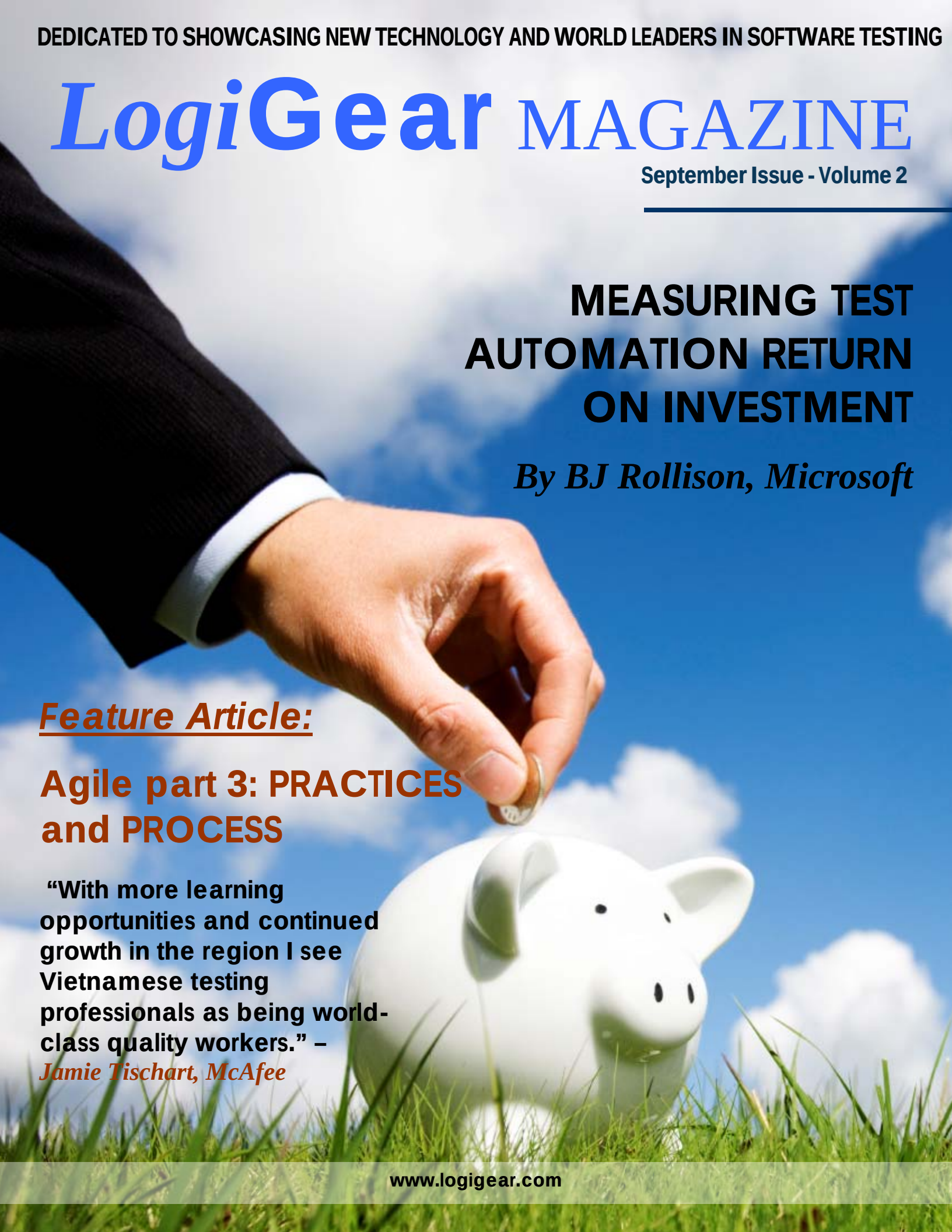
DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

September Issue - Volume 2



MEASURING TEST AUTOMATION RETURN ON INVESTMENT

By BJ Rollison, Microsoft

Feature Article:

Agile part 3: PRACTICES and PROCESS

"With more learning opportunities and continued growth in the region I see Vietnamese testing professionals as being world-class quality workers." – *Jamie Tischart, McAfee*

LogiGear MAGAZINE

IN THIS ISSUE

September Issue - Volume 2

Letter from the Editor

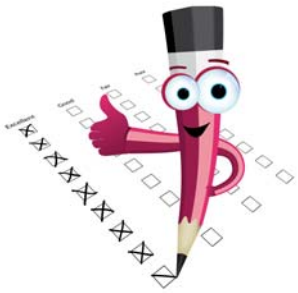
September has been an exciting month for LogiGear!

We launched the first ever software testing conference in Vietnam, VISTACON. It was a resounding success, with well over 200 participants and 20+ speakers from around the globe; each speaking on a wide range ofa cutting-edge testing topics. In this month's magazine, we have uploaded several video recordings of event presentations – giving our readers a small peak into our inaugural event.

Also in this issue we continue Michael Hackett's five part series on Agile for Tester. In part three, Michael talks about what testers will encounter around the various practices and processes while testing an application in Agile. For testers working in an Agile development process, this is essential reading – we look forward to bringing you parts four and five in the coming issues.

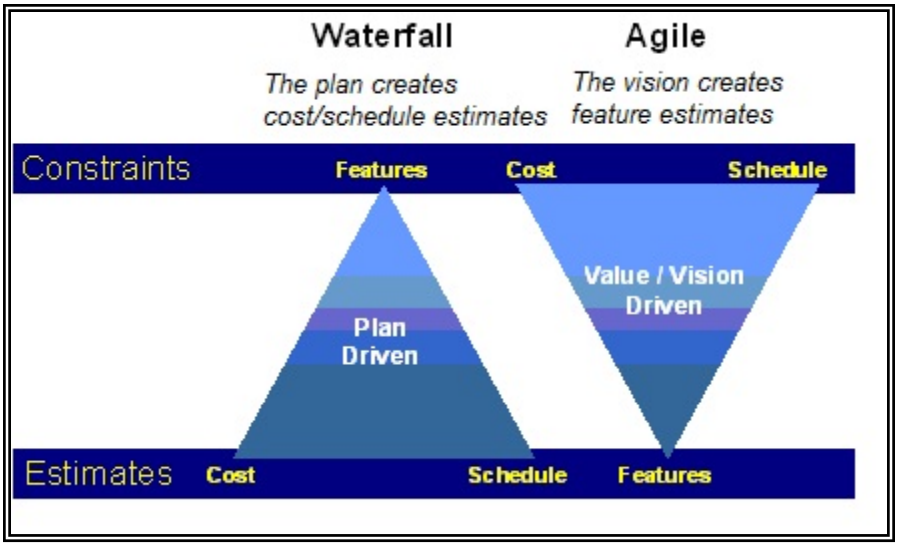
Lastly, I'm excited to announce that LogiGear will begin to solicit outside article contributions for future magazine issues – we feel that getting additional material from testing experts and practitioners in the software testing industry only furthers our mission to publish the most interesting, relevant and unique material on all things software testing.

