DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

October Issue - Volume 3

THE REAL COSTS/BENEFITS OF TEST AUTOMATION

By Frits Bos, PMP, Pm4hire

Feature Article:

Testing in Agile Part 4:
SKILLS and TRANING

"The biggest challenge in software assurance is how to do it economically." –

D. Richard Kuhn, Computer Scientist, National Institute of Standards & Technology

HAPPY HALLOWEEN

www.logigearmagazine.com

LogiGear MAGAZINE

IN THIS ISSUE

October Issue - Volume 3

Letter from the Editor

Hello everyone – I'm hoping each one of us is having a great October. This time of the year is always my favorite, with the changing of the seasons, Fall was always my favorite time of year; it signified change and renewal – but I don't want to digress to much from what's going on here at LogiGear Magazine.

This month, Michael Hackett continues his series on Agile, this time talking about the importance of training. Michael should know, he has been the champion of much of LogiGear's training curriculum and content. He has taught classes and seminars all over the world on topics ranging from leading a test team to how to design excellent tests. His passion these days has been "all about Agile and how it affects testing" and these articles are just the tip of the iceberg when it comes to what LogiGear is doing around Agile.

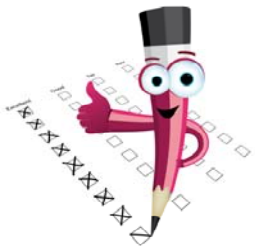
Michael recently developed a new class called "Testing in Agile Development" and he started evangelizing in the summer. We continue to write and talk about how to inject test automation into Agile Development sprints successfully, and most importantly trying to stay ahead of the curve and teaching testers the do's and don'ts in this new software development model.

I recently just got back from speaking at STARWEST, a conference held by SQE, and the major topic was, you guessed it, Agile. So, it's fitting that we continue to write and talk about it here and in conferences that many of the LogiGear executives speak at – so continue to look for more interesting material from LogiGear on Agile Development.

This month we are also introducing a way for others to contribute to our magazine. We have put together an editorial calendar for article submissions from others who might want to reach our testing community. We certainly value others input and opinions in software testing, and there is no better place to that than in LogiGear Magazine.

We hope you enjoy the interviews this month and our other related articles about software testing – we hope our readership finds our articles and materials interesting, and look forward to bringing you more great testing knowledge in the months ahead.

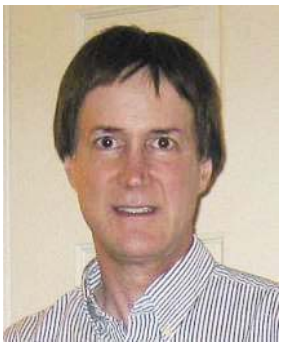
Editor-In-Chief William Coleman



FEATURE ARTICLES

RELATED ARTICLES

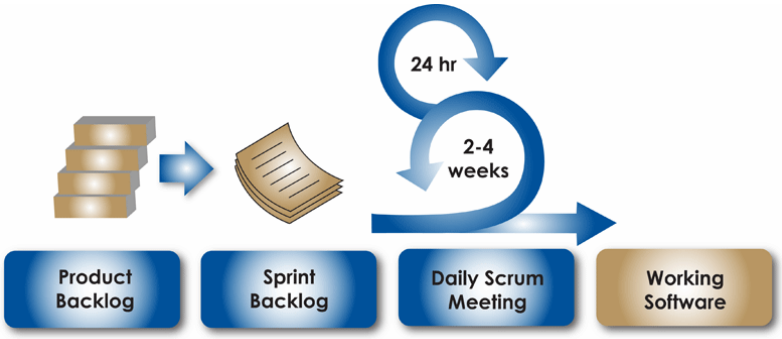
SPOTLIGHT INTERVIEW



IN BRIEF…

Testing in Agile Part 4:

SKILLS and TRAINING



SKILLS

Agile teams need training!

One of the missing links in the implementation of Agile development methods is the lack of training for teams. I noticed in our recent survey on Agile that only 47% of the respondents answered "Yes" that they had been trained in the Agile development process, with over half responding "No." This number is way too high. The "No" respondents should be closer to 0%!

Have you been trained in Agile development?	Percent answered
Yes	47.8%
No	52.2%

Check out last month's Survey Summary Article

A situation that is too common these days is a team says: *We are Agile* because their manger told them they are and they have daily meetings. They have cut down the size of their requirements docs. They call their development cycles *sprints* and their daily meeting a *scrum*. That's about all the *Agile* there is to this development organization. No management backing off, no self-directed teams, same old tools, no more measurable unit testing or Test Driven Development (TDD), and no increased test automation - they now just call themselves *Agile*.

