THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

JUNE 2013 | YEAR 3 ISSUE V

Jerry Weinberg
Direct Observation of Quality

T Ashok
Yin and Yang in Testing

Ben Kelly
The Testing Dead

Johanna Rothman
Integrating the Tester with the Team

Leah Stockley
Addressing the Risk of Exploratory Testing

Joel Montvelisky
Is friendship ruining your testing career???

Over a Cup of Tea with Michael Larsen

Join us this summer in beautiful Madison, WI for....

CAST 2013

"Lessons Learned"

August 26-28, 2013

Monona Terrace - Madison, WI, USA

Full Day Tutorials

Track Sessions

Testing Competition

Keynotes

(by Dawn Haynes & Jon Bach)

MORE ABOUT CAST

CAST 2013 SCHEDULE

REGISTER NOW

Conference of the Association for Software Testing

ast

The Best of Tea Time

Coming soon...

First Indian Testing Magazine to reach 101 Countries in the world !

Created and Published by:

Tea-time with Testers.
Hiranandani, Powai,
Mumbai -400076
Maharashtra, India.

Editorial and Advertising Enquiries:

Email: editor@teatimewithtesters.com
Pratik: (+91) 9819013139
Lalit: (+91) 8275562299

Editorial

By our guest – Michael Larsen

A Degree in Software Testing?

There's been a groundswell in posts and questions lately for ways that software testers can "strut their stuff and prove their worth". There's lots of ways to go about getting skilled, and there are many educational avenues. We know about Computer Science degrees for programmers, we know about Electrical Engineering degrees for hardware hacks. There are Information Technology degrees for those who want to focus on infrastructure and data management. Interestingly, there is little in the way of an undergraduate degree for software testing.

For years, I've contemplated why this is. Is it an omission? Are we too new? Are we not valued as much as other careers? I think the answer may well be "all of the above". Some people say that there should be a clear career path for software testers to follow. There should be a curriculum that, if they were to follow it, learn it, pass exams, and be handed a Bachelors Degree in Software Testing (or an equivalent) we would feel that that person had the acumen, the knowledge, and the ability to tackle software testing challenges at an entry post-graduate job level. It's a nice thought.

Frankly, I'm torn. Software testing is going through a bit of an identity crisis as the moment. Ask ten different seasoned, high level software testers around the world what attributes and skills would make for a world class tester, you're likely to get ten different answers. My idea of important skills and attributes may be totally different than someone else's. Still, I think it's a fair question, and to that end, I'm going to suggest that **everyone** get a degree in software testing!

I see you. You're looking at the screen quizzically, wondering what I'm going about. You're thinking to yourself, didn't he just say there aren't any degrees in software testing? Well, you're right, there isn't. At this point in time, as far as I am aware, there is not to be found an undergraduate software testing program being offered from an accredited university in the United States of America. I can't speak for everywhere in the world, but thus far, I have not heard of one offered. My question to you all is... why wait? Why should we wait for an ossified educational system to make a degree program for us? Why don't we make it ourselves? In short, my question to anyone who tests software, what's stopping you from creating your own degree program?

Programming is a specific skill. It focuses on creating feature, a workflow, a process, an experience, and to do so directly, economically and efficiently. Programmers seek to make the product to do those things. Think of them as though they are the fabricators of a steel box, and that steel box has been left on a table. Their path is clear, make the box. Now, let's look at the tester. What do they need to do? Not much, really... just consider every risk that box represents, and figure out all of the angles possible to understand, discover and mitigate that risk. Are you still with me? Are you thinking, "Wait, that requires an entirely different mental mindset, as well as an entirely different physical skill set"? If so, you'd be right. Turning non-enthusiastic programmers into software testers isn't a good long term solution, so computer science alone isn't going to cut it. Hardware jockeys with electrical engineering coursework learn a lot about physical phenomenon as well as software interactions, but that's not going to cover it all either. So what do we need? I've long argued that the best testers are cross discipline individuals. They come from many different backgrounds.

My path to testing was accidental, but I was an inveterate tinkerer when it came to musical equipment. My studies in philosophy taught me how to question. My studies in political science taught me a lot about group dynamic and working toward plausible solutions. My mathematical studies helped me manage variability and unknowns, and to seek to solve interesting problems as elegantly as possible. My computer programming and computer usage taught me some understanding of systems and how they work, as well as the pain points that humans face using software. My experiences with physical science taught me to understand the physical phenomenon that makes things work in our world, and to understand properties like electricity, motion, temperature, shock, etc. My studies in literature, art, music and creative writing helped me to see the beauty and aesthetics that matter to people (and those that don't) so that I could understand form along with function.

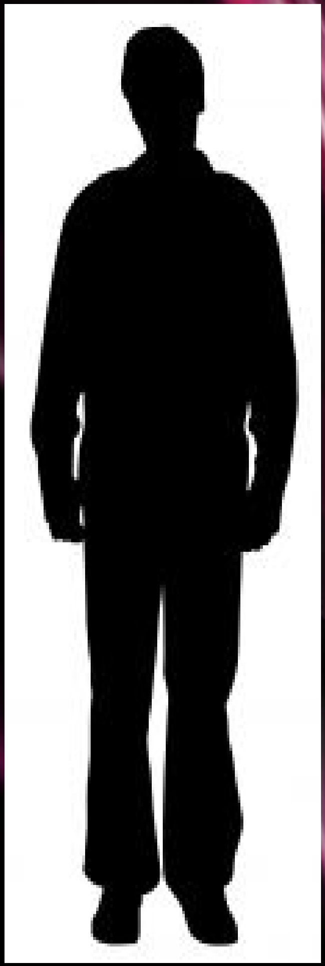
Is your head spinning? That's a lot of stuff to take in, and it's all over the map. In many ways, when we talk about software testers looking to get a degree in testing, we are trying to discover the shortcuts that will get us to the core key knowledge that must be transferred to make testers effective. Therefore, here's my challenge... develop your own degree program. Embrace courses in literature, art history, design, philosophy, rhetoric, mathematics, physics, biology, chemistry, computer programming, music, political science, and foreign languages (whichever one's interest you). Practice using the skills you use in as many areas as you can within your testing responsibilities. Be one who questions, look for the avenues others haven't considered. Keep at it. Encourage other testers to do the same. If you can do this, you will find yourself on a path to amazing skill, artistry, relevance and adaptability. The great news? All of these resources, in both traditional courses or online as self study avenues, are available to anyone with a computer, a web browser, and a hunger to learn and achieve. This is the degree program I've chosen to follow. I figure I will complete it sometime around my 70th birthday, if I'm lucky.

Have you guessed my secret? Is my Degree in Software Testing just another way of saying "keep learning about as much of the human condition as you can"? It might well be. My question to you now... Will you earn that degree? I hope many of you will. **Tea-time with Testers** is designed to be a companion and a helper on that journey. Articles written for each issue focus on real world, nuts and bolts challenges and solutions that you can apply to your own learning and practice. As you read through this issue, I challenge you to think of ways to apply what you learn to your own efforts, and help magnify what you do to the best of your ability. With that mindset, I believe it will be inevitable that that "degree" will be earned. When you have earned it, stop by and wish me a happy birthday sometime in my old age. I'll want to see what you have become, and how much you have transformed the testing world, and the broader world as well.

[illegible]

Michael Larsen,
Software Tester
www.mkltesthead.com

QuickLook



Testing Puzzles by Sebi

AQuA releases new Best Practice Guidelines to assist App Developers with increasing Consumer Privacy Regulations

By – Sneha Konganda

With the Federal Trade Commission recently laying down new guidelines for mobile applications and the creation of the Application Privacy, Protection and Security (APPS) Bill, developers are under increasing pressure to improve privacy policies and data security within their mobile apps. If passed, APPS will require mobile developers to give notice and gain consent for any information gathering they do with mobile apps, as well as maintaining privacy policies and securing any data their app collects. With the recent news that the NSA is gathering consumer data from a range of technology companies the debate over privacy looks set to become the talking point of the mobile industry.

The GSMA has been working with its members and engaging with representatives from across the mobile ecosystem on the GSMA Mobile Privacy Initiative. A core objective of the Mobile Privacy Initiative is to help establish universal guidelines and approaches that address consumer concerns and foster confidence and trust for mobile users.

The App Quality Alliance (AQuA), the mobile industry's trade association for quality app development, which is run by members such as AT&T, Samsung and Motorola, has released its latest Best Practice Guidelines specifically updated with the GSMA Mobile Privacy Initiative guidelines (GSMA Privacy Design Guidelines for Mobile Application Development) to give guidance in the area of customer privacy and mobile apps.

The AQuA Best Practice guidelines set out a range of principles that developers should follow in their development and QA process to ensure they conform with best practice before the deployment of their apps to market.

By amalgamating AQuA's core quality principles with guidance on privacy from GSMA, and previously with the network and battery optimisation guidance from AT&T, the AQuA Best Practice Guidelines are a valuable central resource to help developers navigate privacy requirements without compromising the performance or features of their apps.

New and updated topics include:

1. The requirements for encryption of data
2. Recommended use of passwords
3. Guidance on privacy policies, user rights and correct use of data collected
4. The importance of consent for use of location data
5. Information security and accountability

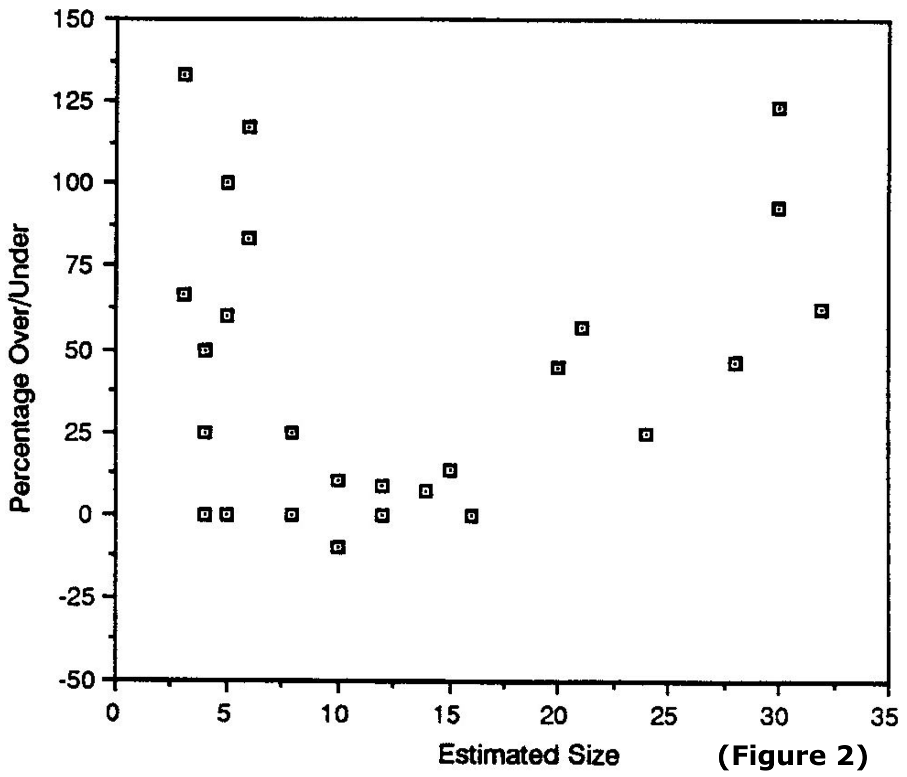
For the GSMA, Pat Walshe (Director of Privacy – Public Policy, GSMA) said:

"The inclusion of the GSMA Mobile Privacy recommendations within the AQuA best practice document further reinforces the importance of consumer privacy as part of the app design process. Privacy-by-design is key to winning and keeping the trust of app users"

You are invited to discuss this on our...

