

Speed Without Consistency Undermines QA Effectiveness

Ethan Givoni, CTO, SmarteSoft, Inc.

QA managers are under intense pressure to keep up with aggressive software development schedules. For years, QA has relied heavily on automated tools to meet this relentless need for speed.

However, speed alone is no longer the major challenge for QA. It is also the constant and accelerating pace of software enhancements. As companies make both tweaks and major updates to their software products, the challenge for QA becomes not only speed, but also consistency and repeatability. This is where traditional QA test tools are falling down on the job and impacting the jobs of QA departments. Because they are the last line of defense against flawed software designs, QA engineers are treated as culprits for not catching bugs before the product is released.

In this paper, we examine some of the issues related to the current generation of test automation methodology and tools, and propose an approach that catapults test automation into the same league of speed, reliability and performance as modern software development tools.

The Status Quo

The leading test tools have been around for two decades or more and have changed little in their underlying architectures. Software test environments on the market today fall into three broad categories:

- Record & Playback is a relatively easy technique, but it produces scripts that are rigid and inflexible. When the application under test (AUT) changes even in a minor way, the scripts require re-recording.
- Manual Scripting is used by many QA departments to try to automate testing and create more flexible scripts by manual coding of numerous functions, but the result is a complex process that involves script code development, debugging and extensive code testing. This greatly increases the maintenance burden associated with the test scripts, as well as the dependence on QA testers with programming knowledge. In addition, these manual script codes cannot be applied to the AUT until the scripts are thoroughly debugged.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

- Manual execution of test cases is often used when QA engineers cannot run automated scripts for certain parts of the target application due to extreme maintenance issues, or because the QA department has never invested in test automation. Manual testing slows down the testing process and also introduces the potential for errors. In addition, because manual testing requires existing knowledge of the AUT, it increases the QA department's reliance on senior QA engineers. When there is staff turnover, the QA department can be at a significant disadvantage with manual test cases.

QA departments continue to compensate for the limitations of these tools with sheer guts and determination, muscling through long hours writing and running test scripts. QA managers need to impose a much tougher definition of "automation" on their test tool vendors. They need a testing suite that is not just fast, but provides consistency and repeatability to support the initial software release *and* all of the successive enhancements, in other words, across the entire lifecycle of the AUT.

A Better Approach

The following are the top three attributes that software test environments should provide:

1. The elimination of subjectivity and variability across tests and testers
In reality, today's "automated" test tools are heavily dependant on the skills of the QA staff. Because there is no underlying methodology built into the test tools to support consistency, each engineer may record scripts and program functions differently, making script creation completely subjective and inconsistent. Noted software testing expert Bret Pettichord has been quoted as saying that record & playback testing tools are "a great idea that rarely works."¹ Subjectivity results in:

- Unexpected and undesirable customization
- Increased maintenance burden
- Slower Time-to-Test cycles
- Unpredictable results
- Sloppy, hard to interpret documentation
- Dependence on the knowledge and skills of individual engineers
- Slow ramp-up time for new engineers

¹ "Seven Steps To Test Automation," Bret Pettichord, originally presented at STAR West, San Jose, November 1999

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

2. A dramatic reduction in the number of scripts generated

With today's complex software applications, business processes in the AUT may branch into other processes. Traditional test environments do a very poor job of regression testing. As the software branches, recorded scripts overlap and repeat the same processes and this proliferation of scripts becomes increasingly difficult to handle. Because of time-to-market pressures, QA engineers may not have enough time to run all test scripts in a regression set. As a result, the test parameters become increasingly inconsistent and harder to duplicate and maintain from release to release.

3. The integration of maintainability into testing

The continuous stream of new application releases routinely breaks existing test scripts. Often QA engineers have to spend significant time modifying, re-recording, debugging and updating test script suites. The number of scripts compounded by the number of application changes will often force QA engineers to discard large numbers of scripts and begin script development again.

Traditional QA tools are based on legacy architectures that continue to impose these heavy penalties on QA departments. The more complex and long-lived the software applications, the more inadequate these tools are becoming. Market leading companies are recognizing the strategic value of modernizing their QA testing environment. The key architectural requirements of a modern QA tool suite are:

1. Built-in methodology
2. Data-driven progression
3. Built-in maintainability

QA test environments that incorporate a built-in methodology and data-driven progression provide QA organizations with the following benefits:

- Greater consistency and predictability for ongoing software releases.
- Reduced Time-To-Test cycles.
- Significantly lower maintenance compared with hard-coded recorded scripts.
- Ensures near 100% coverage with minimal number of scripts (counted in tens rather than hundreds and even thousands for a typical software application).
- Scripts are well documented and clearly identifiable, which makes transfer of knowledge on a project seamless, specifically during times of test ramp ups, staff expansion or downsizing.
- Easily integrates testing with the development lifecycle as advocated by Agile and Extreme Programming proponents.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

- QA engineers can rely on manual QA testers and business analysts to help streamline the testing process by pre-creating the test templates even before the application hits the QA testing cycle.

