Evaluation of QF-Test in January 2006 at CoreMedia

1. Preface

QFS asked me in November 2007 if they could publish my evaluation report which I made available to them. I am happy to do so, as I still like QF-Test very much.

Mind that the information you will find refer to the state of QF-Test in January 2006. E. g. it refers to the old name of QF-Test: qftestJUI. In the Epilogue you will find some statements what I think today of QF-Test (and QFS). But now let's start with the evaluation report:

2. Introduction

qftestJUI is a GUI testtool written in Java and with Jython as scripting language. In December 2005/January 2006 we evaluated this tool if it might serve as new GUI testtool for the Swing-Editor of our CMS-product. This document holds the information what caused the decision for or against the new GUI testtool.

The following text is written in present tense as it got written during the evaluation.

3. Target

The target of the evaluation process is to decide if we want to replace (at least for further development) WinRunner as GUI testtool with qftestJUI and to document the reasons for the decision.

4. Motivation

Current state is that WinRunner has some flaws which make it hard to develop and maintain tests:

- **Lack of Robustness with Java Applications:** E. g. mnemonics-replay handling fails at a random amount of times (while it works perfectly for non-Java applications).
- **WinRunner itself and its Frameworks are very complex:** E. g. you have `mtype`, `obj_mouse_click`, `win_mouse_click`, click, `java_fire_event` are just some examples of doing a mouse-click. But: Most of the time only one of them works. So it's hard to find the function which works for you. And WinRunner is not helpful e. g. in suggesting function names and required arguments (the auto-completion feature has to be disabled to a major bug). And even more: Often arguments (of test-scripts) are hidden in extra

dialogs, so you won't see them by default. All this makes it impossible to develop new tests without regularly browsing the code or reading the documentation.

- **Versioning very difficult:** Even if just working text-based it is very hard to integrate changes from one version to another. E. g. by random WinRunner changes the meta-data of a test-script stored in the file header just because you opened the test-script. When submitting changes you don't know if these are required and it even gets worse on Perforce-integrates. The problem increases with the *EMOS Framework*. It uses Excel-spreadsheets to configure test-runs. And there is no easy way to integrate them between versions. Current state is that all integrates are done by hand which is very error-prone.

In addition we currently create new tests from the scratch. I. e. existing tests will just be maintained while new tests for new features will be written with a totally new framework. This is a good point to look for alternatives.

5. Use Cases

Here is a list of use cases we have to take into account when evaluating qftest.

5.1. Actors

- QA-engineers
- Developers
- optional: Technical Editor

5.2. Required Use Cases

- QA-engineers create GUI-tests.
- Developers create GUI-tests.
- Tests are integrated via Perforce to different versions.
- Support or developers record editor-problems which can be replayed by the assignee of the bug which needs to be fixed.
- Java GUI applications need to be tested. Main focus is the Swing-Editor started via WebStart as this is the most problematic use case in contrast to local installation of the Swing-Editor.
- Run Copy & Paste (or Drag and Drop) tests between Java application and non-Java application (e. g. MS Word)
- As well as Java applications we need to test web-applications like Preview Based Editing (PBE) or the WebEditor.
- Complex Test-Suites need to be handled (developed, maintained).
- Test-Suites need to run automatically in nightly builds.
- Significant test-reports need to be generated in HTML.

5.3. Further Use Cases

- Technical Editors might record sequences for taking screenshots.

6. Requirements

List of requirements derived from the Use Cases:

6.4. Killer priority:

- **Java Applications:** It must (of course) be possible to test Java applications.
- **OS:** The GUI testtool must run on Windows 2000, XP and 2003
- **Batch:** It must be possible to run the tests as batch, i. e. without user-interaction e. g. in nightly builds. Especially the program must end if all tests are finished.