Direct Observation of Quality (Part 2)

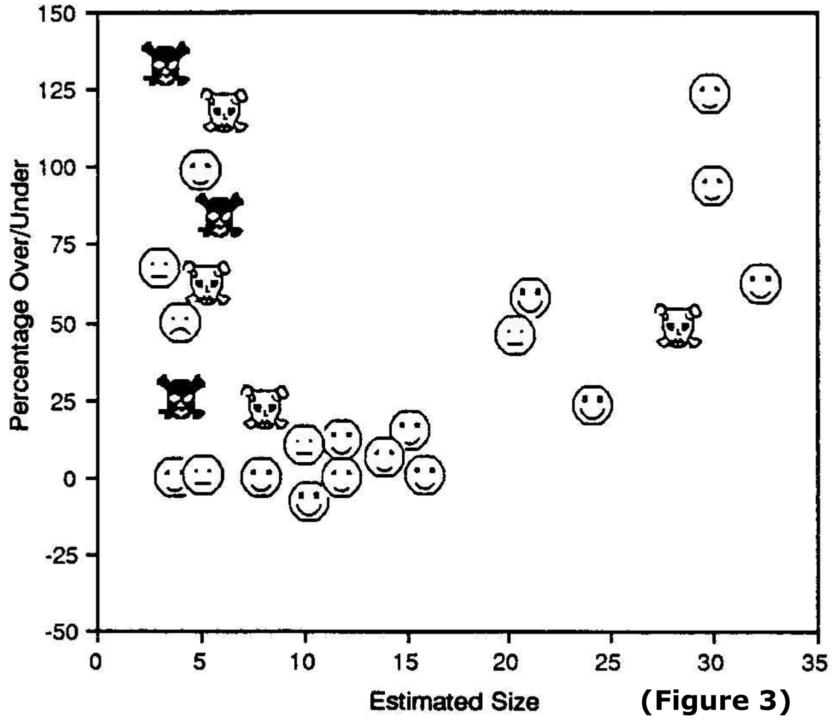
An Example of the Relativity of Quality



(Figure 2)

(This is figure is based on story of company F in Chapter 6 of my book 'How to observe Software Systems') Figure 2 showed the managers that both their large and small projects were often delivered late, but some of the managers argued that that lateness wasn't that important for small projects. Now it's time to see how they were convinced otherwise.

Company F had customer satisfaction figures based on a survey method they had learned in a seminar by Don Gause. One of the outcomes of the survey was a project ranking from 1 to 7 on a Bad-Good scale. The managers retrieved these rankings and put them into seven categories, which they plotted with the ingenious graph (based on an idea by Tufte) shown as Figure 3. Each face represents the customer satisfaction level with one of the project points on the original graph. The facial expressions are easy to interpret at a glance, and paint an interesting picture.



(Figure 3)

Figure 3. A plot of customer satisfaction. A third dimension, customer satisfaction with each project, is represented on the scatter diagram of Figure 2.

The customers—much to the surprise of the IS managers—were not that unhappy with the large projects, even though they were 25% to 125% over their scheduled delivery time. There was one skull-and-crossbones exception, but that was the only project for which delivery schedule was actually critical to the business. The other projects required

extensive customer preparation. The customers themselves were having trouble staying on schedule, so they actually welcomed (for the most part) the software delays.

The IS managers were truly astonished to see Figure 3. I recognized this astonishment as a significant meta-observation, but I had no idea what was behind it. Did our survey method fail? Did the customers lie to the IS managers? Applying the Rule of Three Interpretations, someone came up with the idea that perhaps the IS managers were using an internal definition of quality, one that differed from that of the customers.

To test this hypothesis, we administered the satisfaction survey to the software developers and managers who had worked on each project. Their average satisfaction was displayed in the same format as Figure 3, yielding Figure 4.

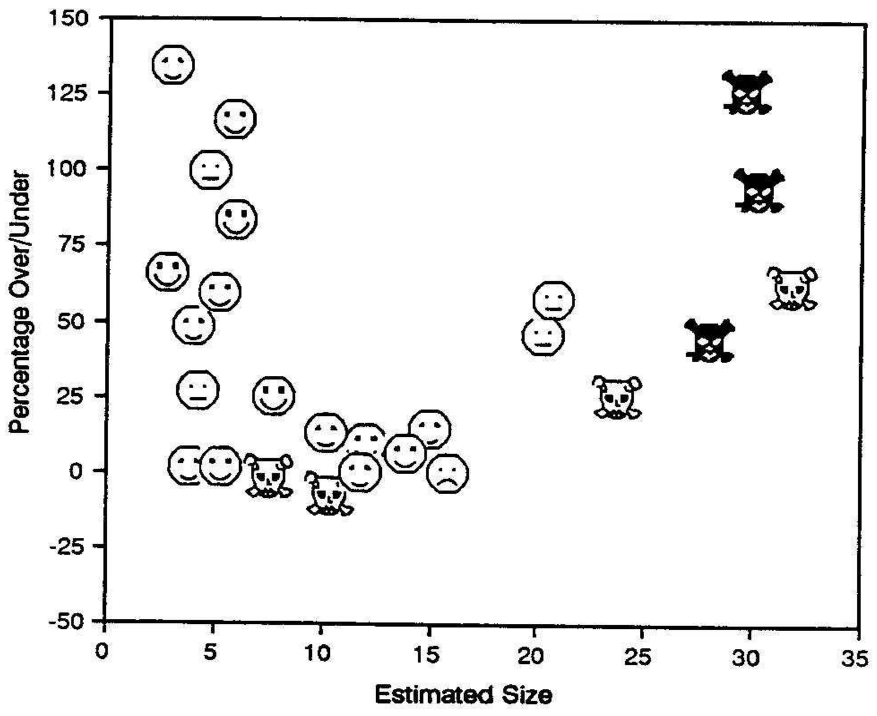


Figure 4. A plot of developer satisfaction.

A third dimension, developer satisfaction with each project, is represented in the scatter diagram of Figure 3. Notice the difference between the picture of developer satisfaction and the picture of customer satisfaction. Comparing the faces on the two scatter plots, it became obvious that the customers and the developers had different criteria for project quality. The customers were concerned about the impact on their business, while the developers attached utmost importance to their own work satisfaction. To the developers, the longer projects represented many months of testing and fixing systems they had long since grown tired of. To the customers, the months during the delay were times of exciting preparation.

Although the smaller systems were similarly over schedule, the developers didn't get bored in the few weeks it took to set them right. For the customers, however, the picture was different. Most of the small systems were needed quickly to analyze some fast-changing business situation. For a number of them (seen as black skulls), the business opportunity had entirely disappeared by the time the software was ready to analyze it.

An Industry Out of Control of Quality

Nobody in the software industry will be surprised to see Company F's trouble meeting schedules. According to data from the SEI surveys, Company F is actually much better than most. Many software organizations today are so overloaded with quality problems that they are no longer coping effectively with their business of maintaining and developing software. In addition to delivery of late, expensive software that customers no longer want, they display all the classic symptoms of overloaded organizations: lack of problem awareness, lack of self-awareness, plus characteristic behavior patterns and feelings. Yet many managers fail to recognize the relationship between this overload and quality problems.

Many managers also fail to recognize the relationship between their own actions and the results they're getting. At best their actions are ineffective, but most of the time they are actually counterproductive. They may know how to develop software, but they don't know the answer to the crucial question that all of us must ask ourselves:

Why don't we do what we know how to do?

One of the reasons we don't do what we know how to do is that we are confused by the multitude of problems.

The first step out of this confusion is to realize that:

Every software problem is a quality problem.

Think about it in terms of the Zeroth Law of Software:

If the software doesn't have to work, you always meet any other requirement.

In more general form, this becomes the Zeroth Law of Quality:

If you don't care about quality then you can meet any other requirement.

Quality, in fact, is producing things of value to some people—meeting their requirements. If you don't have to meet their requirements—if quality doesn't count—you can produce software with any number of features, at any price, as fast as you like. And your developers can think it's great software. In short, if you don't have to control quality, you can control anything else.

In other words, quality is the most direct software measure we can find. And the only direct way to measure quality is with the people whose ideas count, as the following story illustrates:

An executive is going to a weekend conference and he asks his End User Computing (EUC) manager if she has a notebook PC he could take with him. The EUC manager tells him that all the PCs are already taken for the weekend. The only one left must go in for repairs. The executive says that's okay. It does not have to work. He just needs to "have it" for the weekend while he is at the conference.

The EUC Manager is puzzled, but as he is a big shot, she says okay. She remains puzzled until she realizes that the Notebook PC is nothing more than executive jewelry! The executive takes the PC and complains that it's a bit heavy. (It weighs only six pounds, but they don't make executives the way they used to.)

The EUC Manager fixes this "problem" by removing the battery and the hard disk, thereby reducing the weight to a mere four pounds. The executive walks off with the "computer" and a big smile.

What better way to illustrate Zeroth Law of Software? None of the software had to work, so the EUC Manager could "relax" the weight constraints on the hardware! Of course, the EUC Manager didn't think of this as "quality," which is why she told me the story. But her customer was satisfied, and he was an important executive. So, whose ideas about quality should have counted?

Whose Ideas and Feelings Count?

Human satisfaction is the only measure of quality. But if quality is always a matter of people's feelings, whose feelings count? This is the ultimate political question: Who gets to control the definition of quality?

The IBM view of who counts

When it comes to the politics of the information industry, it pays to listen to IBM. Here's what IBM's Chairman, John Akers, had to say about quality:

"Here's another way to think about it: Quality equals excellence equals market-driven. They're all one and the same thing… and we use the words interchangeably." ...

"..., we must commit ourselves to leadership in those markets we choose to serve. We don't intend to be all things to all people. In today's competitive world, we must be selective. And once we've chosen our markets, we must dedicate all the resources it takes to get to, and maintain, the leadership position."

"Market-driven" implies that Akers knows that quality is defined by IBM's customers. Although the first paragraph sounds universal ("Quality equals excellence"), one of Akers's "market-driven" principles says that part of defining quality at IBM is deciding which people we care about, from a business point of view. This leaves IBM in control of the definition of quality, at least in the view from the IBM Chairman.

IBM suffered greatly before and during Akers's reign from its propensity to exclude certain markets as not worthy of IBM attention. Thomas J. Watson almost kept IBM out of the computer business entirely when he predicted a world market of "five computers." IBM was lucky that their major competitor spurned the computer market, too, until IBM could catch up. DEC, on the other hand, was not so lucky. Following Ken Olson's prediction about "home computers," DEC locked themselves more or less permanently out of the personal computer business. As Lord Acton remarked, "Power corrupts." And what power corrupts most thoroughly is the ability to make meaning of observations.

The software developer's view

Sometimes software developers control the definition of quality. Pattern 1 (Variable) organizations may be so far out of control that "It works!" is the only possible definition of quality. Everyone else is too afraid to confront the developers, so whatever they say is law with regard to quality. The arrogance of Pattern 1 developers is often like the arrogance of some IBM marketers: "If you can't get it from us, then you can't get it from anybody." After a while, the response of their customers is often the same, too: "Surprise!" In the case of IBM's customers, the surprise takes the form of considering hardware from some upstart competitor. Sometimes IBM still wins, but the competitors now have their foot in the door. In the case of the Pattern 1 organization's customers, the surprise often takes the form of bringing in a "software Stalin" to attempt to impose Routine discipline. Sometimes the developers still win (if a return to Pattern 1 can be considered winning), but most often the more vocal developers are driven out of the organization.

Non-responsiveness to customers is particularly a problem with organizations that either develop their software in-house or contract it through a central "software bureau"—in short, in any software organization that believes they have a "captive market." When you're competing for customers, non-responsiveness eventually puts you out of business, but in captive situations, it can go on endlessly.

to be continued in next issue...

Back To Index

Biography

Gerald Marvin (Jerry) Weinberg is an American computer scientist, author and teacher of the psychology and anthropology of computer software development.

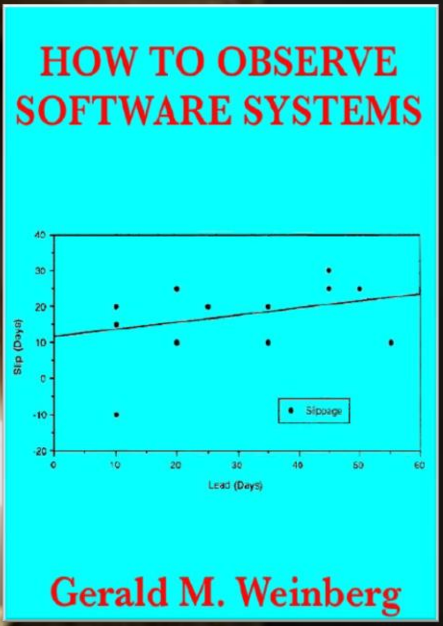
For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming. His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis and Design, The Handbook of Walkthroughs, Design.

In 1993 he was the Winner of the **J.-D. Warnier Prize for Excellence** in Information Sciences, the 2000 Winner of **The Stevens Award** for Contributions to Software Engineering, and the 2010 **Software Test Professionals first annual Luminary Award.**

To know more about Gerald and his work, please visit his Official Website here .