A built-in methodology is the key to consistency

In order to create efficient and consistent scripts, QA engineers must adhere to a common methodology. A simple methodology-driven approach, as described below, eliminates subjectivity by taking advantage of databases and spreadsheets in conjunction with object and data mapping.

First, the QA engineer defines the test flow in a block diagram (as shown in Figure 1) based on the major functional test requirements and then defines the test flow of all the independent modules. This isolates parent processes (i.e., the main business processes) from child processes (i.e., multiple sub-process under a main parent process), which will simplify the test cases refactoring process of the regression test down the line. Each block in the diagram represents one or more business process of the AUT as well as describes the functionality to be tested, the function library, data files, and the database template it uses.

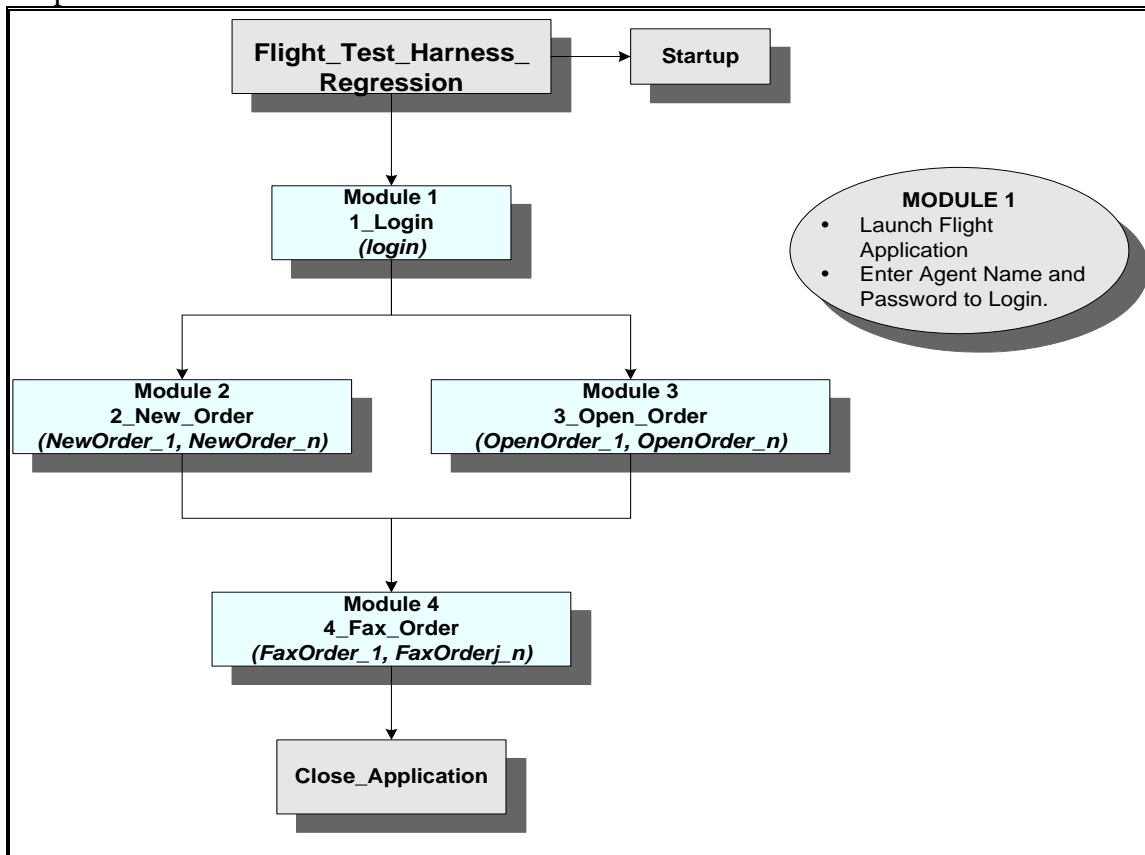


Figure 1: The block diagram provides a well-defined, organized and documented test

Copyright 2009 by SmarteSoft, Inc.