Editor-In-Chief William Coleman



Agile part 3:

PRACTICES and PROCESS



SUMMARY:

Remember that Agile is not an SDLC. Neither are Scrum and XP for that matter. Instead, these are frameworks for projects; they are built from practices (for example, XP has 12 core practices). Scrum and XP advocates will freely recommend that you pick a few practices to implement, then keep what works and discard what doesn't, optimize practices and add more. But be prepared: picking and choosing practices might lead to new bottlenecks and will point out weaknesses. This is why we need to do continuous improvement!

That being said, there are some fundamental practices particular to test teams that should be implemented; if not, your chances of success with agile could be doomed.

Just by being aware of possible, even probable, pitfalls or even by implementing the practices most important to traditional testers does not guarantee agile success and separately testing success, though it should avoid a complete collapse of a product or team -- or potentially as harmful, a finger-pointing game.

This 3rd part of our Testing in Agile series focuses on the impact to testers and test teams on projects implementing agile, XP, and Scrum. In this installment, we'll only focus on the practices that are the most important to software testers.

And, in the final analysis, even if you implement all the important processes -- and implement them well -- you need to review how successful they are in your retrospective and keep what works then modify, optimize, and change what doesn't work. Your scrum master or scrum coach helps with this.

PRACTICES

In this section, we will focus mainly on XP practices, which are the development practices, rather than on the Scrum/project management practices, since it is the development practices that effect test teams most.

To be blunt, if you're developers are not unit testing, and the team does not have an automated build process, the team's success in agile will be limited at best.

UNIT TESTING / TDD AND AUTOMATED USER STORY ACCEPTANCE TESTING

Unit testing and **test-driven development** are so fundamental to agile that if your team is not unit testing, I cannot see how testers could keep up with the rapid release of code and fast integration in agile projects. The burden of black and gray box testing in the absence of unit testing in very fast, 2-to-4 week sprints should frighten any sane person who is knowledgeable in software development. Your developers need to be unit testing most if not all their code in rapid, agile projects – there is no way around it.

Automated user-story acceptance testing is secondary to unit testing in importance. This test validates that the task or goal of the user story has been achieved rather than validating code as in a unit test. Having that kind of test automated and available to re-run on successive builds and releases will enable a test team to focus on more effective exploratory testing, error guessing, scenario, workflow, error testing, varieties of data and alternative path testing that unit testing and user-story validation tests rarely cover. This leads to finding better bugs earlier and releasing higher quality software.

100% developer unit testing is one of the most significant advancements within the agile development process. We all know this does not guarantee quality, but it is a big step toward improving the quality of any software product.

From our Testing **State-of-the-Practice** Survey that we conducted on **logigear.com**, we can get a rough idea about how close we are to the ideal.

Here is one question regarding unit testing from our survey:

QUESTION - ***What percentage of code is being unit tested by developers before it gets released to the test group (approximately)?***

total # of unit tests developed	Percent answered
100%	13.6%
80%	27.3%
50%	31.5%
20%	9.1%
0%	4.5%
No idea	13.6%

A vast majority of agile teams are unit testing their code, though only a fraction are testing all of it. It's important to know that most agile purists recommend 100% unit testing for good reason. If there are problems with releases, integration, missed bugs, and scheduling, look first to increase the percentage of code unit tested!

AUTOMATED BUILD PRACTICE AND BUILD VALIDATION PRACTICE / CONTINUOUS INTEGRATION

With the need for speed, rapid releases, and very compressed development cycles, an automated build process is a no-brainer. This is not rocket science and not specific to Agile/XP. **Continuous integration** tools have been around for years; there are many of them and they are very straightforward to use. It is also common for test teams to take over the build process. Implementing an automatic build process by itself is a step forward, but a team will realize more significant gains if they add to automated builds with continuous integration.

Continuous Integration includes the following:

- Automated build process
- Re-running of unit tests
- Smoke test/build verification
- Regression test suite
- Build report

The ability to have unit tests continually *re-run* has significant advantages:

- It can help find integration bugs faster
- Qualifying builds faster will free up tester time
- Testing on the latest and greatest build will save everyone time.

The positives for continuous integration far outweigh any resistance to implementing them. Test teams can take on more responsibility here: they can be more in control of the testing process -- on how many builds they take and when they take them -- for starters.

HARDENING SPRINT / REGRESSION SPRINT/ RELEASE SPRINT

The most successful agile teams have implemented a sprint that, in effect, is specifically designed and tailored to just *test*, or quite simply a "testing sprint". Although this testing or integration sprint can go by many names, a regression sprint or hardening sprint are the most common types. Prior to releasing to the customer, usually, someone has to do security, performance, accessibility, usability, scalability, perhaps localization, or many other types of tests that are most effectively done once the product is fully integrated. In most cases, this is when end-to-end, workflow, user-scenario tests are done and when full regression suites are executed. It is a great bug-finding and confidence-building sprint. But! Its late into the development cycle. "Bugs" found here may go directly into the backlog for the next release or cause a feature to be pulled from a release.

ESTIMATING AND PLANNING POKER INCLUDES TESTERS.

Testers participating in the sizing and estimating of user stories is very basic to agile success. A few times I have run across companies trying to scope, size and rank a backlog without test team input. This is a gigantic mistake. Let me tell a quick story.

I was at a company that was doing a good job at implementing scrum, which they had piloted across a few teams. They still had some learning to do but were still implementing processes -- overall, doing a good job to start!

The group that had the toughest time adopting to scrum, though, was the testers. This was because in their previous SDLC, the test team was viewed as adversarial, composed mainly of outsiders. Some of those attitudes persisted to the point where the product owner (a former "Marketing person") excluded testers from the user-story sizing and the estimation process.

During a coaching session, I was reviewing some user stories with them. We were doing some preliminary sizing and doing a first pass, assigning only four estimates: small, medium, large, and extra large. (Note: it's a great way to start. Some people call it grouping by shirt size to roughly estimate what can get done in a sprint.)

