ACHIEVING THE FULL POTENTIAL OF TEST AUTOMATION

TABLE OF CONTENTS

ABSTRACT

Software test automation has the capacity to decrease the overall cost of testing and improve software quality, but most testing organizations have not been able to achieve the full potential of test automation. A number of groups have implemented test automation plans, only to later become disregarded as expensive "shelfware." Often such teams continue their automation efforts, yet become burdened with huge costs in maintain large suites of automated test scripts that are of questionable value.

This paper will first discuss the key benefits of software test automation, and then examine the most common techniques used to implement software test automation. It will then analyze prevalent reasons why test automation efforts fail to meet their potential.

Finally, this paper concludes with an examination in how using a keyword-driven approach to test automation allows organizations to avoid problems inherent in other approaches and benefit from test automation. Action Based Testing™, the latest methodology from the original architect of the keyword method, and the TestArchitect™ toolset will be presented as proven real-world examples of why test automation is the optimal solution.

THE BENEFITS OF SOFTWARE TEST AUTOMATION

Most software development and testing organizations are well aware of the benefits of test automation. A quick glance at the website of any test automation tool vendor will point out a number of the key benefits of test automation.

Some of these benefits include:

Reduced test execution time and cost:

Automated tests take less time to execute than manual tests and can generally execute unattended. A tester must simply start the test and then analyze the results when the test is completed.

Increased test coverage on each testing cycle:

Automated tests can allow testing teams to execute large volumes of tests against each build of their application, achieving a level of coverage that would not be possible with manual testing.

This increased coverage can help teams uncover bugs more quickly than manual testing. Test automation can allow teams to test more features in each cycle (breadth), and also to test features using more permutations of inputs (depth).

Increased value of manual testing effort:

So long as applications are meant for human end users, test automation will never entirely replace the need for human testers. Human testers will instantly notice subtle bugs that are almost never detected by test automation, particularly usability bugs. Automated test tools are not instinctual— and test teams typically discover bugs using exploratory and ad-hoc testing techniques. By freeing manual testers from having to execute repetitive, mundane tests, test automation grants them to use their creativity, knowledge and instincts to discover important bugs.

PITFALLS: WHY TEST AUTOMATION PROJECTS FAIL TO ACHIEVE THEIR POTENTIAL

Despite the clear benefits of test automation, many organizations are unable to build effective test automation programs.

Test automation as seen as a costly effort finding few bugs and questionable worth. There are a number of reasons why test automation efforts are unproductive:

Poor quality of tests being automated:

Mark Fewster explains this problem very well:

It doesn't matter how clever you are at test automating a test or how well you do it, if the test itself achieves nothing faster." [Fewster, Software Test Automation, 1.1, (Addision Wesley, 1999)}

Many organizations simply focus on taking existing test cases and converting them into automated tests.

There is a sense that if 100% of the manual test cases can be automated, then the test automation effort will be a success. In trying to achieve this goal, many organizations may have automated many of their manual tests, but found it to be a huge loss in time and money discovering a few bugs.

The plain fact is that a poor test is a poor test, whether it is executed manually or automatically.

Lack of good test automation framework and process:

Many teams acquire a test automation tool and begin automating as many test cases as possible, with little consideration of how they can structure their automation in such a way that is both scalable and maintainable.

Little consideration is given to managing the test scripts and test results or creating reusable functions or separating data from tests, and other key issues which allow a test automation effort to progress successfully.

LogiGear

After some time, the team realizes that they have large quantities of test scripts and countless separate test result files combined with the additional work of maintain the existing scripts while continuing to automate new scripts.

Ultimately, the organization required a larger test automation team with higher costs resulting in no additional benefit.

Inability to adapt to changes in the system under test:

As the team proceeds towards their goal of automating as many existing test cases as possible, they often don't consider what will happen to the automated tests when the application under test (AUT) under goes a significant change.

In lacking a well conceived test automation framework that considers how to handle changes to the system under test, teams often find the majority of their test scripts will usually result in skyrocketing numbers of false negatives, since the scripts are no longer finding the behavior they are programmed to expect. With teams hurrying to update the test scripts to account for the changes, project stakeholders lose faith in the results of the test automation efforts.

Minimal value in test automation causes a decision to scrap the existing test automation effort and start from scratch, using a more intelligent approach that will produce incrementally better results.

GENERATIONS: TEST AUTOMATION EVOLUTION

Software test automation has evolved through several generations of tools and techniques:

Capture/playback tools record the actions of a tester in a manual test and allow tests to be run unattended from many hours each day, greatly increasing test productivity and eliminating mindless repetition of manual testing. However, even small changes to the software under test require that the test be recorded manually again. Therefore this first generation of tools is not efficient of scalable. Using this method, nontechnical testers and business analysts can develop

executable test automation using "keywords" that represent actions recognizable to end-users, such as "login" while automation engineers devote their energy to coding the low-level steps that make up those actions such as "click'," "find text box A in window B", "enter UserName" etc.