6.5. High priority

- **Framework Support:** The tool should support easy development and usage of frameworks and structures: e. g. packages: It should be possible to organize test-scripts into packages to ease finding needed functions.
- **Quick Test Development:** Test development must be possible on the fly, i. e. it shouldn't (always) take more time to develop a feature or fix a bug than to write a test for it. And writing test with the tool should be easy to learn.
- **Stability & Reproducibility:** We must trust the replay mechanism (e. g. Mnemonics: Replay of mnemonics must be possible.)
- **Support:** Quality of Support given by company and other users

6.6. Medium priority:

- **Acceptance:** We need a high developer acceptance of the GUI testtool. E. g. Scripting Language: The scripting language should be widely known to have access to a broad user experience.
- **Perforce:** Integration of test-scripts with Perforce must be possible (and easy). All data for the tests should be stored in text-files which can be "diff"ed (for Perforce Integration).
- **WebStart:** It must be possible to test Java WebStart applications (especially handle default dialogs).
- **Costs:** License, Introduction, Support & Subscription

6.7. Low priority:

- **Web Applications:** It should be possible to test web-applications, too.
- **Java-Access:** Direct access to the Java application e. g. to query the HTML-structure of the JXHTMLTextPane. (Functional tests via JUnit!)
- **Clipboard:** It must be possible to fill Clipboard with content from non-Java applications like MS Word.
- **Report:** It would be nice if a customized HTML-report (with CoreMedia's Corporate Identity) can be created.
- **Additional OS:** Also runs e. g. on Mac and Linux

7. Pros and Contras

This is a list of notes written down during evaluation. Of course it is with respect of the requirements but with no exact mapping towards them.

7.8. qftestJUI alone

Pros

- **Easy to use:** First tests are written very fasten even without reading too much of the documentation.
- **Common Script-Language:** With Jython you can use a very powerful script language with direct access to Java.
- **Easy configurable via Property-Files:** qftest offers by default nodes to read property-files which then can be used to control the behavior of the tests to be run.
- **Developer Feedback:** According to a developer this is a nice tool. Sounds as if the acceptance to the developers would be higher than for WinRunner.
- **No Name Collisions between Packages:** In contrast to WinRunner you don't stumble into name collisions. E. g. in WinRunner you cannot have two TSL-scripts defining a static (somehow comparable to private in Java) function with the same name. Even these private functions have to be unique in the whole project. In qftestJUI you can easily organize everything in packages to prevent such collisions…
- **Packages:** … and the explicit support of packages in qftestJUI is another big plus
- **Easy to find procedures/components:** If you organize your test in different libraries it is very easy to find the procedures to call. Just some clicks, some tabs and you are there. No browsing anymore through unknown code how these procedures might be named.
- **Refactoring I:** To some extend refactoring is supported. I. e. if you carefully handle includes and dependent includes and you rename e. g. a component it is also possible to rename all its occurrences.

