软件自动化测试框架设计参考准则

软件自动化测试框架设计参考准则 测试框架是在所有不同的 测试自动化 阶段定义的一整套指导准则： 需求分析 阶段、脚本设计阶段、执行阶段、报告和维护阶段。框架即对于内部复杂架构的一种包装，这样的包装可以使得最终用户方便的使用。框架还具有对于流程标

软件自动化测试框架设计参考准则
测试框架是在所有不同的**测试自动化**阶段定义的一整套指导准则：**需求分析**阶段、脚本设计阶段、执行阶段、报告和维护阶段。框架即对于内部复杂架构的一种包装，这样的包装可以使得最终用户方便的使用。框架还具有对于流程标准的强制执行性。

目前为止，还没有一种关于如何**开发**测试框架以及在开发过程中需要考虑哪些因素的准则。有很多记载着各式各样的测试框架以及它们各自是如何工作的白皮书，但是这些白皮书中还没有任何一篇文档是记录着测试框架设计共同需要考虑的因素。本文基于测试框架需求，涵盖了测试框架各个方面以及一些必备的基本要素。

1. 自动化测试框架的类型 – 目前普遍存在的框架有以下几种：

- 数据驱动框架 – 当测试对象流程固定不变（仅仅数据发生变化），可以使用这种测试框架。测试数据是由外部提供的，比如说Excel表、XML等等

- 关键字驱动框架 – 这种自动化测试框架提供了一些通用的关键字，这些关键字适用于各种类型的系统。它还为自动化**测试工具**和被测系统提供了抽象性。举个例子，它可以使用相同的**测试用例**来测试类似的Web和**Windows系统。**

- 混合型的框架 – 混合型自动化测试框架同时具有数据驱动型和关键字驱动型框架的优点。这种测试框架不但具有通用的关键字，还有基于被测系统业务逻辑的关键字。例如“登录”、“退出”是可以被使用的仅局限于某系统的关键字。

2. 不要过分的改造 – 自动化测试框架应该尽可能的使**自动化测试工具**发挥它自己强大的功能，而不是通过实现新的关键字来重新定义整套语言。开发一套关键字驱动的自动化测试框架的代价是很大的而且非常耗时。开发一套混合型的自动化测试框架的代价就相对较小而且开发周期短。

3. 可重用性 – 测试框架应该尽最大可能提高可重用性。把单独的Action组合成业务逻辑可以提供可重用性。举个例子，可以把类似于“输入用户名”、“输入密码”和“点击登录”这些Action组合成一个可被重用的模块：“登录”

4. 支持系统的不同版本 – 自动化测试框架应该允许重复使用基线化脚本，这样可以保证这份脚本能被用来对被测系统的多个版本进行测试。对不同系统的支持有两种方式：

- 复制和修改 – 这种方法包含了新建基线脚本的一个拷贝、修改这份拷贝用以测试特定版本的项目。51Testing**软件测试**网 ^b1o.N8W0Y

- 重用和升级 – 这种方法包含了重用基线脚本、提供一个此脚本的升级和优化用以测试特定版本的项目。这样做可以最大化的保障可重用性，这也是推荐的方法。

5. 支持脚本版本化 – **测试脚本**应该被储存在类似于**CVS**、微软的**VSS**等**版本控制**工具中。这样做可以保障在灾难发生的时候可以被恢复。

6. 将开发和发布环境分开 – 自动化应当和其它开发项目同等看待。测试脚本应当在一个**测试环境**下创建和调试。一旦测试脚本测试通过后唯一该做的就是将它部署到发布环境。在紧急发布版本的情况也同样适用这种方法。

7. 外部可配置性 – 脚本的可配置项应当被保存在一个外部文档中。系统的URL、版本、路径等都可以被视作可配置项放在外部文件中。这样做可以使得在不同的环境中都可以执行测试脚本。需要注意的是外部配置文件的路径不要写死，如果把它写死了虽然在任何环境中都还是可以运行脚本，但是每次只能在一个环境运行。配置文件的路径使用相对路径即可解决这个问题。