Scripting, a form of programming in computer languages specifically developed for software test automation, alleviates many issues with capture/ playback tools. However, the developers of the scripts must be highly technical and specialized programmers who work in isolation from the testers actually performing the tests. In addition, scripts are best suited for GUI testing and don't lend themselves for embedded, batch, or other forms of systems. Finally, as changes to the software under test require complex changes to the associated automation scripts, maintenance of the ever-larger libraries of automation scripts becomes an overwhelming challenge. Data-driven testing is often considered individually as an important development in test automation.

This approach simply but powerfully separated the automation scripts from the data for input and returned from the software under test.

This allows the data to be prepared by testers without relying on automation engineers, and vastly increases the possible variations and amount of data that can be used in software testing. This break-down of the problem into two pieces is very powerful. While this approach greatly extends the usefulness of scripted test automation, the huge maintenance chores required of the automation programming staff remain.

Keyword-based test automation compartmentalizes work even further in an advanced structures and elegant approach. This reduces the cost and time of test design, automation, and execution by allowing all members of a testing team to focus on what they do best.

Keyword-based test design can actually begin based on documents developed by business analysts or the marketing department before the final details of the AUT are know. As the test automation proceeds, bottlenecks are removed and the expensive time of highly trained professionals is used effectively.

The keyword method cost benefits become even more apparent as the testing continues. When the software under test undergoes changes, revisions to the test and to the automation scripts are necessary.

Organizing test design and test automation with the keyword framework, eliminates time previously allocated to maintaining large libraries of scripts and rewriting entire scripts anew after major changes to the software under test.

With the keyword method, the necessary changes are far fewer. Many changes do not require new automation and can be completed by non-technical testers or business analysts. When required, changes to automated keywords can be completed by automation engineers without affecting the rest of the test. Hans Buwalda, Chief Architect at LogiGear Corporation, developed the keyword automation concept and first presented this subject to the software testing community in 1994. Mr. Buwalda began implementing his ideas in Europe throughout the rest of the 1990s with the TestFrame™ method and tool, and ultimately continuing its development as Action Based Testing™ in the USA.

This method is the foundation of LogiGear's test automation framework, TestArchitect™, which not only organizes scripting around keywords, but also offers built-in actions that make it possible to automate many tests without scripting of any kind.

ACTION BASED TESTING: A PROVEN APPROACH

Action Based Testing (ABT) provides a powerful framework for organizing test design, automation and execution around keywords. In ABT keywords are called "actions "to make the concept absolutely clear. Actions are the tasks to be executed in a test. Rather than automating an entire test as one long script, an automation engineer can focus on automating actions as individual building-blocks that can be combined in any order to design a test.

Non-technical test engineers and business analysts can then define their tests as a series of these automated

keywords, and execute their tests automatically without creating any additional code.

Traditional test design begins with a written narrative that must be interpreted by each tester or automation engineer working on the test.

ABT test design takes place in a spreadsheet, with actions listed consecutively in a clear well-organized sequence. Actions, test data and any necessary GUI interface information are stored in their own spreadsheets from which they can be called by the main test module.

Test are then executed from within the spreadsheets, using Test Architect's built-in automation or a customer-build test harness.

To achieve the try power of Action Based Testing, it is important to use high-level actions whenever possible in test design. High level actions are understandable by those familiar with the business logic of the test. For example, when the user inputs a number the system makes a mortgage calculation or connects to a telephone.

A good high-level action may not be specific to the system under test. "Enter order" is a good high-level step that can be used generically to refer to specific low-level steps that take place in many tests or many different applications. Automation is then completed through the coding of low level actions. TestArchitect usually provides all the low-level actions necessary through its built-in automation feature, so there is typically no need to write any additional code.

Creating the high-level action required by the test design involves dragging-and-dropping a few low-level actions to create that high-level action. The low-level actions behind "enter order" would be the specific steps needed to complete that action via various interfaces such as HTML, the Windows command line, etc. An example of low-level action would be "push-button."

Whenever coding by an automation engineer is required, breaking this work down into reusable low-

level actions saves time and money by making future code changes necessary even when the software under test undergoes major revisions.

A reshuffling of actions is usually all that is required. If more coding is necessary, it involves only rewriting individual actions rather than revision of entire test scripts and the resulting accumulation of a vast library of old automation is significantly reduced. Action Based Testing allows testing teams to create a much more effective test automation framework, overcoming the limitations of other methods.

Full involvement of the Testing Team in the Test Automation:

Most testing teams consist primarily of people who have strong knowledge of the application under test or the business domain, but do not have a background in programming. The team members who are fulfilling the roles of the test automation engineer are often people with a software development or computer science background, but without a strong understanding of testing fundamentals, the software under test, or the business domain.

Action Based Testing allows both types of team members to contribute to the test automation effort by allowing each person to leverage their unique skills to create effective automated tests. Testers define tests as a series of reusable high-level actions. It is them the task of the automation engineer to determine how to automate the necessary low-level actions and combine them to produce the required high-level actions, both of which can often be used in future tests. This approach allows testers to focus on creating good tests, while the automation engineers focus the technical challenge of implementing actions.