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

process that helps eliminates subjectivity by centralizing the test process flow so that all QA engineers follow the same approach.

Next, the QA engineer creates test cases for the test scripts. Test cases are the building blocks for testing each module. Each test case may encompass a single, independent business process. For example, the QA engineer may identify 20-25 modules in a small-size application versus 40-50 modules mid-size applications. Very large-scale ERP applications (such as SAP, Peoplesoft, Oracle financials, etc.) may require greater than 100 modules per business unit (e.g., Accounting, Inventory, Shipping, Manufacturing, HR, etc). In this case, department modules can be organized as sub-modules, so that each top-level operation will have sub-modules for its internal business processes. This operation enables executing regression sets based on business functions.

Note that with a modern testing suite the process is optimized to *minimize* the number of test cases that need to be generated. The fewer the test cases, the more consistent the testing process overall.

Data-driven progression reduces the number of test scripts

Progressive data-driven scripting is the essence of true QA automation. A progressive data-driven test replaces all hard-coded values with string variables. This is an approach called “parameterization.” When all objects and data are completely parameterized, there is no need to revise the scripts, only the database tables, forms or spreadsheets. This greatly reduces the time associated with testing subsequent software releases. When additional test data is required for testing, data can be imported from the AUT database directly into the test data files/tables. In contrast, record & playback scripts require connecting scripts to data files, but the correlation of the data tables to the GUI objects is not visible to the user, resulting in guesswork and script debugging.

For example, Figure 2 shows how a data-driven progressive approach truly automates the QA test process. First, all of the objects in the AUT are arranged in an Access database or spreadsheet. Next, test data is imported into the tables. Because each row can be configured as a test case with different data and objects, it becomes easy to configure as many test cases as necessary to achieve the level of test coverage required. Fewer scripts are created by refactoring all of the test cases and optimizing the flow and data being tested. Here are two examples of how refactoring can eliminate test case overlapping:

Example 1: When every test needs to login to the AUT every time and perform different test functionality, the login function becomes a separate test that may be called when required.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

Example 2: When a test case requires branching into another business process to perform additional functionality and then return to the original business process, the branched business process becomes a separate test that may be called when required.

Data Of Flight	Fly From	Fly To	Flights Table - Schedules				Name	Class	Tickets
06/30/07	Denver	London	17596	LON	06:57 PM	LAX 07:41 PM NW	John O'donnell	First	2
05/30/07	Frankfurt	Sydney	17596	LON	06:57 PM	LAX 07:41 PM NW	Brian Smith	Economy	1
05/30/07	Los Angeles	Seattle	19130	LON	12:48 PM	LAX 01:33 PM AA	Joe Smith	First	1
			17731	LON	05:45 PM	LAX 06:29 PM NW			
			19122	LON	08:00 AM	LAX 08:45 AM AA			
			19126	LON	10:24 AM	LAX 11:09 AM AA			
			19130	LON	12:48 PM	LAX 01:33 PM AA			
			19133	LON	03:12 PM	LAX 04:42 PM DL			
			19134	LON	03:12 PM	LAX 03:57 PM AA			
			19135	LON	03:12 PM	LAX 06:42 PM AF			
			19225	LON	02:09 PM	LAX 02:53 PM NW			

Figure 2: Test data in Access database/spreadsheet grid.

Note that each row in the grid becomes a different test case scenario where various data types may be used:

- 1) Positive data, i.e., the 'happy' path of the application where the data is correct and the test case is expected to pass;
- 2) Negative data, i.e., the 'unhappy' path of the application where the data is incorrect and the test case is expected to fail;
- 3) Boundary data, i.e., the data string characters or length may be varied and the test case is expected to fail.