One certain story got sized as a large and another at medium. I picked those two stories out from my experience with their product and pointed out that the one story ranked as a large was a very straightforward test. Any tester knowledgeable about this area could do an effective test job pretty quickly. But this other user story they sized as a medium was a test nightmare! I quickly ran through a list of situations that had to be tested -- cross-browser, data, errors, etc -- all of those things that testers do! The team believed me and we proceeded to pull in the test engineer to review our results. The tester quickly said the same thing as I had and pointed to this as a sore point for testers. The stories would have been sized completely wrong for the sprint (as had been the problem for the previous test team) if the test team continued to be excluded from the sprint planning and playing poker session.

This does not mean that it would reduce the complexity (to develop or move the story from a large to a medium). But it would have moved the medium complexity to a large or even an extra large and this would have impacted testing! The lesson learned here is that the test team needed to be included in the planning game. Attitudes had to change or costly estimating mistakes would be made.

This practice is also crucial to the XP values of trust and respect! Sadly, in many situations I have seen testers excluded from the planning meetings and invariably it is always a trust problem! Deal with the trust and respect problem and get involved in the complete planning process!

DEFINITION OF DONE

We're all used to milestone criteria, entrance criteria, and exit criteria in whatever SDLC we're using. The term people on agile projects use that relates to milestone criteria is the definition of *done*. Most often, the problem with milestone criteria is that it is routinely ignored when schedules get tight. This often leads to frustration, bad quality decisions, and ill feelings.

I want to show a simple description of agile that will help us in the discussion.

We are all familiar with the traditional three points of the project triangle that every project must juggle: features (number of features, quality, stability, etc.), cost (resources), and schedule. Before agile, projects committed to feature delivery then would add people (cost) or push out dates (schedule) and sometimes release buggy features to meet whatever constraint project managers felt needed holding!

Agile is different.

In agile, the cost, namely the size/resources of the team, is fixed. We know that adding people to a project reduces efficiency (reference Peopleware section, **Agile Series Part 1)**; and, the schedule is fixed. Never extend the time of a sprint. What can change, and what is estimated, is the *set of features*. This leads us back to the definition of *done*.

What gets done, the user stories/features, gets released. What does not get done gets put into the backlog for the next sprint. Since sprints are so short, this is not as dramatic as pulling a feature from a quarterly release. The customer would have to wait another quarter to get that functionality. If a feature gets pulled from a sprint and put into the back log, it can be delivered just a few weeks later. How do we know what's done? A primary value of XP:

- Working software is the primary measure of progress.

The definition of *done* will change among various groups. There is no one definition, though it commonly includes the following at a minimum:

- the ability to demonstrate functionality
- complete /100% unit testing
- zero *priority 1* bugs
- complete documentation

Many teams also include a demonstration of the user story or feature before it can be called *done.*

In the past, for most teams, it seemed like a nice idea to have a set of milestone criteria but it was routinely ignored. In agile projects, though, with rapid release, the risk of slipping on done milestone criteria could be catastrophic to the system. *Done* is a safety net for the entire scrum team and actually the entire organization.

In the past, many test teams had been the entrance and exit criteria police since, in later stages, milestone criteria are often based on testing, bugs and code freeze -- items testers see and are responsible for reporting on. Now, it is the Scrum Master who enforces the "Done" criteria, not testers. It is much better to have the Scrum Master be the enforcer, rather than have testers act as naysayers and complainers! Every team needs a Scrum Master!

SMALL RELEASES

- Deliver working software frequently.

In Scrum it is recommended that sprints last from 2 to 4 weeks maximum. The practice of small iterative releases is the very core of agile development. I have seen companies rename their quarterly release a sprint and say: "we're agile!" No.

A three month sprint is not a sprint at all. Sprints are meant to be very narrow in focus, able to demonstrate functionality before moving on to the next sprint, and have a prioritized and realistic backlog. These among many reasons should keep your iterations short. Some companies have begun agile implementations with four-week sprints and a plan to reduce the sprint time to three or two weeks over a year, after some successful releases and retrospectives with process improvement. Ken Schwaber and Jeff Sutherland, the original presenters of Scrum recommend beginning with a 2 week sprint.

MEASURE BURNDOWN AND VELOCITY

I have brought up the phrase *sustainable pace* and burndown charts a few times.

Let's briefly discuss these practices.

First, two guiding ideas:

- We have to work at a sustainable pace. Crazy long hours and overtime lead quickly to job dissatisfaction and low quality output (see peopleware definition on Wikipedia) -- the main way we get an idea of sustainable pace is through measuring burndown.

- Burndown charts are one of the very few scrum artifacts.
- The only scrum artifacts are the product backlog, the sprint backlog, and the burndown chart. Velocity is not *by the book* scrum but we will talk about this as well.
- To briefly describe a burndown chart, here are some points about them and their usage:
- they measure the work the team has remaining in a sprint and whether they can get done with its planned work
- they quickly alert you to production risks and failed sprint risks
- they alert you to potential needs to re-prioritize tasks or move something to the backlog
- they can be used during a sprint retrospective to assess the estimating process and in many cases the need for some skill building around estimating the line
- Here is an example of one from Wikipedia, which might help you visually understand what we have just discussed.
- they measure the work the team has remaining in a sprint and whether they can get done with its planned work
- they quickly alert you to production risks and failed sprint risks
- they alert you to potential needs to re-prioritize tasks or move something to the backlog
- they can be used during a sprint retrospective to assess the estimating process and in many cases the need for some skill building around estimating the line

Here is an example of one from Wikipedia, which might help you visually understand what we have just discussed.

To have healthy teams and high quality products, people need to work at a sustainable pace. View this time using burndown charts and velocity.

Burndown charts count the total number of hours of work remaining in a sprint on the y axis against the day-by-day total on the x-axis.

Velocity measures the estimated total of successfully delivered user stories or functionality of backlog items. If this is measured over many sprints, a stable, predictable number should emerge.

Velocity can be used to gauge realistic expectations by both "chickens and pigs" for future purposes. Velocity is measured in the same units as feature estimates, whether this is "story points", "days", "ideal days", or "hours" - all of which are considered acceptable. In simple terms, velocity in an agile world is the amount of work that you can do in each of the iterations. This is based on experience from previous iterations. The Aberdeen Group, a IT research firm, which has covered and published material on Agile Development, makes the claim that "**[w]hen cost / benefit is measured in terms of the realization of management's expectations within the constraints of the project timeline, it is the ability to control velocity that drives the return on investment**."

