DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

January 2011 | Volume V

TestArchitect™ – AUTOMATION TOOLSET

by Hans Buwalda

TEST DESIGN FOCUSED ON EXPEDITING FUNCTIONAL TEST AUTOMATION *by David W. Johnson*

"Automated software testing applied with a return-on-investment strategy built collaboratively with subject matter and testing experts, provides speed, agility and efficiencies to the software development team. " ***Elfriede Dustin***

Read her interview inside!

HAPPY NEW YEAR 2011

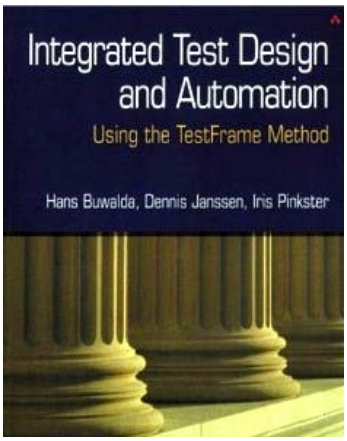
www.logigearmagazine.com

IN THIS ISSUE

January 2011 | Volume V

CONTENTS

Hans Buwalda, CTO, LogiGear Corporation



In Brief

Feature Article

LogiGear presents its new automation toolset with TestArchitect ™. The product uses Action Based Testing as the foundation for its architecture to provide testers a software overseeing all facets of testing including design, automation and management.

Related Articles

David W. Johnson explains the factors needed for a test design paradigm that best expedites functional test automation with a conclusion the piece focusing on how to appropriately implement the paradigm in both an agile and waterfall system development lifecycle.

Hans Buwalda introduces his theory of "Three Holy Grails of Test Design" emphasizing the importance of an effective breakdown of tests, the correct approach for each test module and appropriate test specification.

10 Spotlight Interview

Elfriede Dustin of Innovative Defense Technology discusses her views on test design, scaling automation and the current state of test automation tools.

EDITOR'S LETTER

Happy New Year from LogiGear to those of us who celebrated New Years on January 1! And for our lunar calendar followers, an almost Happy New Year come February 3rd.

We look forward to an exciting and full 2011 as its predecessor was a tough year for many in the software business. At LogiGear Magazine, we view 2011 as a brighter year filled with innovation and advancement. We continue to bring you current articles and video on the latest trends in software quality.

Our editorial calendar is filled with engaging monthly themes addressing software developments, ongoing research and leading individuals in the industry. We are pleased to announce that our upcoming issue will feature a ground-breaking series from Hans Buwalda on Agile Test Automation. As an influential and respected visionary, his articles will highlight the direction of software testing.

As it is difficult to keep pace with the speed of which technology advances, I would like to invite those who have a story in mind and interested in publishing to our readers–be it an interesting perspective or memorable experience on software testing, we are interested in publishing articles or columns from professionals in the field. Our focus for 2011 is getting the next generation of software quality experts published and in the news! Please visit our website as we have just posted our 2011 editorial calendar. Our submission guidelines and contact information are also online.

As the new year moves forward and software testing progresses, we at LogiGear Magazine strive to cover the highlights along the way.

