

STeP-IN SUMMIT 2007

International Conference On Software Testing

Designing Better Test Suites

by

Jayatheerthan Krishnamurthy
IBM India Software Labs
Bangalore, INDIA

jayatheerthan@in.ibm.com

Copyright: STeP-IN Forum and Quality Solutions for Information Technology Pvt. Ltd.

Published with permission for restricted use in STeP-IN SUMMIT 2007 in agreement with full copyrights from owner(s) / author(s) of material. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior consent of the owner(s) / author(s). This edition is manufactured in India and is authorized for distribution only during STeP-IN SUMMIT 2007 as per the applicable conditions.

Practices Experience Knowledge Automation

Produced By

STeP-IN
Forum

www.stepinforum.org

Hosted By



www.gsitglobal.com

ABSTRACT

The expectations from a Test Suite are no longer the same as what they were a few years ago. With so much of advancement in testing tools & techniques, and with the ever increasing complexity of the products being tested, test suite design must take a paradigm shift and should move towards addressing the concerns of testers and test suite developers.

While the primary focus of test suite design must revolve around its ability to unearth product defects, an equally important aspect to be considered during test suite design is its ability to adapt to future changes in the product being tested as well as to the changes in underlying test automation framework. This paper addresses the common problems faced by a tester and a test suite developer by offering some of the best practices in test suite design that have worked successfully for so many test projects.

1. INTRODUCTION & MOTIVATION

With very tight schedules and short deadlines always around the corner, a test suite developer does not get enough time (or the motivation) to even think about writing a nicer code. His aim would most likely be to develop a test suite which works just fine, rather than aiming for a code that is flexible enough to adapt to future changes. However, if there are some easy-to-follow tips, frameworks and design patterns that make one's life easy, then any one would be more than willing to implement them while developing test suites.

This paper discusses common challenges in test suite automation and provides solutions that worked for my test project and various test projects in my company.

Section Description

- Section 2, 3, and 4 : Challenges with test automation
- Section 5 : Frameworks that help to overcome these challenges
- Section 6 : Case study showcasing successful implementation of the best practices
- Section 7 : Enhancements to the frameworks discussed in section 5

2. COMMON PROBLEMS WITH AUTOMATED TEST SUITES

Test suites that are not well designed are always an obstacle in delivering good quality product to customers. A tester using an ill-designed test suite faces the following problems:

a. Lack of portability

Test suites that are strongly coupled with an underlying test automation framework are not compatible with any new advancement in the automation framework technologies and hence a tester is unable to leverage the core advantages of latest technologies. Test suites that are strongly coupled with a particular test automation framework will have to be rewritten to adapt to a new automation framework with better capabilities (be it home-grown or open standards such as STAF). This results in redundant efforts in test suite development.

b. *No indication about type of runtime problems*

A test suite that is not well designed may not show clear indication of the type of the problem encountered - Test code defects Vs Product defects. Most often, the test executor is different from the test case developer and hence such ill-designed test suites increase the dependency of the tester on the test case developer to determine the type of the problem.

c. *Lack of adaptability to changes to product interfaces*

This problem is very common in GUI testing. The current age GUI testing tools use the concept of 'Record and playback' to perform functional verification of GUI panels. This means that the test code relies on the position of UI widgets and/or their containment hierarchy to lookup GUI objects at runtime. While the formal test execution cycle starts after the coding phase is over, the test suite development work overlaps with the product development so that testers can start using the test suite once the product code is ready for testing. However, the UI panels might have undergone some changes from what they were when the test code was developed and by the time the actual test cycle started. This requires few changes to the test suite to adapt to the changes to UI panels. While all of us would accept the fact that such kind of changes to the UI are unavoidable, the amount of rework required to adapt the test suite to the new changes is directly proportional to the quality of test suite design.

d. *Not a plug and play model*

A test suite that does not expose well defined interfaces cannot be plugged into a different test automation framework in future.

So, what are the characteristics of a well designed automated test suite? The next section explores these characteristics with the sections that follow offering more insights into them.

3. CHARACTERISTICS OF A WELL DESIGNED TEST SUITE

- ✓ Should have well defined interfaces. There must be a standard way of getting input parameters and a standard way of reporting back the test results
- ✓ Should be customizable to run a subset of the test suite. This is very helpful if only a portion of it needs to be retested due to previously seen failures. Also if only critical test cases need to be run, then the test suite should provide a facility to select a subset of test cases for execution
- ✓ Should not be coupled with any test automation framework. Many times, a test code would have the need to make a call to STAF command or a command offered by the underlying test automation framework. If the test code makes direct calls to these commands, then it is not portable across multiple automation frameworks
- ✓ Should be designed as per the layered approach (as described in Section 5) to facilitate adaptability to future changes in the product being tested
- ✓ Should help the tester to differentiate, if an error seen during test execution is due to test code bug or a product defect
- ✓ Should be capable of suggesting solutions to the runtime problems (self healing)

- ✓ Should be a building block in constructing more complex aggregate test suites

Since the term ‘Test suite’ covers a vast area, we are going to restrict the rest of our discussion in this paper to GUI test automation. In the sections that follow, let us discuss about the challenges in GUI test automation and about the frameworks that help in building successful GUI test automation suites. Many success stories follow the discussion.