The correlation process requires associating the test data in the database or spreadsheet for each row with the test script. However, to save on script coding, most of the correlation can be done within the database by creating Forms. Figure 3 correlates all the objects in the table of Figure 2 to the mirror-image of the application UI arranged in the database form, making the interaction with the test setup friendlier and easier to configure by any user. The end-result is that the script retrieves data from the database tables and uses it to drive the test during execution.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

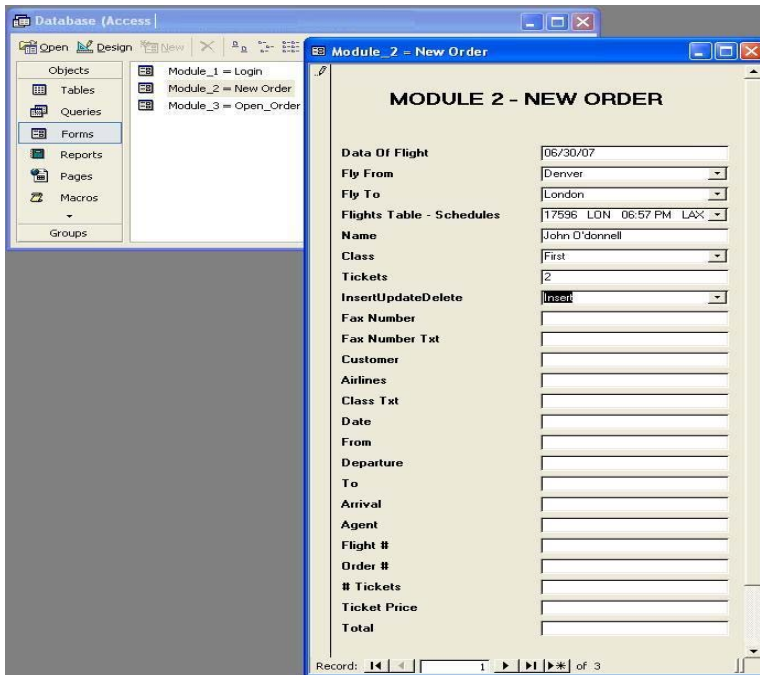


Figure 3: A mirror of the Application UI in Access database

Integrating maintainability into testing supports applications extensibility

A testing suite must be easy to maintain and flexible enough to expand as the AUT changes throughout the application lifecycle. It is important to make the business process flow visible to the user in the database table or spreadsheet at all times. The benefit of this attribute of the modern test suite environment is that it enables any testing resource to quickly visualize and identify what the test case is doing without the need to view coded scripts.

To reduce maintainability and increase testing flexibility, the QA engineer develops a list of objects in an object tree for all GUI objects that take part in the target business process under test, as shown in Figure 4. Creating the object tree translates the logical name of the object and window in the application under test to physical descriptions that the operating system uses.

	Parent Object	object	label
▶	windows taskbar	Push button	Start
	windows taskbar	Menu Item	Run...
	Run Window	Text Box	open:
	Run Window	Push Button	OK
*			

Figure 4: GUI object information contained in Access table/spreadsheet grid

Copyright 2009 by SmarteSoft, Inc.

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

Once the object tree is constructed, it is connected to the Access database tables. When the test script runs, data can be put into edit fields, a push/radio button can be activated and combobox items can be selected as in the application. The QA engineer may implement the code based on the methodology to create the test harness with parameterized scripts for every module (business process) previously identified. In addition, function libraries and startup functions are created to automatically invoke the application, load the object tree, execute the scripts, and compare and report the results.

As the fully parameterized scripts have been developed, this test harness model is ready to execute scripts and test the AUT with different object/data permutations. If object/data selection changes due to new application version, any user can update the object tree map according to the application new version, add, delete, and change names, as well as rearrange the test flow in the database/spreadsheet templates.

Finally, the QA engineer develops function libraries which report test status based on the test results indicating pass (green text) or fail (red text) in a test report. As shown in Figure 5, the report can then be expanded to include details about the type of failure(s), iteration, expected versus actual results, objects state at the time of test, etc.