Gerald can be reached at hardpretzel@earthlink.net or on twitter @JerryWeinberg

HOW TO OBSERVE SOFTWARE SYSTEMS is one of the most famous books written by Jerry.

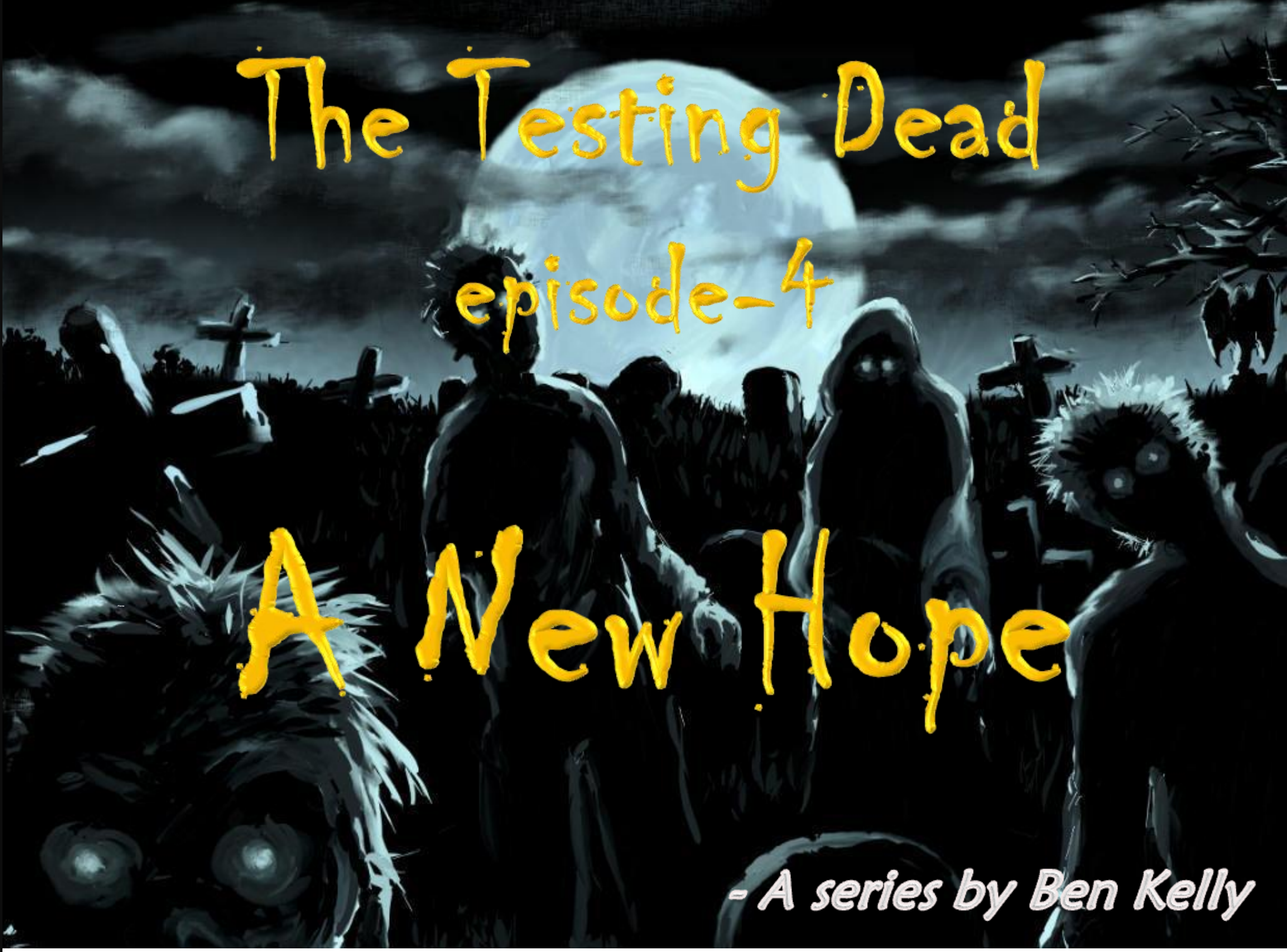


This book will probably make you think twice about some decisions you currently make by reflex. That alone makes it worth reading. "Great to understand the real meaning of non linearity of human based processes and great to highlight how some easy macro indicator can give info about your s/w development process." An incredibly useful book. Its sample can be read online here.

TTWT Rating: ★★★★★

To know more about Jerry's writing on software please click here .

Speaking Tester's Mind
- straight from the author's desk



[Please read part 1, 2 and 3 of this series in previous issues of Tea-time with Testers]

If it's true that zombie testers are being churned out faster than we can rehabilitate them, then what do we do about them? Asked in a perhaps less provocative way, how do we go about making sure that zombie-like testing behavior is neither encouraged, nor rewarded?

When you begin speaking with management types who have thus far only experienced zombie testing, when you engage them about thinking testing, you may well meet reluctance, disbelief and suspicion. A more highly skilled testing group sounds like a good thing, but how do you measure it? How do you make sure that people are doing what they are supposed to be doing? Skepticism is okay.

Testers should be able to explain themselves and their actions. Sometimes it's a little more than that though. Sometimes it's a deeply ingrained cultural issue, demanding adherence to procedures. To some extent, dealing with that sort of mindset and company culture goes beyond convincing them of the non-viability of zombies – there are apparently such things as zombie managers also. They tend to be the ones that roll out that little 'What gets measured, gets done' chestnut – to which I invariably reply 'what gets measured gets gamed'. Nonetheless, thinking testing can be an accountable activity.

If someone wants to be able to see numbers on how effective testers are being, then have members of a project answer a brief questionnaire on how the testers did during the project. How well did they identify and report on risk during design? How well did they find and convey information during hands on testing with delivered builds? Did testers provide information that allowed you to make informed decisions about the project? and so on – if you ask these questions of people the testers interacted with and have them rate them on whatever scale you like (as well providing a few open-ended questions such as 'what else could testers do / what information could they provide in order to be more effective' Then you can put that into whatever pretty charts and things you like and you have a basis from which to begin conversation that doesn't rely on nonsense like test cases executed or bugs reported. As an aside, you should be asking these questions throughout the project anyway. If you do, not only will you be able to alter your strategy when needed, you'll give your peers points of reference when they're filling in said questionnaire later.

We need the people we work with and the people we serve to understand what testing is in a way that is valuable to them – I spoke about this a little bit in the second post on this subject – not just within our own organization, but in a broader sense also. We need to make sure that programmers in general understand the value testing adds, the program managers, analysts, upper management understand the value of testing and to hold software testers to a higher standard, or at least to expect a higher standard from them.

My **previous article** in this series talked about taking steps in your own backyard. What about the wider world?

I semi-regularly attend gatherings for developers. I've given presentations on what developers should be expecting from testers. Initial reactions were interesting. I had some people say to me afterwards things like 'I'd expect anyone with that sort of skill set to be managing a development group or test group' or 'you're so wasted in testing, we need to get you into a programming role'. I think both comments stem from an overly low expectation of what value a thinking tester can add and my reply to both statements was basically that.

Now that they see that I am a tester by choice and I'm not looking to bridge into a 'better' role, they can see that value of having someone around that can speak the same language and brings a different set of skills to the table. Now they contact me for advice or ask me to help them hire skilled testers. I'm happy to do it. If you're not attending gatherings by non-testers, then have a look around for some in your area and turn up. Spend time with them. You don't necessarily have to evangelize or proselytize, just spend time and let them get to know you and what you do.

Attend non-testing conferences. This is a tough one. I'd like to see the **Association for Software Testing** put a grant together to let testers do exactly this. I think there is a lot of good that we could do by attending, and perhaps better still, presenting at non-testing conferences.

Offer to give guest lectures at your local universities. Rather than have young computer scientists indoctrinated to believe that testing is either synonymous with debugging, or something that the unwashed masses do. Help them understand how deep and diverse testing can be.

If you can't find any gatherings for non-testers (or even if you can), invite non-testers to your tester gatherings. Even if you have to bribe them with the promise of alcoholic beverages. Encourage your testing peers to do the same. They may come for the latter, but they'll likely stay for the conversation. Relationships formed off the clock can have a deep impact when it's time to clock on again, whether that be in your own environment, or someone else's.

The fight against zombie testing will not be won overnight and it won't be won by taking out the zombies already in the field. Thinking testers need to get amongst the thinkers of our non-testing peers and help them understand how much more we can do. When they start expecting a higher standard from the testers they work with, then zombie testers will start to find it more and more difficult to find work. Getting rid of the zombies will come down to thinking testers doing our part in whatever way we can. To mangle a phrase (probably) by Edmund Burke – The only thing required for zombies to triumph is for thinking testers to do nothing.

Ben Kelly is a software tester living and working in Tokyo, Japan.

He has done stints in various industries including Internet statistics , insurance and most recently online language learning.

When he's not agitating lively discussion on other people's blogs, he writes sporadically at testjutsu.com and is available on twitter @benjaminkelly.

Back To Index

There was a time when people did not have compass to find right direction. The only guide they had was that guiding star up in the sky.

Do you think that you are also stuck somewhere with technical issues? Do you need help in decision making or want guidance?

Well, the wait is now over . Introducing….

"The Guiding Star"

The panel of our experts is now here to help you.
Send us your questions around software testing and our Guiding Stars will help you out.

E-mail your question on –

theguidingstar@teatimewithtesters.com

Please Note :

1. This is not a job portal.
2. Typical interview questions will not be answered.
3. Questions should be on Software Testing or related topics only.

Image courtesy : MMVI New Line Production

International Conference on QA & TEST Embedded Software Testing

12th International Conference on Software QA and Testing on Embedded Systems

This year we offer a special program: **3 Keynotes, 2 Tutorials, 8 Tracks**, from renowned companies**.** Visit our website -www.qatest.org- to **discover the Programme!**

QA&TEST, the international Conference on Software QA and Testing on Embedded Systems, will be held on 29, 30 and 31 October in Bilbao, Spain, and gives you an excellent opportunity to increase your business network and establish contact with others professionals of the industry.

The main objective of QA&TEST is present the last technological developments in Software Testing and Quality Assurance, and showcase successful best practice to reduce costs and give companies a lead in global competition.

www.qatest.org

October 29 · 30 · 31
2013
Bilbao · Spain

Organiser

SQS SOFTWARE QUALITY SYSTEMS

Sponsor

innovalia ASSOCIATION

In the school of Testing
for your better learning & sharing experience

Addressing the RISK of Exploratory Testing part 3

- by Leah Stockley

In my experience, by far the biggest risk to successful implementation of Exploratory Testing (ET) is the complete misunderstanding of what is meant by the term. In this series of articles I've shared the concerns that have arisen whilst implementing ET and some potential solutions to overcome them.

In the first article I addressed how to help the test team gain the confidence to Exploratory Test. In the 2nd, I shared my experience getting stakeholder buy-in. Once teams & stakeholders understand and agree that ET allows for better testing, it's up to the Test Manager to ensure that the team deliver. This last article will make some suggestions for:

How to manage a team of Exploratory Testers

There are several concerns a Test Manager may face when contemplating the implementation of Exploratory Testing. Not least of which is…

Where to start?

Imagine embarking on a brand new project. Your stakeholders have bought into the efficiency gains so you are free to use Structured ET throughout your testing…. so, 'where to start' may not be an issue for you. That said, if you have never worked with a team of exploratory testers before, you might still be looking for a suitable start point from which to build your own confidence?

Have you considered when you may use ET already? Realising where we already use ET can help us gain confidence. What happens when the developers give you a bug fix release? Do you create new scripts for each bug? Or do you trust your testers have the knowledge to retest the bug? In this situation, a tester not only tests the fix, they also make informed decisions in order to perform regression testing. Understanding and testing whether the fix has broken something else is a perfect example of how teams use ET daily, yet rarely talk about it.

Perhaps you run your smoke/ sanity test on each new release from a checklist rather than a script? If so, your team are already using Structured ET (they have a documented mission to achieve, but some freedom in how to execute the test each time). When you understand how you already use ET, it becomes clearer where to start allocating official time on your testing project plan.

Consider trying ET at the beginning and again at the end of your testing cycles. Allow it at the beginning so your team can root out the more obvious showstoppers early on. Using ET at the end gives your testers a 'last chance to break it' session, once all other tests are complete. Either of these will allow the safety net of your usual scripted tests, plus the chance to gain metrics on how many more defects/ issues were found when you used Exploratory testing methods.