This is wrong. This is not Agile. This will lead to problems. What you want to prevent is a wave of the magic wand and… "abracadabra, you're Agile!"

Training solves problems

Before starting any Agile implementation, the team must be trained in a variety of ideas, practices and methods. Such as: What are Agile, Scrum, and XP? Why are we doing it? How should you do it? What are the new "roles?" How do I do my job differently now?

To make a case for training lets put together a few foundation ideas of Agile development. Teams and individuals must be trusted to do their jobs. Without proper training - that trust will not happen. Simple. Also, in my experience with Agile teams, one common hard learnt lesson is the importance of changing and improving the way teams work and develop software through the use of retrospectives and continuous improvement; that is, teams constantly reflect back on ***what worked***, ***what did not*** and change practices, processes, and sometimes, even teams. This is **key** to Agile success. Teams need training and coaching here. This is not your old fashioned post-mortem.

There is a common phrase in the Agile world: ***Fail early, fail often!*** This means, fail fast - find out what does not work for you, remove inefficiencies right away and find the practices that do not work for your team and get rid of them fast. Try something - fail at it, fix it, make the process better, change how you do things till people are comfortable with how things work.

To do this crucial piece of the Agile/Scrum/XP most effectively, get training about what is Agile and why are these practices used? When your team has problems - all teams do - having good training on the practices, and the processes to recognize and take action on problems as well as recommended or alternative practices will help you ***fail fast*** and fix problems - getting more productive and happier faster!

Agile training for whom?

The main audience for this article is test teams. Most test teams do ***not*** dictate management training. But if you have the ability to make a suggestion, strongly suggest your company's management teams get training in understanding Agile and critical management topics such as: ***Chickens and Pigs***, the importance of product owners, customers or customer representatives working with the Scrum team daily, empowering self-forming/self-directing teams, and team ownership of quality - rather than the traditional single point QC by a test team. Having management understanding of these topics is a minimum starting point and support for productive teams. Management needs Agile training too!

The whole development organization and Scrum team will need training and a new glossary. The practices and glossary will make a laundry list of training topics fast: sprint, spike, scrum of scrums, ***done***, release planning vs. sprint planning, what happens at a scrum meeting and what does not, cross-functional and self-directed team roles, the role of the scrum master, etc.

For other non-test team members there can be significant training, like unit test training for developers and choosing a good unit test harness, TDD (test driven-development) vs. unit testing, scrum master, and product owner training.

Let's pause here for a second, since I want to make sure you are aware that I talk in detail on many of the above skills, topics and methods in my last three articles in this "Testing in Agile" article series. I encourage you to review them since they will provide additional context to some of these suggested training areas.

Here are the links to those articles:

Testing in Agile Part 1 - Intro to Agile for Testers
Testing in Agile Part 2 - Agile is About People
Testing in Agile Part 3 - People and Practices

Skills for test teams

Let's move on -- specific to test teams, in most organizations, after a whole-team training on Agile development, the biggest burden for technical training and testing skills is the ability to be much more dynamic and self-directed in test strategy.

The testing skills needed for Agile test teams are more responsive, dynamic, *thinking on your feet* methods rather than following-a-plan type testing many test teams are accustomed to. The test team needs to move away from validation and into investigative, exploratory, aggressive, unstructured methods, such as "error guessing." Test teams need new strategies that are more responsible for reviewing design as well as hunting down bugs and issues!

Training is also needed for when and how you test. Teams need to split their time and tasks between developers and testers for unit testing, user story validations, UI testing and devoting focus on strategies around manual and automated testing. There are also documentation changes and ideas for more lean and efficient practices -- avoiding documentation waste. Agile teams must have new approaches to writing test cases. These issues get solved with training.

In successful Agile organizations, traditional style test plans are useless and not nearly dynamic and flexible enough for the speed of Agile projects. This does not mean people who test stop planning! It means, instead that a different, leaner, more efficient set of test artifacts need to be created. This, again, is a training issue.

And, as mentioned in other places, good test teams should and can take on more technical tasks, such as design review and build management/continuous integration – read more about continuous integration in my last article on people and practices. This should be achievable for well-staffed test teams with good training and skill development.

Also important for most development teams is team building, communication, and "feel good about working together" type training. Many organizations are moving from situations where the relationship between project management, development and testing is sometimes strained. These frayed relationships generally occur due to poor communication and are often exacerbated by individuals with different motives or unrealistic timelines. In Agile development frameworks, these bad dynamics kill any chance of success. Soft skill training is yet another key to Agile success.