- **Refactoring II:** It is very easy to restructure e. g. packages as you can copy/cut & paste a whole bunch of nodes to even another Test-Suite. If the includes are updated, too no other code needs to be refactored e. g. if you move a procedure to another test-suite.
- **Documentation:** Integrated documentation-feature for procedures, i. e. generation of HTML documents.
- **Documentation:** On the fly documentation of the test-suites as every node can have its own documentation. As this is stored in meta-data it does not disturb reading of the code (which in fact is: reading of the code-tree)
- **Company Size:** QFS seems to be a very small company. It seems as if in fact there is only one Person working at QFS right now: Gregor Schmid. This might be a disadvantage (see below) but also an advantage: Gregor Schmid responds very fast on Support Requests and is open for suggestions for changes to qftestJUI. And by the way: Gregor Schmid is very friendly
- **Focus on Java:** While the focus on testing Java applications is also negative (see below) it is also an advantage. Java is very well supported and e. g. mnemonics work perfectly (in contrast to WinRunner) and it can be assumed that with upcoming new Java Versions qftestJUI will very fast adopt to them (at least much faster than WinRunner)
- **Integrated Jython Terminal:** The integrated Jython terminal allows to quickly test Jython expressions and the modules created.
- **Undo/Redo:** Undo/Redo available with no restrictions.
- **Platform Independent:** According to a developer qftestJUI runs on any platform even with Mac OSX. This might be interesting as current efforts are to add Mac OSX to our supported platforms but there are currently no automatic GUI-tests available for this platform.
- **Compact:** qftest offers the possibility to remove unused components. This helps a lot when trying to keep tests compact and easy to maintain. To some extent this is also done in SUT scripts.
- **Framework Support I:** Because of the tree view you can always easy navigate through the code and remove any unnecessary information by collapsing nodes.
- **Framework Support II:** First of all there is a powerful search to find components, raised exceptions, etc. Then e. g. if you want to call a procedure you will get a dialog what procedure to call and what parameters it has. And if you have a procedure call to a procedure included from another suite you can just jump to it with Ctrl+P. All this makes learning of a framework very simple.
- **Timeout Support:** You can easily create sequences with a timeout. And in case the timeout fails you may e. g. raise an exception. In e. g. WinRunner this is hard to implement and often leads to duplicated code.
- **Startup/Teardown:** Startup and Teardown is supported just as in JUnit even more powerful. Startup/Teardown can be used for any sequence of sequences at any level and even in procedures.
- **Manual:** Very good manual. Easy to read and to understand.
- **Mnemonics and Shortcuts:** You can reach many (if not all) functions of the qftestJUI-editor with mnemonics and shortcuts.

- **Wait for a Window/Dialog to disappear:** qftest by default already offers a way to wait for a component to disappear. This makes sometimes synchronization of test-runs much easier.

Contras

- **Cannot handle WebStart-Default-Dialogs:** With the current version 1.08.4 it is not possible to control WebStart's default dialogs like the Security Warnings. This is required for automated tests of WebStart. Received an appropriate patch from QF-Support: qftestJUI-1.08.4-patch8.zip. Install it to qfs\qftest\qftestJUI-1.08.4\qflib. Problem which still exists: The JVM is sometimes restarted between WebStart's default dialogs and the final start of the WebStart application. Current solution seems to work to catch ClientNotConnectedException and reconnect to the SUT. But: The patch still does not allow interacting with error-dialogs e. g. when the server does not run which provides the JNLP.
- **Company Size:** As already mentioned in the Positive part (see above) QFS seems to be only one real Person: Gregor Schmid. While there are some positive points on this, there is also a negative point: The danger that QFS will die is higher than that Mercury Interactive (WinRunner) will do. But we need a long-term solution which is guaranteed to be supported a long time. E. g. with the upcoming Java 6 we need to be sure that qftestJUI will work with this, too, let's say in a year or two.
- **Focus on Java:** While the focus on Java is also positive (see above) it is also negative: You cannot control other applications than Java applications. This is e. g. a problem with some WebStart dialogs and of course a problem if you want to test interaction between the Swing-Editor and non-Java-applications e. g. for Copy & Paste.
- **Cannot handle non-Java-applications:** qftest cannot control non-Java-applications. So it is e. g. not possible to start Copy & Paste- or Drag & Drop-Tests between MS Word and Swing-Editor for example. Workaround: Use bridge2java to fill the clipboard with contents from MS Word File. Copying of files can be simulated by setting the FileList-flavor in Java.
- **Small Knowledge Base:** QFS offers a mailing-list for discussion of problems with qftestJUI. This is comparable to the user forum as it is offered by Mercury for WinRunner. But the number of users which take part in the discussion seems to be rather small and thus the information contained in the archive is very sparse. And: The Mailing-list is the only archive of additional knowledge provided to the users despite the User Manual.
- **Limited GUI-configuration:** While WinRunner offers a lot of properties to identify windows including their activity state, qftest by default has only a limited number of settings. Sometimes this makes creating GUI-components in the suite more difficult. Well, there is an API which might be used to work around this problem. It has not yet been evaluated how complicated it is to use this API.
- **No On-The-Fly GUI-map configuration:** It is sometimes useful to modify GUI-components on the fly while a test runs. E. g. if you login as admin to the Swing-Editor you might change the window-title of the Suite's GUI-component for the User-Manager to match on `.*(admin)`. Perhaps it is possible to simulate this behavior with variables.

