const OFN_FILEMUSTEXIST = &H1000
   Global Const OFN_CREATEPROMPT = &H2000
   Global Const OFN_SHAREAWARE = &H4000
   Global Const OFN_NOREADONLYRETURN = &H8000
   Global Const OFN_NOTESTFILECREATE = &H10000
   Global Const OFN_SHAREFALLTHROUGH = 2
   Global Const OFN_SHARENOWARN = 1
   Global Const OFN_SHAREWARN = 0
```

8.12 实用脚本

就像这本书的前面说到的那样，Andy Tinkham 开发了一个实用脚本（并且我们也使用了），将 Excel 电子数据表的内容转换成连续的 SQABasic 文本文件。对于不使用 Rational Robot 的人来说，除了可以看到 Andy Tinkham，Excel 访问例程的原始开发者怎样建立这部分代码之外，这个脚本没有太大的用处。而对于使用 Rational Robot 的人来说，转换脚本不仅显示了访问 Excel 电子数据表的方法，而且显示了怎样建立和使用 SQABasic 支持的、强大的对话框功能。

转换程序允许选择 input.xls 文件的路径名。在运行对话框的时候注意内嵌的浏览器按钮、这个程序也允许指定输出文本文件（*.txt）。接下来就是选择开始列（这一列包含记录的类型）和使用的最大列数或者最后一列（最长记录的最后一个数据字段），还有开始行（第一个输入记录的行）和选用哪个 Excel 电子数据表，例如 1～6、这决定了在遇到空行时该在何时结束。这个程序被定制成和 CSDDT 使用的七个控制字段一起工作，通过第六个字段决定在给出的行中要读取多少个数据记录。数据从电子数据表中读取，并且转换成 CSDDT 格式，CSDDT 格式在本书的前面已经解释过。

8.13 调试脚本——当测试发现错误的时候

计划和设计测试脚本是很重要的。建立它们是非常有趣的一件事，至少当时会有许多有趣的事发生。可以在这个过程中学到很多东西，尤其是有关所使用的工具和它的优缺点方面。考虑怎样解决具体的问题是非常有意义的，但是，每天这么做却又是另外一种情况了。在主脚本中，我们已经包含了记录计数器。每读取一个记录就将记录计数器的值写人 Robot 的日志文件中。在决定哪个输人数据记录引发错误时这是一个非常重要的工具。不过可以想像，从几百行日志信息中确定引发错误的记录该有多难。虽然没有把行号作为 CSDDT 输人记录格式的一部分，但包括 Robot 中的编辑器在内的大多数好的文本编辑器都显示了行号。所以如果把每个输人数据记录的第 7 个控制字段提供的注释包含在内，就很容易确定测试在做什么，错误发生在哪里（当注释文本包括在文本日志中时更是如此）。

8.14 实现 CSDDT 模板脚本

注意：

如果已经实现了 global.sbh 文件，不要对它进行复制覆盖。相反，用文本编辑器打开被包括的 global.sbh 文件，在 Edit 菜单中点击 Select All，点击 Copy 复制，然后将它粘贴到现存 global.sbh 文件的末尾。

只有在读完并且理解了这个注意事项后，你才能开始实现脚本。

步骤1 将 * .sbh 和 * .sbl 文件复制到默认的 sqabas32 文件夹。在 Rational 2001 中文件夹应该是在默认路径下，例如：

```
C:\your repository folder.....\TestDatastore\DefaultTestScriptDatastore\TMS_Scripts\SQABas32
```

步骤2 将 * .rec 文件复制到默认脚本文件夹。在 Rational2001 中，除非管理员改变路径，否则文件夹应该在默认路径下，例如：

```
C:\your repository folder.....\TestDatastore\DefaultTestScriptDatastore\TMS_Scripts
```

步骤3 复制完脚本文件后，Robot 还不会意识到这些文件的存在。必须打开 Robot，选择 New Script。键入的脚本文件中的名字与被复制的文件名字相同。点击 OK，文件就会显示出来，而此时脚本命令已经加进去了。

可以用类似微软 Notepad 的标准文本编辑器打开并编辑 * .sbh、 * .sbl 和 * .rec 文件。

在所有文件被创建并且关闭后，转到 Robot 的 File 菜单，选择 Compile All。假设所有文件都被复制到了正确的位置，脚本这时就应该可以运行了。

8.15 DD 脚本

在本书支持站点上可以找到下面的模板文件。

```
'DDMain Template Version 2.0

'This is a template file updated May 24, 2001
'updated May 25, 2001  added the openfile browser for selecting the input file
'updated May 31, 2001  change call function window_select to CallScript DDWindow_Select.
                       Added Global Genral_Return_Code in DataDriven.sbh
'updated June 6,2001  change call function perform action and process error

'$Include: "global.sbh"
'$Include: "DataDriven.sbh"

   Declare Function ReadFields BasicLib "global" (File_Num, Fld_Cnt, GFlds())
   Declare Function Key_Word_Sub  BasicLib "DDKeyWord_Sub" (Fld_Cnt)

   Dim Rec_Type                'Record Type
   Dim Fld_Cnt                 'Data Field count
   Dim Comment As String
   Dim Child_Open As String
   Dim record_counter

Option Compare text                    'allows LIKE to be case insensitive

Sub Main
    Dim Result As Integer

    On Error GoTo Error_Handler1
```

```
    Child_Open="No"
    current_win=""
    current_tab=""
    record_counter=0

'other ways to initialize the InfileName variable
    'InFileName="c:\sqatemplates\testdata\testdata1.txt"
    'InFileName=InputBox$("Enter the name of the input file to process.","Data Driven_
     'Testing",InFileName)

    If txt_filename="" then
        CallScript "OpenFile_Txt"
         'open a text file for test input data this script opens a file browser window
    End if

    InFileName=txt_filename                    'from the Openfile_Txt routine

    If InFileName="" then
        Exit Sub     'the test is canceled no filename was supplied
    End if

    Open InFileName For Input As #1
    txt_filename="" 'clear the name for the next time the script is started

    Do while Not EOF(1)   'Main process loop. Read and Process input records
                                     ' .until there are none left.