Here are some skills I have often found lacking in testers who are involved in an Agile development project:

- good, on the spot, rapid fire estimation skills - for people who test ***and*** people who write code, product owners and scrum masters
- depending on the existing skill set of the many test teams, some testers will need help and training to successfully contribute to software design critique
- many times in this series we have discussed the need for massive test automation. That most teams need help looking for easier, more effective, faster ways to do more automation is clear. Test Automation in Agile deserves its own white paper – I will however, talk a little bit about automation playback tools for Agile in my next article, the last one in the series.

Good Agile trainings should be full of learning games and learning activities. That Agile is "learn-by-doing" is clear. Training adults needs to be lighter, more fun, engaging - and immediately related to their job! There are significantly higher levels of retention with learning activities than with ***only*** lecture or reading. But most importantly, the activities and games are a perfect way to present a new way of thinking, teamwork dynamics and principles to foster good team dynamics.

The Real Costs/Benefits of Test Automation

By Frits Bos, PMP, Pm4hire

Are you frustrated with vendors of test automation tools that do not tell you the whole story about what it takes to automate testing? Are you tired of trying to implement test automation without breaking the bank and without overloading yourself with work? I experienced first hand why people find test automation difficult, and I developed useful ways to cut testing costs. We must focus on simple tools that produce results. Testing is like systems development. If you want quality results, start with quality requirements. You should not start with test automation; you start with an organized approach to QA testing that will facilitate test automation. This paper explains how you can succeed when you address the REAL Costs/Benefits of Test Automation.

Testing, or test automation, is not rocket science. Some people make it more complicated than necessary. All you need to succeed is a good testing process that embodies a vision of what works and what does not work. As you select applications ripe for test automation you will find some that are better tested with manual approaches. Your overall approach must be consistent. You must avoid the cost of duplication. Most people are not even sure what they mean by test automation. What is test automation? Is test automation simply capture/replay processing? Is test automation programming in some script-like language?

Capture/replay works to fill time slots on a glut of TV channels, since all you change are the commercials to be aired. For software testing, think of the corollary of an old saying: "The more you want things to stay the same, the more things have to be changed". If you want to replay tests, isn't that because you want to review the effects of change in the application? You will soon discover that updating playback scripts is a maintenance nightmare, since each case is a unique recording.

Contrary to what some experts claim, test automation should not be program development. Linda Hayes called that "writing programs to test programs" and she rightly classified that idea as absurd. Some test automation product vendors want us to believe in a programming paradigm. Many experts lament that testers are too busy with manual testing to write test automation scripts. Some testers may not be qualified to write such test scripts. Is anyone ready to propose we ask development to double their workload? I didn't think so.

To develop any software application, you have to start with fundamentals. Script writing is labor-intensive. It is difficult to maintain scripts. Generally, our initial testing is best done manually, so that we can stabilize application interfaces before we attempt test automation. If we use different OS/Browser combinations, we face the challenge of automating so many interfaces that it will seem a lot less work to just test combinations manually. Of course, the fact that so many tools are only concerned with GUI testing should be a concern to us. Don't we test batch systems? Don't we need to test individual layers in an N-tier server structure?

People may demonstrate "relative payback alternatives" of manual testing vs. automated test execution in terms of how many test cycles for automation to pay off. What does that mean? Why not focus on the challenge of employing automated tools for any kind of testing? Why not engineer solutions that eliminate a supposed inherent duplication of first establishing manual test scripts and then repeating the effort to produce automated test scripts? Well, IBM has published study results that claim 75% of all testing is still done manually. That means no solution is complete unless it addresses this larger part of the testing needs. I will explain how I created a solution that dramatically reduces the effort of test script creation, and especially of test script maintenance, that provides scripts for manual testing as well as for test automation.

Think about those analyses of breakeven points of manual vs. automated testing that cite a number of automated test sessions after which automated testing becomes cheaper. They seldom account for the need to maintain both manual scripts (as confirmed by IBM) and automation scripts. Many break-even analyses do not tell you how to account for the costs of manual script production plus for the incremental costs of added automation. What I will explain is how we can bring the cost curve down for manual testing by streamlining the scripting, and how some manual testing can be replaced by test automation (but not in terms of replacing all manual testing as some breakeven points imply). The biggest obstacle in the way of progress is to set unrealistic expectations of what people may gain from a specific decision. Overselling the benefits of test automation does nobody any good.

I focus on creating test scripts and eliminating unnecessary duplication. Test automation should be viewed like any other business process automation. I want a solution that meets the needs of my business. If that solution must simplify manual script writing, then that is the focus of automation. I use a practical solution for creating and maintaining test scripts by separating the fixed and variable aspects of scripts. A small set of scripts may be combined with a table of alternate data values so you do not have to replicate scripts for individual combinations. You may add a second table to list test database access keys for individual test cases. The reason is that, over time, a test database tends to change. The indirect approach lets you access the right test database records, while you avoid test script changes due to test database changes.