So you are convinced that, with the right testers, ET will definitely add value. If you are still reticent, may be you are really concerned about …

Are my team good enough to use ET?

Traditionally Test Managers see the number of new test scripts created each day. We may consider the output to see if our team are being productive. Numbers could make us challenge if John only wrote 5 tests today but 10 the day before. But can we realistically check the quality of every written test?

Structured ET actually allows a Test Manager to gain far greater insight into the quality of their team. By attending the Analysis sessions I talked about in article 1, the TM gains first-hand experience of the quality of test ideas and the knowledge levels of the team members. It offers you the chance to question their test design, and even help to improve it (before the work has been done). Being in the analysis session adds major benefits for the test manager. Not only closer, verbal interaction with team members, which is shown to improve morale, but what better way to get quickly up to speed with the project. Not all TM's have the chance to fully analyse the specs. If you have more than 10 testers working for you, how could you individually check the quality of everyone's work? By taking part in these sessions you not only improve your project/system knowledge but if any of your team is struggling to contribute, you will learn very quickly and have chance to take action before testing starts. If your team use mind-maps to create a visual test model (see http://lnkd.in/hxhwpm), you can also make a better assessment of the quality of your team's work.

Regular Debriefs

Another aspect of structured ET, which aids the test manager, is the Debrief sessions. At a logical point each day, the whole test team meet. Each member has no more than a couple of minutes each to discuss:

1. What they tested today & deviations from plan
2. Any challenges faced / risks identified
3. What they are testing tomorrow

You may feel too busy to attend this meeting regularly, but Test Managers who allow their testers to use ET are publicly stating, "I appreciate that things (scope, system behaviour) will change during the project, and our plan will need to adapt accordingly". To successfully manage an ET team, the TM should attend this meeting daily, so they keep up with what has been learned by the team. This is very similar to a Daily Stand up in scrum/ agile and just as essential to the success of ET. It's another chance for the team to question/ challenge each other on how they tested features and to share knowledge about problems faced. Lastly it's a great way to ensure nothing gets missed… and if it has been, at least you find out the same day.

Coaching Skills

We all hope that our manager will be a great coach. Someone we can look up to who can help us improve our skills. Encouraging ET usage in the team not only improves morale, but also gives fantastic opportunities to foster collaboration and coaching within the team. Pair testing is a great tool for this. Even 2 testers with no system knowledge will likely achieve more in an hour of pair testing than if they were left to work alone. If you have not been a hands-on tester for a while, why not get 'back to the floor' and spend an hour with each of your testers to see how they test and offer some coaching. If that sounds daunting (especially if you can't remember the last time you tested something yourself), challenge yourself! Be open to learning something new. Set an example. Foster a team spirit that allows some mistakes. Appreciate that they can be learned from. Make sure the team know there is no failure, only feedback that can help us all improve. Get your team discussing testing and various techniques/approaches. Even set mini-challenges/ competitions as incentives to demonstrate an improvement in test skills. Building confidence and awareness of testing skills is essential in an exploratory testing team … and will create a team of testers that you can be proud to manage.

What about new joiners?

How can we train new joiners without scripts? This question comes up in every course I run. I counter it with a show of hands to the question "Did you learn your current project by executing/reading other peoples' scripts?" Less than 10% learn by this method, yet us testers seem to think test scripts are the best source of knowledge. We have several teams who no longer use detailed test scripts. But, as per article 1, this does not mean we don't have any documentation. We build wiki's containing business/ technology information. We build Visual Test Models that are easy to learn from. We have checklists to guide testing. Combine this with pair testing sessions with an experienced tester and your new joiner will immediately feel part of the team. I've seen the evidence that our new joiners are now delivering value to the team faster than before. We have even had feedback from developers that our new remote-location testers are asking excellent questions! They are seeking information about the product, instead of asking dev to tell them how to test it. We are certain this is because knowledge was not handed to them. The expectation was set on day one that to contribute to this team, they must ask questions and develop their own test ideas. Additionally, they are immediately made responsible to ensure the wiki is up to date.

Obviously you need to ensure when interviewing new joiners that they are up to the challenge. They do not need prior ET experience but they do have to show they are capable of original thought and demonstrate questioning skills. And why would you want anyone in your test team who could not do this?

Over a period of some months, the ideas above start taking shape. You and your team are now confident that structured ET can enable efficient testing and earlier defect detection. You're ready to use it more widely but want to know…

How can a Test Manager plan and measure test coverage without scripts?

Simply put, by using high-level test scenarios, checklists or charters. These act like test scripts, allowing the TM to estimate, plan coverage and allocate testing amongst a team. If you aren't comfortable to change everything on your project, you can still map requirements and defects to these scenarios. If necessary, they can be used like scripts in all other purposes, just that they have less written detail and therefore take less time to create and maintain (time that can be better spent on analysis and capturing important info on the wiki). Of course this structure is needed in most companies, hence the term Structured Exploratory Testing. Another skill to develop is the ability to take good test evidence. My advice here is to take a look outside of the usual test management tools. Some excellent test evidence capture tools have emerged recently, which are either free or very affordable and feature rich. If you are also concerned how to report, read the section in my 2nd article, that explains how to report coverage without needing scripts.

Like the rest of technology, new solutions are developing all the time. As testers we also have to continually improve. We must develop skills that allow us to react to the changes around us. We must show our projects that we are a service to them, not a blocker. That we add value and are true system experts … else we may find ourselves sympathising with the dinosaurs!!

Structured Exploratory Testing (and the excellent analysis which must precede for it to be successful) has been the differentiator for the testers I work with. Every team who has taken on the challenge of learning and implementing SET, those who work hard to improve the way they communicate with the rest of the project team about testing, have achieved universally successful results, not just improved quality & efficiency in testing but in their reputation with the project team and the morale too.

You know you can achieve this in your team too, can't you?

Leah has more than 14 years experience in Software Testing for consultancies, banks and software houses across multiple industries.

Her Context-Driven awakening began 2 years ago. In that time she has become passionate about inspiring and coaching testers to be innovative, empowered and to continually improve the art of testing.

Currently working at a large global bank, Leah is responsible for rolling out the CDT approach across the Global Test Centre and developing solutions to the concerns that arise when introducing & applying CDT.

When not at work, Leah shares her experiences on www.inspiredtester.com

Back To Index

Hiring Great Testers [A How-to Guide]

by Johanna Rothman

Integrating the Tester with the Team

As the last part in this series, you've thought about what made a great tester for your team. You've considered how technical a tester had to be. You interviewed testers and made a decision to hire one of the candidates along with the rest of the team. You checked the candidate's references and extended an offer. The candidate accepted! Now, it's time to bring that person onto the project team.

You don't want to just dump the new person into unfamiliar territory, and hope the new employee figures out what's going on. But you know you don't have a lot of time to bring someone new up to speed. What are you going to do?

Advance Planning Helps

First, prepare for a great first day. Before your new tester even arrives for her new job, you want to ensure she has everything she needs. In the templates for *Hiring Geeks That Fit*, I share several checklists that I have found useful in my work. Note that the work for a new employee starts when the new employee has accepted your offer.

Orientation Checklist, Once the Candidate Accepts Your Offer

Activities to complete once the offer is accepted	**Check off when task is complete**
Order a badge, keys, and key cards, as needed.	
Identify suitable office space, and verify the space is clean and ready for a new employee.	
Verify the office has a desk, lamp, chair, phone, computer, and everything works.	
Order any needed furniture, office supplies, or computer equipment missing from the office.	
Order an email address, a voicemail extension, and physical mailbox.	

Why do I start that early? Because, sometimes these things take forever to accomplish. I once had a tester available to start three days after I extended the offer. And, it took our facilities people five days to organize the voice mailbox. I explained the situation to my new hire, and told him we still wanted him, and that I was more organized than our facilities people. He laughed and told me he understood the problem. I was lucky, he was so understanding.

Prepare, Prepare, Prepare

I use a different checklist to prepare in advance of the first day.

Orientation Checklist to Prepare for Day One

Activities to complete in preparation for Day One	**Check off when task is complete**
Stock the office with basic supplies such as pens, paper, pencils, wastebasket, scissors, stapler, staples, and staple remover.	
Verify email address works and computer is hooked up to the network.	
Verify the phone and email directories and location maps are available; add the employee's voicemail extension to the phone list.	
Identify the locations for all the applications and templates for the employee's work.	
Supply the employee with explanations for how to find help for the applications.	
Assign a buddy who can be available for the first month or so to answer the new hire's technical questions about how the team works and non-technical questions about staff, neighborhood, rules, and traditions peculiar to the specific environment and culture.	
Prepare a welcome letter and orientation package, including all HR forms.	

I use the checklist to make sure the new employee has everything he or she needs to be ready to work. I don't want someone to have a computer, but not be able to log in. And, HR always needs forms, so I want those ready and available. Sometimes, HR will provide a separate orientation, where they will have the forms ready. That's great—that's one less thing I need to do.

I ask for volunteers for the buddy system. When you bring someone into your project or your team, you will lose productivity. If you manage that loss by having one primary person as that new employee's buddy, you are more likely to bring the new employee up to speed quickly and to reduce the overall productivity loss. But you can only do this if you have a volunteer. Mandate this and you have angry people on your hands.

Be Ready for a Great First Day

Now, you are ready for the first day checklist:

Activities to complete when the new hire arrives on Day 1	**Check off when task is complete**
Introduce the new employee to project members, executives, personnel, and administrative staff.	
Show the employee instructions for calling meetings, for booking a conference room, and for other administrative procedures.	
Paperwork to collect for the new hire to fill out on Day One, to be packaged with a welcome letter and orientation packet.	
IRS, INS, and immigration forms, as applicable.	
Health, dental, and life insurance forms, as applicable.	
Benefits forms (long-term disability, short-term disability, and pension or retirement plans) as applicable.	
A nondisclosure agreement, if applicable.	
An emergency contact form.	
Direct-deposit and check-cashing forms.	
Business card forms.	
Paperwork copies to give to the new hire to keep.	
Maps, floor plans, and directions.	
Parking, public transportation, and commute information.	
Personnel and HR policies (conflict of interest policies; sickness, holiday, and vacation policies; lateness and absence policies; sexual harassment policies; conflict of interest policies; medical and personal leave policies; birthday lists).	
Any other paperwork that a new hire needs.	

I know, it seems as if all we do is paperwork. It seems as if we've covered some of this before. Well, depending on where in the world you work, you might have even more paperwork! If you complete it on the first day, you don't have to worry about it anymore.

Now, for the best part, the buddy system.

Integrate the New Employee with a Buddy

When you first assign a buddy to a new employee, the buddy acts as a guide, to explain where everything is. If you've never had a buddy system in place, this is a good time to document the locations of everything, including the physical facilities and tools. Once you've documented the locations, the buddy can explain once where to discover more information, and you've already farther ahead for the next new person.

On the first day, it's helpful if the buddy can take the new employee to lunch, preferably with your team or the project team in your cafeteria. If you have no cafeteria, choose a lunch location that most people use. Sometime during the first day, the buddy can demo your products or applications to the new hire.

After the first few days, the new employee knows where the cafeteria and the bathrooms are, and has probably started reading product and tool documentation. The buddy is available to answer questions about which test tools are where and how to use them, or how you use certain databases, how to submit a bug report, and so on.

During the first few weeks, discussing architectural or design tradeoffs with new developers, test strategies and choices with new testers, or project monitoring specifics with new project managers will help the new employee understand the context of how you work, which choices you've already made, and how to make sure the new employee's work fits into your environment. As the discussion moves past the basics, the buddy has moved from guide to mentor.

You may choose to have a formal ending to the buddy relationship after one month. If so, one way I've found useful is to mark the new hire's one-month anniversary in a group meeting and say, "John is now experienced enough to not need the formal services of a buddy. Thank you Steve, for taking on the role of the buddy. John, you can ask questions of any of us at any time, and hopefully you can take on the role of a buddy to a future new hire."

Agile Makes the Buddy System Easier

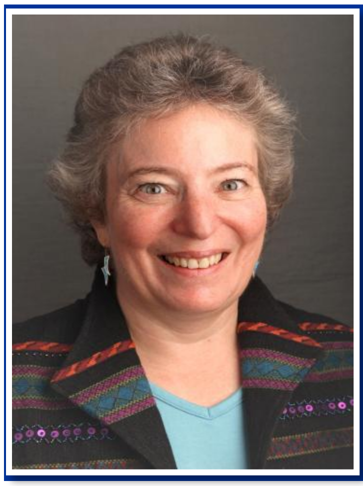
If you have an agile project team, it makes buddying even easier. You can pair the new tester with anyone on the team and the pair can buddy. You might not need a month before the new tester is independent.

