AutoRunner 泽众自动测试引擎软件

技术白皮书

Version4.0

上海泽众软件科技有限公司

2018 年 1 月

目录

1.总述

1.1 背景

随着软件规模的发展和对软件系统的依赖，人们发现：软件的质量对应用系统的影响日益增加，质量存在问题的软件会导致帐务出错，客户信息丢失，用户的服务出错。因此，提高软件的质量成为一个重要的问题。而测试正是提高软件质量的有效手段。数据显示，在一个软件开发过程中，测试占到整个工作的 40%－60%。所以，如何能够在较低成本的情况下大幅度提高测试的质量，对软件的最终质量起到非常重要的作用。

另一方面，当应用软件投入使用之后，随着应用的不断发展和变化，将会提出大量的新增需求。新功能对用户非常重要，能够给用户不断发展的业务提供更强大的支撑。当开发人员修改软件的功能、增加软件功能，新增功能部分导致原有系统运行不够稳定的几率必然增加，可靠性降低：由于修改一个小错误造成大量业务无法正常运行的情况。这就需要做大量的回归测试来保证系统的可靠性，通过回归测试验证以往的功能是正确的、可靠的。传统的回归测试是由人工来实现的，需要大量的人员来完成固定的输入和输出检查。

人工测试带来了一系列问题：

1、测试质量难以保证。临时参加测试的人员很多都是没有受到足够培训的人员，对应用软件本身的理解不够充分，对测试技术也不够了解，往往导致测试案例设计不够严密，测试的质量难以得到保证。

2、测试的成本很高。组织一次大规模的回归测试会导致大量成本发生：包括人工成本和管理成本等等。首先是测试人员自身的成本，然后是由于测试进度比较缓慢造成的开发人员延误造成的成本，此外，管理测试人员，协调测试和开发人员之间的关系也需要很多的工作和成本。

3、测试进度难以控制。由于人员、系统稳定性等众多方面的原因，导致测试的进度比较难以度量和控制，造成项目风险防范困难。

4、测试案例无法保存和管理。作为一个项目，测试案例是一个重要的财富：同软件代码一样，是具有版本和价值的。传统的做法是使用文档来保存测试案例，无法提供一种标准格式来保存测试案例。这样就会导致测试案例的存放非常困难，使用非常困难，

造成了资源的浪费：但需要再做一次回归测试的时候，往往原来的案例都被丢弃了，还需要重新设计和完成测试案例。

1.2 解决方案

企业可以建立一整套软件自动测试体系，包括：需求管理、测试分析、测试管理、缺陷跟踪，并且把这个过程纳入整个软件项目开发和软件产品开发过程。

实际上，在 CMM 的规范中，测试本身就是 SQA 的一部分。自动测试的基础就在于测试工具，只有采用了优秀的自动测试软件，才能够解决自动测试的问题。自动测试工具能够在两个阶段给软件开发企业带来价值：第一，对于软件开发人员来说，高效率的自动测试工具能够提供给程序员自己完成开发过程中的冒烟测试，便于在频繁修改的软件过程中迅速完成测试，保证编码的稳定性；第二，对于项目和产品的测试阶段来说，能够提供稳定的回归测试，保证产品的可靠性。众所周知，在测试阶段发现问题的投入，相对与在软件投产之后出现错误再去解决问题要小的多。

上海泽众软件科技有限公司开发出了国内第一个拥有自主知识产权的自动测试软件——自动测试引擎(AutoRunner)，能够帮助用户实现自动化测试。

1.3 概述

1、本技术白皮书适用于上海泽众软件科技有限公司自动测试工具（AutoRunner)。

2、本技术白皮书是上海泽众软件科技有限公司自动测试工具（AutoRunner）的技术说明，也是技术谈判的主要内容，是采购方询价、系统选型以及系统测试和验收的主要技术依据。

3、本技术白皮书是根据信息产业部颁布的有关技术体制和技术政策并结合上海泽众软件科技有限公司的实际情况制定的。本技术白皮书没有提出而信息产业部的技术体制以及技术标准已有具体规定的内容，应按信息产业部的技术体制以及技术标准执行，如果存在不一致应以信息产业部颁布的最新技术体制及技术标准内容为准。