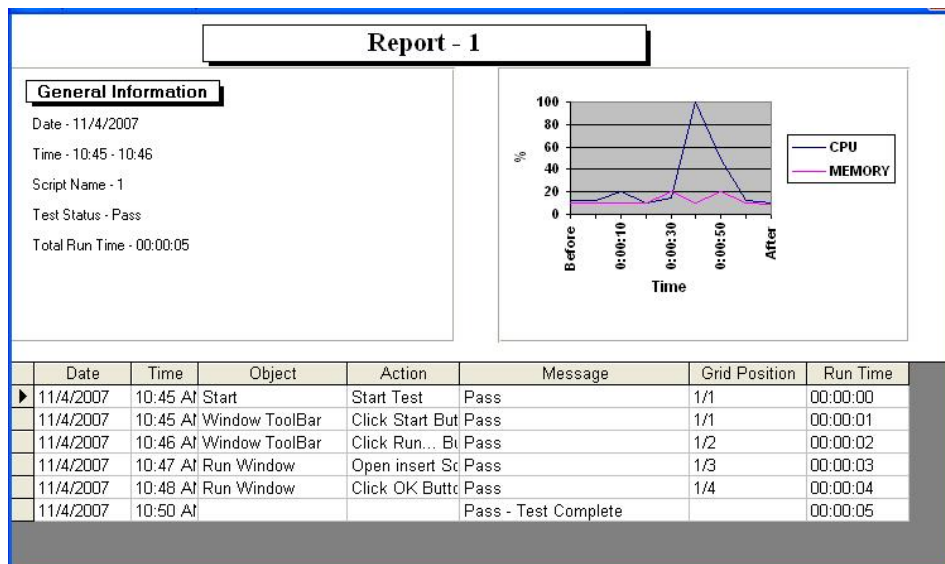


Figure 5: The results of execution of the script

If the AUT changes or expands (when developers are adding new functionality), this modular approach can be easily expanded to add the new modules to the test suite reporting functions and test the new application functionality.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

Case study

The case study outlined below is based on an enterprise application that produces proprietary language for web applications, prevalent in today's marketplace.

Traditional Record/Playback Scripting:	Methodology-based Testing Approach:
<ul style="list-style-type: none"> • Develop 1,500 Test Cases & Scripts 	<ul style="list-style-type: none"> • 50 Test Scenarios* <p>* Test Scenario = Business Process that encompasses many test cases with object & data permutations</p>
<p>Costs:</p> <ul style="list-style-type: none"> • 1,500 test scripts = 1,500 man hours • Average man hour = \$40.00 • Actual cost = \$60,000.00 	<p>Costs:</p> <ul style="list-style-type: none"> • 50 Test Scripts = 100 man hours • Average man hour = \$40.00 • Actual cost = \$4,000.00
<p>Maintenance: Every minor application change or new defects found in the application and/or the testing generate a cost factor.</p>	
<p>Costs:</p> <p>Average 0.5 hours per script maintenance</p> <ul style="list-style-type: none"> • 1,500 test scripts = 750 hours • Average man hour = \$40.00 • Actual cost = \$30,000.00 	<p>Costs:</p> <p>Average 0.5 hour per script maintenance</p> <ul style="list-style-type: none"> • 50 test scenarios = 25 hours • Average man hour = \$40.00 • Actual cost = \$1,000.00
<p>Total costs using Record/Playback Scripting \$90,000.00</p>	<p>Total costs using Test Harness \$5,000.00 Savings: \$85,000</p>

The above case study used SmarteScript™ to implement the methodology based approach, and another popular automation tool for the traditional approach. SmarteScript™ integrates and embodies the methodology-based testing approach, and enables QA engineers to create new test cases as easily and quickly as editing a spreadsheet, delivering five times the speed of scripting. This modern test environment provides all the flexibility, automation and coverage of scripting and none of the complexity, subjectivity and maintenance burden. SmarteScript™ is being used by leading companies in transportation, retail, healthcare, financial services and other industries, providing documented savings of over 60%.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.

Conclusion

The software world has changed dramatically since the current generation of test automation tools were first developed in the early 90's. Software development has sped up enormously, and software has become easier to develop and less error prone, but test automation has not kept up and has not changed much in that period. The advent of methodology based testing (as embodied in tools like SmarteScript™ and SmarteLoad™) brings test automation up to date in a giant step, making the approaches used by most of the industry today, and the generation of products that currently dominate this space, relics of a bygone era. Modern test suites give QA departments the ability to handle QA demands with confidence and deliver the assurances that their companies depend on. This enables QA departments to easily keep pace with software development, minimize bugs and costly reworks, especially with sophisticated applications with frequent updates and upgrades. It is time for the QA industry to move on to its next phase of efficiency, speed, and reliability.

If you have any comments or suggestions regarding this document, please send them via e-mail to ethan.givoni@SmarteSoft.com. Ethan is always happy to discuss his passion, test automation, with anyone, at any time.

Copyright 2009 by **SmarteSoft, Inc.**

All rights reserved. All text and figures included in this publication are the exclusive property of SmarteSoft, Inc., and may not be copied, reproduced, or used in any way without the express permission in writing of SmarteSoft. Information in this document is subject to change without notice and does not represent a commitment on the part of SmarteSoft, Inc.

This document may contain Registered Trademarks, Trademarks and Service Marks that are owned by their respective companies or organizations. SmarteSoft disclaims any responsibility for specifying which marks are owned by which companies or organizations.