Pairing might not work if no one pairs on your agile team, but swarming might work. Swarming is when everyone works on one feature together, to move the feature forward. In that case, the new employee can work alongside a more experienced employee, learning what the more experienced employee does.

Because the features tend to be smaller on agile projects, it's easier to get feedback on your work. If you buddy with a tester on an agile project, the tester can get feedback faster on automated and exploratory tests. The learning cycle is faster.

Hiring Ends When the Tester is Integrated on Your Team

When is your hiring done? When your new tester doesn't feel new anymore. Sometimes that takes six months. Sometimes it takes only three months. You want it to take much less than a year. What do you do to integrate testers on your team? Let me know your approaches.



Johanna Rothman is the author of Hiring Geeks That Fit, https://leanpub.com/hiringgeeks.

See her other books at http://www.jrothman.com/books/.

She writes an email newsletter, the Pragmatic Manager,

http://www.jrothman.com/pragmaticmanager/

Just like that… ;-)

OK SHARON - I DID SIX HOURS OF DEVELOPER TESTING, SO IT'S READY FOR YOU TO...
BING!
FOUND A BUG.
COULDN'T YOU AT LEAST WAIT UNTIL I GET BACK TO MY DESK?
I'LL HAVE ANOTHER ONE FOR YOU BY THE TIME YOU GET THERE.
D. WILBORN
©2009 DAVID WILBORN
WWW.URBANJUNGLECOMIC.COM
7/30/2009

Back To Index

Taking
a break?

a click here
will take you there

UNICOM
WORLD CONFERENCE – NEXT GEN TESTING
08-12 July, 2013 | Le-Meridien, Bangalore
register@nextgentesting.org
Rex Black
Keynote Speaker

UNICOM's Next Generation Testing Conference has been the most successful and widely acknowledged gathering of Software Testing Professionals. In 2012, the Next Generation Testing Conference took place in London, Bangalore, Mumbai, Chennai, New Delhi and Colombo. With over 850 attendees participating in India last year, it proved to be a highly informative and innovative event for software testers.

UNICOM is organizing "**World Conference - Next Generation Testing**" in Bangalore from **08-12 July 2013**. The first three days will consist of workshops presented by world acclaimed testing experts, and these workshops provide the highest standards of training available. Workshops will take place from Monday to Wednesday from 8.30am-5.00pm. The conference and exhibition takes place over the following 2 days on Thursday and Friday.

This conference is expected to have over 500 delegates, 45+ presentation, 25+ speakers, 10+ Workshops, 4 Parallel Tracks.

To view the Program& Workshops click here

To Register click here

When he was younger he had no idea what he wanted to do with his life. He started with a band that turned into his borderline compulsion for almost eight years….

A voice that has rocked concerts and shows years ago…is now one of the leading voices of Software Testing community. He is known for his multiple qualities in community. Some people know him because of his passionate blogging; some identify him as Headmaster of BBST classes and most of the people know him as 'Testhead'!

Meet… **Michael Larsen.**

What helped him reach there? What are his skills? What keeps him motivated?

Know it from the man himself….

over a cup of Tea with Michael Larsen

We would like to know about your journey as a tester. What made you choose software testing? (Do you mind if we ask about mention for your singing background?)

I don't mind at all. Truth is, when I was younger (read: about age 18), I was feeling fairly aimless. I had no idea what I wanted to do with my life, or who or what I wanted to be when I grew up. I met up with some friends at the college I was attending at the time, and our love of music helped us determine we wanted to start a band. We thought it would be a lark, but that lark turned into a borderline compulsion of mine for almost eight years. I learned several degrees worth of "life education" in those years. How to be my own boss. How to run my own company? How to be creative. How to sell. How to market. How to design. How to communicate. How to do public relations. Oh, yeah, and I learned how to sing and write music, too ;).

While all of that was wonderful, what it didn't do was provide me with much financial stability. Around the end of 1990 (when I had turned 23) I decided I need to do something that was more stable than what I had been doing (which was cleaning people's houses by day and rehearsing or performing at night). My drummer at the time suggested that I go to the temp agency he had contracted with, as they helped him land a very lucrative job some months earlier. He's a sales and design rep for audio systems, and he's really good at it, but back then he was just getting started in that, and the temp agency placed him at the start of his career. I figured, what did I have to lose? I put myself out there, willing to do anything, with two stipulations. First, I had to work days so I could perform at night, and I had to work somewhere that would be cool with me having long hair (seems funny now, but in 1990, I was the singer for a "hair band", so having long hair was officially part of the job ;-))

That conversation got me placed in March of 1991 at a then little known software and Hardware Company called Cisco Systems. I came in as a temp to help arrange their engineering library, and here is where being a housekeeper full time for several years totally came to my aid! A job they anticipated would take two weeks took me two days. With a lot of time on my initial contract, I was introduced to the Release Engineering team. They asked if I knew anything about networking or computers. I explained that my knowledge extended to Desktop Publishing of flyers with PageMaker, keeping track of expenses in Lotus 1-2-3, keeping a mailing list in dBase III Plus, and writing lyrics occasionally in WordPerfect.

The entirety of my computer experience, in fact, resided in four 2.5" floppy disks that I carried with me always. I knew a little bit of DOS, played around with Macs, never touched a UNIX system, but they figured I knew enough that I could at least do some things. With that, I was shown how to type into a UNIX terminal that was next to a Bytek EEPROM programmer. If I was willing, I could help them by peeling the labels off of used EEPROMS, putting them into a foam tray and exposing them to high intensity UV light, then taking them out and checking to see if they were truly erased. From there, by typing some commands into the monitor, I could "program" the EEPROMS. I then could push the EEPROMS into a system board, power on a system, and see if the image worked. From there, I'd give the EEPROMS to someone in the Release Engineering group, who would do something else with them.

Long story short, I learned how to do that well enough and quickly enough that the Release Engineering team asked if I'd be game to work with them longer, i.e. for six months or so. I agreed and, through that process, came to learn who the Release Engineering team was. Yes, some of them were build engineers, who compiled software into system images, but most of them, when you got down to it, were software testers. Thus, I came to software testing in a very lateral way, and stayed with it because I thought the people doing it were awesome (and many of them I am still friends with 20+ years later :-))

'Live Blog? He must be Michael Larsen!', is what we hear. What keeps you motivated? Writing 20 blogs a month, isn't it a big number?

My posting ability ebbs and flows. Sometimes I can post a lot, sometimes I post once or twice a week. It depends on a lot of what's going on at the time, and how engaged I am. "Live Blogging" was something I started doing because I found it frustrating to try to take notes and then make posts after the fact. I felt like I was leaving way too much good information behind, plus doing follow-on posts was more difficult, since I had to rely on both recall and a quiet time late at night or early in the morning to put together traditional blog posts. In 2011, I decided "why don't I just state up front that I'm going to "stream of consciousness" write what I am experiencing, and give people the choice to follow along?" I knew it would be chaotic, potentially messy, and at times less than coherent (and rife with spelling errors, too). The idea would be that I would come back later and "clean up" what I originally wrote, rather than try to create something from scratch from my memory. The response to doing this was very favorable, so now I will often "Live Blog" from events I am attending, ranging from local meet-ups to international conferences. What keeps me motivated to do it? I like to learn, and I like to encourage others.

Sometimes the best way to encourage others is to show my own struggles. Many blogs are presented as though the person writing is an authority or expert. I've never intended for that to be the approach for TESTHEAD. I've always strived to be a student, and to show what I am learning as I am learning it. Sometimes, it will make me look a little foolish or ignorant, and that's OK. The end goal is that I want to learn from my experiences, and if that learning helps inspire others to do the same (or something similar) then that's a bonus.

How much technical you think a tester should be? And what would you advise any tester to become good at it?

This is a multi-part question, and it can be approached from many different angles. When you say technical, do you mean that testers should be programmers in their own right? Do you mean that they should be experts in their particular knowledge domain? Do you mean that testers should be every bit as knowledgeable about the product as the end users of that product? Each means something a little bit different. Let's take them one at a time.

Technical == Programming Chops

Essential? Perhaps not. Helpful? Definitely! Does a software tester need to be as well versed as the software developers who are writing the actual code for the product? No, I don't think so, but if you have that interest and want to be, then go for it. If you are not enthused about programming, I do think you need to have some level of skill and understanding of computing to be able to communicate effectively with developers. That doesn't have to be a difficult or huge commitment. Programming at the Linux command prompt using bash or another shell scripting language can teach you a lot about conditions, flow control, variables, branching, functions, etc. Even if that were all the programming you ever learned how to do, you could be very helpful to your team just with that level of knowledge. If you can learn the programming languages that your development team uses, then that will certainly help you do more and be more effective with your team, but we shouldn't equate programming skill with effective testing. They are not the same thing.

Domain Knowledge

This I feel is critical. You could be an expert tester in one area, but that doesn't mean that you will necessarily be an expert in another area. I tested networking equipment for a decade, and that made me feel I could test anything. I was quickly corrected of that fact when I went to work for a company that made OS virtualization software. My networking skills helped a bit, but it was a totally different product and space, and it was an uphill climb to learn how to be effective in that space. I later worked with a company that made capacitance touch devices, and much of the conditions that made the product work had to do with physics and the way that the human body conducted electricity. That was a struggle, and ultimately I was unsuccessful there, because physics was not something I knew much about at the time. Later on, I worked with a company that made a product that helped immigration attorneys manage caseloads for people looking to get visas or apply for citizenship. Over the course of several years, I learned a lot about immigration law. Not enough to practice law, but enough so that I could understand the majority of the workflows that attorneys and paralegals had to go through. That experience helped make me a much more effective tester.

Application Knowledge

Software testers should be as skilled in the product as their top 5% of users. They don't need to be in the top 1%, or know everything about all aspects, but they need to be very in tune with the application, and as knowledgeable about it as a general Power User would be. In my mind, a truly effective tester should be able to step onto the customer service phone system and be able to talk to any customer about their problems, and be able to help get to the bottom of issues. Note, I'm not saying that a software tester needs to be a tech support person, but they should be knowledgeable enough about the product to slide into that role if needed. Customer Support engineers bring some additional skills to the table that some software testers will never be able to match, such as the finesse needed to talk a customer down off the ledge and calm them down, while at the same time helping them solve the problem, provide a workaround, or get to the bottom of an intractable (to them) issue. To this end, I have also often said that, if you want to, in a pinch, find some great software testers in your company without actually hiring another dedicated tester, check out the people working in tech support. Some amazing testers are already there!

Your opinion on the ROI of test Automation and the need of it?

Test Automation is a double edged sword. It’s important, but it’s over-sold, and often it’s improperly sold. Test Automation itself is not a silver bullet. Well crafted test automation, covering a broad area of mission critical areas, can be a lifesaver if a change to a code base breaks something important. I’m a huge fan of automated tests for regression purposes. Those tests have more than once helped me to determine if a new change has broken something important elsewhere. It doesn’t happen often (maybe 99% of the time or more, the tests just run and nothing of interest is noted) but during those rare occasions when something does happen, and we have determined that the error is not a spurious failure (hey, those happen too), they have pointed to real issues that needed to be fixed. The return on investment can seem really low if they don’t find any new bugs, but people who are looking at test automation to find new bugs are, in my opinion, missing the point. Test automation isn’t an investment in the classic sense. It’s a hedge. You are putting these tests in place with the idea that, for a vast majority of the time, they are not going to tell you anything new or exciting about your product... and that’s OK. Their value is not whether or not they find new bugs, their value is in seeing if newly added code elsewhere broke something that was known to work before. At that moment, your automation can very quickly become priceless.

SummerQAmp. Please tell us more about it.

SummerQAmp is an initiative here in the U.S.A. where students ages 16-24 can learn about software testing, and after doing so, get placed with companies as interns where they actually practice the skills that they have learned, with the goal to introduce these skills to students who may not have had a traditional STEM (Science, Technology, Engineering, Mathematics) education. Because of that, we try to make the topics we present start from the basics and work their way up. Currently we have modules that Cover “What is Testing”, What is Bias”, “What is Context”, “What is the Software Development Lifecycle” and “How Does the Web Work”. We also have a module that is in mid development on “Bugs and Bug Reporting”. I am also working with others to develop a “Lesson 0” topic around the Scientific Method, as we feel that would be a huge step towards helping potential testers really understand what it is that we do, and use science and history as a way to understand how software testing helps advance our knowledge.