4、本技术白皮书在内容或技术指标上如果存在错误（包括印刷错误），经双方确认后可对该错误内容或技术指标进行修正。

5、自动测试工具（AutoRunner）版本升级之后，上海泽众软件科技有限公司有权对

本技术白皮书进行修改，并不需要主动通知用户。

6、本技术白皮书以下内容为用户重点考察内容：

软件的功能、性能、技术指标和环境要求；

设备容量计算和配置方法；

所提供的数据库的功能和性能指标；

软件安装要求；

提供软件的接口、协议等工程技术要求；

乙方供货范围、交货能力和时间、运输、安装、调测验收和培训等项内容的日程安排；

其他有关技术资料。

7、本软件对涉及专利、知识产权等法律条款承担有限责任。

8、本技术白皮书提供了对上海泽众软件科技有限公司的自动测试工具（AutoRunner）的相关技术描述，由于用户使用造成损失，上海泽众软件科技有限公司不承担责任。

9、本技术白皮书以中文编写，未经上海泽众软件科技有限公司同意或授权的其它语言或形式的技术白皮书无效。本技术规范书的解释权归上海泽众软件科技有限公司。

2.系统概述

2.1 系统定位

AutoRunner 是一个自动测试工具的集合，也是一个自动测试框架，加载不同的测试组件，就能够实现面向不同应用的测试。AutoRunner 支持浏览器测试和其他各种技术平台的 AUT（applicationunder test），包括：浏览器（IE）、java、win32、silverlight、flex、.NET 等。

2.2 自动功能测试工具的概念

- **测试脚本**

自动测试，就是使用一个程序来测试另一个程序（被测试的应用系统）功能的正确性。如果用来测试的程序本身非常复杂，也需要被测试，或者编写困难，那么自动测试

就失去了意义。

因此，用来测试另外一个程序的程序往往是非常简单的，我们把这个程序称为“测试脚本”。测试脚本通常在测试工具的 IDE 里执行，并且获得 IDE 的支持。

- **自动记录**

当我们编写测试脚本的时候，往往发现编写脚本本身是很困难的：了解脚本的语法、了解测试过程、把测试过程转换称为测试脚本语句。自动记录，就是通过记录一个操作过程来获得测试脚本的功能。通过自动记录，我们就能够得到一个操作的基本的脚本，通过修改这个脚本，我们得到更通用的测试脚本。

- **同步点**

在执行测试脚本的时候，测试脚本语句的操作对象是 GUI 的组件。测试脚本通过这个组件的属性（如：名称、位置、winclass、disable 等）来确定那个组件是我们需要操作的组件。

这个查找组件的过程如果失败，意味着：

第一，应用系统的响应比较慢，

需要等待一段时间再进行一次定位；或者第二，该组件不存在。这个查找、定位组件的过程，我们称为同步点。

AutoRunner 的同步点都是隐含方式的：在操作对象的时候进行自动同步，自动设置同步点。

- **检查点**

测试的目的是检查数据是否正确。在测试的过程中，我们需要检查某个组件的某些属性满足某个条件。这个检查的位置和条件，我们称为检查点。在 AutoRunner 中，使用 check（“objectname”， “property”， “期望值”）来作为检查点的脚本语句，它检查对象 objectname 的属性 property 是否和期望值一致。

在使用使用中，可以使用检查点来检验系统的各个方面，如数据库、GUI 属性等。

- **参数化与数据驱动**

测试脚本是针对一个测试过程的。一个测试过程往往需要众多的数据来测试。通过自动录制得到的脚本，所有的输入数据都是常数，是固定的。如果需要使用一个测试脚本测试多组数据，就需要对脚本进行参数化，把固定的常数修改为来自数据源变量。这个过程我们称为参数化。采用了参数化的脚本，我们称为数据驱动的模式。

2.3 业务提供

所谓业务提供，就是指使用本自动测试工具能够提供的功能。

2.3.1AutoRunner 适用性说明

AutoRunner 是一个产品家族，不是一个单个的产品。这个产品家族的特点是共同使用了一个标准的 IDE，并且使用相同的测试脚本语言。从脚本的角度上看，他们是完全相同的。所不同的是相同的 IDE 采用不同的测试 plugin 组件。我们提供不同的组件以满足不同的测试需求：

2.3.2 自动化的功能测试

AutoRunner 的基本功能，就是对软件进行功能测试。功能测试本身是面向需求的黑盒测试工具。

它以需求点为出发点，为了满足需求点（即需求），进行测试分析，得到测试案例。然后使用测试工具得到测试案例库（测试案例库包括测试脚本和案例数据），并且根据测试案例库对功能进行测试，得到被测试软件的错误报告和缺陷跟踪报告，进而反馈给软件开发人员，帮助他们确定问题，修改错误，提高软件的质量。