8. 自身可配置性 – 理想的测试框架应该是自身可配置的。一旦部署到系统中之后应当不需要再做任何手工配置，脚本应当自动配置完成一些必要的设置。

9. 任何对象改动引起的变动应该是最小的 – 自动化过程中最为常见的问题是对象识别的变更。测试框架应该支持可以很容易的来完成这些修改。这可以通过将所有的对象识别设置储存在一个共享文件来实现。这个文件可以是XML文件、Excel文件、**数据库**或者自动化所特有的格式。这里有两种可能的方式来实现对象识别设置的方式：51Testing软件测试网PX6}0j&re

- 静态方法 – 这种方法中，所有对象定义都在测试最初被加载到内存中。任何对象定义变更只能通过停止和重新运行测试来实现。

- 动态方法 – 对象定义是通过需求拉动的。这种方式和静态方式比较而言显得较为缓慢。但是对于非常大的脚本而言，并且对象识别需要在运行时做修正的情况下，动态方法是适用的。

10. 测试执行 – 自动化测试框架也许需要满足以下需求（基于实际需求）

- 执行单独的测试用例；

- 批量执行测试用例（一组测试用例）；

- 只执行失败的测试用例；
- 可以在前一个（一组）测试用例执行结果的基础上，执行下一个（一组）测试用例；

根据实际需求还会有许多其他情况。一个框架可能无法实现所有这些需求，但它应具有足够的灵活性以适应今后此类需求。51Testing软件测试网?#q!U l!Y%K1n

11. 状态监测 - 一个框架应允许实时监控执行状态，一旦失败能够发出警报。这样可以在出现failure之后迅速反馈。

12. 报表 – 不同的系统对报表有不同的需求，有时候需要一组测试的整体报表，有时候只需要单个测试用例级别的测试执行报表。一个好的测试框架应该有足够的弹性来按需支持不同的报表。

13. 发生更改的时候对测试工具尽量小的依赖性 – 一些测试脚本的更改可能只能在打开测试工具后，在测试工具中进行修改，然后保存。测试脚本应该在没有测试工具的情况下也可以对脚本进行更改。这样的实现可以减少license的购买从而为公司节省开支。这样的实现还可以让所有想去修改脚本的人无需安装测试工具也可以很方便的对脚本进行修改。

14. 方便的调试 – 调试在自动化过程中占据了大量的时间，因此在调试这个过程中需要加以特别的关注。关键字驱动的测试框架因为使用了外部的数据源（比如Excel数据表）去读取脚本中的关键字和**测试过程**，所以较难调试。

15. 日志 - 生成日志是执行的重要组成部分。在一个**测试案例**的不同点生成调试信息这是非常重要的。这些信息有助于快速地找到问题的范围，同时缩短了修改时间。

16. 易用性 - 该框架应易于学习和使用。对框架的人员培训费时且昂贵。有一个好文档的框架更容易理解和使用。

17. 灵活性 - 框架应该足够的灵活，以适应任何改进，而不会影响已有的测试案例。

18. **性能**的影响 – 框架还应考虑对执行性能的影响。一个复杂的框架会增加脚本的加载或执行时间，这一定不是我们所期望的。像缓存技术，当执行时编译所有代码到单个库中等...只要可能都应该用于性能的改善。

19. 框架支持工具 – 开发一些外部工具来完成任务，这对框架设计会有帮助。这是一些例子：51Testing软件测试网:N v7nsq

- 从本地文件夹上传脚本到**QC**51Testing软件测试网F}'Z]F&J Ah
- 结合库文件到当前打开的脚本51Testing软件测试网+[SF1nR"]N
- 同步本地和QC上的脚本文件

20. 编码标准 - 编码标准应确保脚本的一致性，可读性和易于维护。编码标准应包含下列内容：

- 命名规范（变量、子程序、函数、文件名、脚本名称等），例如i_VarName为整数变量， fn_i_FuncName为返回值是整数的函数；
- 库、子程序、函数的头部注释。这应包含,如版本历史，创建者，最后修订者，最后修订日期，说明，参数，示例；
- 对象命名规范。例如txt_FieldName为一个文本框；

我们应该把自动化测试看作是一个开发项目，而不仅仅是记录和回放事件。先有一个良好的框架，再开始自动化测试，这样可以确保较低的维护成本。当你在开发一个框架，进行框架的需求分析时，可以参考本文谈到的这些准则。