Measuring the burndown rate and calculating velocity will give you reasonable amounts of work for a team to do at a pace that is conducive to happy teams, releasing higher quality software. To repeat from the introduction piece to this series on Agile, "**Teams working at a reasonable pace will release higher quality software and have much higher retention rates -- all leading to higher quality and greater customer satisfaction."**

When the team feels really good about their abilities it encourages them to do better. The business starts to believe in the team and this sets the team up *in the zone*. When it gets *into the zone*, the team can generally sustain its steady-state velocity month after month without burning out. And better yet, they get to enjoy doing it. Geoffrey Bourne, who writes for Dr. Dobbs Journal notes, "The essence of creating a happy and productive team is to treat every member equally, respectfully and professionally." He believes Agile promotes this ethos and I agree with him.

In conclusion, being agile is implementing practices to help product and development teams work most efficiently and be happy. There are many, many practices (again, XP has 12 core practices). Here, we discussed only the key practices for testing success. If your team is calling itself agile and has not implemented some of these practices, it is crucial to bring them up in sprint retrospectives and talk about their benefits and the problems that not doing them has caused in your product and customer.

There are other practices that need to be in place for success, and specifically for test teams to be successful and not be pointed out in a blame game. These are covered in other installments, namely:

- You have to have a scrum master.
- Automate, automate, automate.
- Use sprint retrospectives for process improvement.

More Agile Questions and Results from "The State Of The Practice Survey"

AGILE SERIES SURVEY RESULTS FROM THE STATE OF THE PRACTICE SURVEY 2009

Question 1

Have you been trained in Agile Development?	
Yes	47.8%
No	52.2%

**** this was out of 100 respondents***

The fact that more than half of the respondents answered "no" here is troubling in many ways; let's just stick to the Practices issue. It is clear some of these organizations are calling themselves "agile" with no reality attached. Whether you want to call them "ScrumButts" or refer to them as Lincoln's 5-legged dog, calling yourself "agile" without implementing practices is just not agile! Attempting to be agile without training *all* the team in the why and how of these practices will fail.

Question 2

Since you move to Agile Development, is your team doing:	
More Unit Testing?	50%
Less Unit Testing?	6%
The Same Amount of Unit Testing?	28%
I have no idea?	16%

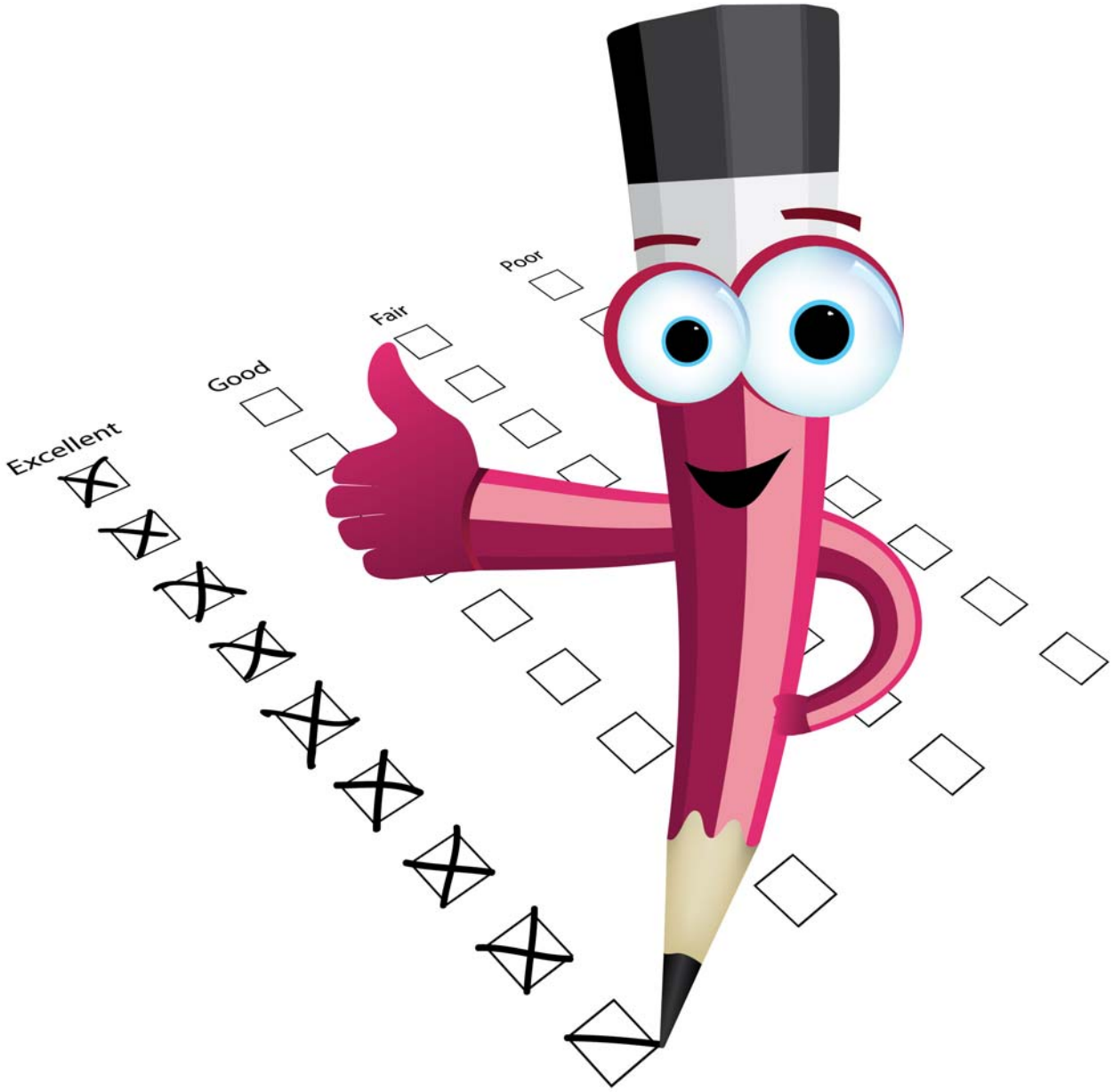
Ideas to take from this are many: That more "unit" testing is happening in 50% of the responding organizations is a good thing! That more "unit" testing is happening at only 50% of the organizations is a problem. More troubling to me is that 16% have no idea! This is un-agile on so many levels -- a lack of communication, no transparency, misguided test efforts -- a lack of information on test strategy, test effort, test results -- and a lack of teamwork!

Question 3

Does your team have an enforced definition of *done* that support an adequate test effort?	
Yes	69.6%
No	30.4%

**** this was out of 100 respondents***

This is encouraging. Hopefully the 30% without a good *Done* definition are not "ScrumButts" and will be implementing a useful definition of *done* very soon!



Survey Response Overview

LogiGear Corp. conducted a wide-scale survey in 2009-2010 to get ideas on the current state of the practice of testing. Here is what I intended and what I learned.