You can use my solution manually, but then testers must consult 3 sources of data in order to execute those scripts + data values + profile definitions. I created an automated solution that merges these 3 sources dynamically just prior to a test execution, so that testers see a "virtual" instance that contains the current data values. It is possible to cut script creation costs by 50% using this strategy, and maintenance by at least 75%. Database changes just prior to the start of testing are no problem: you can update the profiles and generate a new set of scripts that same afternoon, rather than to search through a stack of test scripts to find all the instances that will have to be changed.

I implemented the process used to support this "manual test automation" solution as a VBA macro within Excel™. I added data generation capabilities with that logic, to identify data by attributes, to identify pairs/triplets/quads for optimized combinations, to set mapping tables for equivalent values, and to define Governing Business Rules that determine what are testable condition combinations. I believe in fundamental testing, to relate test cases and scripts to business requirements and functional specification and to incorporate risk-based testing priority setting. These capabilities support complex script requirements with almost no extra effort on the part of the QA analyst.

By contrast, compute-bound data generation efforts are generally avoided in most manual script preparation initiatives. Shortcuts can seriously compromise the credibility of what is actually tested. With my tools I can account for what I test and why, which is an important automation benefit. I can provide reassurance that all that testing is based on fundamental requirements for the application, not based on what developers thought they had to build, which can be a major source of functionality bugs. Because manual scripting tends to be done in a hurry, QA analysts may look for the easiest source of requirements, so that tests are influenced by what the developers think they should be implementing.

Functional testing is not the only game in town. We need unit testing and integration testing that may not be done properly due to the effort involved in producing needed test cases. With a test framework (such as modeled after JUnit), I can use my tools to produce test data using the same test cases that the application must be able to handle in black-box testing. I output data into "*.CSV" files that can be input into a test framework to present alternative tests and to validate the results as well, so developers can stop bugs from infecting the code before it even reaches QA.

I demonstrated test execution automation with Certify™ (by Worksoft) because the internal working tables from my tools are directly compatible with record-sets in Certify™. A large part of the test automation engineering was done with cut-and-paste operations. I wrote a simple, reusable driver script for Certify™ that mimics my table-driven script generation process. That driver script invokes simple task oriented action scripts, such as specific screen dialogs, and Certify™ inserts the relevant data for each instance that must be tested. The Certify™ keyword-driven architecture allows me to produce automated script segments in hours. My data-driven architecture produces full execution automation within days of debugging the manual scripts. I use one common input, my source specifications, that I can update in order to regenerate scripts and record-sets in minutes. This all but eliminates script maintenance concerns and removes the logistics of managing manual test scripts in parallel with automated test scripts. I know that not everybody uses Certify™. Some people already bought into more elaborate products (perhaps based on writing test dialogs the hard way in VB Script). Initial feedback was that my approach was fine for a select user community, but not for a majority of users of test automation software. Some skepticism is healthy, too much is paralyzing! I like to think of solutions that are easily understood by most testing analysts. I want quick payback from test automation.

There is no benefit from products that end up as "shelf-ware" because they consume more resources than what they return in benefits. Certify™ is good, but really no single product is sufficient. Look at the complex IT solutions with different products integrated into a complex production environment. Ask if it is realistic to want a one-tool-solves-all solution. Look beyond individual tools and consider multiple tools. You need more software and additional learning, which adds complexity to integrate those tools to meet specific testing challenges.

To support other test automation products I created a second tool. I use the same data and I created a generic keyword structure to generate executable scripts for any procedural testing tool. This requires custom OPDEF's (operation definitions) that use keywords to select script code segments. My tool can insert appropriate data values to make each script instance functionally operational. It takes a little more effort than if you use Certify™. First you need to write the same simple task oriented action scripts as explained for Certify™. You also need to write reusable OPDEF's that convert keyword based actions into script code that can then be executed by the targeted test automation product. This way you can continue to use your existing test automation software if you are able to feed it directly from the same inputs we use to generate manual scripts. When changes are presented, you simply pull out the input workbook, you make the changes, and you regenerate your test collateral right down to your favorite test automation scripts, usually within the same day. If you need to work with multiple test automation tools, you can provide multiple OPDEF libraries and compile the appropriate scripts with the right set of definitions for each tool.