2.3.3 自动化的回归测试

由于软件开发是面向用户需求的，而用户需求也是不断变化的。修改软件会经常性的引入错误，根据统计，每修改 3 个错误可能会引入 1 个错误。虽然修改了很小的一部分，却存在引入巨大错误的风险。防范风险的手段就是回归测试。

手工回归测试往往需要大量的人力才能够实现，这就出现了：减少测试（降低了成本）就增加了风险；降低了风险（引入大量测试人员进行全面的回归测试）就会增加成本。采用 AutoRunner 的自动化测试工具就能够解决这个问题。

2.3.4 每日构建与冒烟测试

程序员往往通过单元测试来对他（她）所负责的部分进行测试。当测试完成后，又

需要进行集成测试（即几个模块组装在一起之后的测试）。单元测试是白盒测试，往往和最后的功能测试存在一定的差异。

目前，很多先进的做法（如微软）都采用每日构建和冒烟测试的方法，就是在每天程序员都需要提交自己的代码，并且构建一个版本进行测试，第二天把测试的结果反馈给开发者。每日构建和冒烟测试能够很大程度上提高软件的开发效率，并且对与 SQA 而言是增加了软件度量的指标。每日构建和冒烟测试必然要建立在自动测试工具的基础上，依靠人是无法在每天晚上完成一次完整的功能测试的。

2.3.5 版本升级测试

新软件开发完毕，即将发布的时候，用户非常关心：新的版本是否能够完成原来来版本的功能、是否和老版本功能兼容。重新测试一边老版本的所有功能是必要的，会提前发现版本兼容的问题、数据的问题等等。实现这个测试的基础就是自动测试功能，基于 AutoRunner 的测试案例能够在很短的时间之内完成一次测试，防止问题发生。

2.3.6 特性概述

AutoRunner 作为自动测试工具，采用最流行的 Java beanshell 脚本语言作为测试脚本，增加了测试人员对测试工具的接收程度，也能够在测试人员学习测试工具的过程中学会 java 的基本知识。

AutoRunner 具有优秀的录制功能，能够一次录制非常完善的脚本和资源，降低了测试人员修改脚本的工作量。对于测试过程中遭遇不断回放错误的测试人员来说，是非常有价值的。

强大的对象识别技术

在测试执行的过程中，由于版本不同，可能会导致各个版本之间的组件发生名称、位置、属性等方面的变化，从而导致上一次录制的脚本无法工作。AutoRunner 的对象识别技术，能够不以来于对象的位置，并且具有自动识别的功能，在无法精确定位组件的情况下，能够选中一个最可能的组件，使得脚本的更改下降到最低。

简便的脚本

对于使用 java 语言作为脚本，很多测试人员可能会担心过于复杂。实际上，所有的测试脚本都是继承一个标准的类 TestCase，并且使用它提供的基本方法，因此是非常

简单的，没有复杂的 java 成分，便于那些已经学习过其他测试工具的测试人员迁移到这个工具上来。

自动化的数据驱动

AutoRunner 提供了自动化的数据驱动功能：在录制脚本的时候，已经在脚本中实现了数据驱动。用户可以把不需要的数据驱动修改为常量。因此，测试人员不需要自己来编写复杂的数据驱动。另外 AutoRunner 还提供了一个数据驱动框架,便于测试人员使用。

关键字驱动

AutoRunner 实现了关键字驱动，IDE 提供关键字视图和专家视图（编辑测试脚本），不熟悉脚本的用户通过拖拽的方式也能够编辑测试脚本；经过编辑的测试脚本也能够被转换成关键字来查看和编辑。关键字驱动的引入，降低了自动测试的难度，使得不熟悉测试脚本的测试工程师也能够编写和维护测试脚本。

对象指示器

AutoRunner 的对象指示器能够在录制脚本的时刻，指示识别到的对象，并且保存此对象的图片，作为对象属性的一部分。在关键字视图上，能够查看每个语句关联的脚本；在对象浏览器上，可以在查看对象属性的同时显示对象的图片，便于用户理解当前对象的位置信息。

灵活的验证方式

案例执行的正确与否需要数据比对来验证。AutoRunner 提供了强大验证方式，用户可以非常简便的通过编写脚本来使用，如提供对字符串的正则表达式验证。

良好的扩展性

一般的脚本虽然很简便，但是对于特殊的测试，往往需要更复杂的功能，例如：需要对网络上的另一台系统中的数据库的某些数据进行同步。基本的 AutoRunner 不提供这个功能。由于 AutoRunner 使用了标准的 java（目前为最新的 JDK1.6）那么用户可以自己编写一个同步方法（或者类）加入到系统中来使用，只要是 java 已经提供的功能，都可以得到完善的支持。