I am Senior Vice President at LogiGear. My main work is consulting, training, and organizational optimization. I've always been interested in collecting data on software testing – it keeps us rooted in reality and not some wonkish fantasy about someone's purported *best practice!* Just as importantly, many software development teams can easily become myopic as to what they do as being normal or "what everyone does." I also wanted to do a survey to keep my view on testing practices current. I am always looking to incorporate data I glean about new ideas and methods in my training and consulting programs when I work on for clients at LogiGear.

In 2009 and 2010, I conducted a large survey called "What is the current *state-of-the-practice* of testing?" I opened the survey in the first part of 2009 and collected data for an entire year. Invitations were sent out to testers from around the world – since software testing is a global trend with experts and engineers having all sorts of ideas on how to do certain methods and practices differently – I wanted to capture and understand that diverse cross-section of ideas.

Some of the data was pretty much what you'd expect, but for some of the sections especially around outsourcing, offshoring, automation and Agile to name a few; the answers were quite surprising.

This first article is designed to give you an introduction into my approach and some preliminary findings. I hope to share with you over the coming months more data and my interpretations of those results.

GOAL

My goal in doing the survey was to move away from guesses about what is happening, and what is common, and move to using actual data to provide better solutions to a wider variety of testing situations. First, we need to better understand the wide variety of common testing practices already in use and how others are using these processes and techniques for success or failure. In order to make positive changes and provide useful problem solving in software development and specifically testing, we need to know what is actually happening at ground level, not what a CTO might *think* is happening or *wants to* happen!

Also, when I write a white paper or article I want it to reference and contrast *real*-world testing and software development. I hope this will help many teams in checking their practice against some broader test/dev situations as well as give realistic ideas for improvement based on what other teams and companies may *really* be doing!

The Questions

I wrote the survey on a wide variety of current topics from testing on agile projects; to opinions of offshore teams; to metrics. The survey also featured more question sets on the size and nature of teams, training and skills, the understanding of quality, test artifacts and the politics of testing.

The Sample Set

This was a very large survey, with over 100 multiple choice questions combined with several fill-in essay type responses. The survey was meant to be cafeteria style, that is, testers could choose sections that applied to their work or area of expertise and ignore or skip those that did not apply to them, professionally or by interest. For example, there were sections for "teams that automate," and "teams that do not automate," teams that "self-describe as *agile"*, offshore teams, onshore teams, etc. So no one was expected to complete the entire survey.

Some Sample Responses

Here are some preliminary findings from my survey. Analyzing the entire survey will take more time - but I did want to put out a selection of findings to give you an idea of what type of information I will be sending out. I picked some responses that were interesting because they confirmed ideas or surprising because they are rarely discussed, poor planning, old ideas, or just surprising! I've broken them down into four sections, "answers that I expected", "conventional wisdom that seems validated", "answers that did not appear uniform", and some "surprising data" that was in some cases unexpected.

We received responses for 14 countries!

So here we go:

Answers that were along the lines I expected:

Question: Test cases are based primarily on	Answer
A - Requirements Documents	62%
B - Subject Matter Expertise	12%

The overwhelming majority of teams still begin their work referencing requirements documents. However good, however bad, *complete* or too vague - most people start here. I did think the number of teams starting with test cases with workflows, user scenarios - using their subject matter expertise would be higher. How a user completes some transaction or some task - I guess, is still secondary to the requirement.

Convention Wisdom that was validated:

Question: What is the name of your team/group?	Answer
A - QA	48.8%
B - Testing	20.5%

This is conventional wisdom, but surprised me. It is definitely a trend - at least in Silicon Valley - to move teams away from the outdated term "QA." Since the people who test rarely ever, almost never, really do QA. If you are a tester and you think you do QA, please return to 1985. It is interesting, though, that this number calling themselves QA has dropped below 50% -- as time goes on this number will continue to drop.

60% of all respondents write test plans for each project

Here is some more conventional wisdom – this can be a great point of interest when you are debating - should I/we write a test plan for each project?

Far from Uniform Answers:

Question: Educational Level (selected responses)	Answer
A - High School	3.0%
B - Bachelors of Arts/Sciences	40.0%
C - Some Graduate Work	19.0%
D - Masters Degree	24.6%
E - PhD.	3.0%

It seems conventional wisdom that the vast majority of people who test have university degrees, but I am surprised at how many have done post graduate work, have a master's degree and have PhDs. It runs against conventional wisdom that people who test are the least trained on the development team, perhaps they are the *most* educated!

Surprising Data:

34% of all respondants indicated that their regression testing was entirely manual

A very big surprise to me! The lack of automated regression! Wow. That is one of the biggest and most surprising results of the entire survey! Why do 1/3 of teams *still* do all manual regression? Bad idea, bad business objective.

52% do not test their application/system for memory leaks

The number of teams *not* doing some variety of memory, stress, DR (disaster recovery), buffer overflow (where applicable) load, scalability, etc. testing was another big surprise. We need to look further into this. Is it bad planning? Lack of tools, skill, lack of knowledge, *keeping your fingers crossed*? In many cases I bet this is bad business planning.

87% of respondants rank offshoring or outsourcing as "successful"

Such a very high number of people responding that offshoring and outsourcing was successful goes against conventional wisdom that it's the managers who like outsourcing/offshoring but production staff (the people who actually *do* the work), are not happy with it!

37% of teams say they do not currently automate tests, with 10% indicating they've never tried to automate

That over 1/3 or respondents currently do not automate tests is in line with what I see in my work at many companies but is contrary to popular belief and any sort of *best practice.* What I see out in the business world is teams that automate think everyone automates and they automate enough. Teams that do not automate see automation as *not* common, too difficult, not something *testers* do. This number is way, way too high. Any team not automating has to seriously look at the service they are providing their organization as well as the management support they are receiving from that organization!

Agile Series Survey Results From "The State of the Practice Survey"

As part of my on-going series on Agile for Testers – see this month's article on People and Practices, I wanted to include the data I collected Agile development and testing and give you a chance to view them.

Question 1

Have you been trained in Agile Development?	
Yes	47.8%
No	52.2%

The fact that more than half of the respondents answered "no" here is troubling in many ways; let's just stick to the Practices issue. It is clear some of these organizations are calling themselves "agile" with no reality attached. Whether you want to call them "ScrumButts" or refer to them as Lincoln's 5-legged dog, calling yourself "agile" without implementing practices and training on what this is all about is just not agile! Attempting to be agile without training all the team in the why and how of these practices will fail.