4. CHALLENGES WITH GUI TEST AUTOMATION

a. *Managing object repositories*

A major challenge with GUI test automation is the management of UI object repositories. The repository grows in volume as the test suite starts supporting more number of test cases. Also, there are chances of object reuse across test cases. Hence a well designed object repository is the only solution for efficient test suite development. Section 5 discusses object repository management techniques.

b. *Identifying reusable task sequences*

Identifying reusable tasks and organizing them well is a key to developing good quality test suites. Automating a test case by choreographing reusable tasks helps in developing easy-to-maintain test suites.

c. *Handling slow response time of remote test machines*

This is a very important aspect to be kept in mind while designing GUI test automation. Not all systems have same response time. For example, a geographically remote test system with low end hardware would definitely have very slow response times than a local test system. Hence, the page refresh rates are going to be different for these two cases. The test suite should be designed such that it is configurable to adapt to different response times of test machines. Practical examples and solutions are given in Section 6 – Case Study.

d. *Web Application testing - Supporting multiple web browsers*

If the GUI test automation is for a web based application, then the test scripts must be browser independent. The tester should be able to seamlessly run the test suite on any given browser. Case study given in Section 6 elaborates more on this.

5. FRAMEWORKS THAT HELP IN GUI TEST AUTOMATION

This section describes IBM’s ITCL framework^[1] for GUI test automation as well as a framework that I built along with my test team to provide portability for my test suite across multiple test automation frameworks.

5.1 IBM’s ITCL Framework

The IBM framework, formerly known as the ITCL framework, was developed by the Quality Software Engineering team in collaboration with experienced automation teams throughout IBM. The framework consists of a three-tiered architecture implemented through the *appobjects*, *tasks* and *test cases* packages.

The principles underlying the appobjects, tasks, and test cases packages are:

- Layered architecture
- Separation of the "what" from the "how"
- Code reuse
- Consistent and clear organization

Here's an explanation of appobjects, tasks, and test cases:

- **AppObjects:** This is where you will store information about your application's GUI elements. It is also where you will write your 'getter' methods, which return objects enabling the caller to query and manipulate these GUI elements. Typically, these methods are called within the *Task* layer.
- **Tasks:** This is where you will write reusable methods that exercise common functions in your application. It is also where you will write methods to manipulate and query complex, application specific controls. Methods in a *Task* are called by *Test Cases*.
- **Test Cases:** These are methods that navigate through an application, verify its state, and log results

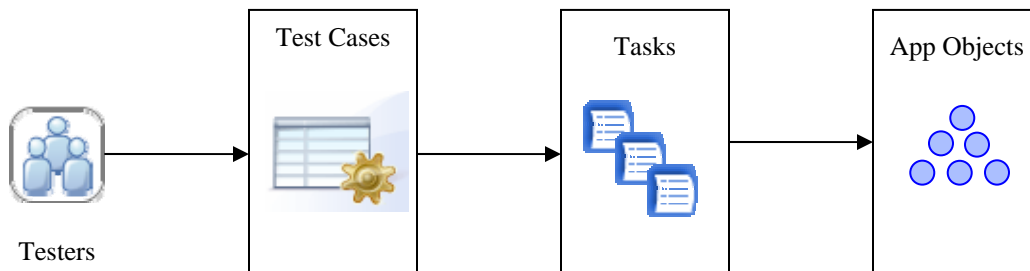


Figure 1: Three Layers of ITCL framework

Advantages of ITCL framework

- ✓ Separation of test suite into 3 layers enables easy management
- ✓ Test code is completely transparent of changes to GUI objects
- ✓ Future changes to GUI panels requires minor modifications to AppObjects and Task layers and hence the Test suite is now future proof
- ✓ Tasks can be reused in multiple test cases, thus avoiding redundant code
- ✓ ITCL provides a Widget class library and many object lookup utilities which help in easy development of GUI test automation suite

ITCL framework resolves the problem described in *Section 2c*. ITCL framework has been implemented in many test projects in IBM and has proven very successful. One of the tools where ITCL framework can be implemented is IBM Rational Functional Tester (RFT). My test project uses RFT and ITCL framework. The combination of these two can create wonders for any GUI based Functional Verification Test project.