Significant Reduction of Test Automation Maintenance:

Many organizations build a significant test automation suite with traditional methods and begin to see benefits but encounter more maintenance efforts when the application changes. Action Based Testing significantly reduces maintenance by allowing users to define their test at the business process level. Rather than defining tests as a series of interactions with the UI, test designers can define tests as a series of business actions. For example, a test of a banking application might contain the actions "open new account,"

"deposit," and "withdraw."

Even with the underlying UI changes, these business processes will still remain the same, so the test designer does not need to update the test. It will be the job of the automation engineer to update the actions affected by the UI changes, and this update will only need to be made once, rather than in multiple test scripts.

Improved Quality of Automated Tests:

In Action Based Testing, test designers follow a top-down approach which ensures that there is a clearly stated purpose for every test. The first step is to determine how the overall test automation effort will be broken down into individual test modules.

Such Test Groupings include:

- Different function areas of the same application.
- Varying test types (positive, negative, requirements -based, end-to-end, scenario-based, etc.).
- Diversified quality attributes under test (business process, UI consistency, performance, etc.

Once the test modules have been identified, the next step is to define explicit test requirements for each module. Test requirements are critical because they force test developers to consider what is being tested in each module, and to explicitly document it. Once the rest requirements are defined, they serve as both a roadmap for developing the test cases in the module, and the documentation for the purpose of the tests. Test cases are associated explicitly to test requirements.

By explicitly stating the test requirements, it is possible to easily determine the purpose of a test, and to determine if a test does not sufficiently meet those test requirements. This process ensures that the tests being automated have a clear purpose that can be used to determine in the future if the test needs maintenance or even retirement.

Test developers can be precise and concise in their test creation, creating enough tests to meet their stated requirements without introducing redundancy.

Many test automation teams spend more time maintaining their existing tests than actually creating new tests. This high maintenance burden is due to the

fact that automated tests are highly dependent on the UI of the UI of the application under test; when the UI changes, so must the test automation. It is usually the case that the core business processes handled by an application will not change, but rather the UI used to enact those business processes changes.

After explicitly defining the test requirements, the test developers implement the test scenario using either predefined actions or by defining new actions. Test developers can define their tests as high-level business processes allowing the tests to be more readable than those defined using low-level interface interactions.

Facilitates Test Automation Strategy

Many testing teams dive into test automation without considering how they should approach test automation. A very typical approach is to acquire a test automation tool, and then try to start automating as many existing test cases as possible. More often than not, this approach is not effective. Action Based Testing provides a framework that integrates the entire testing organization in supporting effective test automation. Business analysts, test engineers, automation engineers, test leads and QA managers all work within the framework to complete test planning, test design, test automation and test execution. With the right framework in place, the organization can respond most effectively to everything from marketing requirements to software development changes.

Enable Effective Collaboration by Distributed Teams

With testing teams often distributed to low-cost areas across the country and around the world, the challenge of sharing information, sharing test
libraries and sharing automation libraries is multiplied many times over.

Action Based Testing provides a strategic framework for organizing tests with a very clear structure enabling a strong measure of control over the disruption caused by distance and time zone differences. TestArchitect™. as a test automation framework supporting the Action Based Testing methodology, takes this to the next level by remotely sharing database repositories of test modules, actions and other test assets, and provides clear control and reporting to managers of access, changes and results.

CONCLUSION

As with other areas of software development, the true potential of software test automation is realized only within a framework that provides a truly scalable structure.

Since its introduction in 1994, the keyword based method of test automation has become the dominant approach in Europe and is now taking the United States by store precisely because it provides the best way to achieve this goal.

Action Based Testing offers the latest innovations in keyword-driven testing from the original architect of the keyword concept. Test design, test automation and test execution are all performed within a spreadsheet environment, guided by a method focused on an elegant structure of reusable high-level actions.

TestArchitect, a test automation framework from LogiGear with features ranging from action organization to globally distributed team management, offers the full power of Action Based Testing to the entire testing organization including business analysts, test engineers, automation engineer, test leads and managers.

LOGIGEAR SOLUTIONS

LogiGear is a leading provider of software testing services, test automation and application development and maintenance. We help organizations deliver better products while saving time and money. Since 1994, we have completed testing projects with hundreds of companies from early stage start-ups to Fortune 100, across a wide range of industries and technologies. For more information visit Logigear.com or contact us at (650) 574-1400 or sales@logigear.com.

TYPES OF TESTING

- Manual
- Automated
- Database
- Exploratory
- Black box
- Gray box
- Whitebox

About *Logi*Gear

LogiGear Corporation provides global solutions for software testing and offers software testing training programs worldwide. LogiGear is a leader in the integration of test automation, offshore resources and US based project management for fast, cost effective results. Founded in 1994 by top thought-leaders in software quality, LogiGear offers the best possible combination of low cost, fast turnaround, and advanced testing expertise. LogiGear serves clients around the world ranging from the Fortune 500 to early-stage startups in a wide range of industries. Be it consulting, training, high-volume and cross-platform test automation, outsourced testing, or automation technology, we partner with organizations to create approaches that precisely meet their needs. For more information on how we can help your business, contact us today.