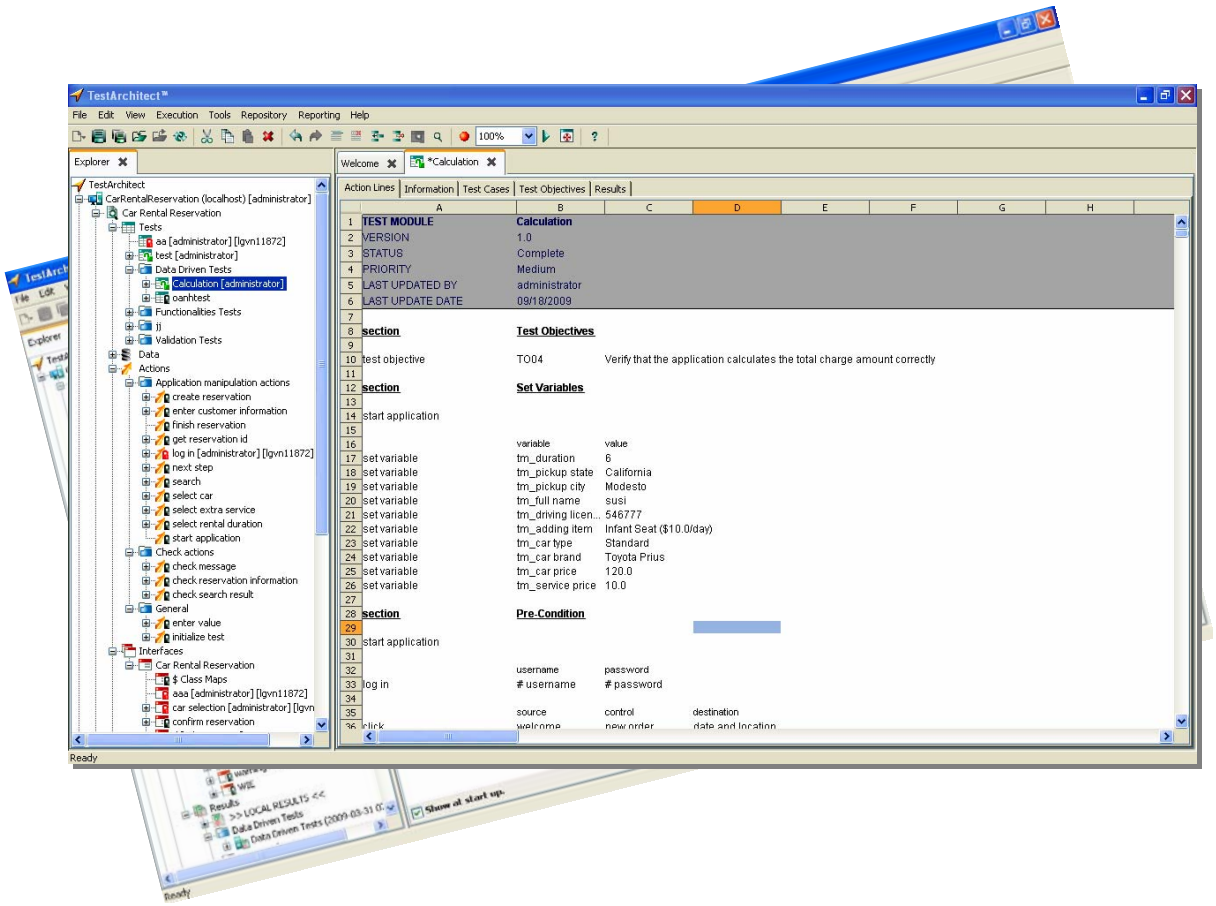
Michael Hackett
Senior Vice President
Editor

In next month's issue

- **Exploratory Testing Using Test Automation** by Anne-Marie Charrett
- **Spotlight Interview:** Jonathan Kohl, Kohl Concepts Inc.
- **Are testers' ethnographic researchers?** by John Stevenson

EDITORIAL CALENDAR

- **February:** Exploratory Testing
- **March/ April:** Agile Test Automation
- **May:** Test Process Improvement
- **June:** Test Methods and Metrics
- **July:** Agile Testing
- **August:** Multi-platform Automation
- **September:** Offshoring or Outsourcing Software Testing
- **October:** Regression Testing
- **November:** Mobile testing and mobile test automation
- **December: Improving Capability through** Test and QA Training



TestArchitect™

AUTOMATION TOOLSET by *Hans Buwalda*

TestArchitect™ is the name we have given to our automation toolset. It reflects the vision that automated testing requires a well-designed architectural plan allowing technical and non-technical elements to work fluidly in their capacity. It also addresses the continual missing link of all test automation tools of how to design tests. In TestArchitect the test design is located in the center column flanked by the test automation execution engine and test management.

What is often misunderstood is that automated testing is not the same as automating manual testing. For automated testing to be successful, tests need to be designed with their automation already in place. In my experience, it is this simple missing factor that is the source of many test automation failures.