8. WinRunner alone

Pros

- **Huge Knowledge Base:** WinRunner has a great support site with a four-level support:
  - Knowledge Base of problems and their solutions managed by Mercury
  - User Knowledge Base: Solutions offered by users of WinRunner
  - Discussion Forum: Forum with a huge number of users who discuss their everyday-work with WinRunner.
  - Support-Requests: Support requests you directly send to Mercury. Support Requests are stored in the Web-Interface by Company so that even other users from CoreMedia can access e. g. my support-requests.
- **Reuse of Test-Suites:** Test-Suites written for the Swing Editor could be reused for the WebEditor. Just the basic implementations of the functions need to change not the test-suite itself (unless things are just not available in the WebEditor like e. g. the User-Manager).

Contras

- **Undo/Redo restricted:** Undo and Redo is only available until next save. If a file is saved, the undo/redo-buffer is cleared.
- **Proprietary Script Language:** TSL is only used by Software from Mercury Interactive (LoadRunner, WinRunner). Thus there is only one editor which can handle it and the number of user-forums is very restricted.
- **Platform Specific:** WinRunner runs only on Windows. An alternative might be Mercury's product XRunner.
- **Perforce:** Most frameworks require having at least some data in Excel spreadsheets. Changes to them are horrible to handle on integration tasks. The same is true with the header-files which contain meta-information of the test-scripts. Often they change without reason and so you cannot be sure if you need to integrate them or not.
- **Crashes during debugging:** WinRunner crashes very often during debugging-sessions. If a test fails (e. g. because of a syntax error) and you want to restart the test after it has been fixed, WinRunner crashes very often. This is especially annoying as you have to close the SysTray-Application manually and all files being open upon crash will be marked as locked which leads to a warning-dialog each time you open such a file.
- **Inconsistent Last Directory:** If you open a test in the editor and want to open another test in the same directory most of the time WinRunner won't start the file-chooser-dialog at this directory but at strange places. To be exact: The start-location is the place where you last saved e. g. a GUI-map-file to. This often leads to a lot of clicks until you are at the directory where you wanted to be. Especially if you need to adjust a bunch of files this is very annoying. A bug-report to Mercury did not reveal much more than "That's normal" (although they raised a feature request).
- **Many Ways which possibly work:** If you have a task to do there are many ways to do it. E. g. to click on an object you may use the function `click`, `mtype`,

`obj_mouse_click`, `win_mouse_click` and then there are also more possibilities depending on the application (e. g. for Java you may also decide to fire events directly). One very time-consuming task is to find the one which works best and is most robust to changes in the GUI. Just an example: I spent nearly one complete day just to find a way to right-click on any given text on a web-page just because not every logically possible way would do.

- **Bad documentation:** WinRunner itself but also the EMOS Framework are very bad documented. While for WinRunner you can use the very good Support Website as quick reference you cannot do that for EMOS. Although there are mailinglists it is hard to find any valuable information. Most has to be derived from the documentation bundled with EMOS which states already in its introduction, that it is outdated.
- **Meta Data clutter Perforce Changes:** Files like header and all .GUI-files often are detected as changed without any obvious reason. The problem is that they don't only store such important information like test-script-parameters (header) or map-entries for components (.gui) but also references to Excel-tables which don't have anything to do with the test-script (header) or the last state of the GUI-Map-Browser (i. e. if the GUI-Map nodes in the tree were opened or not). This makes it hard to decide whether to submit such changes or not (may be some are important) and if we just submit them without review they will clutter integrates unnecessarily

9. Two GUI Testtools

Pros

- **Use Strengths of both Applications:** In fact if one tool is better in critical points but cannot fulfill all required (GUI-)QA tasks we should combine both applications.