Question 2

Since your move to Agile Development, is your team doing:	
More Unit Testing?	50%
Less Unit Testing?	6%
The Same Amount of Unit Testing?	28%
I have no idea?	16%

Ideas to take from this are many: That *more* "unit" testing is happening in 50% of the responding organizations is a good thing! That *more* "unit" testing is happening at *only 50%* of the organizations is a problem. More troubling to me is that 16% have no idea! This is un-agile on so many levels -- a lack of communication, no transparency, misguided test efforts -- a lack of information on test strategy, test effort, test results -- and a lack of teamwork!

Question 3

Does your team have an enforced definition of *done* that support an adequate test effort?	
Yes	69.6%
No	30.4%

This is encouraging. Hopefully the 30% without a good *Done* definition are not "ScrumButts" and will be implementing a useful definition of *done* very soon!

Question 4

What percentage of code is being unit tested by developers before it gets released to the test group? (Approximately)?	
100%	13.6%
80%	27.3%
50%	31.5%
20%	9.1%
0%	4.5%
No Idea	13.6%

I won't respond again about the *No Idea* answer, as that was covered above, but it's important to know that most agile purists recommend 100% unit testing for good reason. If there are problems with releases, integration, missed bugs, and scheduling, look first to increase the percentage of code unit tested!

The Result

The overriding result is that the current testing practice is quite diverse! There is *no single test practice*, no one way to test, and no single preferred developer/tester ratio. Everyone's situations were different and even some similar situations had very different ideas about their product quality, work success and job satisfaction!

My Future Plans

I plan to continue to commission surveys as a regular part of my desire to take a pulse on what is really happening in the software development world -- with regard to testing rather than postulations from self-described experts. As noted above, as a result of this being a very large survey, I will be publishing sections over the next few months. I look forward to bringing you exciting as well as troubling trends that I've postulated from the data I've collected.

Automation Return on Investment (ROI) ticker

The LogiGear Automation Return on Investment (ROI) ticker, the set of colored numbers that you see above the page, shows how much money we presumably save our customers over time by employing test automation as compared to doing those same tests manually, both at the design and execution level.

We've segmented this page into three sections: one is to clarify our assumptions; then, to provide our audience some definitions, and finally to offer an example of our approach. We hope this helps clarify how we view test automation ROI.

Definitions:

What is a test case and how does LogiGear define this? A *test case* is a self-contained experiment -- a sequence of operational and verifying actions, with a specific set of data, towards the application under test -- with a defined outcome.

At LogiGear we reflect test cases as a series of actions or lines of actions. This is quite different from what most testers are used to.

In our ticker, a *New Test Case* is a test case that is created once during the day by our test engineering teams. We capture this number in every 24 hour cycle.

In our ticker, a *Modified Test Case* is a test case that has already been created, using the above criteria, but has been modified to accommodate new changes in the application. This might be a new verification point or a new navigation method. We might change that test case several times during the day, and we again, update this number in every 24 hour cycle.

How do we reflect our presumed savings?

The presumed **Money Saved** is calculated by comparing the total number of test cases that are developed and run automatically to the same number of tests that would be run and developed manually. As noted above in our assumption section, we make some assumptions in our calculation. One being the time it takes to develop and run our keyword based automation vs. the time it takes to develop and run a manual test case. We also factor in low cost offshore services that maximize not only the resource cost, but the implementation costs.

Here is our simple formula reflecting costs:

$$\begin{pmatrix}\text{Total number}\\ \text{of new and modified test cases}\end{pmatrix} \times \begin{pmatrix}\text{Total number of Hours}\\ \text{To Develop and Run Tests}\end{pmatrix} x \begin{pmatrix}\text{Rate per Hour}\\ \text{for resources}\end{pmatrix} = \text{Total Testing Cost}$$

As noted above - in our services organization, we generally realize a *five (5) fold improvement* in test execution and at least a *two (2) fold improvement* in designing and updating new and modified tests through our keyword method.

Assumptions and Clarifications:

The ticker reflects test cases that are developed and modified daily by our offshore service teams in Vietnam. LogiGear has been helping companies with software testing and test automation for many years. Over those years, we've compiled the number of tests we do for our clients as a matter of pride and internal tracking. In that time-frame we have developed over 8 million tests on 3000+ projects (yes, that's million with an M!).

On to our numbers...

I'm sure we can all agree that executing a manual test takes more time. What we tried to do is average out the time it would normally take to run through a standard manual test case (with the assumption that these tests are required to be executed multiple iterations). We also looked at the time it would take to develop a manual test case, using the standard manual test case narrative approach.

LogiGear's assumption is that it takes roughly *10 minutes* to execute and document a standard test case manually and *6 minutes* to write that same test. Again, these are averages - we're sure some would agree or disagree with these figures.

In test automation, if your test cases are not running faster, then you are doing something wrong. Automation should allow you to execute your test cases at least 3 to 5 times faster than conventional manual testing or *2 minutes for each test* in our assumptions. LogiGear also feels that our keyword test design approach, Action Based Testing, should allow testers to write and develop tests faster; this improvement in time should be at least 2 times faster, or *3 minutes for each test* in our calculations.

Lastly, we factor in the cost of developing and running these test cases both manually and automatically. The rate we use to calculate is our costs to do this work offshore in our Vietnam Testing Facility, which provides low cost high expertise testing services. The rate we use as a comparison to reflect the savings, is doing this same work in the United States.

Here is a simple matrix of our data we used for our calculation:

Data for Equation:	
Time for Creating a Manual Test Case	6 Minutes
Time for Running a Manual Test Case	10 Minutes
Time for Creating an ABT Test Case	3 Minutes
Time for Executing an ABT Test Case	2 Minutes

An Example of our Approach:

Let's use a test example of *checking a website registration system*. This system has a registration dialogue and we want to test it for different inputs. Your organization might write test cases in a classic test case narrative format, like the example below:

classic test case format

Most values are implicit. The tester has to figure them out during execution....

Execution instructions are repeated in multiple test cases

test steps	expected results
Enter a user id that is greater than 10 characters, enter proper information for all other fields, and click on the "Continue" button	There should be an error message stating that "User Id must be less than 10 characters".
Enter a User Id with special character's), enter proper information for all other fields and click on the "Continue" button	An error message should be displayed indicating that "User Id cannot contain some special characters".
Enter the information, with a password of 4 characters and click on the "Continue" button	Check for an error message saying: "Password must contain at least 5 characters".

The above method is not very friendly to automation, nor is it very efficient test case design.

However, for keyword automation, not only is the test case designed for automation, but it's also a very efficient way for documenting test.