To tackle automated testing we have come up with a vision that consists of three elements:

- an effective keyword-driven method for test design and automation called Action Based Testing
- a toolset to support—TestArchitect—that has all the features commonly found in test tools including a shared repository organization and flexible web-based "dashboards" for managing test projects. However, this article will only focus on Action Based Testing aspects
- a set of guidelines and best practices that can be found at our website, www.logigear.com/hans/

Action Based Testing method was a concept devised 16 years ago as a complex yet very critical project that is now widely used in software testing. The tests are clustered in "test modules", and are described as a series of "actions" varying from inputting a value, clicking a button or capturing and verifying a value. The actions are named with "action keywords" or "action words" and have arguments for the input and expected output values. Part of a test module is also a set of "test objectives" that detail the goals of the test. Action Based Testing contains many directions on how to determine the test modules and the test objectives, but that falls beyond the scope of this article.

A key concept in Action Based Testing is that the automation efforts don't focus on the tests but solely on the actions as named by their action words. In most projects a relatively small set of actions can be used to describe a large set of tests. The result is that the tests are readable for humans and at the same time are fully automated. The automation is very maintainable, since in the case of a change in the system under testing generally only the implementation of one or more actions has to be modified; most of the tests can be left alone.

Do you need TestArchitect to do Action Based Testing? Not necessarily. In fact, a simple interpreter written in the scripting language of a test playback tool can do the job quite well. The outline of such an interpreter would consist of:

"repeat
read a test line
look up the keyword
execute the function mapped to the keyword
write results in the result file
until no more test lines"

TestArchitect, however, was developed to facilitate Action Based Testing test development. It organizes the test modules and the actions, and has tools similar to a test editor to quickly create the action based tests. Also popular is the feature of "action definitions", where existing actions can be used to create new ones without additional programming.

An added feature allows actions to be shared across test projects by having a project "subscribe" to another "supplier" project. The subscriber can then use all actions of the supplier project freely and any project the supplier itself subscribes to are automatically available to the subscribing project. Subscriptions can be recursive and mutual: a supplier can subscribe to yet another project, in which case those actions are available for its subscribers as well, and even a supplier project can subscribe to one or more of its own subscribers.

The underlying concept in Action Based Testing is to design tests explicitly and to not rely on recording of tests. However, for practical reasons TestArchitect does come with a recording tool called the "Action Recorder." Instead of generating scripts this tool produces action lines that can be inserted anywhere in a test module or action definition.

Another important aspect of TestArchitect is test result management. After a test run, results are initially stored locally. The user must take an explicit action to store them in the repository visible to others and later have an impact on metrics and reports. This process prevents repositories cluttered with results of the numerous ad hoc dry-runs users supervise during testing or automation development.

Once in the repository, results are further organized with proper naming or storage placement. Lastly, results are linked to the test modules and test cases they have tested.

Automation in TestArchitect is organized as a set of libraries that work completely separate from the test management and test design components allowing you to program the actions in virtually any programming language. It is also possible to execute the tests in a third party playback tool using its scripting language to program the actions.

All in all, TestArchitect is an ecosystem for testers to design, automate, and manage tests grounded in its core design of Action Based Testing. Without this method, I believe it's just another playback similar to its counterparts. It's my firm belief that TestArchitect serves to improve the quality and implementation of tests. This to me is at the very core of our primary objectives as testers: to write great tests that find meaningful bugs and help improve overall application quality.

Test design focused on expediting functional test automation

By David W. Johnson

Test organizations continue to undergo rapid transformation as demands grow for testing efficiencies. Functional test automation is often seen as a way to increase the overall efficiency of functional and system tests. How can a test organization stage itself for functional test automation before an investment in test automation has even been made? Further, how can you continue to harvest the returns from your test design paradigm once the test automation investment has been made? In this article we will discuss the factors in selecting a test design paradigm that expedites functional test automation. We will recommend a test design paradigm and illustrate how this could be applied to both commercial and open-source automation solutions. Finally, we will discuss how to leverage the appropriate test design paradigm once automation has been implemented in both an agile (adaptive) and waterfall (predictive) system development lifecycle (SDLC).

