June, 2009

testing experience

The Magazine for Professional Testers

printed in Germany
print version 8,00 €
free digital version
www.testingexperience.com
ISSN 1866-5705

Security Testing

Project-Based Test Automation

by David Harrison

This article sets out a project-based experience of Automated Testing in the context of software developed with the Java-Swing programming framework. In terms of business domain, the software represented an advanced costing tool for use by underwriters in the re-insurance business. However, we focus here on the *pattern of solution* more than the domain-specific re-insurance aspects.

This article is based on material from the forthcoming book "*Automated Functional Testing for Java-Swing*", to be published by the author in April, 2009.

Structure of Project QA Activities

The structure of the Quality Assurance (QA) activities in this project-based development setting, can be summarized as shown in Figure 1, below:

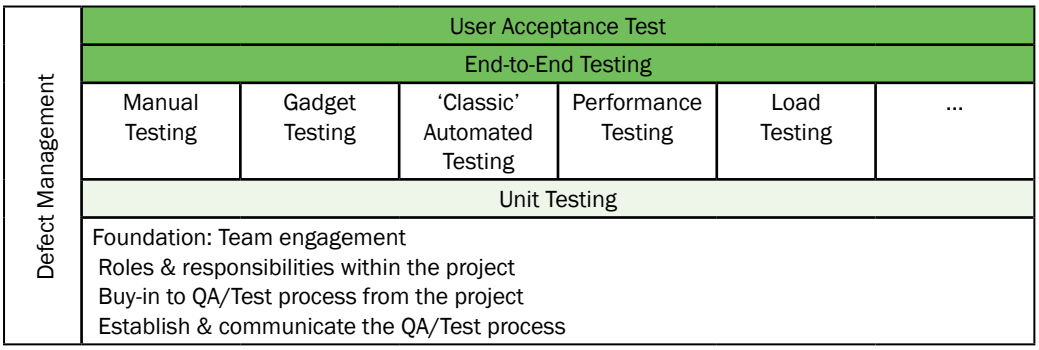

Figure 1 – QA/Test Pattern

In this pattern you can see that the 'classic' automated testing, of the type that we will address here, is only *one part* of the overall Quality Assurance effort. The target of our QA work on this project was, from the outset, to develop a strategy which could be termed Agile, in that it fitted in with the iterative build cycle on the project, as well the highly seasonal nature of the development/deployment process itself.

Let's take a look at each of the parts of this pattern in a little more detail:

Foundation

This part is really crucial to the success of a project-based QA/Test effort. Its role in the pattern is to set out the fundamental project issues which must be achieved in order to maximise the success of the overall QA effort. This naturally affects also the possible successful outcome of any automated testing effort. For an automated testing effort to be given at least some chance of success, the QA/Test team needs strong support from both the IT as well as the business parts of a project. The collaboration from the IT group is particularly important for the pattern presented here, as we need IT involvement in specializing the application slightly to enable access to the internals of the application.

In order for good collaboration to happen, such things as team engagement and having a clearly established set of roles & responsibilities, is *essential*. Having a clear technical process set out is one thing, but this must be communicated to *all project members* by the QA workstream.

Unit Testing

A key part of the process, underpinning all the efforts that follow, is that of Unit Testing. Typically, the developers will use a standard unit testing framework, which for Java would most likely be JUnit[1].

Defect Management

This part of the QA/Test process reflects that part which turns testing into a genuine Quality Assurance process. As hinted in the Foundation part of the pattern, it is important for a development project to recognize the importance of defects, not just as negative things to be got rid of, but as *extremely positive*, naturally occurring items of project life. Indeed, defects are to be aggressively sought within a project if the *deeply negative* characteristics of them are to be avoided that of their impact on the user community if they are deployed to the productive environment.

Manual Testing

Yes – Manual Testing. This is always a theme in the project-based QA/Test story, particularly for projects that are at the very beginning of their development. In these cases, it is quite usual that a large part of the functionality of the software will have to be manually tested. This arises from the unfortunate fact of project life that software in its initial development phase is usually highly volatile in terms of its user interface design, thus precluding automated testing.

Gadget Testing

This part of the testing process is a catch-all for the testing which uses specialist executables that fulfill specialist testing tasks. This form of testing would be used if the software being

1 http://www.junit.org

developed has a mathematical result, for example, which needs to be asserted. A specialist tool which, using known input data, causes the calculation to be performed and then compares the result to a known expected result, would be a good example of a "Gadget".

Classic Automated Testing

This form of testing is the subject of this article. Suffice it to say here that this testing is characterized by exercising the built software through its User Interface (UI). We add here that, from the outset, the task of test automation was seen as essentially a coding task. With this mindset, we looked for "efficiency" in just the same way as any software development workstream would. Often test automation is viewed in quite the opposite way, the coding aspects being set to the background or hidden behind simplistic definitional "spreadsheets" and the like. Taking on board the mental model of "coding" releases us from seeking the false dawn associated with alternative approaches and endlessly struggling with the fundamentals in order to achieve some sort of successful, repeatable outcome.