''''''''''''''''''''''''''''''
'Read input test data file
''''''''''''''''''''''''''''''
        Input #1,Rec_Type,WinCd,TabCd,ActCd,ErrCd,Fld_Cnt,Comment

        if Fld_Cnt<> 0 then

            ReDim DFld(Fld_Cnt)    'set the appropriate size of the array for the number of
                                   'data fields in this record

            'pass file number, fld_cnt, address of DFld
            Result=ReadFields(1, Fld_Cnt, DFld())
            if Result=0 then
                MsgBox "ReadFields function failed. Program terminating"
                Close #1  'close the open file
                Exit Sub
            end if
        end if

''''''''''''''''''''''''''''''''''''''''''
'Log which record we are processing
''''''''''''''''''''''''''''''''''''''''''
        record_counter=record_counter+1
        SQALogMessage sqaPass, "last record read ="&Cstr(record_counter), ""

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Determine the record type
'
' H=header record, skip this record do not process
' B=Breakpoint stop for program debugging
' X=Terminate prior to the end of the input data file
' G=Good record. Process normally
```

```
' K=Good record. Pre-Process for Keyword substitution,
'   then process normally
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
        if Rec_Type LIKE "H" then
            goto NoProcess
        elseif Rec_Type LIKE "B" then
            stop            'data breakpoint
        elseif Rec_Type LIKE "X" then 'terminate the program
            Close #1
            Exit Sub
        elseif Rec_Type LIKE "G" then
            Goto Start                  'good record
        elseif Rec_Type LIKE "K" then
            key_word_sub(Fld_Cnt)
            'this call is used when keywords are used in the data such as 'AutoDate"
        else
            MsgBox "The record type is invalid. The program is terminating"
            Close #1
            Exit Sub
        end if

Start:
''''''''''''''''''''''''''''''''''''''''''''''''''
'Determine if we need to open a new window or not
''''''''''''''''''''''''''''''''''''''''''''''''''
        if Child_Open="No" then
            'go open the selected window. This is executed at first pass only.
            CallScript "DDWindow_Select"
            if Gen_Return_Code=0 then
                close #1
                exit sub
            end if

            Current_Win=WinCd       'update and save the current window selection
            Child_Open="Yes"           'we have now opened a window
        else
            if Current_Win<>WinCd then
                CallScript "DDWindow_Select"
                if Gen_Return_Code=0 then
                    close #1
                    exit sub
                end if
                Current_Win=WinCd   'update and save the current window selection
                Child_Open="Yes"       'we have now opened a window
            end if
        end if

'TAB SELECT
''''''''''''''''''''''''''''''''''''''''''''''''''
'Determine if we need to open a new tab or not
''''''''''''''''''''''''''''''''''''''''''''''''''
            if TabCd <> "" and TabCd <> "none" and TabCd <>_
                "TabCd" and Current_Tab<>TabCd then

                CallScript "DDTab_Select"
                if Gen_Return_Code=0 then
                    close #1
                    exit sub
                end if
```

```
                Current_Tab=TabCd
            end if

''''''''''''''''''''''''''''''''''''''''
'Perform the specified Action
''''''''''''''''''''''''''''''''''''''''
        CallScript "DDPerform_Action"
        if Gen_Return_Code=0 then
           MsgBox "Specified Action Could Not be Performed"
           close #1
           exit sub
        end if

''''''''''''''''''''''''''''''''''''''''
'Process the Error Code
''''''''''''''''''''''''''''''''''''''''
        if ErrCd="" or ErrCd="none" or ErrCd="ErrCd" then 'Error is not expected
            goto NoProcess
        else
            CallScript "DDProcess_Error"
            if Gen_Return_Code=0 then
               MsgBox "Specified Error Processing Could Not be Performed"
               close #1
               exit sub
            end if
        end if

NoProcess:
    Loop 'end of main loop
    Exit Sub

Error_Handler1:
    MsgBox "Error number "&Cstr(Err)& " occurred at line: "&Erl &" -- "&Error$
    Exit Sub
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Error code  Error Text as defined in SQABasic for runtime errors
'        5   Illegal function call
'        6   Overflow
'        7   Out of memory
'        9   Subscript out of range
'       10   Duplicate definition
'       11   Division by zero
'       13   Type Mismatch
'       14   Out of string space
'       19   No Resume
'       20   Resume without error
'       28   Out of stack space
'       35   Sub or Function not defined
'       48   Error in loading DLL
'       52   Bad file name or number
'       53   File not found
'       54   Bad file mode
'       55   File already open
'       58   File already exists
'       61   Disk full
'       62   Input past end of file
```