Managing Object Repositories

Object repository management is an important aspect in GUI test suite development. The GUI objects are subject to change and hence we need a proper planning here. We must first classify the types of objects that we need to capture and organize them into packages. Some of the GUI automation tools (such as Rational Functional Tester) provide options to customize the object lookup parameters; assign weight for each test object which will be used to match against the actual GUI object at run time.

5.2 Test Suite Adaptability Framework

This is a framework designed by me and my test team, which is implemented in our test project and is successfully in use for the past 2 years! This framework is also widely used by many of the test teams across the globe within IBM. This framework offers flexibility to the test suites to adapt to changes in the underlying test automation framework as well as changes to any external components that test suite may interact with. This framework addresses the problems discussed under *Section 2a*.

The idea is very simple, but the application of it is very powerful! The framework mandates that any calls that are made external to the test suite must route through a gateway that determines how to dispatch the call.

For example, a test case that tests a Shopping Cart web application might want to query the backend database to verify if the items purchased by the customer are persisted in the database. Since the database call is external to the test suite, it gets routed through a Database gateway which determines how to dispatch the database call and how to retrieve results from the backend. The advantage of the gateway layer is that the test suite is transparent to changes to the database layer.

Figure 2 explains in the simplest form, the layered approach of this framework that enables the test suite to adapt itself to changes to the underlying test automation framework.

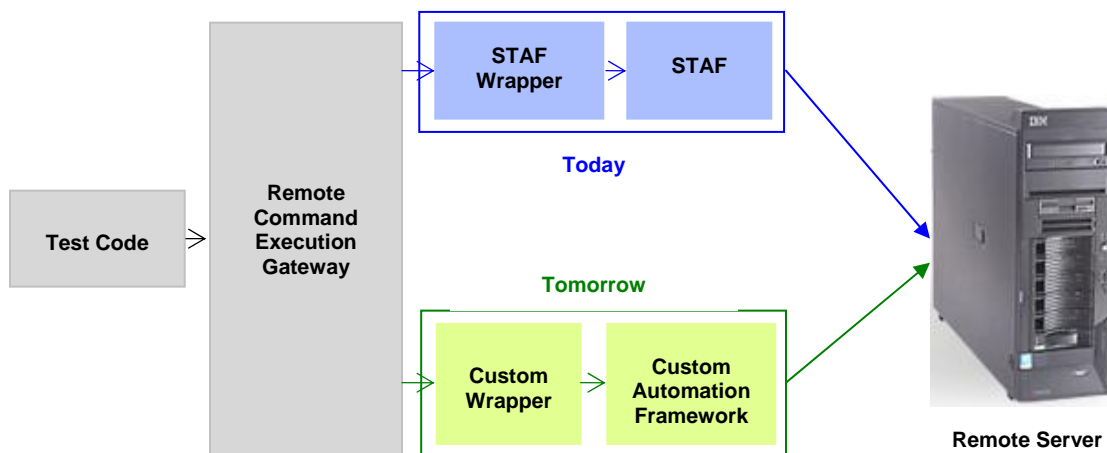


Figure 2: Simplest form of Test Suite Adaptability Framework

As Figure 2 explains, if the test code runs on STAF today, then it can be run *as it is* by plugging in a different custom automation framework (or) any other framework tomorrow.

The ‘Remote Command Execution Gateway’ has well defined interfaces that would be implemented by the Automation Framework Wrapper classes (for example, STAF Wrapper and Custom Wrapper as shown in Figure 2 above). Plugging in a new test automation framework (or) command execution framework is as simple as implementing the Command Execution Interface.

Advantages of Test Suite Adaptability Framework

- ✓ Test code is unaware of changes to underlying test automation framework

- ✓ Adaptable to future changes to underlying command execution framework
- ✓ Write once, run anywhere

6. CASE STUDY

The frameworks discussed in Section 5 were implemented in my test project – Administrative Console Testing. This is a GUI based testing which tests administrative console of IBM products. This involves verifying of GUI panels as well as executing some product commands to cross check the verification.

We used IBM ITCL framework and Test Suite adaptability framework for this project. The following section describes how we implemented these two frameworks in our project.

1. We created Test Object repository to record GUI widgets. This helped us in isolating test object lookup from the test cases and hence test code was made transparent to changes to UI panel changes.
2. Used ITCL utilities to generate ‘getter’ methods for looking up GUI test objects
3. Identified reusable tasks and grouped them under the tasks package
4. Wrote test suites by choreographing tasks
5. Used the Remote Command execution gateway to dispatch calls to the remote test machine. This gave the flexibility of adapting to changes to underlying automation framework

We also had a requirement to support multiple browsers. Though the DOM object model is same for multiple browsers, the HTTPS authentication challenge window is different for different browsers. This was a challenge for us. We created separate Object repositories for supporting multiple browsers, which helped to overcome the challenge.

