LogiGear Expert Testing... Real World Solutions

LogiGear® Whitepaper Series

Achieving the Full Potential of Software Test Automation:
The Strategy For Reducing Costs and Speeding Time-To-Market

October 25, 2004

Contents

1. Abstract 1
2. The Benefits of Software Test Automation 2
3. Pitfalls: Why Test Automation Projects Fail to Achieve Their Potential 2
4. Generations: Test Automation Evolution 4
5. Keywords: Action-Based Testing 6
6. Conclusion 10

Diagrams

Figure 1: *The ABT Framework* 6
Figure 2: *Action Based Testing in TestArchitect™* 9
Figure 3: *An action definition in TestArchitect™* 10

1. Abstract

Software test automation can dramatically reduce costs and speed up time-to-market, but its full potential cannot be achieved without a carefully planned, scaleable and maintainable process. Without the right strategy, expensive test automation tools may become "shelfware."

This paper will first discuss the key benefits of software test automation, and examine the most common implementation tactics. It will then analyze the key reasons why test automation efforts often fail to meet their potential. Finally, it will show how using a *keyword-based* test automation strategy enables organizations to avoid those problems.

Action Based Testing™, featuring the latest improvements to the keyword-based strategy from the original architect of the keyword method, and the TestArchitect™ toolset will be presented as real-world examples of how the full potential of test automation can be realized.

 © LogiGear Corporation 2004. All Rights Reserved.

2. 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 Web sites 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 fully execute than manual tests, and can generally run unattended. A tester simply starts the test, and then analyzes the results when completed.

Increased test coverage on each testing cycle: Automated tests enable 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 can help teams uncover bugs in existing functionality much more quickly than through manual testing. Test automation enables teams to test more features in each cycle (breadth), and to test those features with a larger set 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. No matter how sophisticated test automation tools become, they will never be as good as human testers at finding bugs in an application. Human testers will instantly notice subtle bugs that are almost never detected by test automation, particularly usability bugs. Automated test tools cannot 'follow their instincts' to uncover bugs using exploratory and ad-hoc testing techniques. By freeing manual testers from having to execute repetitive, mundane tests, test automation enables them to focus on using their creativity, knowledge, and instincts to discover more important bugs.

3. Pitfalls: Why Test Automation Projects Fail to Achieve Their Potential

Despite the clear benefits of test automation, many organizations are not able to build effective test automation programs. Test automation becomes a costly effort that finds fewer bugs and is of questionable value to the organization.

There are a number of reasons why test automation efforts are unproductive. Some of the most common include:

Poor quality of tests being automated

Experts in the field agree that before approaching problems in test automation, we must be certain those problems are not rooted in fundamental flaws in test planning and design.

> "It doesn't matter how clever you are at automating a test or how well you do it, if the test itself achieves nothing then all you end up with is a test that achieves nothing faster." Mark Fewster, Software Test Automation, I.1, (Addison Wesley, 1999).

 © LogiGear Corporation 2004. All Rights Reserved.

Many organizations simply focus on taking existing test cases and converting them into automated tests. There is a strong belief 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, organizations find that they may have automated many of their manual tests, but it has come at a huge investment of time and money, and produces few bugs found. This can be due the fact 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 it is scalable and maintainable. Little consideration is given to managing the test scripts and test results, creating reusable functions, separating data from tests, and other key issues which allow a test automation effort to grow successfully.

After some time, the team realizes that they have hundreds or thousands of test scripts, thousands of separate test result files, and the combined work of maintaining the existing scripts while continuing to automate new ones requires a larger and larger test automation team with higher and higher costs and no additional benefit.

> "Anything you automate, you'll have to maintain or abandon. Uncontrolled maintenance costs are probably the most common problem that automated regression test efforts face." Kaner, Bach and Petticord, *ibid*, p. 114.

Inability to adapt to changes in the system under test

As teams drive 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 system under test (SUT) under goes a significant change.

Lacking a well conceived test automation framework that considers how to handle changes to the system under test, these teams often find that the majority of their test scripts need maintenance. The outdated scripts will usually result in skyrocketing numbers of false negatives, since the scripts are no longer finding the behavior they are programmed to expect.