Test design - selection criteria

The test design selection criteria should be grounded in the fundamental goals of any functional automation initiative. Let us assume the selected test automaton tool will enable end-users to author, maintain and execute automated test cases in a web-enabled, shareable environment. Furthermore, the test automation tool shall support test case design, automation and execution "best practices" as defined by the test organization. To harvest the maximum return from both test design and test automation the test design paradigm must support:

- Manual test case design, execution and reporting
- Automated test case design, execution and reporting
- Data-driven manual and automated test cases
- Reuse of test case "steps" or "components"
- Efficient maintenance of manual and automated test cases

Test design – recommended paradigm

One paradigm that has been gaining momentum under several guises in the last few years is keyword-based test design. I have stated in previous articles that:

"The keyword concept is founded on the premise that the discrete functional business events that make up any application can be described using a short text description (keyword) and associated parameter value pairs (arguments). By designing keywords to describe discrete functional business events the testers begin to build up a common library of keywords that can be used to create keyword test cases. This is really a process of creating a language (keywords) to describe a sequence of events within the application (test case)."

The Keyword concept is not a silver bullet but it does present a design medium that leads to both effective test case design and ease of automation. Keywords present the opportunity to design test cases in a fashion that supports our previous test design selection criteria. It does not guarantee that these test cases will be effective but it certainly presents the greatest opportunity for success. Leveraging a test design paradigm that is modular and reusable paves the road for long term automation – not only that, it moves most of the maintenance to a higher level of abstraction: the keyword. The keyword name should be a shorthand description of what actions the keyword performs. The keyword name should begin with the action being

performed followed by the functional entity followed by descriptive text (if required). Here are several common examples:

- Logon User - Logon User
- Enter Customer Name - Enter Customer Name
- Enter Customer Address - Enter Customer Address
- Validate Customer Name - Validate Customer Name
- Select Customer Record - Select Customer Record

Test design – keyword application

Keyword test case design begins as an itemized list of the test cases to be constructed –usually as a set of named test cases. The internal structure of each test case is then constructed using existing or new keywords. Once the design is complete, the appropriate test data (input and results) can be added. Testing the keyword test case design involves executing the test case against the application or applications being tested.

At first glance this does not appear to be any different than any other method for test case design but there are significant differences between keyword test case design and any freehand/textual approach to test case design. Keyword test case designs are:

- Consistent – the same keyword is used to describe the business event every time
- Data Driven – the keyword contains the data required to perform the test step
- Self Documenting – the keyword, description contains the designers' intent
- Maintainable – with consistency comes maintainability
- Automation -- supports automation with little or no design transformation (rewrite)

Test design – adaption based on development/testing paradigm

There are two primary development and testing approaches being used by development organizations today: adaptive (agile) and predictive (waterfall/cascade). Both approaches certainly have their proponents–though the increasingly adaptive (agile) system development lifecycles are gaining precedence. The question becomes how does this affect the test design paradigm? The answer appears to be that it really does not affect the test design paradigm but it does affect the timing.

Predictive (waterfall/cascade) development lifecycles can be supported by a straight-forward design, build, execute and maintain test design paradigm that may later support automation. Eventually, one would expect the predictive testing team to design, build, execute, maintain and automate their test case inventory. This could be accomplished using both Tier 1 commercial automation tools and open source automation tools. As long as the automation tools support modular based design (functions) and data driven testing (test data sources) keyword-based automation can be supported–the most significant difference being the time and effort required to implement the testing framework. Adaptive (agile) development lifecycles come in several flavors–some support immediate keyword-based functional test design and automation while others do not. Agile test driven development (TDD) using FitNesse™, a testing framework which requires instrumentation by and collaboration with the development team, certainly supports keyword-based test case design and automation. Other agile paradigms only support instrumentation at the unit test level or not at all, i.e. a separate keyword-based test case design and automation toolset must be used. The challenge for non-TDD agile becomes designing, building, executing and maintaining functional tests within the context of a two to four week sprint. The solution is a combination of technique and timing. For the immediate changes in the current sprint, consider using exploratory testers and an itemized list of test cases with little, if any content–basically a high-level check list. Once the software for a sprint has migrated to and existed in production for at least one sprint, a traditional set of regression test cases can be constructed using keywords. This separates the challenge into sprint-related testing and regression testing.