```
'        63   Bad record number
'        64   Bad file name
'        68   Device unavailable
'        70   Permission denied
'        71   Disk not ready
'        74   Can't rename with different drive
'        75   Path/File access error
'        76   Path not found
'        91   Object variable set to Nothing
'        93   Invalid pattern
'        94   Illegal use of NULL
'       102   Command failed
'       429   Object creation failed
'       438   No such property or method
'       439   Argument type mismatch
'       440   Object error
'       901   Input buffer would be larger than 64K
'       902   Operating system error
'       903   External procedure not found
'       904   Global variable type mismatch
'       905   User-defined type mismatch
'       906   External procedure interface mismatch
'       907   Pushbutton required
'       908   Module has no MAIN
'       910   Dialog box not declared
```

有关这个 DDPerform _ Action 例子的注意事项：

首先，记住单引号标记了注释的开始。

续行格式是下划线、空格，然后新行。

And 字符（&）用作连接字符。

此版本 DDPerform _ Action 的基本格式是程序——Sub Main 直到 End Sub——后面是一个函数——Function Perform _ Action（ActCd）直到 End Function——后面是另一个函数——Function Execute _ Pref _ Command 直到 End Function。此程序的方便之处在于不用从 DDMain 中调用就可进行代码调试。

既然在 Robot 中，这是一个程序或脚本，那它就能在单独模式下执行。在一般操作中，DDMain 获取所有 DDPerform _ Action 需要的信息，然后调用它。为了调试，我们可以在从本地 sub main 调用 Perform _ Action 函数之前初始化一些它需要的值。注意调试代码区域行——虽然接下来的几行目前标记为注释，但如果去掉注释它们就会成为可执行代码。它们会初始化 ActCd 和 TabCd 变量，调整 dfld（）的大小，然后模拟来自外部文件的输入数据记录，用一些值初始化这个数组。然后我们可以查看与 ActCd 和 TabCd 值相关的代码，确保 dfld（）值确实作了应该做的事情。调试结束后，我们可以将调试代码行用注释符注释起来，直到下次测试不同的用例时再用不同的值。

一点有关 AUT 中目的窗口的背景知识：你可能猜到了，从菜单栏中激活的窗口叫做“Preferences”。它包括几个（7 个）不同的标签页，每个有许多不同的控件。因为这是工作在客户端的产品代码的复制，所以所有的

标签、注释和实际的标签页名字都改了，这样才能被包括在这里。

注意，在这个特殊例子中，标签页代码用来决定哪段代码应该先被执行（基于哪个标签页被选择）；然后请输入数据，在目的标签页上选择预期的对象。这个窗口上有三个共同按钮：Apply、Cancel、OK。这些按钮对窗体上所有七个标签页来说都通用，所以把代码放在一个当数据输入后就会被调用的独立函数中就更有意义，这些代码执行所有可能的动作，这些可能的动作，包括 PREF _ APPLY、PREF _ CANCEL、PREF _ OK 和纯粹输入数据的动作（PREF _ DataEntryOnly）。为了能不与任何特定标签页交互，只执行命令，我们也提供了 Command _ Only 动作的 TabCd 代码（参见最后一个用例声明）。

```
'DDPerform_Action Version 2.0

'Copyright Archer Group, 2001 All Rights Reserved

'This is a template file updated May 24, 2001
'updated June 2001

'$Include: "DataDriven.sbh"
'project specific include files

'$Include: "MyApp_global.sbh"

Option Compare text                                  'allows LIKE to be case insensitive

Declare Function Perform_Action(ActCd)
Declare Function Execute_Pref_Command

Dim Result as Integer

Sub Main
    Dim Result As Integer

    'Initially Recorded: 6/5/2001  1:26:55 PM
    'Script Name: DDPerform_Action

'Debug code area

'    ActCd="PREF_DataEntryOnly"
'    TabCd="TAB-1"
'    redim dfld(14)
'    dfld(1)="<None>"
'    dfld(2)="Open"
'    dfld(3)="Some Value"
'    dfld(4)="Other Data"
'    dfld(5)="5 little bytes"
'    dfld(6)="6 little bytes "
'    dfld(7)="7 little bytes "
'    dfld(8)="8 little bytes "
'    dfld(9)="9 little bytes "
'    dfld(10)="10 little bytes "
'    dfld(11)="11 little bytes "
'    dfld(12)="12 little bytes "
```

```
'    dfld(13)="13 little bytes "
     dfld(14)="14 little bytes "

    Gen_Return_Code = Perform_Action(ActCd)
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function Perform_Action(ActCd)

    Perform_Action=1  'set the initial return value from this function to PASS

    Select Case (TabCd)

        Case "TAB-1"

            'Input test data

            Window SetContext, "Caption=Preferences", ""

            'first control
            if (dfld(1)="1" and ShowTime="False") or (dfld(1)="0" and _
                                        ShowTime="True") then
                CheckBox Click, "Text=Time"
            end if

            if (dfld(2)="1" and ShowDate="False") or (dfld(2)="0" and _
                                        ShowDate="True") then
                CheckBox Click, "Text=Date"
            end if

            if (dfld(3)="1" and ShowKeyboard="False") or (dfld(3)="0" and _
                                        ShowKeyboard="True") then
                CheckBoxClick, "Text=Keyboard Indicators"
            end if

            if (dfld(4)="1" and ShowRecCount="False") or (dfld(4)="0" and _
                                        ShowRecCount="True") then
                CheckBox Click, "Text=Selected Record Count"
            end if