标准化

AutoRunner 符合测试工具的基本要求，如：同步点、验证点、错误报告等，都遵守了国际化测试标准，便于用户理解和使用，也便于用户比较各个不同测试工具之间的差异。

2.4 产品设计目标

1.提高回归测试的覆盖率，提高测试质量。对于功能已经完整和成熟的软件，每次发布一个新的版本，其中大部分功能和界面都和上一个版本相似或完全相同，这部分功能特别适合于自动化测试，从而可以让测试达到测试每个特征的目的。通过 AutoRunner 来编写回归测试的测试案例，并且再每次发布版本的时候通过执行所有的测试案例来进行回归测试，能够覆盖大量的功能——人工测试无法进行测试的功能。

2.每日测试的高效率。DCC 版本的发布周期往往比较短，也就是开发周期只有短短的几个月，而在测试期间是每天/每 2 天都要发布一个版本供测试人员测试，一个系统的功能点有几千个上万个，人工测试是非常的耗时和繁琐，这样必然会使测试效率低下。AutoRunner 通过高效率的自动执行测试案例，允许每天对版本进行测试，提高测试效率。

3.具有一致性和可重复性。由于每次自动化测试运行的脚本是相同的,所以每次执行的测试具有一致性, 人是很难做到的. 由于自动化测试的一致性，很容易发现被测软件的任何改变。

4.更好的利用资源－－周末/晚上。理想的自动化测试能够按计划完全自动的运行,在开发人员和测试人员不可能实行三班倒的情况下, 自动化测试可以胜任这个任务, 完全可以在周末和晚上执行测试. 这样充分的利用了公司的资源,也避免了开发和测试之间的等待。

5.解决测试与开发之间的矛盾。通常在开发的末期,进入集成测试阶段,由于每次发布一个版本的初期,测试系统的错误比较少,这时开发人员有等待测试人员测试出错误的时间. 事实上在迭代周期很短的开发模式中，存在更多的矛盾， 但自动化测试可以解决其中的主要矛盾。

6.将烦琐的任务转化为自动化测试。大量重复的测试是非常繁琐的，并且需要消耗大量的人力才能够完成。自动测试能够很好的解决这个问题，不需要繁琐的劳动，不需要大量的人员。