KEY PRINCIPLES OF TEST DESIGN

By Hans Buwalda

Test design is the single biggest contributor to success in software testing. Not only can good test design result in good coverage, it is also a major contributor to efficiency. The principle of test design should be "lean and mean." The tests should be of a manageable size and at the same time complete and aggressive enough to find bugs before a system or system update is released.

Test design is also a major factor for success in test automation. This is not that intuitive. Like many others, I initially also thought that successful automation is an issue of good programming or even "buying the right tool." Finding that test design is the main driving force for automation success is something that I had to learn over the years–often the hard way.

What I have found is that there are three main goals that need to be achieved in test design. I like to characterize them as the "Three Holy Grails of Test Design"–a metaphor based on the stories of King Arthur and the Round Table as the three goals are difficult to reach mimicking thestruggle King Arthur's

knights experienced in search of the Holy Grail. This article will introduce the three "grails" to look for in test design. In subsequent articles of this series, I will go into more detail about each of the goals.

The terminology in this article and the three subsequent articles are based on Action Based Testing (ABT), LogiGear's method for testing and test automation. You can read more about the ABT methodology on the LogiGear web site. In ABT, test cases are organized into spreadsheets which are called "test modules." Within the test modules the tests are described as a sequences of "test lines," each starting in the A column with an "action" while the other columns contain arguments. The automation in ABT does not focus on automating test cases, but on automating individual actions which can be re-used as often as necessary.

The Three Goals for Test Design

The three most important goals for test design are:

1. Effective breakdown of the tests

The first step is to breakdown the tests into manageable pieces, which in ABT we call "test modules." At this point in the process we are not yet describing test cases; we simply place test cases its its appropriate "chapters." A break down is good if each of the resulting test modules has a clearly defined and well-focused scope which is differentiated from the other modules. The scope of a test module subsequently determines what its test cases should look like.

2. Right approach per test module

Once the break down is done, each individual test module becomes a mini-project. Based on the scope of a test module we need to determine what approach to take to develop the test module. By approach, I mean the choice of testing techniques used to build the test cases (like boundary analysis, decision tables, etc.) and who should get involved to create and/or assess the tests. For example, a test module aimed at testing the premium calculation of insurance policies might need the involvement of an actuarial department.

3. Right level of test specification

This third goal is deciding where you can win or lose most of the maintainability of automated tests. When creating a test case try to specify only the high-level details that are relevant for the test. For example, from the end-user perspective "logi" or "change customer phone number" is one action; it is not necessary to specify any low-level details such as clicks and inputs. These low-level details should be "hidden" at this time in separate, reusable automation functions common to all tests. This makes a test more concise and readable, but most of all it helps maintain the test since low-level details left out will not have to be changed one-by-one in every single test if the underlying system undergoes changes. The low-level details can then be re-specified (or have their automation revised) only once and reused many times in all tests.

In ABT this third principle is visible in the "level" of the actions to be used in a test module. For example, in an insurance company database, we would write tests using only "high-level" actions like "create policy" and "check premium," while in a test of a dialog you could use a "low level" action like "click" to see if you can click the OK button.

Regardless of the method you choose, simply spending some time thinking about good test design before writing the first test case will have a very high payback down the line, both in the quality and the efficiency of the tests.

Elfriede Dustin of Innovative Defense Technology, is an author of various books on testing including *Automated Software Testing, Quality Web Systems*, and her latest book *Effective Software Testing*.

Elfriede Dustin

LogiGear: *With Test Design being an important ingredient to successful test automation, we are curious at LogiGear Magazine how you approach the problem? What methods do you employ to insure that your teams are developing tests that are ready or supported for automation?*

Dustin: How we approach test design depends entirely on the testing problem we are solving. For example, we support the testing of mission critical systems using our (IDT's) Automated Test and Re-Test (ATRT) solution. For some of the mission critical systems, Graphical User Interface (GUI) automated testing is important, because the system under test (SUT) has operator intensive GUI inputs that absolutely need to work – we approach the test design based on the GUI requirements, GUI scenarios, expected behavior and scenario based expected results.