            '5th control          ComboBox Click, "ObjectIndex=1", "Coords=196,7"

            ComboListBox Click, "ObjectIndex=1", "Text="&Cstr(Dfld(5))

            '6th control
            EditBox dblClick, "Label=control 6", "Coords=16,10"
            InputKeys Dfld(6)

            'Execute the command
            Execute_Pref_Command

        Case  "TAB-2"

            '1st control

            Window SetContext, "Caption=Preferences", ""
            EditBox DblClick, "Label=control 1:", ""
            InputKeys Dfld(1)
```

```
'2nd control
ComboBox Click, "Label=Control 2:", "Coords=204,9"
ComboListBox Click, "Label=Control 2:", "Text="&Cstr(Dfld(2))

'3rd control
ComboBox Click, "Label= Control 3:", "Coords=204,11"
ComboListBox Click, "Label= Control 3:", "Text="&Cstr(Dfld(3))

'4th control
EditBox DblClick, "Label=Control 4:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(4)

'5th control
EditBox DblClick, "Label=Control 5:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(5)

'6th control
EditBox DblClick, "ObjectIndex=10", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(6)

'7th control
EditBox DblClick, "Label=Control7:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(7)

'8th control
EditBox DblClick, "ObjectIndex=8", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(8)

'9th control
EditBox DblClick, "Label=Control 9:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(9)

'10th comtrol
EditBox DblClick, "Label=Control 10:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(10)

'11th Control
EditBox DblClick, "Label=Control 11:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(11)

'x
EditBox DblClick, "Label=XXXXXXXX:", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(12)

'y
EditBox DblClick, "ObjectIndex=3", ""
InputKeys "{HOME}+{END}"
InputKeys Dfld(13)

'z
```

```
        EditBox DblClick, "ObjectIndex=2", ""
        InputKeys "{HOME}+{END}"
        InputKeys Dfld(14)
        Execute_Pref_Command

    Case "TAB-3"

        Window SetContext, "Caption=Preferences", ""
        'A
        EditBox DblClick, "Label=A:", "Coords=18,8"
        InputKeys Dfld(1)
        'B
        EditBox DblClick, "Label=B:", "Coords=19,4"
        InputKeys Dfld(2)
        'C
        EditBox DblClick, "Label=C", "Coords=19,10"
        InputKeys Dfld(3)
        'D
        EditBox DblClick, "Label=D:", "Coords=21,7"
        InputKeys Dfld(4)
        'E
        EditBox DblClick, "Label=E:", "Coords=46,9"
        InputKeys Dfld(5)
        'F
        InputKeys "{TAB}"
        InputKeys "{HOME}+{END}"
        InputKeys Dfld(6)
        'G
        EditBox DblClick, "Label=Width:", "Coords=42,6"
        InputKeys "{HOME}+{END}"
        InputKeys Dfld(7)
        'H
        InputKeys "{TAB}"
        InputKeys "{HOME}+{END}"
        InputKeys Dfld(8)
        'comments
        InputKeys "{TAB}"
        InputKeys "{HOME}+{END}"
        InputKeys Dfld(9)

        Execute_Pref_Command

        Case "TAB-4"

            'Tab 4 stuff

            Window SetContext, "Caption=Preferences", ""
            EditBox DblClick, "Label=DP M:", "Coords=52,10"
            InputKeys Dfld(1)
            '
            ComboBox Click, "Label=D S M:", "Coords=168,9"
            ComboListBox Click, "Label=D S M:", "Text="&Cstr(Dfld(2))
            'Default Selection
            ComboBox Click, "Label=Default Decision:", "Coords=164,12"
            ComboListBox Click, "Label=Default Decision:", "Text="&Cstr(Dfld(3))
            '
            InputKeys "{TAB}"
            InputKeys Dfld(4)
```

```
            Execute_Pref_Command

        Case "TAB-5"
            Window SetContext, "Caption=Preferences", ""
            'Sxx
            EditBox DblClick, "Label=Sxx:", "Coords=42,8"
            InputKeys Dfld(1)
            'Dyy
            ComboBox Click, "Label=Dyy:", "Coords=168,9"
            ComboListBox Click, "Label=Dyy:", "Text="&Cstr(Dfld(2))
            'Z12
            ComboBox Click, "Label= Z12:", "Coords=165,13"
            ComboListBox Click, "Label= Z12:", "Text="&Cstr(Dfld(3))
            'Default Decision
            InputKeys "{TAB}"
            InputKeys Dfld(4)
            Execute_Pref_Command

        Case "TAB-6"
            Window SetContext, "Caption=Preferences", ""
            TabControl DblClick, "ObjectIndex=1;\;ItemText=TAB-6", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-6", ""
            InputKeys "{TAB}"
            InputKeys "{HOME}+{END}{DELETE}"
            InputKeys Dfld(1)
            Execute_Pref_Command

        Case "TAB-7"
            Window SetContext, "Caption=Preferences", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-7", ""

            InputKeys "{TAB}"
            Inputkeys Dfld(1)
            Execute_Pref_Command

        Case "Command_Only"
            Execute_Pref_Command

        Case Else
            Msgbox "Action Selection not found for ActCd: "&Cstr(ActCd)&" Terminating"
            Perform_Action=0  ''return value FAIL
            SQALogMessage sqaFail, "Action Selection not found for: "&Cstr(ActCd)&"", ""
    End Select