Below, we took the same set of tests that we created above using a traditional narrative approach with steps and expected outcomes, and reflected them as keywords.

At LogiGear, our tests look like this (plus!! we then automate these keywords):

Keywords using Action Based Testing

steps are hidden inside action — expected results

check registration dialog	user id: aaaaabbbbbc	message: User Id must be less than 10 characters
check registration dialog	user id: résoudre	message: User Id cannot contain some special characters
check registration dialog	password: test	message: Password must contain at least 5 characters

This is a very efficient way of designing tests and they are in format that allows for easy test automation.

MEASURE TEST AUTOMATION RETURN ON INVESTMENT

BJ Rollison,
Test Architect at Microsoft

I just finished reading Implementing Automated Software Testing by E.Dustin, T. Garrett, and B. Gauf and overall this is a good read providing some well thought out arguments for beginning an automation project, and provides strategic perspectives to manage a test automation project. The first chapter made several excellent points such as:

- Automated software testing "**is software development**."
- Automated software testing "and manual testing are intertwined and **complement** each other."
- And, "The overall objective of AST (automated software testing) is to design, develop, and deliver an automated test and retest capability that **increases testing efficiencies**."

Of course, I was also pleased to read the section on test data generation since I design and develop test data generation tools as a hobby. The authors correctly note that random test data increases flexibility, improve functional testing, and reduce limited in scope and error prone manually produced test data.

There is also a chapter on presenting the business case for an automation project by calculating a return on investment (ROI) measure via various worksheets. I have 2 essential problems with ROI calculations within the context of test automation. First, if the business manager doesn't understand the value of automation within a complex software project (especially one which will have multiple iterations) they should read a book on managing software development projects. I really think most managers understand that test automation would benefit their business (in most cases). I suspect many managers have experienced less than successful automation projects but don't understand how to establish a more successful automation effort. I also suspect really bright business managers are not overly impressed with magic beans.

Magic beans pimped by a zealous huckster are the second essential problem with automation ROI calculations. Let's be honest, the numbers produced by these worksheets or other automation ROI calculators are simply magic beans. Now, why do I make this statement? Because the numbers that are

plugged into the calculators or worksheets are ROMA data. I mean really, how many of us can realistically predict the number of atomic tests for any complex project? Also, do all tests take the same amount of time, or will all tests be executed the same number of iterations? Does it take the same amount of time to develop all automated tests, and how does one go about predicting a realistic time for all automated tests to run? And of course, how many of those tests will be automated? (Actually, that answer is easy....the number of automated tests should be 100% of the tests that should be automated.)

Personally, I think test managers should not waste their time trying to convince their business manager of the value of a test automation project; especially with magic beans produced from ROMA data. Instead test managers should start helping their team members think about ROI at the test level itself. In other words, teach your team how to make smart decisions about what tests to automate and what tests should not be automated because they can be more effectively tested via other approaches.

In my next post I will outline some factors that testers, and test managers can use to help decide which tests you might consider automating. Basically, the bottom line here is that an automated test should provide significant value to the tester and the organization, and should help free up the testers time in order to increase the breadth and/or scope of testing.

From I.M.Testy (BJ Rollison's blog)

How to become a software tester?

David S. Janzen *- Associate Professor of Computer Science Department California Polytechnic State University, San Luis*

LogiGear: How did you get into software testing and what do you find interesting about it?

Professor Janzen: The thing I enjoy most about computing is creating something that helps people. Since my first real job building telecommunications fraud detection systems, requirements and design were the most fun. Then when I heard about this thing called test-driven development, and something just clicked. Using unit test specification to do design made sense to me -- plus, you get this great side effect of all these automated tests to make refactoring and maintenance easier. I guess I got pulled into software testing by way of software design.

LogiGear: What kind of work are you doing and how did you pick those specific testing topics?

Professor Janzen: My PhD research focused on how test-driven development (TDD) affects the internal, or design quality of software. I did a bunch of experiments with students and software professionals that provided some evidence about the benefits of TDD. The experiments are pretty straightforward:

*Get two groups of essentially equivalent programmers, ask them to complete the same tasks, but have one group use the approach you are studying while the other uses the "traditional" approach. Then collect metrics and surveys to see how the two approaches varied.*Having become convinced that TDD is a great way to design and build software, my recent efforts have moved toward incorporating TDD into computing education. I think TDD is taking the same path that objects took in the early '90s. Folks in the industry started adopting objects so the broader academic community took notice. However, they mostly considered object-oriented programming to be an advanced concept, so it started to appear in graduate and upper-level courses. As educators became more comfortable with the approach, objects eventually made it down to first-year courses. I think the same is happening with TDD, so I am building tools and doing experiments to demonstrate that TDD can be taught to beginning programmers with great success. I call the approach Test-Driven Learning (TDL).

My current project is a web-based development environment for beginning programmers that will incorporate TDL-inspired labs. It will soon be available at http://web-ide.org.

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

Professor Janzen: Software testing is such a great field to study. My students who are good at software testing often get the best job offers. Most undergraduate computer science and software engineering programs don't have separate courses in software testing. The best route is to do well in your core programming courses, and then take as many software engineering courses as you can. Many software engineering courses involve team projects.

Volunteer to be a software tester or quality assurance manager. Or, take the role of system integrator or build manager.

These roles will give you exposure to many of the automation tools that software testers use, and will help you start thinking about how to *break* software and not just how to *build* it.

LogiGear: What sort of graduate programs should college graduates consider? 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 5 years?

Professor Janzen: There are two routes you might consider in graduate school. If you want to be involved in software development in companies and lead software teams, look at masters in software engineering programs. Many of these programs cater to software professionals who are already working in industry, by offering courses in the evenings or on weekends. If you are interested in more cutting-edge research, such as building new software testing tools, or developing new software testing methods, consider going to a traditional Computer Science PhD program. There are lots of smart researchers working on really interesting testing topics.

In five years? Well a lot of work seems to be focused on automatically generating automated tests, and also on tools for working with models. Look for some of the top software testing conferences and try to attend. Read software testing trade journals, there are plenty online, and many are still published in magazine format. Try to identify and follow interesting and cutting edge active researchers – and maybe if you're lucky you'll get a chance to meet these pioneers in person – which can be an exciting and thought provoking experience.

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

Professor Janzen: I am interested in some of the work being completed by Tao Xie and Laurie Williams at North Carolina State University. Tao is doing a lot of work with automated software testing and data mining. Laurie has working on static analysis tools, reliability, security, and mutation testing.

LogiGear: Thank you, Professor Janzen.