Contras

- **Not possible to share tests:** I. e. if we would only use WinRunner we could write a function to create a directory in the Explorer-Window of the Swing-Editor and call this function from many tests. If we then want to test the Web-Editor we just have to adjust this very function and the same tests will also run in the Web-Editor.
- **Lethargical change between applications:** It is very hard to understand the EMOS Framework (which we currently use to organize our WinRunner-Tests) and the extensions we made to it. There is always a warm-up time necessary until you can work fluently with the framework. If we now switch between the frameworks (suggesting that we also have a kind of framework in qftest) it will always take some time before the warm-up time is over. And it is needed nearly every time. Of course it will become shorter but from my experience at least the EMOS Framework will still have at least a warm-up time of a day or two.

10. Decision Matrix

Priority	WinRunner	qftestJUI	Comment
Killer			
Java Applications	+	+	**WinRunner** requires a Java Add-In which lacks support for most recent Java versions. **qftestJUI** has to *instrument* the Java version, i. e. modify the installed Java version by adding hooks to it. For automatic tests this can be done from the command line with new qftestJUI-versions.
OS	+	+	Required OS are supported by WinRunner and qftestJUI. But **qftestJUI** also supports platforms like Linux and Mac OS X
Batch	+	+	
Total	**3**	**3**	
High			
Framework Support	+	+	While both applications may be used to build huge Frameworks for testing only **qftestJUI** helps efficiently in using such frameworks. First it offers *packages* by default (in WinRunner you have to introduce naming conventions for this) and qftestJUI can be used to easily navigate through available functions and even search for them. In WinRunner you have to *remember* the name of the function and its arguments. Other advantages of qftestJUI are built-in documentation-generation and separate name-spaces for functions inside packages.
Quick Test Development	−	+	As already stated in Requirement Framework Support **WinRunner** is very complex and does not really help in finding the functions and their arguments needed, especially in huge Frameworks. To write your first (replayable) test in WinRunner always takes a lot of more time than to do it in qftestJUI.
Stability & Reproducibility	⚠	+	**WinRunner** lacks robustness of test results. One example are mnemonics which are replayed inconsistently. The only workaround in WinRunner is to run a failed test again and only record it as *failed* if both calls failed. This is the current implemented workaround for mnemonics. Up to now there is no such problem with qftestJUI.
Support	+	+	Both supports of Mercury Interactive and Quality First Software (QFS) are very different and both have advantages and

			disadvantages. To sum it up: *Both supports are very good.* **Mercury** offers a huge knowledge-base which often solves a problem but in many different ways (i. e. some times too much information). A ticket-system ensures that you get into direct contact with Mercury's support engineers. But if you need WinRunner to get modified (e. g. to fix a bug) it takes an unpredictable amount of time until this request is fulfilled. In contrast you have a very direct support at **QFS**. It happened to me many times that I either got the answer that the change will be available with the next version and sometimes even that I got a patch just the other day.
Total	**2.5**	**4**	
Medium			
Acceptance			In the past developers at CoreMedia using **WinRunner** often reported uneasiness using WinRunner. The experience of me is that one reason might be that WinRunner offers many ways to solve a given task… but only one of the ways will work. And as record & replay often does not work with Java Applications there is not something like *learning by doing*. In contrast **qftestJUI** already has been evaluated by a CoreMedia developer and has been rated high. And also my experience is that you will have first results very fast as record & replay works perfectly. Thus you have a good start for learning how to work with qftestJUI. In addition the documentation of qftestJUI is very good.
Perforce			**WinRunner** has two main problems which already got described in the "Motivation", i. e. Excel-spreadsheets and unpredictable changes in meta-information of TSL-Sources. This hasn't been evaluated for **qftestJUI** by example but in theory qftestJUI is much better when needing to integrate changes. The tests are stored in XML-files and you can easily move huge functional blocks to Python module files which of course are very easy to integrate. And qftestJUI offers by default functions to read data from (text) properties-files so it's also easy to keep the whole configuration in text-based-files in contrast to Excel-spreadsheets.
WebStart			As already tested by example **WinRunner** is able to start an application via WebStart and to click away default-dialogs like desktop-integration and such. In contrast **qftestJUI** has not been able in the originally evaluated version to handle these dialogs. But I quickly got appropriate patches to work around this problem, which is one big  for the support of QFS.