In summary, my solution requires only one level of maintenance effort that we minimize by using a 3-pronged approach of scripts + data + profiles. Any extra effort to produce the initial input data to ensure that your testing is sufficiently thorough adds extra cost to the script preparation effort. With my approach, the cost of initial scripting will often be reduced, and maintaining that script base is clearly much more efficient than if done manually. Consider:

- Scripts change because physical interfaces change. We can design scripts so that we use the fewest lines of unique code per GUI screen segment (or transaction file, etc.). Such changes are simple to implement with a tool like Certify™ that is able to "re-learn" a screen. We can eliminate duplication of low-level dialogs to dramatically lower the testing costs. The risk of script changes is so common that projects wait as long as possible before they start scripting. You cannot escape that risk in regression testing. Unless you avoid duplication of low level scripts, maintenance efforts will be significant. With my reusable OPDEF architecture you cannot minimize scripting any further unless you use a keyword-driven architecture that we use in Certify™.
- Data change when new functions support new (or additional) conditions or options. If data are entrenched in scripts it is difficult to find what you must change or add in the scripts to reflect new conditions. By keeping the data separate, it is easy to focus on data instances that are affected by application changes. We can update those data and regenerate scripts relatively quickly (typically in an hour or two after we receive the changes). This includes all the testing collateral necessary for the execution of test cases that reflect the new version of the application, which is a significant advantage.
- Profiles change because databases are not static. We make test scripts adaptable by using "profile" references in the data and by providing separate "definition" data to map each profile to actual database keys. There are many aspects to this concept. As data become stale, they are edited or replaced, and you reflect that in the profile definitions. Since you can describe data attributes for a profile you keep a clear focus of what you are testing, while account references become confused when the underlying data are subject to change. The extra step of using a profile is a critical step that safeguards your investment in the test collateral by formalizing the design documentation.

These concepts are lacking in most test automation tools. This explains why test projects fail due to multiple duplication opportunities. You need manual testing and test automation, and perhaps different test automation tools to test different aspects in different environments, and transactions files, and updates to database files. Unlike other approaches, I recognized the key issue as creation and maintenance of test scripts, transactions, and test data.

Maintenance of test collateral is onerous. It can consume over 75% of the total costs of small upgrade projects. You think not? Ah, what probably happens is that your testers use only 10% or so of the scripts for regression testing, and they only update that 10% of scripts. As a result that script base becomes corrupt after a few upgrade cycles. I recommend that you keep 100% of all scripts updated, even if you use 10% of those scripts in a given regression testing session. Test automation lets me regression test 100% of all scripts each time. A full regression provides a better level of quality assurance than what you can achieve with a purely manual approach, and at a fraction of the cost.

Keep your efforts to code automation scripts to a dull roar. Avoid "writing programs to test programs" as Linda Hayes cautioned. This is especially true if you need multiple tools to deal with different types of applications under test, because that would take an army of programmers with different skills to keep up with the scripting needs. Keep in mind the limitations of test automation when you deal with web-based applications that must be validated for compatibility with a myriad of customer configurations. You still need manual scripts. My solution uses Excel workbooks, IntelliData.xls and IntelliScript.xls, that automate the procedures described above. I have documentation and data-driven / keyword-driven training materials that I use in my business. Like many "simple" solutions, this did not evolve overnight: it is the result of many trial and error versions of code to solve specific problems. What I see in many testing tools is a constant effort to update the tools due to technological changes. What I see in most testing organizations is a desire to keep the application-specific test scripts as static as possible. Based on that premise my tools bridge the gap between testing tools and business needs and they satisfy the primary goals and objectives for test automation: to cut costs in manual testing as well as in test execution automation. I will be happy to provide you with additional information about my tools and my testing methodology.

*Article from QA Vision - **Frits Bos** (Frits@pm4hire.com) is a veteran of over 30 years in IT, and a witness to the tendency to reinvent the wheel. He provides contract services in Project Management, Business Analysis, QA, and BCP. He also develops training seminars and creates development and testing tools, and is an active contributor at the www.stickyminds.com site for QA testing professionals.*

Does Agile development need specialists

By John Morrison, QA Manager at Sun Microsystems

"Agile is to software development what poetry is to literature.
It is very difficult to do well, very easy to do poorly,
and most people don't understand
why a good poem is good and a bad poem isn't..."
- from the web

Transition from a traditional development model to an agile methodology is often met with some degree of skepticism and doubts by testers. Books and programs on Agile development tend to emphasize the need for multi-skilled generalists who can take up different functional roles as needed. This tends to cause specialist testers to worry about losing their identity in an agile world. Is there a need for specialists in agile or is the agile world inhabited by generalists, the proverbial *jack-of-all-trades* ?

To answer this question, look no further than your favorite team sport. Agile software development is a team process involving members from the different functional groups coming together as part of one team to produce software. No longer are they members of distinct teams such as development, testing, technical writing, i18n, l10n, etc. Back to our earlier question on whether agile needs specialists or is the new world full of generalists?