Other mission critical systems we are testing have a need for interface testing, where hundreds or thousands of messages are being sent back and forth between the interfaces – and the message data being sent back and forth has to be simulated (here we have to test load, endurance and functionality). We might have to look at detailed software requirements for message values, data inputs and expected results and/or examine the SUT's Software Design Documents (SDDs) for such data so we can ensure the test we are designing, or even the test that already existed, are sufficient to permit validation of those messages against the requirements. Finally, it's of not much use if we now have automated the testing of thousands of messages, and we have added to the workload of the analysis. Therefore, the message data analysis needs to be automated also and again a different test design is required here.

Many more test design approaches exist and need to be applied depending on the many more testing problems we are trying to solve, such as testing web services, databases, and others.

ATRT provides a modeling component for any type of test, i.e. GUI, message-based, combination of GUI and message based and more. This modeling component provides a more intuitive view of the test flow than mere text from a test plan or in a test procedure document. The model view lends itself to producing designs that are more likely to ensure the automated test meets the requirements the testers set out to validate and reduces rework or redesign once the test is automated.

LogiGear: *What are your feelings on the different methods of test design from Keyword Test Design, Behavior Driven Test Development (Cucumber Tool) to scripted test automation of manual test case narratives?*

Dustin: We view keyword-driven or behavior-driven tests as automated testing approaches, not necessarily test design approaches. We generally try to separate the nomenclatures of test design approaches from automated testing approaches. Our ATRT solution provides a keyword-driven approach, i.e. the tester (automator) selects the keyword action required via an ATRT-provided icon and the automation code is generated behind the scene. We find the keyword / action-driven approach to be most effective in helping test automators and in our case ATRT users build effective tests with ease and speed.

LogiGear: *How have you dealt with the problems of scaling automation? Do you strive for a certain percentage of all tests to be automated? If so, what is that percentage and how often have you attained it?*

Dustin: Again the answer to the question of scaling automation depends [if we] are we talking scaling automated testing for message-based / background testing or scaling automation for GUI-based automated testing. Scaling for message based / background automated testing seems to be an easier problem to solve than scaling for GUI-based automated testing.

We don't necessarily strive for a certain percentage of tests to be automated. We actually have a matrix of criteria that allows us to choose the tests that lend themselves to automation vs. those tests that don't lend themselves to automation. We also build an automated test strategy before we propose automating any tests. The strategy begins with achieving an understanding of the test regimen and test objectives then prioritizes what to automate based on the likely return on investment. Although everything a human tester does can, with enough time and resources, be automated, not all tests should be automated. It has to make sense, there has to be some measurable return on the investment of automation before we recommend applying automation. *"Will automating this increase the probability of finding defects?"* is a question that's rarely asked when it comes to automated testing. In our experience, there are ample tests that are ripe for achieving significant returns through automation so we are hardly hurting our business opportunities when we tell a customer that a certain test is not a good candidate for automation. Conversely, for those who have tried automation in their test programs and who claim their tests are 100% automated my first question is "What code coverage percentage does your 100% test automation achieve?" One hundred percent test automation does not mean 100% code coverage or 100% functional coverage. One of the most immediate returns on using automation is the expansion of test coverage without increasing the amount of test time (and often while actually reducing test time), especially when it comes to automated backend / message testing.

LogiGear: *What is your feeling on the current state of test automation tools? Do you feel that they are adequately addressing the needs of testers? Both those who can script and those who cannot?*

Dustin: Most of the current crop of test automation tools focus on Windows or Web development. Most of the existing tools must be installed on the SUT thus modifying the SUT configuration. Most of the tools are GUI technology dependent. In our test environments we have to work with various Operating Systems in various states of development, using various types of programming languages and GUI technologies. Since none of the currently provided tools on the market met our vast criteria for an automated testing tool, we at IDT developed our own tool solution mentioned earlier, ATRT. ATRT is GUI technology neutral, doesn't need to be installed on the SUT, is OS independent and uses keyword actions. No software development experience is required. ATRT provides the tester with every GUI action a mouse will provide in addition to various other keywords. Our experience with customers using ATRT for the first time is that it provides a very intuitive user interface and that it is easy

to use. We also find, interestingly, that some of the best test automators with ATRT are younger testers who have grown up playing video games coincidentally developing eye-hand coordination. Some of the testers even find working with ATRT to be fun taking them away from the monotony of manually testing the same features over and over again with each new release.