As the team hurriedly works to update the test scripts to account for the changes, project stakeholders begin to lose faith in the results of the test automation. Often the lack of perceived value in the test automation will result in a decision to scrap the existing test automation effort and start over, using a more intelligent approach that will produce incrementally better results.

 © LogiGear Corporation 2004. All Rights Reserved.

"Test products often receive poor treatment from project stakeholders. In everyday practice, many organizations must set up complete tests from scratch, even for minor adaptations, since existing tests have been lost or can no longer be used. To achieve a short time-to-market, tests need to be both easy to maintain and reusable." Buwalda, Jannsen and Pinkster, Integrated Test Design and Automation: Using the TestFrame Method, p. 8 (Addison Wesley, 2002).

4. 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 execution, and allow tests to be run unattended, greatly increasing test productivity and eliminating the mind-numbing 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 or scalable.

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 these 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 but don't lend themselves to 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 ever-larger libraries of automation scripts becomes an overwhelming challenge.

Data-driven testing is often considered separately as an important development in test automation. This approach simply but powerfully separates the automation script from the data to be input and expected back from the software under test. Key benefits to this approach are that the data can be prepared by testers without relying on automation engineers, and the possible variations and amount of data used to test are vastly increased. This breaking 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 breaks work down even further in an advanced, structured 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. Using this method, non-technical 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. Keyword-based test design can actually begin based on

 © LogiGear Corporation 2004. All Rights Reserved.

documents developed by business analysts or the marketing department, before the final details of the software to be tested are known. As the test automation process proceeds, bottlenecks are removed and the expensive time of highly-trained professionals is used effectively.

The cost benefits of the keyword method become even more apparent as the testing process continues. When the software under test is changed, revisions to the test and to the automation scripts are necessary. By using a keyword-based framework, organizations can greatly reduce the amount of maintenance needed, and avoid rewriting entire test scripts. Many changes do not require new automation at all, 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, and can be swapped into other tests in the library as needed.

The keyword method has become dominant in Europe since its introduction in 1994, where it was incorporated into the TestFrame method and tool, and is now coming into its own in the USA. LogiGear's Action Based Testing™ represents the continued evolution of this approach under the guidance of the original architect of the keyword method. This method is the foundation of LogiGear's test automation toolset, TestArchitect™, which not only organizes test design and test automation around keywords, but also offers built-in actions that make it possible to automate many tests without scripting of any kind.

Hybrid testing tools merit a brief discussion due to the intense marketing efforts of several vendors that are helping to bring awareness of keyword-driven technologies to the industry. In these products, a keyword-like user interface is layered atop a traditional automated testing tool. Most of these tools simply provide a GUI window listing the library of low-level functions that automation engineers have produced in scripting language. Many attempt to offer "scriptless" automation through the use of GUI views, templates and business rules-based test design. One example emphasizes a graphical interface that enables non-technical users to create tests by specifying low-level actions. The tool then performs automatic code generation in a target scripting language. These canned keyword scripts must be regenerated each time a change is made to test, and direct editing is not recommended.

These hybrid solutions, in attempting to oversimplify test automation engineering, are unable to offer the power, flexibility and customization necessary to automate tests for complex systems. In addition, because these tools implement a keyword-based interface without an underlying testing method supported by the full keyword framework, the manual creation and maintenance of tests is rather labor intensive due to the use of only low-level keywords.