Like a team sporting activity, for a team to be successful it cannot be - 1) all members of one particular type or specialize in one activity e.g. development or testing alone 2) all members who are generalists and knowing part of all functions but not specialists of any function. How would you rate the chances of your favorite sports team if it were comprised of either types of members – a) all players who specialize in just one area e.g. of cricket which I follow: a team of just batsmen or just bowlers b) all generalists e.g. again of cricket: a team of just all-rounders – it might be better than the first choice with only players of one type but still not the best choice. An agile team takes a step towards success when it comprises specialists from the different functions coming together and working together collaboratively to produce software. Each member of the agile team brings to the table their unique set of skills that influence the software's development and success of the project. However, an additional requirement for these specialists that are part of the agile team is to be able and willing to share in some of their non-core tasks. For example, a tester who can debug defects and make small fixes if need be, a developer who can also document their feature or do some testing, etc.

There is truth to the statement that agile needs generalists. However, it is better to have specialists who can go beyond their functional domains to help towards the release rather than specialists who just restrict their involvement to their functional area of specialization. Ultimately, agile development is a team effort and testers like others on the team must own the release from the beginning without merely trying to police it at the end.

***

From **John Morrison's Blog**

How to become a software tester?

D. Richard Kuhn - Computer Scientist, National Institute of Standards & Technology

LogiGear: How did you get into software testing? What did you find interesting about it?

Mr. Kuhn: About 10 years ago Dolores Wallace and I were investigating the causes of software failures in medical devices, using 15 years of data from the FDA. About that time, Raghu Kacker, in NIST's math division, introduced me to some work that his colleague at Bell Labs, Sid Dalal, had done on pairwise and interaction testing for software. The idea behind these methods is that some failures only occur as a result of interaction between some components. For example, a system may correctly produce error messages when file space is exhausted or user input exceeds some limit, but crashes only when these two conditions are both true at the same time. Pairwise testing has been used for a long time to catch this sort of problem. I wondered if we could determine what proportion of the medical device failures were caused by the interaction of two or more parameters, and just how complex the interactions would be – in other words, how many failures were triggered by just one parameter value, and how many only happened when 2, 3, 4, etc. conditions were simultaneously true. We were surprised to find that no one had looked at this question empirically before. It turned out that all of the failures in the FDA reports appeared to involve four or fewer parameters. This was a very limited sample in one application domain, so I started looking at others and found a similar distribution. So far we have not seen a failure involving more than 6 parameters. This does not mean there aren't any, but the evidence so far suggests that a small number of parameter values are involved in failures. The reason this is significant is that if we can test all 4-way to 6-way interactions, we are likely to catch almost all errors, although there are many caveats to that statement and we don't want to oversell this.

LogiGear: You are working in very special field of software testing. It's quite different from common software testing like: Unit tests, Automation testing, GUI Testing, etc…. So what kind of work are you doing? How did you pick those specific topics? Can you help explain this type of testing to our readers?

Mr. Kuhn: NIST's mission is to develop better measurement and test methods, so Raghu and I proposed an internal project to build on the empirical findings discussed earlier. The first problem that had to be solved was to find ways of efficiently generating tests that cover all t-way combinations, for t=2, 3, 4, etc. This is a hard mathematical problem that has been studied for a century or more. Jeff (Yu) Lei had developed a very efficient algorithm to cover 2-way combinations as part of his dissertation at NC State, so we talked to him about extending the algorithm for t-way combinations. He did this successfully and we worked with him to incorporate the algorithm into a practical tool for testers, which is now being used at more than 300 companies.

The tool, ACTS, makes it easy for testers to enter parameter names and values for each, then generates test values that cover 2-way through 6-way combinations. For realistic applications, this can produce thousands of tests in some

cases, so we have also worked on ways to automate the oracle problem for testing – how to determine the expected results for a particular set of input. This allows testing to be (almost) fully automated, so running hundreds or even thousands of tests can be done at a reasonable cost.

LogiGear: How can a college student prepare to go into software testing and become really good at it? What should he/she look for in teachers, courses, and methods?

Mr. Kuhn: The biggest challenge in software assurance is how to do it economically. Assurance methods used for life-critical software, such as in aviation, are too expensive for most applications, yet every year we become more dependent on software working correctly. One of the best ways to reduce cost, as in other fields, is to bring more science and technology to bear on the problem, so courses that focus more on the science than on managing testing will be important. Testing is only one part of assurance. An important area of research is how to integrate formal methods for specification and proof with testing, including combinatorial interaction testing.

LogiGear: What sort of graduate programs are available for fledging software engineers?? Also, in your opinion, what are some of the more interesting research questions people are asking now and what do you think they'll be researching in, say, 5 years?