Performance Testing

In this form of testing we seek to establish that the software meets the performance metrics defined for it, usually per business operation or functionality. There are a number of commercial and open-source tools to help in this area.

In our approach we have used our approach to perform simple performance testing, establishing performance metrics for "Search" from distant global locations.

Load Testing

This form of testing is concerned with ensuring that the software performs as expected under concurrent user load. Often this type of testing is mandated when the software architecture is Client-Server. The Server-side will need to be highly resilient and fail-safe, as the number of "logged-in" users grows and performs specified key business tasks. As in the case of Performance Testing, there are a number of tools on the market for performing this type of test.

Where's the "Agile" Part?

With project-based testing, especially where development is performed in a relatively short time frame, its important that the QA/Test approach and any Automated Testing approach takes account of this project feature. For test automators its important to have a strong technical basis, on which to incrementally extend the range of functionality that is tested; and for this to happen, we need to look for agility in the same way that a software development team might. The interplay between the QA/Test and development workstreams must be based upon iterative builds and a constant flow of new and expected fixed defects at the build points, which in turn underlines the vital role Defect Management has in achieving an Agile process.

This rapid cycling of defects to-and-fro between QA and development is the hinge of our Agile approach, leading to strong defect reduction over the full project cycle. Alternative strategies, such as V-Model/Waterfall cannot render this kind of positive dynamic.

Which Tool & Why

As the product under development was Java-Swing based, the decision was taken to adopt QF-Test[2]. There are a range of tools that could have been chosen, so why this one?

This tool has the following *striking* characteristics which make it stand out from the crowd:

- strong and intelligent connection with the Java VM, object creation and destruction are events that get announced over this connection
- has first-class Exceptions reflecting things happening in the test project itself as well as what is happening in the Software Under Test (SUT)
- has a powerful, extensible Name Resolving architecture, which allows a wide range of name replacements to be performed for cases that are very common, but quite challenging, in real-life software
- has the Jython language as an extension to its built-in programming paradigm (e.g. can instantiate objects from SUT), which enables some very elegant tactics to be employed to get at objects of the UI and their properties
- embraces the use library structure in test projects, which allows a structure to be introduced as between project-specific and generic concerns
- allows the development of data-driven tests using its built-in programming paradigm, which allows a very wide set of data to be used in tests. This represents a significant feature of "scaling-up" in automated testing.

Taking Exception

One of the powerful characteristics of QF-Test is its use of *first-class exceptions* which relate directly to what's happening in the SUT as well as in the test project itself. Being able to code/script tests using real exceptions makes the eventual design very well patterned as well as effective.

What's in a Name?

One of the major difficulties encountered when attempting to automate testing a real-world project context, is the ability to reliably assign names to objects that appear in the UI of the Software Under Test (SUT). QF-Test contains a *very effective and extensible* Name Resolving architecture. A range of visual elements in the SUT of the project in question demanded that the automation tool had a strong capability in dealing with object naming.

2 www.qfs.de

Talking to Objects

The project for which the automated testing solution was developed, involved an advanced graphical UI, in which the "real estate" of visual objects extended beyond the physical viewport of the screen. The extent of this real size was governed by a range of characteristics within the model being displayed. The objects viewed by the user have associated editing dialogs, which naturally are things we as test automators need to get at to perform assertive testing. The resolution of this problem critically rested on the provision of a special object within the main application frame, which contained methods which could be invoked by Jython to open the editor panel for an object specified by its "path" within the overall visual structure, and thus allowed us to gain access to these otherwise off-screen visual objects. The overall visual structure is also available by means of these built-in object methods.

Generic Controls

In order to meet our target of Agile test automation, an early sub-project in the QA workstream, was the development of a Generic Library, which would enable the software-specific workflows to have tests "coded" as fast as possible. We could benefit (as in normal software development practice) from "code" re-use and general applicability, depending only upon the values of parameters. The importance of such a Generic Library cannot be overestimated in our achievement of an Agile automation process.

Where are we?

The pattern of solution for test automation offered here, embracing both the detail of how we perform this often very challenging task, as well as its relationship and fit with other parts of the overall project landscape, has now been in place for a number of years. The pattern takes full account of the fundamental challenges of test automation - which usually render the outcome in alternative approaches to rather less like testing and more like driving the software as a benign and careful "user" - is today fully part of the target project.

The software has undergone, from inception, 4 years of successful global deployment. Test automation was begun with the development of the Generic Library capability as a key precursor to the main task, in version 2 of the software.

The automation project itself is in the process of being handed over to the central support groups for them to use as part of their ensuring that maintenance changes will not adversely affect the correct operation of a crucial business tool which plays a central role in the Underwriters day-to-day activities.

Biography

David Harrison works as an independent software QA/Test Manager – currently at SwissRe, Zurich, Switzerland within the tools development group. This group has the mandate to develop and deploy reinsurance costing tools to the actuary and underwriter desktop. He can be contacted at dharrison_ch(a-t)yahoo(dot)co(dot)uk.
This article is based on material from his forthcoming book: "*Automated Software Testing for Java-Swing; A Pattern of Solution*", to be published in June, 2009.