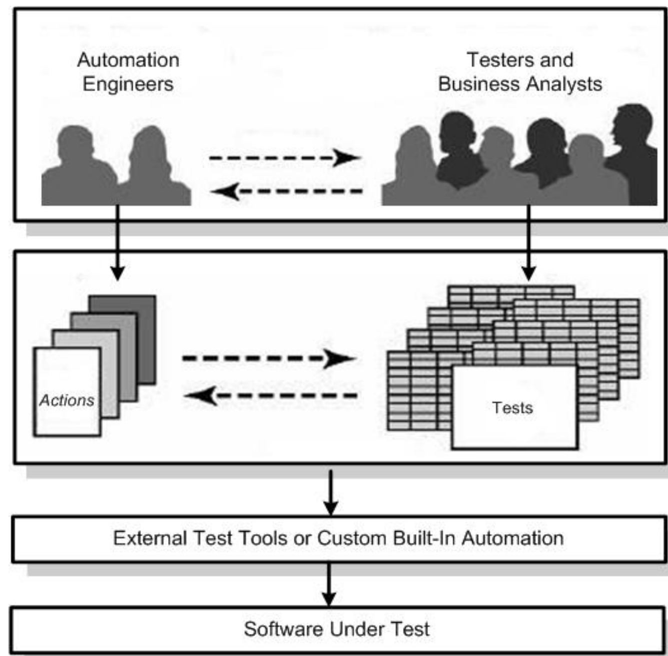
 © LogiGear Corporation 2004. All Rights Reserved.

5. Keywords: Action-Based Testing™

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, concentrating on the test rather than the scripting language.

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 in a clear, well-organized sequence. Actions, test data and any necessary GUI interface information are stored in separate spreadsheets, where they can be referenced by the main test module. Tests are then executed from right within the spreadsheet, using third-party scripting tools or TestArchitect's own built-in automation.

Figure 1: The ABT Framework



To realize the full 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 of many different applications.

Automation is then completed through the scripting (programming) of low-level actions. TestArchitect provides a comprehensive set of the low-level actions necessary through its built-in automation feature. In that case creating a high-level action required by the test design would involve only drag-and-drop of 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 GUI, etc. An example of a low-level action would be "push button".

 © LogiGear Corporation 2004. All Rights Reserved.

Whenever scripting by an automation engineer is required, breaking this work down into reusable low-level actions saves time and money by making future scripting changes unnecessary even when the software under test undergoes major revisions. A reshuffling of actions is usually all that is required. If more scripting is necessary, it involves only the rewriting of individual actions rather than revision of entire automation scripts and the resulting accumulation of a vast library of old automation.

> "The organization develops test standards which can be reused in the next test. The test itself, and the various tasks involved, is therefore more clearly defined. The costs of the test are known in advance and it is clear what has been tested to a specific level of detail. In addition, insight into the approach and the status of the test process can be gained at all times, ensuring that the test process can be adjusted in a timely manner if necessary. This method enhances the quality of both the test process and the test products, resulting in higher quality for the tested system." Buwalda, Jannsen and Pinkster, *ibid*, p. 9.

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 Test Automation

Most testing teams consist primarily of people who have strong knowledge of the application under test or the business domain, but with light expertise in programming. The team members who are fulfilling the role of test automation engineer are often people with a software development or computer science background, but lack a strong expertise in 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 enabling 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 then 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 reused in many future tests. This approach allows testers to focus on creating good tests, while the automation engineers focus on the technical challenge of implementing actions.

Significant Reduction of Test Automation Maintenance

Many organizations build a significant test automation suite using older automation methods and begin to see some benefits, only to get stuck with a huge maintenance effort when the application changes. Test automation teams end up spending 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 application under test; when the UI changes, so must the test automation. It is usually the

 © LogiGear Corporation 2004. All Rights Reserved.

case that the core business processes handled by an application will not change, but rather the UI used to enact those business processes changes.

Action Based Testing significantly reduces the maintenance burden by allowing users to define their tests 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 if 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 often needs to be made in only one place.

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*. Some common ways of grouping tests include:

- Different functional areas of the application.
- Different types of tests (positive, negative, requirements-based, end-to-end, scenario-based, etc.).
- Different quality attributes being tested (business processes, UI consistency, performance, etc.).

Once the test modules have been identified, the next step is to define *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.