Mr. Kuhn: Graduate programs with an established program in software engineering will be good choices for work in software assurance. In addition to how to integrate formal methods with combinatorial testing, important questions include: how to order or prioritize tests to find faults more quickly, how to identify the particular combination(s) that caused a failure, how to extend these methods to very large problems, and above all, determining the effectiveness of these methods in the real world. It would be nice if these problems were solved quickly, but I'm sure they will still be important in 5 years. Combinatorial testing has grown very rapidly in the past 10 years, from around 4 or 5 papers a year in the 1990s though 2002, to more than 50 per year recently, so it seems to be attracting a good deal of attention from researchers.

LogiGear: Lastly, who do you consider to be some of the leaders in this field and what are they doing?

Mr. Kuhn In addition to Jeff Lei (U. Texas, Arlington) and Jim Lawrence (George Mason U.), who have been part of our team since the beginning, Charles Colbourn, at ASU, is the expert on t-way covering arrays and algorithms. We're working with Renee Bryce (Utah State) and Sreedevi Sampath (U. of Maryland Baltimore County), who are looking at test prioritization. Myra Cohen (U. Nebraska-Lincoln) has done a good deal of work on both the theory and application of these methods. There are many other leaders including Sid Dalal, George Sherwood (from former AT&T, Bellcore, SAIC), and Alan Hartman (from IBM)

A lot of people are now working on getting these methods into practice. Among those we are working with are Jim Higdon at Eglin Air Force Base and Justin Hunter of Hexawise. Turning research ideas into something that works in the real world can be more challenging than the research in some ways.

LogiGear: Thank you, Mr. Kuhn.

D. Richard Kuhn

Computer Scientist, National Institute of Standards & Technology

Expertise

- Combinatorial Testing
- Role Based Access Control
- Quantum Information Networks

Honors & Awards

- 2009: Excellence in Technology Transfer Award
- 2008: Best Standards Contribution, NIST/ITL
- 2007: Best Journal Paper Award, NIST/ITL
- 2002: Gold medal award for scientific/engineering achievement, U.S. Dept. of Commerce

VISTACON 2010

Matt Elias
Software QA Test Lead

Livescribe, Inc.
7677 Oakport St., 12th Floor
Oakland, CA, USA, 94621

LogiGear: What made you decide to join the conference?

Matt Elias: Actually I was here with friends and we heard about a new testing conference VISTACON, so, obviously being a tester, I was curious. Our company works with KMS and we are outsourcing some software development and test automation. This conference just happened to coincide nicely with our trip and we were able to take advantage of the extra 3-day overlap while in Vietnam. Some of the information that we're learned about at VISTACON is actually directly contributable to our software development and testing processes at LiveScribe – so it's been a very, very valuable conference.

LogiGear: What impressed you most at the conference?

Matt Elias: I would say the diversity and experience of the speakers and topics. When

you have companies like Microsoft, EA, and Halliburton in attendance – they bring such a diverse set of product experience to the table coupled with many different perspectives on software development and testing. They talk about processes and methods that I probably would have never thought of myself -- probably nobody in my company would have thought of them – so, having this diversity was extremely helpful and interesting.

LogiGear: How do you feel about the speakers, organizations, participants and the conference materials?

Matt Elias: I would say all of the materials and participants really stood out for me. One example, BJ Rollison, from Microsoft provided thoughts and offered interesting, tangible test examples based on his real-world experiences – I really enjoy hearing about practical employment of tools and methods – it's more tangible. From his talk, we were able to correlate and think about what he is doing at Microsoft and apply some of those methods to our own products and processes at LiveScribe.

LogiGear: What did you get from VISTACON?

Matt Elias: VISTACON provided the ability to do some great networking with many professionals in my industry. It also provided me some insight into various improvements for my own company. For example, the specific Agile processes and problems that most companies face when they're building from the ground up -- because I work for a start up. A lot of problems we face are the same problems that these start-ups face – so hearing how others approach these problems helped. Also, being able to learn from some of the industry leaders on how they were able to mitigate these software development and testing issues and really grow into industry leaders was very valuable.

LogiGear: Would you recommend VISTACON to others?

Matt Elias: Yes, most definitely. I am actually going to try to get some more of my company to come out with us next year.

LogiGear: Thank you, Matt.

"What did I get from VISTACON other than the ability to do all this networking with all these professionals? I would say a lot of insight into improvements for my company" – *Matt Elias, Livescribe*

AUTOMATION TOOLS

Win Runner

What is WinRunner?

HP **WinRunner** software was an automated functional GUI testing tool that allowed a user to record and play back UI interactions as *test scripts*. As a functional test suite, it worked with HP QuickTest Professional and supported enterprise quality assurance. It captured, verified and replayed user interactions automatically, in order to identify defects and determine whether business processes worked as designed.