End Function

------------------------------------------------------------------------------------------
Function Execute_Pref_Command
        If ActCd="PREF_OK" then
            Window SetContext, "Caption=Preferences", ""
            PushButton Click, "Text=OK"

            if ErrCd LIKE "none" then
                Result = WindowVP (DoesNotExist, "Caption=Preferences",_
                            "VP=Window Existence;Wait=2,30;Status=NORMAL")
                Current_Win="none"
                Current_Tab="none"
            end if
        elseif ActCd="PREF_APPLY" then
```

```
                Window SetContext, "Caption=Preferences", ""
                PushButton Click, "Text=Apply"
                if ErrCd LIKE "none" then
                    Result = WindowVP (Exists, "Caption=Preferences",_
                                 "VP=Window Existence;Wait=2,30;Status=NORMAL")
                end if
            elseif ActCd="PREF_CANCEL" then
                Window SetContext, "Caption=Preferences", ""
                PushButton Click, "Text=Cancel"
                if ErrCd LIKE "none" then
                    Result = WindowVP (DoesNotExist, "Caption=Preferences",_
                                 "VP=Window Existence;Wait=2,30;Status=NORMAL")
                    Current_Win="none"
                    Current_Tab="none"
                end if
            elseif ActCd="PREF_DataEntryOnly" then
                'do nothing
            end if
End function
```

如果愿意，你也可以像以下代码段一样在 case 语句中选择组合值。

```
Function Perform_Action(ActCd)
    Perform_Action=1
    Select Case (ActCd & TabCd)
        Case "PREF_OK"&"TAB-1"

--------------------------------------------------------------------------------
 DDProcess_Error  Version 2.0

 This is a template file updated May 24, 2001
 updated June 2001

'$Include: "DataDriven.sbh"

    Declare Function Process_Error(ErrCd)

Sub Main
    Dim Result As Integer

    'Initially Recorded: 6/5/2001  1:53:10 PM

    'Script Name: DDProcess_Error
    Gen_Return_Code=Process_Error(ErrCd)
End Sub

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Function Process_Error(ErrCd)

    Process_Error=1

    Select Case ErrCd

        Case "PREF_GEN_ERR"
```

```
            Result = WindowVP (Exists, "Caption=Validation Errors", "VP=Window_
                            Existence;Wait=2,30;Status=NORMAL")
            Window SetTestContext, "Caption=Validation Errors", ""
            Result = LabelVP (CompareText, "ObjectIndex=1", _
                        "VP=Alphanumeric;Wait=2,30;Type=CaseSensitive")
            Window ResetTestContext, "", ""
            Window SetContext, "Caption=Validation Errors", ""
            PushButton Click, "Text=OK"

        Case "x"            ' a place holder for the next case to be inserted

        Case Else
            Msgbox "Error Code not found for ErrCd: "&Cstr(ErrCd)&" Terminating"
            Process_Error=0  ''return value FAIL
            SQALogMessage sqaFail, "Error Code not found for: "&Cstr(ErrCd)&"", ""
    End Select
End Function

'DDTab_Select  Version 2.0

'This is a template file updated May 24, 2001
'updated June 2001

'$Include: "DataDriven.sbh"

Declare Function Tab_Select(TabCd)
Dim Result

Sub Main
    Dim Result As Integer
    'Initially Recorded: 6/5/2001  1:16:14 PM
    'Script Name: DDTab_Select
    Gen_Return_Code = Tab_Select(TabCd)
End Sub
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
Function Tab_Select(TabCd)
    tab_Select=1 'initially set the return code to "Success"
    Select Case TabCd

'preferences tabs

        Case "TAB-1"
            Window SetContext, "Caption=Preferences", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-1", ""

                Result = TabControlVP (CompareProperties, "ObjectIndex=1", "VP=OP TAB-
                            1;Wait=2,30")
                Window ResetTestContext, "", ""

            Case "TAB-2"
                Window SetContext, "Caption=Preferences", ""
                TabControl Click, "ObjectIndex=1;\;ItemText=TAB-2", ""
                Result = TabControlVP (CompareProperties, "ObjectIndex=1", "VP=OP
                            YTD;Wait=2,30")
                Window ResetTestContext, "", ""

            Case "TAB-3"
                Window SetContext, "Caption=Preferences", ""
```

```
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-3", ""
            Result = TabControlVP (CompareProperties, "ObjectIndex=1", _
                       "VP=OP NSI;Wait=2,30")
            Window ResetTestContext, "", ""

        Case "TAB-4"
            Window SetContext, "Caption=Preferences", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-4", ""
            Result = TabControlVP (CompareProperties, "ObjectIndex=1",_
                        "VP=OP NFD;Wait=2,30")
            Window ResetTestContext, "", ""

        Case "TAB-5"
            Window SetContext, "Caption=Preferences", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-5", ""
            Result = TabControlVP (CompareProperties, "ObjectIndex=1", _
                       "VP=OP ICB;Wait=2,30")
            Window ResetTestContext, "", ""

        Case "TAB-6"
            Window SetContext, "Caption=Preferences", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-6", ""

            Result = TabControlVP (CompareProperties, "ObjectIndex=1",_
                       "VP=OP TAB-A;Wait=2,30")
            Window ResetTestContext, "", ""