7.增加软件信任度。只有经过大量测试案例测试过的版本才是可靠的，而只有使用自动测试才能够保证在段时间内完成大量的测试案例。

3.系统体系结构特性要求

3.1 系统要求

操作系统环境

WindowsXP

Windows2000

Windows2003

Windows7/Windows8

注：理论上对于安装了 jdk1.6 的 windows 系统都提供支持。

系统要求

JDK1.5

IE5.5 以上（针对 IE 的 plugin）

测试案例数据格式

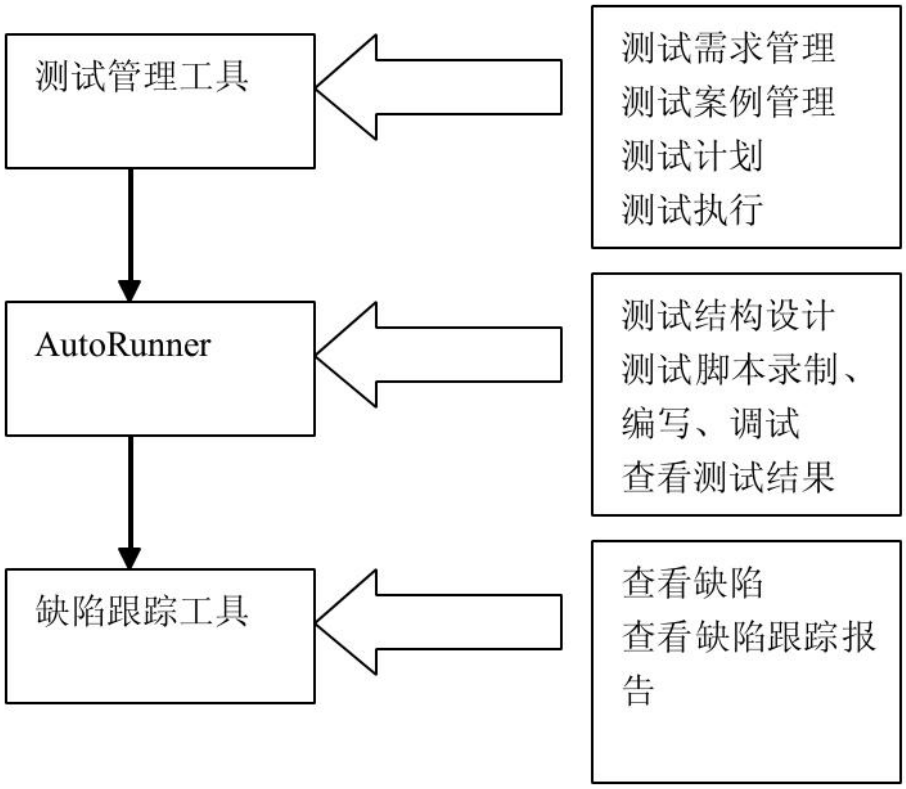
XML

EXCEL

注：理论上支持 jdbc 接口的数据库。

AutoRunner 是一个自动化的功能测试工具，它可以和测试管理工具、缺陷跟踪工具

一起来使用，以达到更好的效果。



3.2 系统性能

AutoRunner 针对与系统的功能测试自动化，对性能要求不高：自动测试的脚本执行速度，超过人工执行的速度。

3.3 扩展能力

- **扩展验证点**

所谓的验证点，就是用来验证被测试系统返回数据或者状态是否和预期一致的点。

AutoRunner 提供了完整的验证点功能，用来验证字符串、bitmap 文件是否正确，对字符串可以验证是否符合定义的“正则表达式”。当然，由于验证往往是非常复杂的，例如：当我们使用一个功能向 database 中增加一条记录后，通过 jdbc 来查看该记录是否已经被增加。这就需要用户根据具体的数据库来编写一个功能来实现特殊的校验点。系统提供了基本的校验方法，允许用户自己来通过编写一个特殊校验的类，或者一个特殊的方法来定义特殊的校验点（调用的结果如果希望反映的标准的测试报告中，就需要

调用系统提供的基本方法），最终实现对验证点功能的扩展。

- **自动录制时候的针对用户自定义组件的识别**

根据国外测试人员的经验，编写脚本的工作中，大量的工作都被用户的自定义组件消耗了。由于很多的测试工具本身支持一组标准的控件，在自动录制的时候，系统能够根据这些组件来生成测试脚本，并且允许回放这个脚本来执行测试。当用户自定义了一个组件之后，用户定义的组件是基于基本组件的，系统就往往无法自动识别这些组件，导致测试人员录制脚本的时候非常复杂：名称不同、识别困难、运行时刻同步点错误。AutoRunner 提供了对组件的定义功能：所有的组件类型必须被定义，并且只有最上层的已定义类型组件被识别，其他的组件都不会被识别。如果用户定义了自己的组件，那么他只需要把他自己定义组件的：类名、contexttype 增加到组件定义文件中就可以了。AutoRunner 的这个功能大大增强了对用户自定义组件的支持，使得测试人员能够录制正确的脚本、编写正确的脚本，减少差错。

- **对第三方测试管理工具的支持**

AutoRunner 提供了对第三方测试管理工具的支持：通过数据文件或者数据库，就可以传递测试案例信息、测试案例数据信息。AutoRunner 提供了命令行的支持，支持用户在远程启动和调用，这就为第三方的测试管理工具提供了一个执行调用接口。

- **对第三方缺陷跟踪工具的支持**

同样的，AutoRunner 可以提供针对缺陷跟踪工具的 API 的调用，和第三方缺陷跟踪工具达到“无缝连接”。

3.4 可靠性和可用性

系统的可用性和可靠性由几个指标来衡量：

第一，系统的出错处理能力。也就是，当系统出现错误之后，是否能够提供完善的错误处理机制，跳过错误，继续执行允许执行的下一个功能点测试。

第二，系统执行过程中工具不会出现异常，导致测试无法正常执行。

第三，测试脚本出现异常，提供强大的调试功能。

第四，当 AutoRunner 升级之后，原有测试脚本能够兼容，继续使用。

具体到 AutoRunner，如下：

- **系统的出错处理能力**

对所有的测试案例来说，每一个测试案例都是一个继承自 class TestCase 的子类，在测试过程中的动作都是调用父类 TestCase 中的方法来实现的，如：setWindow()，setValue()，getValue()，setProperty()，getProperty()等。这些方法在 出错的时候（一 般都是同步点错误 ），会抛出 一个异 常 syncException。案例只有一个主要的测试过程类：test() throws syncException。当 test()执行的时候，如果出现异常，就会抛出一个 syncException，外部的方法会 catch 到这个 syncException，然后使用一个通用的方法来处理错误。测试人员只需要编写一个标准的错误处理方法就可以完成这些所有的工作。当然，这个测试人员需要对 java 有一定的了解和熟悉，但是这样的人员只需要一个就可以了，因为出错处理程序只需要一个，它用来处理所有的错误，并且使得下一个测试案例可以被执行。