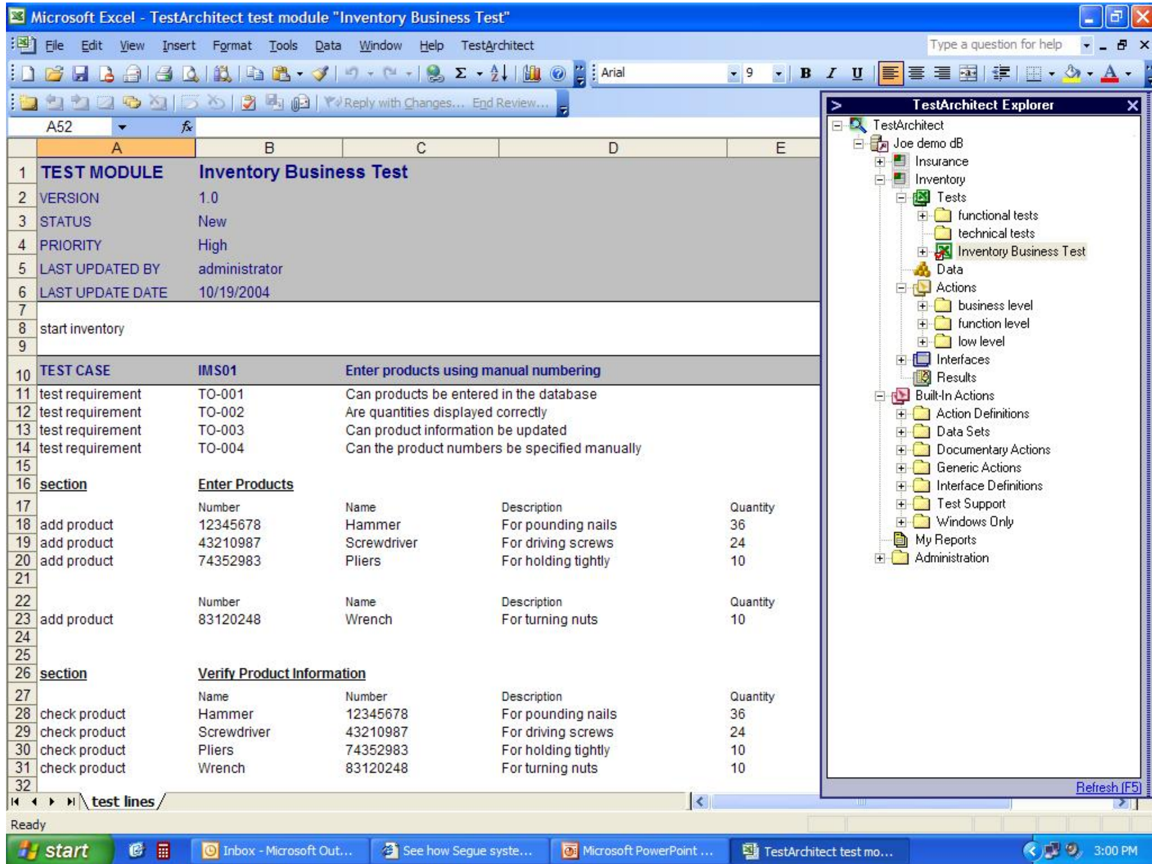
Once the test requirements are defined, they serve as both a roadmap for developing the test cases in the module, and documentation for the purpose of the tests. Each test case is associated to one or more test requirements, and each test requirement should be addressed by one or more test cases.

By explicitly stating the test requirements, it is possible to easily determine the purpose of a test, and to identify if a test does not sufficiently meet those test requirements. Test requirements can be quickly checked to determine 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 unwanted redundancy.

After explicitly defining the test requirements, the Test designers can start implementing the test cases using either predefined actions or by defining new actions. Test designers can define their tests as high-level business processes, which allow the tests to be more readable than tests defined using low-level interface interactions.

 © LogiGear Corporation 2004. All Rights Reserved.

