QualiSystems

Object-Oriented Test Automation

Introduction

Today, network and IT-infrastructure testing organizations find themselves in a bind. Although business imperatives such as increased competition, pressurized profit margins, and the search for efficiencies make automation an important and attractive initiative, many test-automation projects encounter significant challenges that lead to a poor return on investment (ROI). At the root of these challenges are the inherent constraints of traditional, scripting-based automation methodologies. An object-oriented test automation architecture empowers dramatic improvements in automation best practices. The result is breakthrough performance in delivering time to value, lowering total cost of ownership (TCO) and increasing the ROI of test-automation initiatives.

The Challenges of Traditional Test-Automation Approaches

Traditional telecom and IT infrastructure test and lab automation projects often experience difficulties in delivering on their promises because they have been based on programmer-generated scripts. Scripts are a great tool for programmers, and may work well for smaller test-automation projects or for testing environments that do not experience many changes. However, since most testing personnel are non-programmers and testing environments experience regular and continuous changes, script-based methodologies suffer from a number of significant drawbacks:

- **Dependence on programmers and high maintenance costs** – scripts are typically coded to automate a multistep process, generating lengthy files of TCL, Perl, Python or other scripting language. These files are lengthy, complex, and thus difficult to maintain, update or repurpose as conditions in the testing environment change. Since only programmers create or update scripts, the entire automation process becomes very tied to individual programmer knowledge. If there is a change that makes scripts out of date, programmers must then spend significant amounts of time to create new version of the scripts. This process is costly in terms of programmer time and becomes a bottleneck in the productivity of the majority of non-programmer testing personnel.

- **No scalability, low penetration** – due to the high cost of maintenance and because most testing environments experience regular changes that demand continuous and time-consuming script revisions, scalability is very difficult to achieve. As a result, many automation projects never go above 10% penetration of testing processes.

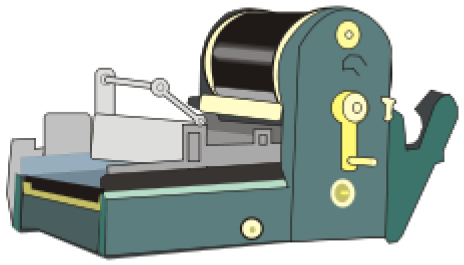
- **Script bloat** – over time, a large number (in fact, the majority of generated scripts) cover only a minority of testing functions. This core of automated tests exists in a multitude of versions that cover minor variations in the ever-changing test environment. The proliferation of scripts becomes a logistical and revision-control problem, which requires the introduction of source version control (SVC) tools. This further burdens programming staff with additional administrative overhead.

- **Vulnerability to disruptions due to changes in personnel** – since the content of the scripts is only known to and maintainable by the programming staff, there is very little systematized knowledge. If programmers change their job or role, their expertise is lost and the viability of the automation project may be threatened.

- **High TCO, low ROI, and project fatigue** – programmers are specialized staff, and therefore expensive. Never-ending, heavy programming requirements make the TCO of traditional automation projects unacceptably high. Project costs become more glaring since due to the low penetration of automation into the body of testing tasks, the overall testing process does not accelerate significantly. This means that the return on investment is poor. Lacking the penetration to reach positive ROI and burdened with a high TCO, automation projects risk losing funding and sustainability.

The Problem with Script-Generating Automation Tools

Cognizant of the disadvantages of pure scripting, some technology vendors have developed automation products that provide a GUI tool to generate multistep scripts through ‘record and replay’ or step-by-step coding methods. The internal code underneath the GUI is typically proprietary, which necessitates throwing away all existing automation scripts and recreating them using the record and replay tool. This is a burdensome cost; but there is an even greater problem with script generators: While the initial script generation may be a bit more user-friendly than pure coding, the script



generator approach to automation still suffers from many of the same high TCO and low ROI problems as programmer-generated scripts.

Script-generator products often try to compensate for some of the complexities of the scripting approach by offering programmer-oriented features like SVC servers to track the myriad of script versions. Ironically, these are usually positioned as highly empowering, when in fact they point to these tools' dysfunctional, traditional roots.

An Object-Oriented Approach to Test Automation

Test organizations can gain a vast improvement relative to the high TCO and low ROI of traditional test automation by implementing an object-oriented model. Instead of creating long, monolithic, hard to maintain scripts, an object-oriented approach enables the capture of all automation elements as building block objects. This includes objects for interfacing with test lab infrastructure resources (compute, storage, network, virtual, cloud), provisioning actions (such as loading OS images), and testing tasks (such as running a traffic load test). An object-oriented architecture offers a quantum leap in maintainability compared to scripts:

- The limited scope of automation objects means that they are easy to capture, maintain, and refactor to meet the requirements of a changing test environment
- A shared library of resource, provisioning and testing objects can be maintained in a systematic fashion. While programmers and data architects are the ideal personnel to build the library, the easy maintainability of the object library reduces risk because there is a greater balance between the expertise residing in the system vs. that in programmers' brains.
- Automation objects can be tagged with arbitrary labels so that they can be easily searched and leveraged by many users from a shared library.

The greatest results come from combining a highly reusable object library with powerful GUI tools. Together, they allow for much more productive and efficient automation processes and practices:

- **Automation driven teamwide** – non-programmers can easily use the object library and powerful GUI tools to drive all day-to-day automation processes. Key automation GUI tool capabilities include:
    - Drag-and-drop test topology design using physical and virtual test infrastructure resource objects
    - Right-click menu-driven provisioning actions from the test topology visualization GUI that leverages provisioning objects
    - Drag-and-drop automation workflow design using a library of user-generated testing objects, and out-of-the-box automation logic objects

- **High levels of reuse** – not only is the object library highly reusable, but generated test topologies, provisioning and test workflows can be saved and shared across teams, promoting a higher level of reuse
- **'Object transfer' vs. knowledge transfer** – going beyond the simple notion of 'sharing' between peers in the same department, an object library approach means that test topologies and workflows provide a highly accurate and efficient method of handing off precise scenarios between developers, architects, QA teams, operations, technical support, field personnel and even customers. This is a huge time saver, as knowledge transfer processes based on verbal descriptions, text writeups, and static diagrams are time consuming and error prone.

These vastly improved processes revolutionize test automation leading to lower TCO and higher ROI:

- Programmer time is maximized, restraining ongoing costs and lowering TCO
- Automation of testing tasks accelerates and achieves very high degrees of penetration – 80% to 90% of tests can be automated
- Testing cycles accelerate speed and expand coverage, leading to a strong ROI, faster time to market and higher quality
- Costly test infrastructure resources are optimally used through improved sharing, leading to huge CAPEX and OPEX savings

Conclusion

As data center and network infrastructures become increasingly virtualized and agile, it is time to evolve testing to achieve similar agility. Traditional script-based approaches cannot achieve the business objectives of technology organizations. A next-generation, object-oriented architecture transforms testing processes and delivers compelling ROI.

QualiSystems offers TestShell, an object-oriented test and lab automation solution that is used by over 100 service providers, technology manufacturers, enterprises, and government agencies worldwide. For more information about QualiSystems, visit our website at http://www.qualisystems.com.