        Case "TAB-7"
            Window SetContext, "Caption=Preferences", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=TAB-7", ""
            Result = TabControlVP (CompareProperties, "ObjectIndex=1",_
                       "VP=OP MAP;Wait=2,30")
            Window ResetTestContext, "", ""
'add new set tabs

        Case "set information"
            Window SetContext, "Caption=MyApp Shell", ""
            Window SetContext, "Caption=Add Set;ChildWindow", ""
            TabControl Click, "ObjectIndex=1;\;ItemText=Set Information", ""
            Window SetTestContext, "Caption=MyApp Shell", ""
            Window SetTestContext, "Caption=Add  Set;ChildWindow", ""
            Result = TabControlVP (CompareProperties, "ObjectIndex=1",_
                       "VP=OP Set Info;Wait=2,30")
            Window ResetTestContext, "", ""

        Case "optional information"
            Window SetContext, "Caption=MyApp Shell", ""

      Window SetContext, "Caption=Add Set;ChildWindow", ""
      TabControl Click, "ObjectIndex=1;\;ItemText=Optional Information", ""
      Window SetTestContext, "Caption=MyApp Shell", ""
      Window SetTestContext, "Caption=Add Set;ChildWindow", ""
      Result = TabControlVP (CompareProperties, "ObjectIndex=1",_
                 "VP=OP Optional Info;Wait=2,30")
      Window ResetTestContext, "", ""

    Case Else
        Msgbox "Tab Select not found for TabCd: "&Cstr(TabCd)&" Terminating"
        Tab_Select=0  ''return value FAIL
        LogMessage sqaFail, "Tab Select not found for tab code: "&Cstr(tabCd)&"", ""
End Select
```

```
End Function
-------------------------------------------------------------------------------
'DDWindow_Select  Version 2.0

'$Include: "DataDriven.sbh"

    Declare Function Window_Select(WinCd)
    Dim result as integer

Sub Main
    'Initially Recorded: 5/31/2001  1:49:50 PM
    'Script Name: DDWindow_Select
    Gen_Return_Code = Window_Select(WinCd)
End Sub

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Function Window_Select(WinCd)
    Window_Select=1
    Select Case WinCd

        Case "preferences"
            Window SetContext, "Caption={MyApp*}", ""
            MenuSelect "Tools->Preferences..."
            Result = WindowVP (Exists, "Caption=Preferences", "VP=Window
                        Existence;Wait=2,30;Status=NORMAL")
            Window SetContext, "Caption=Preferences", ""
            Window MoveTo, "", "Coords=0,0"

        Case "CPWS_N"
            Window SetContext, "Caption={MyApp*}", ""
            MenuSelect "Action->Create PlanningWorkSheet->N"
            Window SetTestContext, "Caption={MyApp*}", ""
            Result = WindowVP (Exists, "Caption={N PWS*};ChildWindow", "VP=Window
                        Existence;Wait=2,30;Status=NORMAL")
            Window ResetTestContext, "", ""
            Window SetContext, "Caption={Ny PWS*};ChildWindow", ""
            Window WMaximize, "", ""

        Case "CPWS_Y"
            Window SetContext, "Caption={MyApp*}", ""
            MenuSelect "Action->Create PlanningWorkSheet->Y"
            Window SetTestContext, "Caption={MyApp*}", ""
            Result = WindowVP (Exists, "Caption={Yield PWS*};ChildWindow", "VP=Window
                        Existence;Wait=2,30;Status=NORMAL")

            Window ResetTestContext, "", ""
            Window SetContext, "Caption={Y PWS*};ChildWindow", ""
            Window WMaximize, "", ""

        Case "AddNSet"
            Window SetContext, "Caption={MyApp*}", ""
            Window SetContext, "Caption={N PWS*};ChildWindow", ""
            Window WMaximize, "", ""
            Window SetContext, "Caption={{MyApp*}", ""
            MenuSelect "PWS->Add N Set..."
            Window SetTestContext, "Caption={MyApp*}", ""
            Result = WindowVP (Exists, "Caption=Add N Set;ChildWindow",_
```

```
                    "VP=Window Existence;Wait=2,30;Status=MAXIMIZED")
        Window ResetTestContext, "", ""

    Case "AddYSet"
        Window SetContext, "Caption={MyApp*}", ""
        Window SetContext, "Caption={Y PWS*};ChildWindow", ""
        Window WMaximize, "", ""
        Window SetContext, "Caption={MyApp*}", ""
        MenuSelect "PWS->Add Y Set..."
        Window SetTestContext, "Caption={MyApp*}", ""
        Result = WindowVP (Exists, "Caption=Add Yield Trial Set;ChildWindow",_
                    "VP=Window Existence;Wait=2,30;Status=MAXIMIZED")
        Window ResetTestContext, "", ""

    Case ""

    Case Else
        Msgbox "Window Select not found for WinCd: "&Cstr(WinCd)&" Terminating"
        Window_Select=0  ''return value FAIL
        SQALogMessage sqaFail, "Window Select not found for: "&Cstr(WinCd)&"", ""
  End Select
End Function
-----------------------------------------------------------------------------------------
```

8.16 SQABasic32 包含文件

```
DataDriven.sbh   version 2.0

'Global definitions for the Data Driven Main, Window Select, Tab Select, Perform Action and
                         Process Error functions