So, do you see an alternative to formal education and costly degrees?

I think it definitely will help some people see the practical application of software testing, and it can be a part of a broader curriculum, whether that be through a traditional computer science degree, or through more self-directed learning. The goal is to help give potential testers both the understanding of what good software testing is, and to spark an interest for them to pursue it as a career in its own right. We also want to give them some solid skills that they can carry to whatever career they ultimately decide to pursue. The fact is, solid testing skills can help in many careers; finance, manufacturing, biology, geology, mining, and medicine all have areas where testing skills and the ability to think like a tester would be of great advantage.

What do you think about the test processes and maturity models? What would be your advice on reducing cost of testing?

I look at the maturity models and test practices the way that I look at test automation. I think they are important, but I also think they are over-sold. I am not a fan of the term “best practice”; I believe that a practice is only as good as the context it is being used in, and the ability to adapt and use different methods and models effectively is more important that following a prescribed plan. Note, in some fields that are regulated and require a certain rigor to allow a product to be released, then yes, there are protocols that need to be followed. Understanding those protocols and where they need to be followed does matter. It can cause a messy legal fight if they are not, so to say that they are unimportant would be ignorant. What I think we can do, though, is look at and evaluate what processes we use and which ones are effective. At the same time, we need to consider which processes are wasteful or are just there because of historical precedent and may no longer be relevant. Costs can go down when we are focused on not just what is required, but to also see what requirements just don’t need to be there any longer, and either get rid of them, or focus our energies on areas where we actually can be more effective.

It's hard to imagine AST EdSIG without you. We would like to know more about your work, experience with AST and Miagi -Do.

First, let’s clarify that the Association for Software Testing’s Education Special Interest Group is what we are talking about here ;) Second, it may surprise some people to know that prior to May of 2010, I had no involvement with AST, the BBST courses, or AST’s EDSIG at all.

In March of 2010, I made a commitment to become more involved with the greater software testing community, and by get more involved, I meant to do something where I actually contributed to the community. That started with creating a blog called TESTHEAD. My whole goal for TESTHEAD was to share what I knew about software testing, plus what I didn't, as well as what I hoped to learn. Through that process, I learned about Miagi-do, which is a zero profit school of software testing. I chose to embark on a journey of discovery with other instructors of the school. For those interested, Markus Gartner was/is my sensei in the school. Even though I'm now a Black Belt/Instructor in Miagi-do, I still consider Markus my sensei, and probably always will :). Anyway, through my becoming a student, Markus suggested I look at AST and the BBST classes, with the initial goal of participating in the classes, but with the stretch goal of teaching the classes. I offered to teach, and Cem Kaner & Becky Fiedler took me up on the offer :). After participating in, assisting in and leading several sessions of all of the classes, I am now one of the Lead Instructors for all of the course offerings. That knowledge, along with understanding the weird quirks of the delivery software that we use to offer the classes, helped make me a good candidate to lead the EdSIG when the opportunity arose. I also wanted to see AST move into offering education at a different level, from which the SummerQAmp initiative came our way. I am hoping to see more opportunities that can help us bridge the gap between the academic and the practitioner. I consider myself much more a part of the latter, but I want to learn as much from the former as I can as well :).

How do you see BBST and other courses by AST helping testers? Could you please tell how different they are from other certifications available in market?

First, the Black Box Software Testing courses (BBST) are skills based, and they are designed with the idea of helping people do what's referred to as "long term learning". While we use workshops and quizzes to help with the learning process, we focus more attention on people writing long form answers to questions, having those answers reviewed by other participants and instructors, and defending their answers (or showing how they have learned something different and incorporated that learning into their answers). We feel that is a great way to learn, and that it's a much deeper experience. It's also a lot tougher to grade.

Because of this, and many other factors, we decided that BBST would not be a "certification", but that they would be standalone courses that, if participants completed them, we would offer a certificate of completion to say that, yes, to our satisfaction, we feel that the participant did sufficient work and showed sufficient understanding that we, as instructors, and as an organization, would acknowledge that. We do not "certify" the skills of testers who have taken the courses, but I do feel that the certificate of completion for each course (Foundations, Bug Advocacy, Test Design and our Instructor's Course) shows a significant investment in their education and commitment to learning, and I commend each and every participant who has one of those certificates. They've earned them!

This discussion would be incomplete without talking about Weekend Testing Americas. Please tell us about it and how it can help new testers.

Weekend Testing Americas is, of course, a chapter that has grown out from the Weekend Testing movement founded in India. Ajay Balamurugadas and Markus Gartner encouraged me, because of my involvement with the sessions taking place in Europe, to help bring the initiative to the Western Hemisphere. Our sessions cover those from Honolulu, Hawaii to Sao Paulo, Brazil, from Nome, Alaska to Tierra Del Fuego, Chile, and everything in between. The idea behind Weekend Testing is that we offer a training ground for testers to come out and practice their craft. Very often, people only test the products that their companies make, and the ability to look at other products, techniques or ideas just isn't there. We offer a place to take on missions and charters, as well as look at interesting applications and technologies, and do so in a way that doesn't require anyone to follow a particular model or approach of testing. My primary role in the Americas sessions has been to facilitate them and to help testers who participate practice software testing, learn some skills and techniques/approaches, and meet some great people in the process.

What benefit do you think attending conferences can add to overall growth of software tester?

Conferences are terrific if you go to actually "confer". Before you sign on to attend a conference, ask yourself "Why are you going"? If it's to hang out with other testers and learn from them, that's cool, but you may find it more beneficial if you go with a handful of questions.

What technique could I use to help get a handle on test case overload? Could I learn more about a particular tool that we are considering? What is different about mobile testing or web testing as compared to traditional applications? Has someone else out there dealt with the pain and frustration of trying to implement a different testing approach? These are examples, but if you go with something like five questions in mind, structure your time so that you can get those questions answered, and give you ideas as to where to go from there, then yes, conferences can be time very well spent. Here's another hint... you may find that you get those questions answered while conversing with someone late at night at the hotel bar, or over lunch, or during one of the "tester games", or out in the hallway between plenary sessions. That's all fine. I have attended conferences where my key takeaways, more often than not, happened outside of the track sessions. Your goal is not to attend sessions; your goal is to get your questions answered and be empowered to do better work with the answers you receive. How you get those answers is up to you ;).

AST, SummerQAmp, Scout camps, Miagi-Do, blogging, singing, presenting at conferences, testing at work and Weekend Testing Americas. Are we missing anything? We are curious to know how you really manage it. And want your advice on how to maintain balance between multiple activities.

I may not be the best person to ask how to balance, because I tend to be an all-in or all-out kind of person. I don't "dabble" in activities, and honestly, I am terrible at apportioning time in a sparing manner. My time management approach is to find out which of my commitments are regular, and which ones are "special occasions". When I have a regular commitment, it's relatively easy to schedule for it. When I was producing a podcast every week, it was simple. I knew that I would need to devote about three to four hours over the course of a week to editing audio, setting up a show and having it ready to go by Thursday morning. It happened every week, so I made it work. I know that I have to report every day on my test status, so I can cover what I am able to between those meetings and put the attention that I need to into it. If I'm running a BBST class, I know when the deadlines are, and what needs to be met and when. The more difficult challenge is focusing on things that may not be regular, or urgent, but the payoff for doing them could be tremendous. Those are nebulous, and sometimes I find it very hard to dedicate time to those endeavors.

Give me a hard and fast deadline, and I'll make it happen; the closer the better :). Tell me "Oh, it's OK, get to that whenever you have time"... it may never come to pass. Sometimes I find it helpful to set up "class schedules" for certain things I want to do. When I do that, I try to give it the same amount of time I would to a class in a university (an hour a day three times a week or an hour and a half twice a week, etc.). The point is that I schedule it and I commit to that period of time. This works great for things that are "nice to do" or "I should really focus on that" kind of initiatives. Often, they are things that are not all that fun. On the flip side, I have to realize when I am spending too much time on the things that I find to be fun. Blogging for me is something I just go with. If I find I'm writing daily, or multiple times a day, I go with it. If I find that I haven't written for a while, then I shift to "office hours" so that I can write something. Newton's Laws of Motion fit very well here. If you start doing something and you do it regularly, you will find that you have more ideas and energy to keep doing it. If you stop doing something, you will find that it can be very hard to get it started again.

What are 5 important skills that you think a tester must possess?

Curiosity: Be someone who takes pleasure in poking something and saying "I wonder what happens if I do this!"

Empathy: You need to walk a mile in the user's shoes to really understand what annoys them.

Persuasion: Developers will consider a bug serious if you can show and persuade them it is serious.

Flexibility: Be willing and able to adapt to changing situations.

Patience: Know that testing can be a long haul, and sometimes it's not a lot of fun. Hang with it, though. Even the worst of times shall pass.

What would be your advice to our readers? And especially for people who want to become the next Michael Larsen?

Please don't aspire to become the next me. Become the first you. That's way more important! If, however, I somehow inspire others to want to become a more dedicated tester, or get more involved in things that matter to them, then that would be a great legacy. I'd suggest finding an initiative you care about; one you would be willing to give your all to. It might be an open source project. It might be an education initiative; it might be trying to make the next great plug-in to some automation tool. Whatever it is, find it, get involved and be of service to the testing (and broader) community.

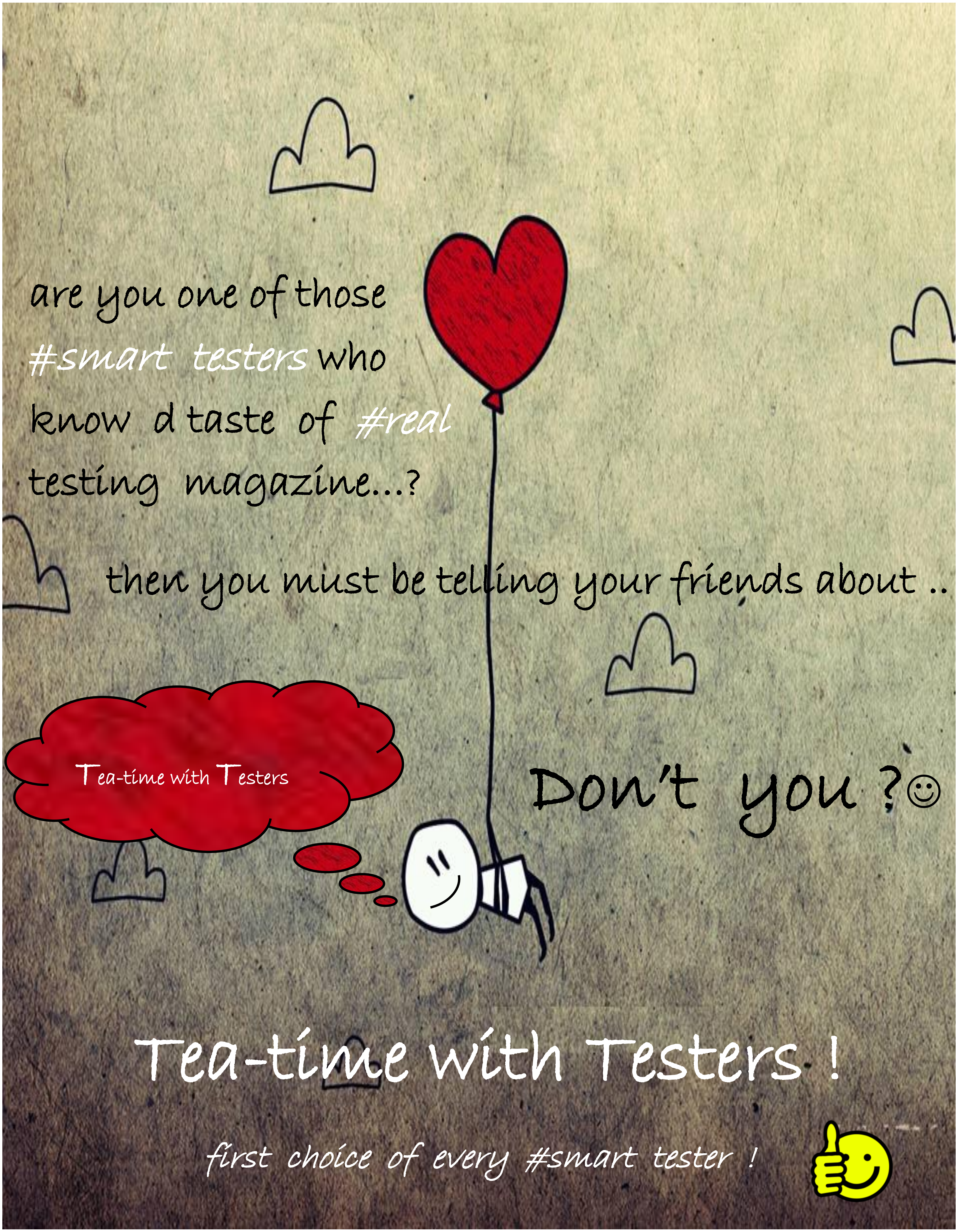
Do you read Tea-time with Testers? We would love to hear your opinion about us.