Another challenge faced was to handle slow response times of remote test machines. The page refresh rates are not same for all platforms. For example, a remote zSeries test machine may respond slower than a local AIX test machine. In order to handle this, we added a few properties to our test suite config files to fine tune the time period the test suite must wait before locating a GUI test object.

With the help of ITCL and Test Suite adaptability framework, we reaped the following benefits:

- ✓ Test suite is resilient to changes to Admin Console GUI panels
- ✓ Supporting multiple browsers was easy
- ✓ Changes to underlying automation framework did not affect our test suite

7. ENHANCEMENTS TO TEST SUITE ADAPTABILITY FRAMEWORK

We have identified few enhancements to the Test suite adaptability framework that will increase its ability to handle runtime problems in a better fashion. The enhancements that we are currently making to it are:

- Using Autonomic Computing ^[2] to make the test suite Self Healing
- Enabling the test suite to differentiate product defects Vs test code defects

Autonomic computing systems have the ability to manage themselves and dynamically adapt to change in accordance with business policies and objectives, enabling computers to identify and correct problems often before they are noticed by IT personnel. Autonomic computing systems have the following behavior:

- ✓ Self Configuring

- ✓ Self Healing
- ✓ Self Optimizing
- ✓ Self Protecting

We plan to make our framework Autonomic enabled so that we reap the above mentioned benefits. With the help of autonomic systems, test suites would be able to suggest solutions for commonly occurring problems during test execution, thus making a tester's life easy.

Also, with Autonomic system, we would be able to differentiate if a problem during test case execution is due to test code defect or product defect. So, test suites that are Autonomic enabled would be very helpful to the testers and would help in troubleshooting problems better.

The diagram below describes the Autonomic computing architecture:

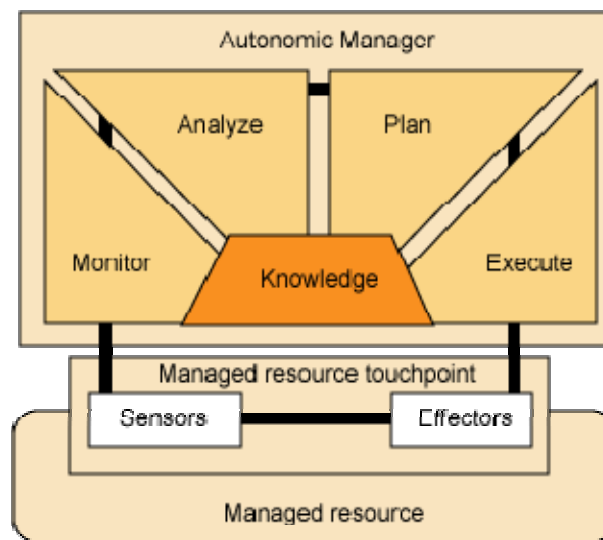


Figure 3: *Autonomic Computing Control Loop*

Our test suite becomes the managed resource and the information about common problems and their solution becomes a part of the Knowledge Base. The autonomic manager would monitor for errors in test suite execution and would analyze the problems. With inputs from the Knowledge Base, it would plan a solution for the problem and would implement the execution of the solution through 'effectors'.

8. REFERENCES

- [1] Implementing the ITCL Framework using Rational Functional Tester, by Aarti Goel.
http://www-128.ibm.com/developerworks/rational/library/06/0822_goel/
- [2] An autonomic computing roadmap, by Nicholas Chase.
<http://www-128.ibm.com/developerworks/library/ac-roadmap/>

Copyright Information

Java is a trademark of Sun Microsystems in the United States, other countries, or both. IBM is a trademark of IBM Corporation in the United States, other countries, or both. WEBSHERE is a trademark of IBM Corporation in the United States, other countries, or both. ZSERIES is a registered trademark of IBM Corporation in the United States, other countries, or both. Rational is a trademark of IBM Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Author Biography



Jayatheerthan is a Tech Lead at IBM India Pvt Ltd, Bangalore, having over 7 years of experience in the IT Industry. He has done his Masters in Computer Applications from Bharathidasan University, Trichy. He is a *Sun Certified Java Programmer* and an *IBM Certified WebSphere Administrator*.

He has played various roles in his career – Developer, Designer, Analyst, Technical Consultant, Tester and a Trainer. He is very fond of designing easy-to-use frameworks for application development. His contributions to his current organization include a state-of-the-art, future-proof framework for test script development, which is well acknowledged and used by test teams across the globe.

His technical areas of interest are Autonomic Computing, SOA, Design Patterns and OOAD.