'This is a template file created May 24, 2001

    Global Dfld()
    Global WinCd        As String
    Global TabCd        As String
    Global ActCd        As String
    Global ErrCd        As String
    Global Current_Win As String
    Global Current_Tab As String
    Global InFileName  As String
    Global Gen_Return_Code As Integer
    Global Gen_Return_String As String
Global Key_word_list(9,1) as String  'the size of this list should match the number of
                         keywords defined
CONST KWListSize = 9

DBase_Util.sbl version 2.0

'update the DB_Num constant as database connections are added
Const DB_Num = 4

Global Available_Databases(4) as String
Global Available_Connect_Strings(4) as String

Sub DB_Util
'For each additional database bump from Available_Databases(1) to Available_Databases(2),
                         etc. supply a name and a connect string
```

```
Available_Databases(0)="(none)"
Available_Connect_Strings(0)="none"

Available_Databases(1)="Local MS Access Test Database"

Available_Connect_Strings(1)="DSN=TestData;DBQ=C:\Local_Repo\TestData\testdata.mdb;DriverId=
                        281;FIL=MS
                        Access;MaxBufferSize=2048;PageTimeout=5;PWD=saturn51;UID=admin;"

Available_Databases(2)="MYAPP xrem" 'description only

Available_Connect_Strings(2)="DSN=xREM;UID=baposey;PWD=xxxxx;DBQ=mrem.abccompany.com;DBA=W;A
                        PA=T;FEN=T;FRC=10;FDL=10;LOB=T;RST=T;FRL=F;PFC=10;TLO=0;"
Available_Databases(3)="XREM - MyApp" 'description only

Available_Connect_Strings(3)="DSN=DREM;UID=meeeeeee;PWD=xxxxx;DBQ=xrem.abccompany.com;DBA=W;
                        APA=T;FEN=T;FRC=10;FDL=10;LOB=T;RST=T;FRL=F;PFC=10;TLO=0;"
Available_Databases(3)="MSDE Local - MyApp" 'description only
Available_Connect_Strings(3)="DSN=MyAppMSDE;Description=MyApp
                        local;UID=ZXZXZXZ;PWD=zxzxzx;APP=Rational Test;WSID=MYAPP02"
End Sub

--------------------------------------------------------------------------------------------
excel.sbh   version 2.0

Declare Function ReadExcelDataSingle BasicLib "excel" (sFileName As String, sCell As String)
                        As String
Declare Function ReadExcelDataMulti BasicLib "excel" (sFileName As String, sCells As String,
                        sData() As String) As Integer
Declare Sub WriteExcelData BasicLib "excel" (sFileName As String, sCell As String, vValue As
                        Variant)


'excel.sbl v3.0--Andy Tinkham, CWC Inc.
'Permission to redistribute has been granted to Dan Mosley of CSST Technologies, Inc., and
                        Bruce Posey of Archer Group.

'*****************************************************************************
'* ReadExcelDataSingle
'*
'* Retrieves the value from a cell on the first sheet in a specified Excel
'* workbook and returns it as a string.
'*
    '* Parameters:
    '*      sFileName -- full path and filename of the spreadsheet to open
    '*      sCell -- cell column/row designator (e.g., "A1")
    '*      vSheet -- optional parameter.  Use this to specify either an integer
    '*          sheet number (where first sheet is numbered 1 not 0) or a string
    '*          sheet name in order to reference the sheet to retrieve data from
    '*
    '* Note:  This will error if the specified sheet of a workbook is not a data sheet
    '*
    '*****************************************************************************
    Function ReadExcelDataSingle(sFileName As String, sCell As String, _
        Optional vSheet As Variant) As String

        'Dimension the needed variables
        Dim objExcel As Object
        Dim objWorkBook As Object

        Dim objWorkSheet As Object
```

```
    Dim sData As String

    'Open up Excel
    Set objExcel = CreateObject("Excel.Application")

    'Open the workbook
    Set objWorkBook = objExcel.Workbooks.Open(FileName:=sFileName)

    'See if a sheet was referenced in the call.  If not, default to the first sheet
    If IsMissing(vSheet) Then
        vSheet = 1
    End If

    'Set up a reference to the sheet in the workbook
    Set objWorkSheet = objWorkBook.WorkSheets(vSheet)

    'Make everything uppercase to ease comparision
    sCell = UCase(sCell)

    'Retrieve the value from the cell
    sData = objWorkSheet.Range(sCell).Value

    'Close the workbook
    objWorkBook.Close

    'Exit Excel
    objExcel.Quit

    'Clear all the references to the objects
    Set objWorkBook = Nothing
    Set objWorkSheet = Nothing
    Set objExcel = Nothing

    'Return the cell value
    ReadExcelDataSingle = sData
End Function
'******************************************************************************
'* ReadExcelData
'*
'* Retrieves the values from a cell or group of cells on a sheet in a specified
'* Excel workbook.  This function fills in an array that is passed in as an
'* argument.  The return value is TRUE or FALSE and should be checked before the
'* array is accessed to make sure that the data was correctly loaded.
'*
'* Parameters:
'*      sFileName -- full path and filename of the spreadsheet to open
'*      sData -- array to hold the values of the cells.  Should be an undimensioned
'*          dynamic array (e.g., one declared like Dim arrExcelValues() As String)
'*      vSheet -- optional sheet name to retrieve data from.  If omitted, the first
'*          sheet of the workbook is used
'*      vRange -- optional range column/row designator (e.g., "A1:B16").  If omitted,
'*          the first table of data is returned (passing a range of "all" also will
'*          return this same table).  A table of data is defined as the
'*          block of cells contiguous to a cell without encountering any blank rows
'*          or columns (blank cells are ok).  The cell used for contiguousness is A1
'*          unless a cell is specified in the sCell parameter
'*      vCell -- optional parameter used to specify the cell used to determine the
```