- **IDE 的稳定性**

在一个大量的测试案例被执行的时候，实际上 IDE 并没有工作，它只是在等待响应。执行测试的过程，就是执行 java 各个不同的类的过程。而 TestCase 是一个非常健壮的类，不会导致系统出现异常。因此，IDE 从理论上是非常坚固的。

另外基于 java 的系统一般而言，稳定性都非常好。特别是所有的测试案例基本上都是继承自 class TestCase。

- **产品升级**

当产品升级的时候，对原有测试案例影响最大的就是 TestCase 类的变化。classTestCase 实际上只是一个 abstract，只实现了一个基本的 interface，实际的功能都是由底层的组件来实现的，这个组件在 IDE 启动的时候被 load，跟测试人员自己编写的测试案例没有任何直接关系。

因此当底层的类发生变化的时候——系统升级可能会带来底层类的变化——对测试脚本没有影响。

3.5 国际支持

支持多种语言 Unicode 编码形式；用户可以选择中英文界面的版本。系统对语言编码的识别是由系统自动完成，用户不必考虑选码的问题。

4.系统基本功能

4.1 测试案例创建与录制

- **创建测试案例**

用户能够创建一个测试案例。创建的测试案例脚本是空的，需要用户自己来加入包的名字、类的名字等等。创建测试案例可以在项目浏览器中使用右键菜单或者系统的菜单。如果用户是一个非常熟悉测试案例的测试人员，他（她）就可以自己手工来编写测试案例的代码了。但是，由于资源文件不存在，所以他（她）如果希望自己编写的测试案例能够执行的化，还需要手工编写对应的 xml 资源文件。创建测试案例的过程都是从录制开始的。

- **通过录制创建测试脚本**

当你从菜单或者工具条启动“录制”命令，系统开始记录你的所有操作，并且在记录过程中把生成的脚本文件显示在编辑器上面。

录制的结果是，你得到了：

1）一个可以被执行的测试脚本文件；

2）测试脚本相关的资源文件，这个资源文件用来记录所有脚本中用到的窗口、组件的属性（如：名称、位置、tabindex、类型等）。

4.2 测试案例编辑

- **测试案例的结构**

测试案例是具有结构的，它能够运行，首先要符合 java 的语法和主程序入口。并且它需要使用测试基本类提供的功能来完成测试。

- **测试案例编辑**

AutoRunner 提供了强大的测试案例编辑功能：

第一， 提供了 java 脚本的关键字识别技术，能够识别系统的关键字，避免语法错误；

第二， 第二，提供了实时语法分析的功能，在编辑过程中动态分析语法，并且对语法错误动态报警，尽量避免编译时刻再出现错误。

4.3 测试案例参数化

- **数据驱动**

录制完成测试案例之后，你就得到了一个测试脚本。如果这个测试脚本只能够被执行一组数据，并且数据是固定不变的，那么你每一次的测试就只能够执行很简单的功能了。边界条件、路径覆盖，需要使用一个脚本、很多组数据输入才能够完成，固定的数据无法满足要求。

数据驱动就是指能够把需要输入（和验证）的数据参数化，通过脚本执行不同的数据，就实现了数据驱动，也就是数据与脚本分离。AutoRunner 实现了脚本与数据分离：脚本使用 java 的脚本，在脚本执行的时候，从数据源中读取数据。AutoRunner 使用了 DataSource 这样一个接口来实现参数化。DataSource 通过外部定义的组件实现对外部数据源的操作功能，从外部获取数据。DataSource 本身就是通过插件来实现的，IDE 只定义了 interface，外部插件决定系统的行为。通过加载不同的插件，用户可以使用不同的数据源来访问数据。如：excel、xml、db 和其他。

- **测试案例参数化**

AutoRunner 在自动录制完成之后，可以通过菜单“参数化”，AutoRunner 会弹出所有的对象树，提供给用户勾选，选中部分进行自动参数化。

参数化的结果：

1）脚本变为参数化脚本；

2）数据池自动增加了选择的参数列表。

在测试案例参数化之后，用户仍然可以手工来修改，实现进一步的参数编辑工作。

- **创建外部数据源**

只有访问数据源的脚本，没有外部数据源，那么所有的脚本访问都会失败。用户需要创建外部的数据源。

有两种方式创建数据源：

第一，自动通过 IDE 创建。在脚本文件中，选中该脚本的右键菜单中的“创建/维护脚本”，IDE 会自动查找所有的 datasource 操作，并且更新数据源。

第二，通过手工创建。需要在外面手工编辑文件。

4.4 增加同步点和验证点

- **同步点的概念**

在进行输入输出之前，就需要对系统进行同步，使得输入和输出能够针对正确的窗口或者组件，以免出现异常和错误。如果同步条件没有出现，系统就需要等待一段时间，来满足运行系统的要求，使得需要操作的组件能够显示出来。