I do read Tea Time with Testers. I like the fact that it is not specifically aligned with a particular school or approach. Too often, we see articles that are all about "best practices" or "Time honored traditions", and while there is a place for that, I want to see what may not be so common, or may be rather fringe in idea or opinion. I'm a big fan of real world and actual practitioner based information. I also like the fact that Tea-time with Testers reaches out to a broad community to help provide that balance. Alongside well recognized voices (James Bach, Johanna Rothman, Jerry Weinberg, etc.) there are also opportunities every issue for people like me to present their ideas and see them get traction with their peers. I consider it a good place to share (and find) real world stories about real world software testing and what actually works. If nothing else, it lets me know that I'm not the only one dealing with certain issues :).

Thank you Michael for your valuable time as well as for kind words about Tea-time with Testers. On behalf of TTwT family, I would like to thank you for the video pitch.

Dear Readers, you can watch below video to know what Michael and other experts have to say about us. Feel free to share it with your friends and fellow testers.

- Editor


are you one of those #smart testers who know d taste of #real testing magazine...?
then you must be telling your friends about ..
Tea-time with Testers
Don't you ?☺
Tea-time with Testers !
first choice of every #smart tester !

testing intelligence

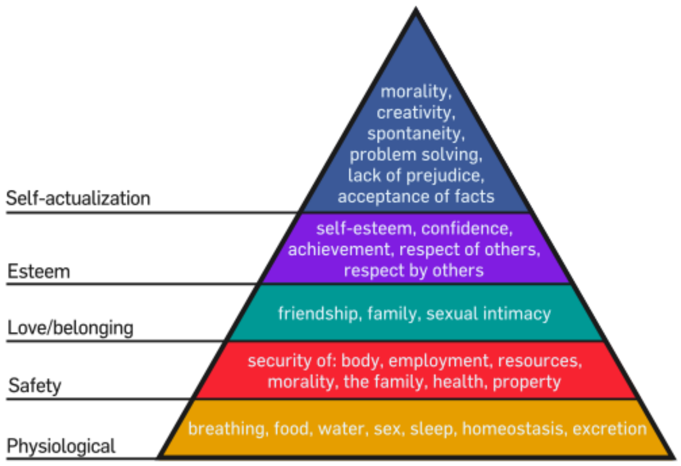
- *its all about becoming an intelligent tester*

an exclusive series by **Joel Montvelisky**

Is friendship ruining your testing career???

I am sorry to say that I have some bad news for you today…

Humans are social beings, and as such we have the instinctive and almost unavoidable need to be liked by our peers.



morality, creativity, spontaneity, problem solving, lack of prejudice, acceptance of facts
Self-actualization
self-esteem, confidence, achievement, respect of others, respect by others
Esteem
friendship, family, sexual intimacy
Love/belonging
security of: body, employment, resources, morality, the family, health, property
Safety
breathing, food, water, sex, sleep, homeostasis, excretion
Physiological

I'm not making this up!

Maslow went as far as listing *belonging* as one of the primary psychological needs in his famous pyramid, on top of *Safety* but less important than *Esteem*.

This means that for you it is (obviously) extremely important to make sure no one is running behind you with a loaded gun; but right after that it is more important for you to feel that your group likes you than it is to know they appreciate you as a professional tester.

If you haven't understood how this is making you a bad tester, then let me explain a little further…

We are paid to criticize our peers

Our main task as testers is to criticize the work of our peers and to provide visibility into the status of our projects.

Criticising in itself is not a bad thing!

The definition of the word is actually "*to consider the merits and demerits of (something) and judge accordingly*". For example think about the movie critic who writes a positive review of a movie, or the food critic who recommends a restaurant to his readers.

For us as testers this criticism is translated, in the simplest ways, to:

- Test the product developed by our programmers and set our tests to pass or fail.
- Report bugs when something was not done in the way it should have been done.
- Raise flags when we understand the team is not working in a way that will allow the project to be released on time, on scope or with the proper level of quality.
- Write reports that give an overview of the status of our project: Good or Bad, and why?

In short we are charged with the job of investigating the work being done by our peers, and reporting to management whether this work is good or bad for the project.

But I don't want to get my friends into trouble…

When the work is good then we obviously have no problem telling this to our bosses.

But when the work is bad: when tests keep failing all the time, when bugs are raised in large numbers, when the application is not on the state it should have been by this stage of the project… Then we need to come to our management with bad news that will most certainly get some of our "friends" into trouble.

It is normal to feel remorse for escalating bad news to management, specially when we know that our friends are not completely to blame for all the delays and issues happening in the development process. So what can you do?

Maybe we need to be "team-players" and help our friends?

How about we stop reporting all the bugs in the Bug Tracking System? Instead we can send "off-the-record" emails with issues found to our developers, and this way the statistics don't look so bad for them.

But then what will happen when one of these bugs escapes from the system and is found by a customer? At that time we will be asked how come we didn't find this critical bug during the testing process???

Or maybe we can delay some of our tests because we know that the system is not ready and most of them will fail.

Back To Index

Then what will happen when the project manager comes and says that we are the one to blame for the delay of the whole release because we were not able to run our tests on time???

All of a sudden, because we became "team players" and didn't do our jobs correctly, we became the focus for all the issues in the project!

The best path is to be transparent and professional in our work

It is understandable that you don't want to get any of your "friends" into trouble, these are the same guys you are having lunch with every day and the same guys with whom you play poker or basketball one night a week.

The best way to help your team mates is by using the same tools that may harm them in order to improve the way they do their work.

For example:

Give them constant visibility into the statistics reflected by your bug findings. Have a process and a mechanism in place that helps them to see the "story" told by the bugs being found and their trends. Help them to use this information for their own benefit and to have the ways of proactively informing management of any issues before you get to them with the "bad news".

Make sure they are constantly aware of your testing approach and schedule. If you are going to be doing extensive tests on the new functionality give your team the time to have the feature ready on time, share with them the tests you will run so that they are not "surprised" by the bugs that you find. Remember that you work together so you don't need to "hide" your scenarios from them in order to find more bugs.

Share your results with the development before you send them management, this will give them time to add their own comments and so help you provide the best possible picture of the situation.

In two words: Working Together

It took me some years to realize this, but as testers we don't work against our developers, we work for them!

Part of our job (among other things) is to help them write beter software, and to do this as quickly and effectively as possible.We need to make sure they understand that we want to help, and at the same time to educate them in the ways in which they can help us to do this job better (but that I will leave how to educate them to a different post).So, the next time you feel that your friendship is getting in the way of your work, remind yourself that you are there to help your friends in their work and not to cause them any harm.

Think about the PROFESSIONAL things you can do in order to assist them in their work.

Joel Montvelisky is a tester and test manager with over 14 years of experience in the field.

He's worked in companies ranging from small Internet Start-Ups and all the way to large multinational corporations, including Mercury Interactive (currently HP Software) where he managed the QA for TestDirector/Quality Center, QTP, WinRunner, and additional products in the Testing Area.

Today Joel is the Solution and Methodology Architect at PractiTest, a new Lightweight Enterprise Test Management Platform.

He also imparts short training and consulting sessions, and is one of the chief editors of ThinkTesting - a Hebrew Testing Magazine.

Joel publishes a blog under - http://qablog.practitest.com and regularly tweets as joelmonte

Call for Articles !

Have you got something to say?

yes, we are listening you...!!!

"Tea-time with Testers" firmly believes that one of the best ways to improve upon software testing is to listen to the lessons learned by others and their experiences too.

So, if you have an interesting story that you'd like to share with the world, contact us at teatimewithtesters@gmail.com.

Submit your articles, stories, thoughts around software testing.

now its your chance to be heard...!

Click HERE to read our Article Submission FAQs!

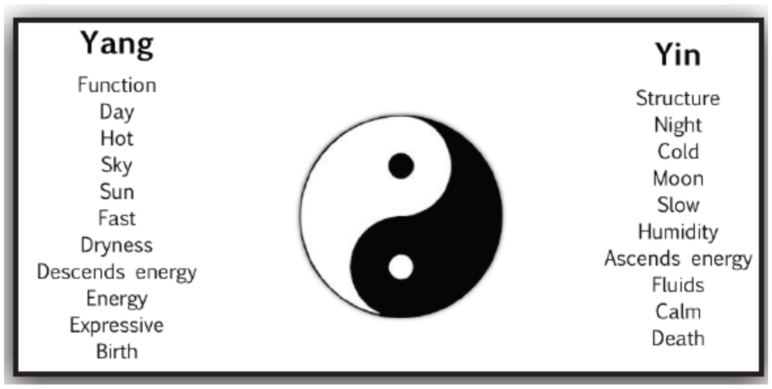
sciencephotogallery

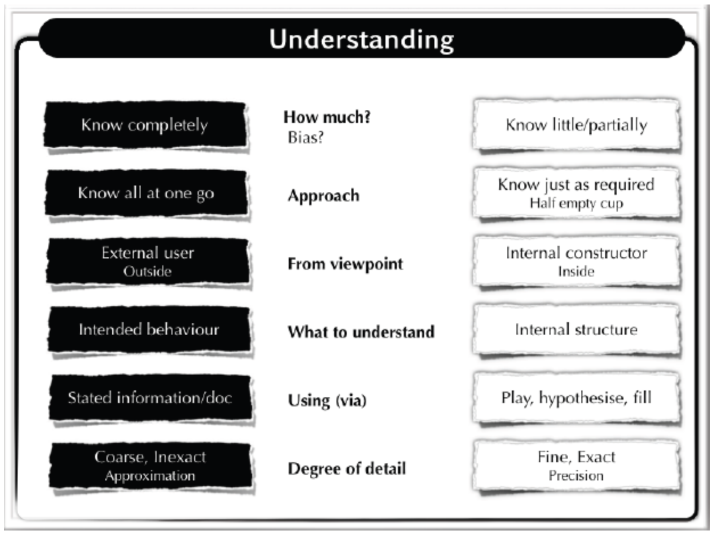
Yin and Yang in Testing: The magic is in the middle

As we mature we see more opposites. Is the objective of testing to find more issues or prevent them by being more sensitive? Is behavioral information more suitable that structural information to design test cases? Is automated testing superior to finding issues?

There are very many things in the discipline of testing that seem contrary, like the Yin and Yang, creating a constant tension as to 'what-to-choose'.

This explores the idea the various opposites and outlines the view that "The magic is in the middle". That it is not a tussle, but a perfect state with the opposites balancing each other.