```
'*          contiguous table.  Ignored if sRange is not omitted or set to "all".  This
'*          parameter must refer to a cell that is in the contiguous range you want
'*          to retrieve, but it can be any cell in that contiguous range
'*
'* Note:  This will error if the specified sheet of a workbook is not a data sheet
'*
'******************************************************************************
Function ReadExcelData(sFileName As String, sData() As String, Optional vSheet _
    As Variant, Optional vRange As Variant, Optional vCell As Variant) As Integer

    'Dimension the needed variables
    Dim objExcel As Object
    Dim objWorkBook As Object
    Dim objWorkSheet As Object
    Dim objRange As Object
    Dim i As Integer
    Dim j As Integer

    'Deal with the optional parameters first
    If IsMissing (vSheet) Then
        vSheet = 1
    End If

    If IsMissing (vRange) Then
        vRange = "all"
    End If

    If IsMissing (vCell) Then
        vCell = "A1"
    End If

    'Open up Excel
    Set objExcel = CreateObject("Excel.Application")

    'Open the workbook
    Set objWorkBook = objExcel.Workbooks.Open(FileName:=sFileName)


    'Set up a reference to the first sheet in the workbook
    Set objWorkSheet = objWorkBook.WorkSheets(vSheet)

    'Set a reference to the specified or default Range of data
    If LCase(vRange) = "all" Then
        Set objRange = objWorkSheet.Range(vCell).CurrentRegion

    Else
        Set objRange = objWorkSheet.Range(vRange)
    End If

    'Redimension the array to hold the data
    ReDim sData(1 to objRange.Rows.Count, 1 to objRange.Columns.Count) As String

    'Read in the data cell by cell (since SQA won't accept an array as a return value)
    For i = 1 to objRange.Rows.Count
        For j = 1 to objRange.Columns.Count
            sData(i,j) = objRange.Rows(i).Columns(j).Value
        Next j
    Next i

    'Close the workbook
    objWorkBook.Close
```

```
    'Exit Excel
    objExcel.Quit

    'Clear all the references to the objects
    Set objWorkBook = Nothing
    Set objWorkSheet = Nothing
    Set objExcel = Nothing

    'Return TRUE to indicate things worked correctly
    ReadExcelData = TRUE
End Function

'*****************************************************************************
'* WriteExcelDataSingle
'*
'* Writes a value to a cell on a specified sheet in a specified Excel
'* workbook.
'*
'* Parameters:
'*      sFileName -- full path and filename of the spreadsheet to open
'*      sCell -- cell column/row designator (e.g., "A1")
'*      vValue -- value to write out
'*      vSheet -- optional parameter used to specify the sheet to write
'*          to.  Can either be an integer specifying the sheet's index or
'*          a string specifying the sheet's name
'*
'* Note:  This will error if the specified sheet of a cell is not a data sheet
'*
'*****************************************************************************
Sub WriteExcelDataSingle(sFileName As String, sCell As String, vValue As Variant, _
    Optional vSheet As Variant)

    'Dimension the needed variables
    Dim objExcel As Object
    Dim objWorkBook As Object
    Dim objWorkSheet As Object

    'Deal with the optional parameters
    If IsMissing(vSheet) Then
        vSheet = 1
    End If

         'Open up Excel
         Set objExcel = CreateObject("Excel.Application")

         'Open the workbook
         Set objWorkBook = objExcel.Workbooks.Open(FileName:=sFileName)

         'Set up a reference to the first sheet in the workbook
         Set objWorkSheet = objWorkBook.WorkSheets(vSheet)

         'Write the value to the cell
         objWorkSheet.Range(sCell).Value = vValue

         'Save the changes
         objWorkBook.Save

         'Close the workbook
```

```
    objWorkBook.Close

    'Exit Excel
    objExcel.Quit

    'Clear all the references to the objects
    Set objWorkBook = Nothing
    Set objWorkSheet = Nothing
    Set objExcel = Nothing
End Sub

'******************************************************************************
'* WriteExcelData
'*
'* Writes an array of values to a range on a specified sheet in a specified
'* Excel workbook.
'*
'* Parameters:
'*      sFileName -- full path and filename of the spreadsheet to open
'*      sCell -- cell column/row designator (e.g., "A1"). This should be
'*          the upperleftmost cell of the range you want to write out. Values
'*          will then be written out in the order they are in in the array.
'*      vValues -- values to write out. Must be a variant two-dimensional array
'*      vSheet -- optional parameter used to specify the sheet to write
'*          to. Can either be an integer specifying the sheet's index or
'*          a string specifying the sheet's name
'*
'* Note: This will error if the specified sheet of a cell is not a data sheet
'*
'******************************************************************************
Sub WriteExcelData(sFileName As String, sCell As String, vValues() As Variant, _
    Optional vSheet As Variant)

    'Dimension the needed variables
    Dim objExcel As Object
    Dim objWorkBook As Object
    Dim objWorkSheet As Object
    Dim objRange As Object
    Dim i As Integer
```