Figure 2: Action Based Testing in TestArchitect™



Facilitates Test Automation Strategy

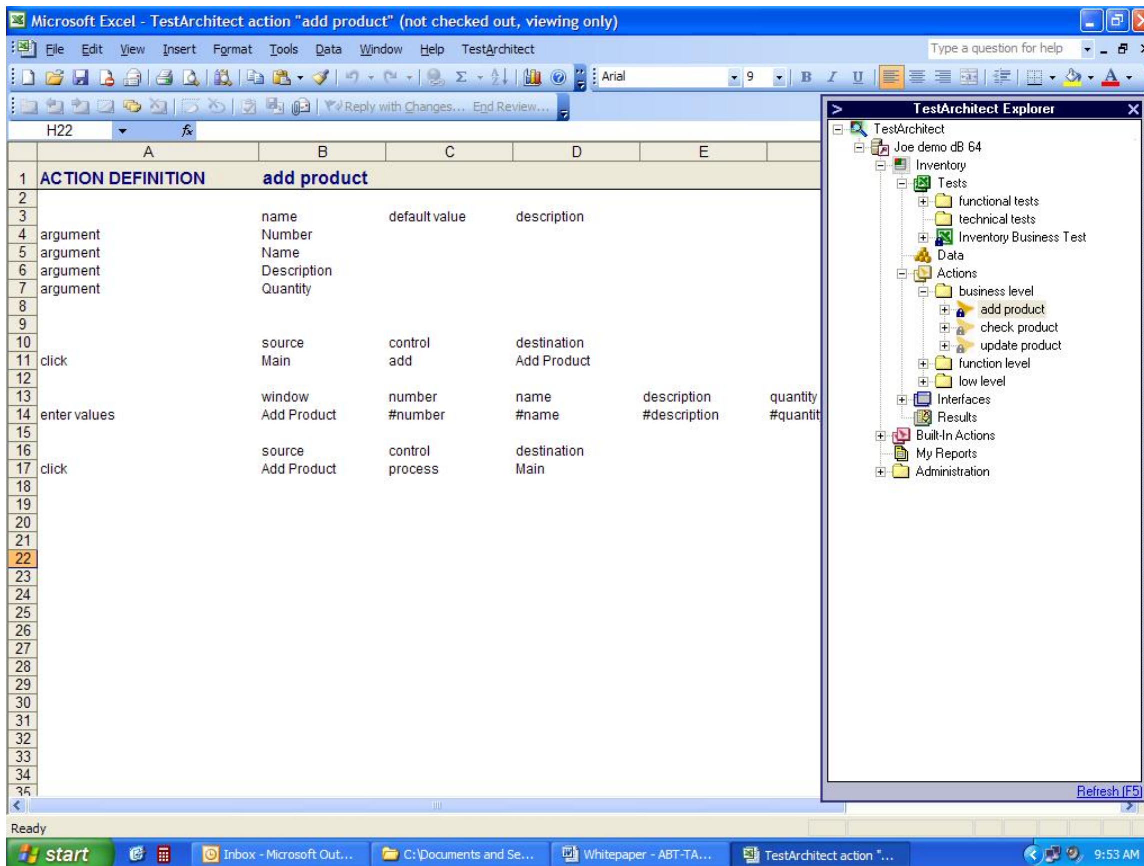
Many testing teams dive into test automation without first considering *how* they should approach test automation. A very typical approach is to acquire a test automation tool, and then try to start automating as many existing test cases as possible. More often than not, this approach is not effective.

Action Based Testing provides a framework that integrates the entire testing organization in support of effective test automation. Business analysts, testers of all kinds, 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.

 © LogiGear Corporation 2004. All Rights Reserved.

Figure 3: An action definition in TestArchitect™



Enables Effective Collaboration by Distributed Teams

With testing teams now often distributed across the country and around the world, the challenge of sharing information, tests and test automation libraries is multiplied many times over. Action Based Testing provides a proven framework for organizing tests and test automation libraries with a clear structure, preventing disruptions that can be caused by distance and time zone differences. TestArchitect™, an ABT-based tool, takes this to the next level by enabling remote sharing of database repositories of test modules, actions and other components, and provides clear control and reporting to managers of access, changes and results.

6. Conclusion

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

 © LogiGear Corporation 2004. All Rights Reserved.

its introduction in 1994, keyword-based test automation has become the dominant approach precisely because it provides the best way to achieve this goal.

Action Based Testing offers the latest in keyword-based 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. The result is a truly scalable, maintainable, reliable software test automation process that allows all contributors in a testing organization to focus on what they do best.

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

For more information about LogiGear®, Action Based Testing™ and TestArchitect™, please visit www.logigear.com, call 1-800-322-0333 or e-mail abt@logigear.com.

 © LogiGear Corporation 2004. All Rights Reserved.