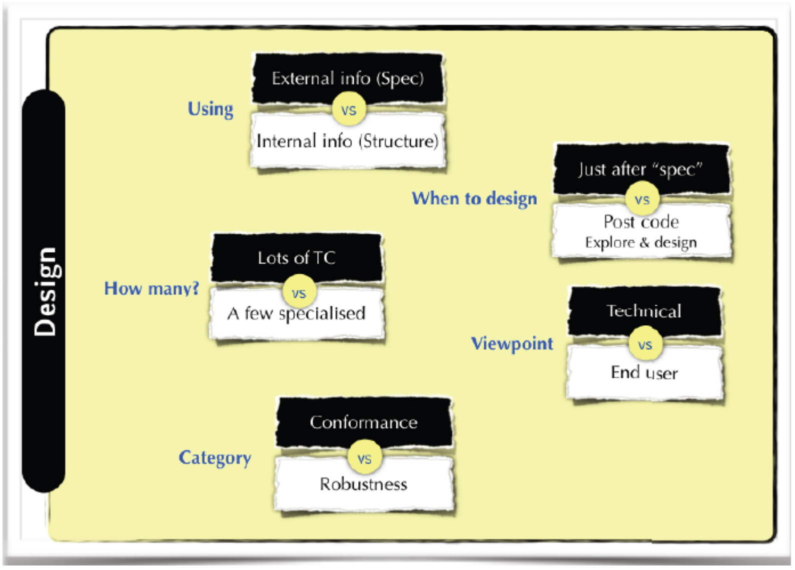


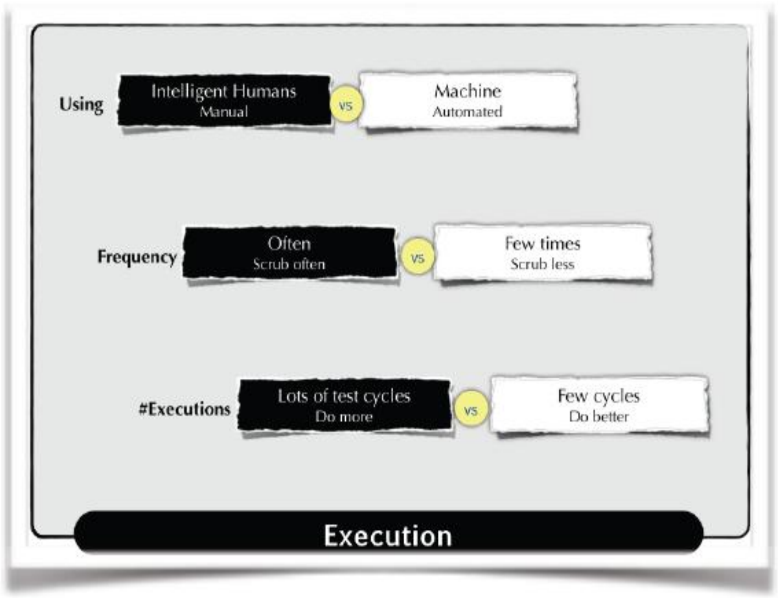
Let us look at the act of understanding ... Should we know the entire system completely, in detail? Will this bias us? Should we attempt to understand the whole at one go or should be understand only as much required and defer the rest to later? What should we understand - how the system should-work/is-working or how it is architected/built? And what is the granularity to which we need we understand - precise or approximate? And do we read the spec and understand or play/experiment and figure it out?

As we can discern from here, it is 'neither this nor that', but something in between. What the middle is; is based on experience. And experience is in knowing the governing variables that enable decision making. Some are these are: (1) state of the product/application (brand new Vs existing), (2) area of complexity (internal i.e. structural or external i.e. behavioral) (3) degree of availability of information. And the applications of general principles like: onion peeling, experiment in addition to trust, approximate and iterate.

Let us to explore the opposites in test design. Should we have a lot of test cases that cover a lot more or have focused & specialised test cases? Which information is more suitable to design test cases - external behavioral information or internal structural information? Should we take the viewpoint of end user or deep technical viewpoint to design? And finally how much positive and negative test cases hold we look forward to? And once again, it is neither the extremes, it is the middle!

And what are the governing variables here ? Some of these are: quality level for which we are designing (early stage building blocks, or end user use cases), #inputs to combine to generate test cases, type of defect we are looking forward to uncover.

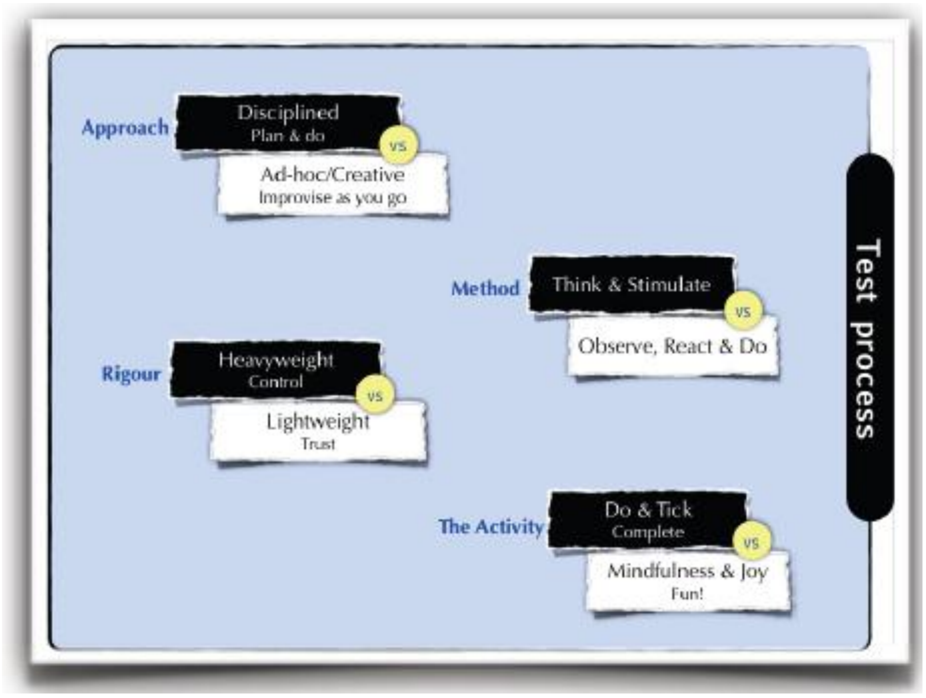




Moving onto the aspect of test execution- Is automated testing better that using intelligent humans? Should we do more cycles of execution or less? And finally should we test more often? i.e. scrub more.

The middle is based on some of these governing variables - is the objective stability/cadence (health check), defect yield, feature addition frequency.

Lastly examining the process of validation, how heavy/light should the process be, how disciplined versus creative should we be, how much of thinking (a-priori) versus observation & learning should we adopt, and how much compliance & control Vs freedom? A process is most often mistakenly understood as an inhibitor of creativity, when the real intent is to 'industrialise' the common things, ensure clarity in interactions so that we can channel our energies from the mundane stuff to higher plane to deliver performance. Doing this requires us to be mindful, be in the present and 'enjoy the doing'. Some of the variables that govern this are: complexity, error proneness, degree of availability of information, fun factor.



Now the big question - is the objective of testing to detect issues or to enable us to have a heightened sensitivity so that we can prevent issues? You figure this out.

Summarizing, there exist opposites like the yin and yang. The trick is to find the middle that is harmonious, requiring the right mix of two, the polar opposites. Not by a compromise, but via a wise choice of the middle discovered by using by a set of governing variables gained with experience. The harmony that we experience in the middle when we balance the two ends is the "magic".

On a philosophical note, zero is said to be infinity! This means when you are empty, unattached, you are filled with bliss! And they are not really opposites as we perceive, the magic is the middle- 'the bliss'. As a long distance endurance cyclist, I discovered the magic, the bliss when I focused on the front wheel rather than the mile marker during long rides lasting multiple hours. Not to be worried about the distance to destination, or judge the performance in the distance covered, but to enjoy the cycling. The magic is to be in the present, to be mindful. Not the past, nor the future.

The magic is in the middle.

Recognise it; exploit it harmoniously to deliver high performance.

Cheers.

T Ashok is the Founder & CEO of STAG Software Private Limited.

Passionate about excellence, his mission is to invent technologies to deliver "clean software".

He can be reached at **ash@stagsoftware.com**

Back To Index

by

Testing PUZZLES by Sebi

Claim your Smart Tester of The Month Award. Send us an answer for the Puzzle and Crossword bellow b4 15th August 2013 & grab your Title.

Send -> teatimewithtesters@gmail.com with Subject: Testing Puzzle

List all level one directories from https://www.google.com/

Example: https://www.google.com/tagmanager/ (good)

https://www.google.com/tagmanager/web (not good, because is not level one directory)

Note: In the context of security bug bounty here, each main sub-folder is usually a different application, with different functionality. Each new found application can be a juicy ground for bugs, thus rewards.

Back To Index

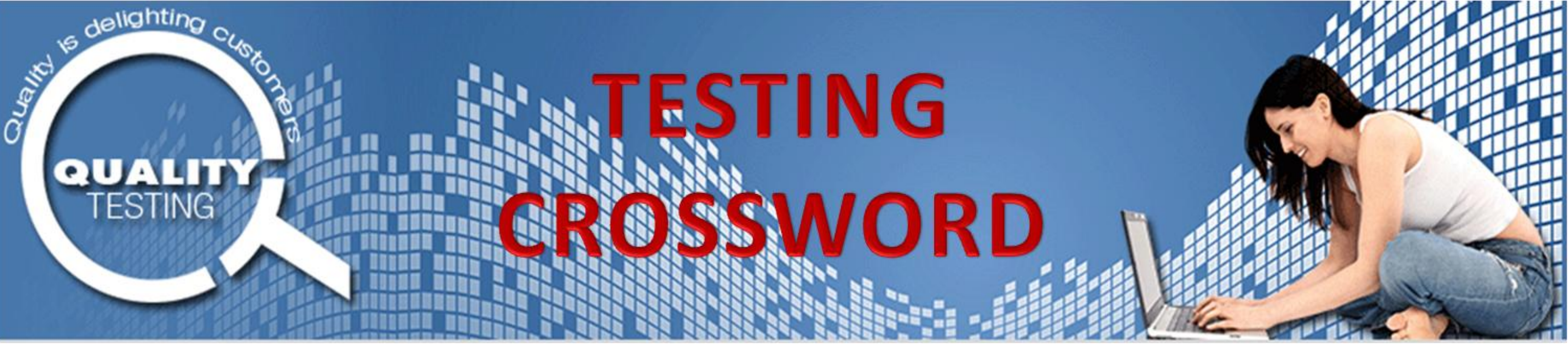
Biography

Blindu Eusebiu (a.k.a. Sebi) is a tester for more than 5 years.

He considers himself a context-driven follower and he is a fan of exploratory testing.

He tweets as @testalways.

You can find some interactive testing puzzles on his website www.testalways.com



TESTING CROSSWORD

1		2		3			4
	■		■		■		■
5		6			■	7	
	■		■	8			■
	■	9			■	10	
11			■	12			■
	■		■	■	■		■
13							

Horizontal:

1. Any data developed for use in tests (8)

5. It is a technique used during the software development life cycle for software component and code error detection prior to application execution, in short form (2)

6. The first executable statement within a component, in short form (2)

7. The first executable statement within a component, in short form (2) Performance testing focused on ensuring the application under test gracefully handles increases in work load, in short form (2)

8. It is often the final step before rolling out the application (3)

9. It is a black-box GUI test automation tool. Its first 3 words (3)

10. Testing the ease with which users can learn and use a product, in short form (2)

11. Verifying a product is accessible to the people having disabilities, in short form (2)

12.You do it when bug needs an immediate fix (3)

13.A device that duplicates the functions of one system using a different system, so that the second system behaves like the first system (8)

Vertical:

1 _______ is the step wise description or activities which are going to be executed in order to validate the application (8)

2. It is a Mobile Test Automation tool for Android, iPhone, Blackberry, Symbian & WindowsPhone 7 (7)

3. It is an open-source web site stress test tool (6)

4. It is a provider of SaaS-based Test Management service for managing and executing manual and automated software tests and for reporting defects (7)

Answers for last month’s crossword:

N	G	R	I	N	D	E	R
B	■	T	■	T	■	P	■
E	G	G	P	L	A	N	T
H	■	L	■	T	■	■	E
A	■	A	G	I	L	E	S
V	T	S	■	T	■	■	S
E	■	S	■	P	■	■	Y
■	F	U	N	T	A	S	Y

We appreciate that you

“LIKE” US !

Join us on Facebook.

You are just a CLICK AWAY

Note- We have declared winners’ names on our Facebook page.

Every Tester
who reads Tea-time with Testers,
Recommends it to friends and colleagues .
What About You ?
Image : vernhart

in next issue

articles by -

Jerry Weinberg

T Ashok

Joel Montvelisky

Bernice Ruhland

Mike Talks

Rajesh Mathur

our family

Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)

Lalitkumar

Pratikkumar

Contribution and Guidance:

Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)

Jerry

T Ashok

Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover page image – Abhijit Juvekar

Core Team:

Dr.Meeta Prakash (Bangalore, India)

Dr. Meeta Prakash

Testing Puzzle & Online Collaboration:

Eusebiu Blindu (Brno , Czech Republic)

Shweta Daiv (Mumbai, India)

Eusebiu

Shweta

Tech -Team:

Chris Philip (Mumbai, India)

Romil Gupta (Pune, India)

Kiran kumar (Mumbai, India)

Kiran Kumar

Chris

Romil

|| Karmanye vadhikaraste ma phaleshu kadachna |
Karmaphalehtur bhurma te sangostvakarmani ||

To get **FREE** copy,

Subscribe to our group at

Google™

Join us on

facebook

Follow us on

www.teatimewithtesters.com

Give Feedback