With ATRT we wanted to provide a solution where testers could model a test before the SUT becomes available and/or see a visual representation of their tests. ATRT provides a test automation modeling component.

We needed a solution that could run various tests concurrently (whether GUI tests or message based/backend/interface test) on multiple systems. ATRT has those features.

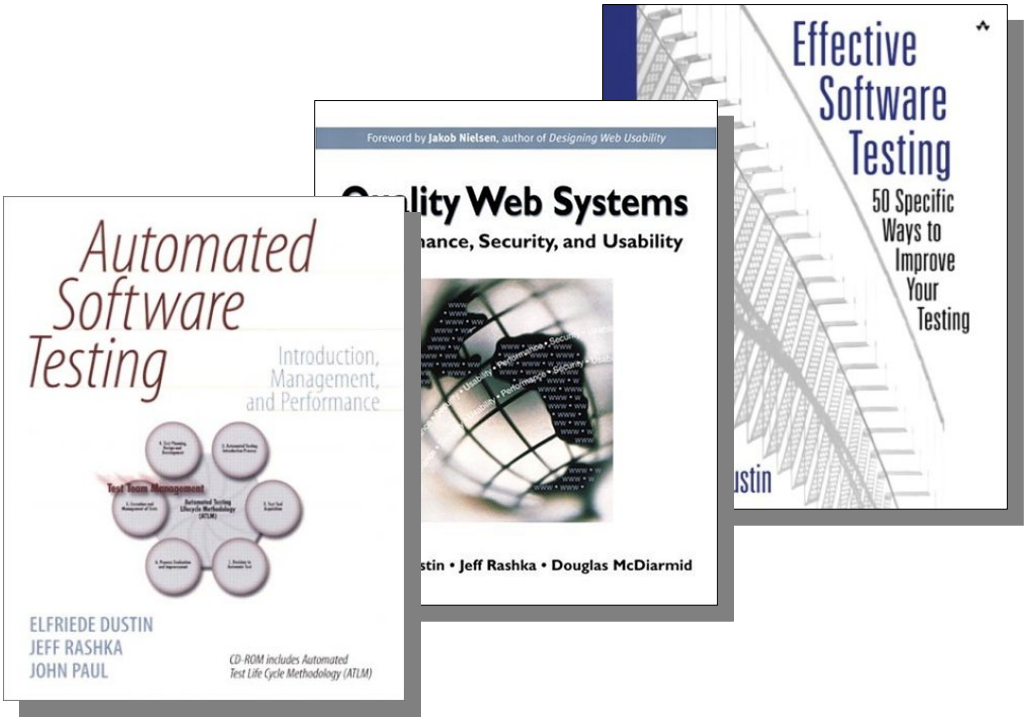
A tester should be able to focus on test design and new features. A tester should not have to spend time crafting elaborate automated testing scripts. Asking a tester to do such work places him/her at risk of losing focus on what he/she is trying to accomplish, i.e. verify the SUT behaves as expected.

LogiGear: *In your career, what has been the single greatest challenge to getting test automation implemented effectively?*

Dustin: Test automation can't be treated as a side-activity or nice-to-have activity– whenever we have some time let's automate a test. Automated testing needs to be an integral, strategically planned part of the software development lifecycle (that is a mini-lifecycle itself) and cannot be an afterthought. See our books on "Automated Software Testing" and "Implementing Automated Software Testing" on the automated testing lifecycle.

LogiGear: *Finally, what is the greatest benefit that test automation provides to a software development team and/or the test team?*

Dustin: Systems have become increasingly complex. We can't test systems the same way we did 20 years ago. Automated software testing applied with a return-on-investment strategy built collaboratively with subject matter and testing experts, provides speed, agility and efficiencies to the software development team. When done right, no software or testing team can test anywhere near as much, so quickly, consistently and repeatedly as with automated testing.



AUTOMATION TOOLS

IDT
Innovative Defense Technologies

What is Automated Test and Retest (ATRT)?

IDT's ATRT Test Manager addresses the complex testing challenges of mission critical systems by providing an innovative technical solution that solves the unique testing problems associated with this domain. It provides an integrated solution that can be applied across the entire testing lifecycle.