With HP WinRunner, your organization gains several advantages, including:

- Testing **time reduced** by automating repetitive tasks
- Broad testing support for **diverse environments**, using a single testing tool
- Increased **return on investment** through modifying
- and **reusing test scripts** as an application evolves

For what types of applications can WinRunner be used to test?

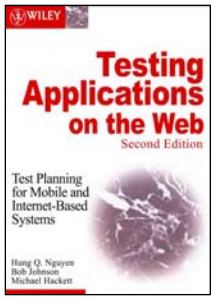
WinRunner is suitable for any kind of testing that involves software. It has been used to successfully test applications built on Windows, Java, Unix, mainframes, the Web, mobile devices, embedded software, graphical software, multi-media, and more.

WinRunner Server Requirements:
Windows® applications (Win32), Visual Basic, Java™, ActiveX

SOFTWARE TESTING BOOKS

Testing Applications on the Web – 2nd Edition

Test Planning for Mobile and Internet-Based Systems



Authors: Hung Q. Nguyen, Bob Johnson, Michael Hackett
Paperback: 432 pages
Publisher: Wiley; May 16, 2003
Language: English
Product Dimensions: 9.1 x 7.3 x 1.5 inches

Testing Applications on the Web was written by a true authority in the field; Hung Q. Nguyen's *Testing Applications on the Web* is a nicely comprehensive guide to virtually every conceivable aspect of software testing. It's filled with must-have background information for any test engineer or manager who's testing thin-client systems. This book also outlines the state of the art in software testing. Notable sections include a short guide to no fewer than 24 distinct types of software tests, how to test browser-based user interfaces effectively, and a thorough guide to Web-performance testing.

REVIEW: "This is a good book. If you test web apps, you should buy it!"
By Dr. Cem Kaner – Director of Florida Institute of Technology's Center for Software Testing Education & Research.

This book has substantial value for what it teaches us about testing on the web. Beyond that, it teaches about thinking clearly and thoroughly when your application interacts in complex ways with other systems. I think his approach will have lasting value and lasting influence long after many of the detailed issues that he describes have been resolved and replaced with new ones.

This book is pretty good, it gives a very good overview of everything in software testing, I wish there was more on testing enterprise, very good for entry-mid level testers or performance engineers, worth it for the mone.

I've been waiting for this book and I must say where was this book a year ago? I have experienced most of what the author has described and yes, it answers questions that I have had.

SOFTWARE TESTING CONFERENCES in November 2010

1	ASTA International Software Testing Conference 2010	02 – 05	HCMC, Vietnam	http://www.vietnamesetestingboard.org/zbxe/?mid=asta2010
2	London Tester Gathering	02 – 11	London, England	London Tester Gathering
3	UNICOM Next Generation Testing Conference	03	London, England	http://unicom.co.uk/product_detail.asp?prdid=1620
4	IEEE International Test Conference	31 Oct – 05	Texas, US	http://www.itctestweek.org/
5	The Dutch Testing Day	04	Leiden, NL	http://www.testdag.nl/
6	Software-QS-Tag 2010	05	Bayern, Germany	http://www.qs-tag.de/
7	ICTSS - Int. Conf. on Testing Software and Systems	08 – 10	Natal, Brazil	http://ictss2010.dimap.ufrn.br/ictss
8	Advanced Software Testing	10 – 12	Roma, Italy	Advanced Software Testing
9	Expo QA	16 – 18	Madrid, Spain	http://www.expoqa.com/en/conference.php
10	Agile Development Practices East	14 – 19	Orlando, Florida, US	http://www.sqe.com/AgileDevPracticesEast/
11	Euro STAR 2010	29 – 02 Dec	Copenhagen, Denmark	http://www.eurostarconferences.com/

SUBMIT YOUR ARTICLES ON LOGIGEAR MAGAZINE:

SUBMISSION GUIDELINES

LogiGear Magazine would like to announce that it is soliciting articles from writers, activists, journalists, and our readers who have an interest in writing about software testing topics. Such topics can include regular articles such as those in our magazine or from posted articles on blogs, websites or newsletters that can reach a wider circulation. These periodicals can be about: features, tips and hints, testing instructions, motivational articles and testing overviews, that will assist our readers in gaining knowledge about software testing's practices and industry.

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 reader's response.

3. Due to staffing, editing of such submissions is limited; therefore, please send only the final draft of your work. After article is received, it may be revised before publishing, but not without your final approval.

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.

Additional information about this process or to submit an article, please email: thuyen.vu@logigear.com

Wish you a

HAPPY HALLOWEEN DAY

2010