- **自动同步和手工同步点**

所谓的自动同步点，是只在操作过程中，由于本身需要执行操作，如对某个组件输入一串字符，而需要等待这个组件出现，这种同步点是系统在操作过程中自动加入的，我们称为“自动同步点”。也有一些情况，需要手工增加一些同步点，当系统执行到一定时候，需要等待一个条件出现再继续执行，这种同步点我们称为“手工同步点”。用户需要关心的是手工同步点，例如：需要等待一个 image 能够正确显示，然后再继续下面的工作。它不是单纯的等待，而是每间隔一段时间就去查看是否满足同步条件，如果满足系统就继续执行，如果不满足而系统超时时间没有达到，就继续等待。如果出现超时，那么就抛出 SyncException。

- **验证点**

测试的目的是看执行一个过程，结果是否和预期结果一致。验证的方法就是查看结果是否一致，这个点我们称作“验证点”。验证成功则继续执行，验证不成功也需要继续执行，并且把结果写入测试报告。AutoRunner 的验证点需要手工加入——AutoRunner 不知道用户需要验证那些内容。

- **增加验证点**

用户可以使用编辑器来增加验证点，AutoRunner 提供了方法让用户来增加验证点。

4.5 测试案例执行

- **测试案例执行**

当测试案例只有能够被执行才有意义。在 AutoRunner 里，测试案例是一个 java 的类（特殊的 java 类）。这个类首先被编译，然后执行。通过菜单上的“执行”项，你可以执行这个测试案例。如果编译出现错误，则会在信息栏中提示错误。执行支持标准输出，并且把标准输出显示在 AutoRunner 下面的输出框里面。

- **多次执行**

当测试用例被执行的时候，AutoRunner 会提示，需要用户输入当前测试脚本被参数化之后，需要使用的数据列表的行号范围。输入之后，会多次执行这个测试脚本，每次使用一行的数据，达到一个脚本中执行多次的目标。测试跟踪调试测试脚本本身也可能出错，也可能由于被测试对象的变化（如缺少了一个对象）而出现错误。因此，定位和排除错误的方法，我们使用了跟踪调试。AutoRunner 使用了 java 作为测试脚本，并且每个测试脚本都是一个 java 的类。因此 AutoRunner 实现了 java 的跟踪体系结构：JDA。AutoRunner 允许用户设置断点、查看本地变量值、查看指定的变量的值，并且提供了单步执行的各种模式。

5.AutoRunner 的特点

评估自动测试工具的关键在于：

第一， 很高的建立测试案例的生产率；

第二， 降低用户的二次开发成本；

第三， 便于维护使用；

第四， 便于测试案例的数据驱动扩展；

第五， 测试案例资源的延续性；

第六， 扩展性。

下面，我们就 AutoRunner 在这几个方面的特点简要介绍：

- **AutoRunner 具有很高的生产率**

自动测试工具建立一个测试案例脚本的时间成本为手工测试一次的 3－10 倍，可见建立自动测试的起始是需要一定的成本的。

如何降低建立测试案例的成本，是自动测试工具的关键。AutoRunner 的优势在于：首先，优秀的自动识别组件功能。脚本能够在录制完成之后直接使用，能够自动适应出现的各种情况，如：窗口位置、title、大小等的变化，组件位置、名称的变化。通过自动识别能够识别处组件，从而降低对编写脚本的要求，提高了自动录制的可用性。第二，提供了数据驱动框架。很多测试工具虽然支持参数化的功能，但是需要手工完成数据驱动框架，才能够实现数据驱动：从指定的文件中获取数据。AutoRunner 自动定义标准的数据驱动模式，定义了标准的数据驱动格式，降低了增加测试案例的成本。虽然建立一

个测试脚本需要一定的时间，但是在测试脚本建立之后增加一组数据的时间却非常短。

- **模糊识别**

AutoRunner 对每种组件定义了标准的模糊识别指标。在录制测试案例之后，系统的资源文件就会根据系统的配置文件生成确定识别权重的指标。在测试脚本被执行的时候，通过权重算法来进行模糊识别和匹配。