Costs			Will be evaluated
Total	**1.5**	**2.5+*Costs***	
Low			
Web Applications			**qftestJUI** cannot handle non-Java-applications. Therefore there is no way to test web-applications like the WebEditor or Preview Based Editing in the CMS 2005 Demo.
Java-Access			WinRunner and qftestJUI both are able to access Java-classes and -methods. But in contrast to WinRunner qftestJUI can do this in a very direct way. In case you want to use Java by **WinRunner** e. g. *outside* the tests, e. g. to read property-files, you have to launch any other Java application beforehand. This is possible and currently practiced for latest WinRunner Tests based on the EMOS Framework. In contrast **qftestJUI** offers direct access to Java and all its powers e. g. to read XML-files etc. as it uses Jython as scripting language.
Clipboard			For some tests we need to paste clipboard-contents from external applications like MS Word to the Java application. As **WinRunner** is able to control non-Java-applications it is a piece of cake to replay this scenario. In **qftestJUI** you have to use a workaround which is to use a Java-class to load data into the Clipboard. Of course the disadvantage is that there is no direct way to do so. But the advantage is that you get independent from third-party-products in your tests thus the robustness of tests will increase.
Report			**WinRunner** offers no way of creating HTML-reports. As introduced by me the tests now generate XML-files of the results which than can be transformed via XSLT. **qftestJUI** offers HTML-reports by default as well as XML-reports. So you have a quick way of getting HTML-reports and you have an easy way to generate custom reports by applying your own XSLT-files to the qftestJUI-reports.
Additional OS			**WinRunner** only runs on Windows systems, which are currently our focus. But recent requirements show, that we sometimes also need to run the GUI-tests on other platforms like e. g. MacOS X. This is only possible with **qftestJUI**.
Total	**3.5**	**3.5**	

11. Conclusion

The conclusion has not been made during the evaluation phase as also other options have been taken into account for creating more tests for the Swing Editor. But eventually we decided to buy licenses of QF-Test.

12. Epilogue

Just as the conclusion which was delayed after the creation of the evaluation report this epilogue also is not part of the original evaluation report. Everything else is as-is, i. e. you just read through the original report from now nearly two years ago.

Today we still use QF-Test – and we still use WinRunner, because we didn't manage to get time for converting the tests in WinRunner to tests in QF-Test. There is no way for automatically converting those tests and thus every test has to be converted by hand. But we are on our way. Some tests already moved.

I am still happy we made the decision to switch to QF-Test. While I still have to maintain WinRunner Tests and recently had strong problems with Java 1.6 and Windows Vista QF-Test just works as fine on Java 1.6/Windows Vista as on Windows XP with Java 1.5 or 1.4.

It's always a pleasure to write tests in QF-Test as they are easy to create. To some extent I might say it's my framework I built meanwhile. It allows me to have tests which read in the code tree just like:

1. Create Document "A"
2. Write some Text into the Document
3. Save
4. Do Checks

Where every single line points to a function/procedure in the framework I built up. But it's QF-Test which enabled me to build such a framework.

Also CoreMedia developers who don't write QF-Tests every day feel very comfortable with QF-Test. Most of the times I only need to introduce them for one or two hours and they start right away.

QF-Test still gets better from release to release. So it's no wonder that some flaws I mentioned I fixed meanwhile. As e. g. the "lazy variable expansion bug" as I call it, which was not a bug but a different approach to handle variable expansion. And great new features were introduced like dependencies I use a lot meanwhile.

And one last word about the company size: Of course I totally misjudged the company size and I am happy to know some more people at QFS now.

Mark Michaelis, Software Engineer Quality Assurance, ISTQB Certified Tester