Highlights of ATRT:

- **Non-intrusive to the System under Test (SUT) -** SUT configuration remains intact
- **Cross-platform and cross-OS compatible –** Web, Client Server, Handheld and Linux, Windows, Solaris
- **GUI Technology neutral –** Independent of any GUI controls (custom or non-custom)
- **Scriptless Automated Testing –** Software development not required
- **Data driven –** Allows same test scenarios to be run repeatedly with different data sets, e.g. boundary testing, equivalence partitioning
- **Model driven Automated Testing –** Allows test flows generated via basic point and click model interface

Other Features:

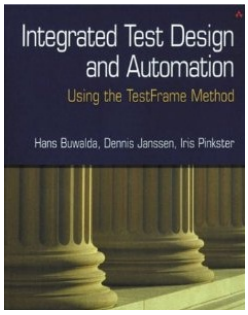
Combining the power of GUI and non-GUI automated testing: ATRT Test Manager provides GUI and message based testing capability for Systems Under Test (SUT).

Distributing concurrent testing over a network: Automated tests can be executed concurrently over a network for test cases where various GUI or message based outputs are network dependent network or must run in parallel.

Batch Processing: For test cases requiring simultaneous execution as a batch job, e.g. endurance or longevity testing.

SOFTWARE TESTING BOOKS

Integrated Test Design and Automation: Using the Testframe Method



Authors: Hans Buwalda, Dennis Janssen, Iris Pinkster, and Paul Watters
Paperback: 224 pages
Publisher: Addison-Wesley Professional, 1 edition (28 December 2001)
Language: English
Product Dimensions: 9.1 x 7.3 x 0.6 inches

Editorial Review:

*This practical guide enables readers to understand and apply the TestFrame method–an open method developed by the authors and their colleagues which is rapidly becoming a standard in the testing industry. With the aid of this book, readers will learn how to * customize the TestFrame method for their organizations * develop reusable testing standards * make optimum use of automated testing tools * reuse and maintain test products * IT managers will learn how to improve the control the test process and assess results, and expert testers will learn effective ways of automating test execution in a structured way.*

Zero-defect software is the holy grail of all development projects, and sophisticated techniques have now emerged to automate the testing process so that high-quality software can be delivered on time and on budget. This practical guide enables readers to understand and apply the TestFrame method.

REVIEW: *"The Test Automation Bible",* March 21, 2002 - By David Edwards

This exceptionally well written volume on automated testing should be read by EVERY software engineer and project manager, not just the Q&A team. Hans Buwalda, world expert on the TestFrame method, has written an excellent overview which is contemporary and explains the Greek metaphor very clearly. I highly recommend the examples given in the book.

SUBMIT YOUR ARTICLE TO LOGIGEAR MAGAZINE:

SUBMISSION GUIDELINES

LogiGear welcomes submissions from new authors, QA and test engineers. We encourage the next generation of thinkers and practitioners in software quality and testing to be published in LogiGear Magazine. Articles can be original works or previously posted articles on blogs, websites or newsletters but within a relevant timeframe. Stories can take form as features, tips and hints, testing instructions, motivational articles and testing about software testing practices and industry trends.

Please note:

1. Feature articles must be 500 words or more, Tips and Tricks must be 200 words or less, and Tool & Technology must be 500 words or more.

2. Only submissions from original authors will be accepted. By submitting this material, you have acknowledged that the material is legal and can be redistributed (book publishers or a public relations firm will need to reference this information at the top of the article). Such articles will not be compensated and will not be accepted if it can be interpreted as advertisements. However, you may place a brief resource box and contact information (but no ads) at the end of your article. Please specify if you wish to have your e-mail included for readers' response.

3. Due to staffing, editing of such submissions is limited and therefore send only the final draft of your work. After an article is received, it may be revised before publishing.

4. Submissions may or may not be used. It is the sole discretion of LogiGear Magazine's editorial staff to publish any material. You will be notified in writing if your article will be published and what edition it will appear in.

For additional information about our guidelines or to submit an article, please send an email to lolita.guevarra@logigear.com