David S. Janzen

California Polytechnic State University, San Luis Obispo - Associate Professor, Computer Science

Expertise

- Empirical Software Engineering
- Agile Methods
- Test-Driven Development
- Object-Oriented Systems
- Design Patterns
- Software Metrics
- Computer Science, Software Engineering, and Information Systems Pedagogy

Honors & Awards

- 2007: Professor of the Year, Computer Science - Cal Poly
- 2006: Paul F. Huebner Memorial Award, Electrical Engineering & Computer Science - University of Kansas

Testing industry in Viet Nam

Jamie Tischart
Director - Sews, Product Delivery

McAfee, **Inc**.
9781 S. Meridian Blvd, Suite 400
Englewood, Co, USA, 80112

LogiGear: How do you feel about your first time speaking at a conference in Vietnam?

Jamie Tischart: I am very proud to be speaking at the conference in Vietnam. I have been working with LogiGear's Vietnam Testing Center for over 6 years and feel that the individuals combine high level skills with an amazing work ethic. I trust my most important projects to my Vietnamese teams and look forward to sharing more knowledge with testers in Vietnam to hopefully help them continue to learn new methods.

LogiGear: What made you decide to speak at this conference?

LogiGear: What do you think about the development of the software testing industry in Vietnam?

Jamie Tischart: I am very excited about the continued growth and development of the software testing industry in Vietnam. The professionals that are in the industry are knowledgeable, hard working, and continue to increase their expertise. With more learning opportunities and continued growth in the region I see Vietnamese testing professionals as being world-class quality workers.

Jamie Tischart: As a long time supporter of software testing in Vietnam, with many different experiences with different testing projects, I felt that it was my responsibility to share my knowledge and expertise with the growing software testing community. Also Vietnam is one of my favorite places to visit in the world, so I could not pass up the opportunity to visit again.

LogiGear: What are you going to present at the conference and what experiences are you going to share with Vietnamese engineers about software testing?

Jamie Tischart: I am presenting two topics at the conference: Performance Testing Strategies and Transitioning Test Teams to Agile.

LogiGear: Thank you Jamie.

"With more learning opportunities and continued growth in the region I see Vietnamese testing professionals as being world-class quality workers." – *Jamie Tischart, McAfee*

AUTOMATION TOOLS

Test Architect™
Your high volume test automation solution

What is TestArchitect?

TestArchitect is a versatile and effective test development and test automation framework that enables teams to improve quality through better test coverage, while reducing time-to-market and reducing costs. Based on the Action Based Testing™ methodology, TestArchitect allows organizations to routinely achieve 95% or more highly automation coverage, while other tools commonly result in 25% or less.

With TestArchitect, everyone on the test team can focus on what they do best with a new level of effectiveness:

- **Test engineers** and **business analysts** design *executable* tests composed of reusable actions
- **Automation engineers** use their programming skills to automate and maintain individual actions, rather than entire tests, resulting in far better maintainability and scalability
- **Managers** and **test leads** stay in control with customizable reporting and version control

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

TestArchitect 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.

TestArchitect Client Requirements: Microsoft® Windows® XP, Vista or 7, also available is a Linux version that will work in most distributions.

TestArchitect Server Requirements: Microsoft® Windows® XP, Vista, 7, Server 2003, Server 2008, Linux

SOFTWARE TESTING BOOKS

Global Software Test Automation:
A Discussion of Software Testing for Executives

Authors: Hung Q. Nguyen, Michael Hackett, Brent K. Whitlock
Paperback: 164 pages
Publisher: Happy About (August 1, 2006)
Language: English
Product Dimensions: 8.4 x 5.1 x 0.5 inches

Global Software Test Automation is the first book to offer software testing strategies and tactics for executives. Written by executives and endorsed by executives, it is also the first to offer a practical business case for effective test automation, as part of the innovative new approach to software testing: Global Test Automation — a proven solution, backed by case studies, that leverages both test automation and offshoring to meet your organization's quality goals.

REVIEW: "Finally, a testing book for executives!"
By Scott Barbe-Chief Technologist, PerfTestPlus

"Happy About Global Software Test Automation: A Discussion of Software Testing for Executives is an absolute must read for any executive in a company that develops, customizes or implements software.

For years, software testing has been notoriously under valued and misunderstood by corporate executives. While leading software testers have been trying to get their message to executives from the bottom up, they have been largely unsuccessful. This book has the potential to change that.

With this book, all it takes is one business trip and you'll be able to engage in risk and ROI based planning to minimize many of the challenges and expenses your company faces related to software through the efficient and effective application and management of software testing."

SOFTWARE TESTING CONFERENCES in October 2010

1	Agile Testing Days	04 - 07	Berlin, Germany	http://www.agiletestingdays.com/index.php
2	Iqnite events	04	London, England	http://www.iqnite-conferences.com/uk/index.aspx
3	Pacific Northwest Software Quality Conference	18 - 20	Portland, Oregon	http://www.pnsqc.org/
4	Toronto *TesTrek*	18 - 21	Toronto, Canada	http://www.qaitestrek.org/2010/
5	Software Test Professionals	19 - 21	Las Vegas, NV	http://www.stpcon.com/
6	International Conference on Software QA and Testing	27 - 29	Bilbao, Spain	http://www.qatest.org/en/index.php
7	Google Test Automation Conference	28 - 29	Hyderabad, India	http://www.gtac.biz/

SUBMIT YOUR ARTICLES ON LOGIGEAR MAGAZINE:

LogiGear Magazine welcomes articles from writers, activists, journalists and also from our readers who have no prior experience in writing, on topics that we deal with regularly or on topics that you think need a wider circulation. We are interested in seeing well-written features, tips and hints, instructions, motivational articles and other articles that will help our audience gain knowledge about Software Testing. You can find the editorial calendar in the follow link www.logigear.com. Moreover, any writings or introduction about automation tools, software testing books, and conferences are welcomed.

Please note:

- We only accept submissions from the original author of the articles. By submitting material, you acknowledge that you are legally entitled to distribute the work and to allow it to be redistributed. (If you are a book publisher or public relations firm with copy to distribute, please include a note to that effect at the top of the article you submit.) We do not pay for articles, and do not accept articles that are primarily advertisements. However, you may place a brief resource box and contact information (but no ads) at the end of your article. Let us know if you wish to have your e-mail address included for reader response.
- Since we have a small editorial staff, we cannot spend much time editing submissions. Please send us final drafts of your work. After receiving your submissions, we might edit before using it. You will be kept informed.
- Your submission might be used or not. In case we have many articles related to one topic, we could not publish all of them.

Please submit your article to thuyen.vu@logigear.com