- **关键字驱动**

AutoRunner 提供了领先的关键字驱动技术，支持脚本编写使用专家视图，不熟悉脚本的用户使用关键字视图，并且实现在脚本视图与关键字视图之间的相互转换，既提升了效率，也提升了易用性，既能够给熟悉脚本的测试工程师提供高效的工作平台，也能够给不熟悉测试脚本的测试工程师使用方便。用户可以对系统配置文件的识别参数进行调整，达到修改整个录制脚本识别参数权重的目标，便于提高整个项目中脚本开发的效率。在用户录制完成脚本之后，可以对对应的资源文件的权重属性进行修改，使系统能够定制具体的模糊识别对象，对脚本组件识别算法作特殊处理。通过模糊识别算法，能够极大地提高脚本执行的可靠性，对于由于类似组件位置、大小等变化之下的脚本执行，能够起到非常良好的效果：用户不需要因为界面小的修改而导致来修改测试脚本。

- **便于维护使用**

案例完成之后，随着应用系统的修改、应用系统版本的提升，同样需要维护这个测试用例库，因此维护使用是非常重要的功能。维护方便性主要体现在几个方面：简洁的框架、容易理解的脚本、方便的调试功能。

AutoRunner 提供了针对测试案例的框架，这个框架包括：案例层次划分（AutoRunner 的案例由 Action 组成，每个 Action 包含对一个 Window 的所有操作，AutoRunner 允许在案例之间共享 Action 来提高系统的可维护性）、数据驱动框架、自动同步、数据校验模式等。使用这些框架能够非常容易的维护测试案例库。

AutoRunner 采用了 java 的语法，测试人员使用的语法非常简单，便于理解和使用。并且，由于系统提供了关键字驱动的框架，所以对一般的维护而言，根本不需要了解 java，只需要知道最基本的操作就可以。

AutoRunner 遵守 JDA 的标准，提供了最强大的系统调试功能：从设置断点、单步执行、变量查看、表达式查看等方面提供支持，便于测试人员容

易排除错误。

另外，AutoRunner 提供了强大的编辑器，在一般编辑器能够动态识别语法关键字的基础上，还能够同时提供语法检查——在编辑的时候从事语法检查，对错误的语法实时提示。这个编辑器对于比较缺乏编程经验的程序员来说，非常重要。

- **测试案例资源的延续性和扩展性**

测试案例库本身也是一种资源它和应用版本是对映的关系，随着应用系统版本的升级，案例库也会升级，那么回归测试的效果才能够最大化。

对于测试工具来说，要保证这个资源，就需要保证：测试脚本的兼容性。另外由于随着应用的发展，测试工具的功能需要大幅度的提升，因此工具的可扩展性也需要保证才能够保证测试案例资源的延续性。

AutoRunner 使用了 java 语言作为基础，并且实现了 java 调试功能，可以随着 java 的发展不断的发展，扩展自己的功能。采用 java 语言是一个巨大的优势，比测试工具自己使用一种语言要方便的多。从根本上说，AutoRunner 不是采用了哪种语言的语法，而是从根本上就是 java 语言。这和采用 vbscript 或者 c 语言语法的工具是截然不同的。在扩展外部功能方面，由于 AutoRunner 使用了 java 语言，允许使用外部的包，也就是说可以任意增加脚本本身的功能而不受语法的限制和工具本身是否支持外部包的限制——在最大程度上提高了扩展性。

6.厂商支持能力

AutoRunner 泽众自动测试引擎软件，我们通过在线 QQ、微信、电话、电子邮件为您提供支持与服务，您也可访问我们的网站 http://www.spasvo.com/寻求帮助；为保证服务质量，确保有效地解决用户的问题，保障用户的项目实施进度，技术支持仅向授权用户和授权试用用户提供。请您在联系泽众技术支持时，告知您的单位名称和服务代码。

技术支持

电话：021-60725088-8007

传真：021-60725088-8017

电子邮件：support@spasvo.com

QQ：1404189128

泽众微信公众号

产品服务

有关培训、产品购买及试用授权方法的问题，请与销售代表联系，或联系泽　众咨询热线.

电话：021-60725088-8006

传真：021-60725088-8017

电子邮件: sales@spasvo.com

提供完备的用户手册，管理员使用手册，系统技术手册并再系统升级后及时修改更新服务。

厂商能够根据在实际应用中的问题，迅速给予解答（2 小时内），并给出解决方案(48 小时内)。